Recording Provenance for SQL Queries and Updates

Stijn Vansummeren*
Hasselt University and
Transnational University of Limburg, Belgium

James Cheney
University of Edinburgh
UK

Abstract

Knowing the origin of data (i.e., where the data was copied or created from)—its provenance*—is vital for assessing the trustworthiness of contemporary scientific databases such as UniProt [16] and SWISS-PROT [14]. Unfortunately, provenance information must currently be recorded manually, by added effort of the database maintainer. Since such maintenance is tedious and error-prone, it is desirable to provide support for recording provenance in the database system itself. We review a recent proposal for incorporating such support, as well as its theoretical properties.*

1 Introduction

Chris, a fan of foreign and domestic beers, constructs a database $R(\mathsf{beer}, \mathsf{kind}, \mathsf{origin})$ listing beers, their kind, and their origin. He proceeds by manually inserting tuples, as well as by copying from the existing general beer database $S(\mathsf{beer}, \mathsf{kind}, \mathsf{origin})$, and from $T(\mathsf{beer}, \mathsf{origin})$, a database that specializes in lagers.

$$\mathsf{insert\ into}\ R\ \mathsf{values}\ (\text{'Duvel'},\ \text{'Strong ale'}, \text{'Belgium'}); \tag{1}$$

$$\mathsf{insert\ into}\ R\ (\mathsf{select}\ *\ \mathsf{from}\ S\ \mathsf{where\ origin} = \text{'USA'}); \tag{2}$$

$$\mathsf{insert\ into}\ R\ (\mathsf{select\ T.beer},\ \text{'Lager'}\ \mathsf{as\ kind}, T.\mathsf{origin\ from}\ T); \tag{3}$$

When inspecting the result, Chris notices that R reports Stella Artois as an American beer, while it is in fact a Belgian one. A friend tells Chris that this error is probably due to database T, which is known for its poor data quality. Perhaps Chris should check the other records inserted from T for their correctness?

Although the scenario above is clearly a simplification for the purpose of illustration, many contemporary scientific databases—sometimes referred to as *curated databases*—are constructed in a similar manner by a labor-intensive process of copying, correcting, and annotating data from other sources. The value of curated databases lies in their organization and in the trustworthiness of their data. As illustrated above, knowing where data was copied or created from—its *provenance*—is particularly important in assessing the latter.

In hindsight, Chris could simply have recorded provenance by adding an extra attribute prov to R and by issuing slightly different update statements. For instance, update (3) would have become

$$\mathsf{insert\ into}\ R\ (\mathsf{select}\ T.\mathsf{beer},\ \text{'Lager'}\ \mathsf{as\ kind}, T.\mathsf{origin},\ \text{'db T'}\ \mathsf{as\ prov\ from}\ T). \tag{4}$$

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Stijn Vansummeren is a Postdoctoral Fellow of the Research Foundation–Flanders (FWO).

This only records provenance of whole tuples, however, and a finer granularity is often required. For instance, the presence of $(\mathsf{Stella\ Artois}, \mathsf{lager}, \mathsf{USA}, \mathsf{db\ T})$ in R would designate T as the provenance of the tuple $(\mathsf{Stella\ Artois}, \mathsf{lager}, \mathsf{USA})$ even though this tuple does not literally occur in T. After all, only Stella Artois and USA were copied from T, but lager was inserted by Chris. On the other hand, recording only the provenance of data values (like Stella Artois, lager, ...) is often not sufficient either. For instance, knowing that all data values in $(\mathsf{McChouffe}, \mathsf{Scottish\ ale}, \mathsf{Belgium})$ were copied from S does not necessarily imply that all of these data values came from the same record in S. As such, it is desirable to record provenance at all levels in a database (data values, tuples, and even whole tables). While it is possible to do so manually by adding enough extra attributes to R and by suitably rewriting the original updates (1), (2), and (3), this approach quickly becomes very tedious, time-consuming, and error-prone, especially when R contains many attributes. Also, the importance of retaining detailed provenance information is often not appreciated until it is too late—perhaps months or years after the data was originally copied into the database. In this respect, it seems preferable to let the user write queries and updates as before, and to let the database system record provenance *automatically*.

Before provenance recording can be automated, however, it is imperative to have a good explanation of the meaning of queries and updates with regard to provenance. This meaning may be obvious for the examples given so far, but what about updates such as the following?

$$\mathsf{insert\ into}\ R\ (\mathsf{select}\ S.\mathsf{beer},\ \text{'stout'}\ \mathsf{as\ kind},\ S.\mathsf{origin}\ \mathsf{from}\ S\ \mathsf{where}\ S.\mathsf{kind} = \text{'stout'}) \tag{5}$$

Is stout created by Chris or copied from S? Both explanations seem reasonable due to the condition that $S.\mathsf{kind} = \text{'stout'}$. A similar situation occurs for updates involving joins:

$$\mathsf{insert\ into}\ R\ (\mathsf{select}\ S.\mathsf{beer}, S.\mathsf{kind}, T.\mathsf{origin}\ \mathsf{from}\ S, T\ \mathsf{where}\ S.\mathsf{beer} = T.\mathsf{beer}\ \mathsf{and}\ S.\mathsf{origin} = T.\mathsf{origin};) \tag{6}$$

Are the beer and origin attributes copied from S or from T? Again, both explanations seem reasonable, but it is unclear which explanation is to be preferred.

In this article, we will follow Wang and Madnick [17] and Bhagwat et al. [2] and define stout to be created by Chris in update (5) because stout appears as a constant in the select clause instead of $S.\mathsf{kind}$. Moreover, we define beer to be copied from S in update (6) because the select clause lists $S.\mathsf{beer}$ and not $T.\mathsf{beer}$. Similarly, origin is taken to be copied from T because the select clause lists $T.\mathsf{origin}$ and not $S.\mathsf{origin}$.

While this provenance semantics is simple and natural, it may not be the particular provenance that a database curator had in mind for the above updates. Nevertheless, this simple provenance semantics has been shown *expressively complete*: for every query or update O that manually records provenance (like (4) above) there exists a normal query or update (like (1), (2), and (3) above) for which the provenance semantics is equivalent to O, provided that O satisfies certain soundness criteria discussed in Sections 2 and 3. As such, if we use this semantics to record provenance automatically, we do not lose flexibility with regard to recording provenance manually. We feel that this property strongly argues in favor of the proposed provenance semantics as the "right" basis for recording provenance automatically.

We should note that although we will restrict ourselves in what follows to the provenance semantics for (a fragment of) SQL queries and updates operating on classical flat relations and only consider provenance at the data value and tuple level, the topic was originally studied for queries and updates operating on *nested* relations, where provenance is recorded at all levels (data values, tuples, and tables) [5]. We refer the interested reader to Buneman et al. [6] for a full exposition.

To put this article in the right context, we should also mention the other forms of provenance recently studied in databases. First, while we are interested in recording *where* data is copied or created from, the *why*-provenance approach of Cui et al. [**?**] and Buneman et al. [**?**] wishes to identify, for each output tuple t of a query, the set of input tuples that caused t to be output. More recently, why-provenance has been refined using program slicing techniques by Cheney et al. [8]. The *how*-provenance approach of Green et al. [**?**] is interested in recording how t was computed from the input (e.g., t could be the result of joining two input tuples). Finally, there has also been interest on *querying* provenance and other forms of annotations rather than recording it [12, 11].

beer	origin	beer^c	origin^c	tup^c
Leinenkugel	USA	c_1	c_2	c_3
Stella Artois	USA	c_4	c_5	c_6

beer	origin	beer^c	origin^c	tup^c
Leinenkugel	USA	c_1	c_2	c_3
Stella Artois	Belgium	c_4	$\perp$	$\perp$

Figure 1: Color propagation for query (7). On the left is $\mathsf{clr}(T)$, the colored version of table T, and on the right is the colored result.

2 Provenance Recording for Queries

Let us first consider provenance recording for queries. Updates will be considered in Section 3. For ease of exposition we restrict ourselves to simple SQL queries of the following form, excluding subqueries; grouping; and aggregation.

$$\begin{array}{rcl} Q & ::= & \mathsf{select}\ r_i.* \ \mathsf{from}\ R_1\ r_1,\ \ldots,\ R_m\ r_m\ \mathsf{where}\ \varphi \\ & | & \mathsf{select}\ a_1\ \mathsf{as}\ A_1,\ \ldots,\ a_n\ \mathsf{as}\ A_n\ \mathsf{from}\ R_1\ r_1,\ \ldots,\ R_m\ r_m\ \mathsf{where}\ \varphi \\ & | & Q\ \mathsf{union}\ Q \end{array}$$

Here, φ is any valid where-clause without subqueries and every a_i is either a constant data value or an expression such as $r_i.C$ that refers to an attribute of one of the tuple variables. Our approach can be generalized to deal with subqueries, grouping, and the connectives intersect and except, but aggregation presents some problems, as we will see. Note that we only allow the wildcard $*$ to be applied to a single tuple variable; selections such as select $*$ from $R,\ S$ that return all attributes of a cartesian product can always be rewritten to mention these attributes explicitly in the select clause.

The provenance semantics. Let us collectively refer to the individual data values and tuples in a database as the database's *items*. To define the provenance semantics, we use a formalization based on the "tagging" or "annotation propagation" approach of Wang and Madnick [17] and Bhagwat et al. [2]. In this approach, each input item is assumed to have an identifying "color" which serves as an abstraction of a system identifier or some other means of referring to part of a database. We can then describe how queries and updates manipulate provenance by means of functions mapping such colored databases to colored databases in which colors are propagated along with their item during computation of the output. The provenance of an item in the output is simply the item in the input with the same color. To illustrate, consider the table $T(\mathsf{beer}, \mathsf{origin})$ from the Introduction with tuples $\{(\mathsf{Leinenkugel}, \mathsf{USA}), (\mathsf{Stella\ Artois}, \mathsf{USA})\}$ in which the data values and the tuples are annotated with colors $c_1, c_2, \ldots$ as shown at the left of Fig. 1. There, beer^c and origin^c store the colors of the data values in the beer and origin attributes, and tup^c stores the colors of the tuples. As such, Leinenkugel is colored by c_1, the first occurrence of USA is colored by c_2, the first tuple is colored by c_3, and so on. We could then define the colored semantics of the SQL query

$$\begin{array}{c} (\mathsf{select\ t.}* \ \mathsf{from}\ T\ \mathsf{t\ where\ t.beer <> 'Stella\ Artois'}) \\ \mathsf{union}\ (\mathsf{select\ t.beer,\ 'Belgium'\ as\ origin\ from}\ T\ \mathsf{t\ where\ t.beer} = \mathsf{'Stella\ Artois'}) \end{array} \tag{7}$$

to map T to the colored table at the right of Fig. 1. This defines the provenance of Leinenkugel in the output to be the corresponding data value in T, the provenance of the tuple $(\mathsf{Leinenkugel}, \mathsf{USA})$ to be the provenance of the first tuple in T, and so on. The "empty" or "blank" color $\perp$ indicates that an item is introduced by the query itself. Hence, this particular colored semantics takes the view that the second select subquery constructs a new tuple rather than copying an existing one.

Intuitively, we will take the view that queries either construct new items or copy complete items from the input. As such, all data values resulting from constant construction as in $\mathsf{select\ 'USA'\ as\ origin\ from}\ T\ t$ are colored $\perp$, as are the tuples returned by queries such as $\mathsf{select}\ A,\ B\ \mathsf{from}\ R$ whose select clause constructs new

tuples. All other items, such as the tuples returned by select t. $*$ from T t, retain their color. This is essentially the same provenance semantics as that of Wang and Madnick [17] and Bhagwat et al. [2], although they only consider provenance for data values, not tuples.

In order to elegantly formalize this intuition, notice that, by storing colors as in Fig. 1, it becomes possible to define functions mapping colored databases to colored tables in SQL itself. For example, if we let $\mathsf{clr}(T)$ denote the colored version of table $T(\mathsf{beer}, \mathsf{origin})$ then the particular colored semantics of query (7) illustrated in Fig. 1 is defined by

$$\begin{aligned}&(\text{select t.}* \text{ from } \mathsf{clr}(T) \text{ t where t.beer <> 'Stella Artois'})\\ \text{union } &(\text{select t.beer as beer, 'Belgium' as origin, t.beer}^{\mathsf{c}} \text{ as beer}^{\mathsf{c}}, \perp \text{ as origin}^{\mathsf{c}}, \perp \text{ as tup}^{c}\\ &\text{from } \mathsf{clr}(T) \text{ t where t.beer} = \text{'Stella Artois'})\end{aligned} \tag{8}$$

To define our provenance semantics it hence suffices to assign, to each query Q, a query $\mathcal{P}[Q]$ mapping colored databases to colored tables. It is important to remark that Q and $\mathcal{P}[Q]$ operate on different views of the database: Q operates on the tables without colors (like (7) above), while $\mathcal{P}[Q]$ operates on colored tables (like (8)). To avoid confusion between the two views, we will range over uncolored tables by R, S, and T, and over their colored versions by $\mathsf{clr}(R), \mathsf{clr}(S)$, and $\mathsf{clr}(T)$. We refer to the attributes that store normal data values in $\mathsf{clr}(R), \mathsf{clr}(S)$, and $\mathsf{clr}(T)$ (like beer and origin) as the *normal attributes* and to the attributes that store colors (like $\mathsf{beer}^c, \mathsf{origin}^c$, and tup^c) as the *color attributes*.

Definition 1: The provenance semantics $\mathcal{P}[Q]$ of a query Q operating on uncolored tables is inductively defined as follows. Let $\mathcal{P}[a]$ denote the blank color $\perp$ when a is a constant, and let $\mathcal{P}[a]$ denote $t.A^c$ when a is an attribute reference $t.A$ with t a tuple variable.

- $\mathcal{P}[\text{select } r_i.* \text{ from } R_1\ r_1,\ \ldots,\ R_m\ r_m \text{ where } \varphi] :=$

  $$\text{select } r_i.* \text{ from } \mathsf{clr}(R_1)\ r_1,\ \ldots,\ \mathsf{clr}(R_m)\ r_m \text{ where } \varphi;$$

- $\mathcal{P}[\text{select } a_1 \text{ as } A_1,\ \ldots,\ a_n \text{ as } A_n \text{ from } R_1\ r_1,\ \ldots,\ R_m\ r_m \text{ where } \varphi] :=$

  $$\begin{aligned}&\text{select } a_1 \text{ as } A_1,\ \ldots,\ a_n \text{ as } A_n,\ \mathcal{P}[a_1] \text{ as } A_1^c,\ \ldots,\ \mathcal{P}[a_n] \text{ as } A_n^c,\ \perp \text{ as } \mathsf{tup}^c\\ &\text{from } \mathsf{clr}(R_1)\ r_1,\ \ldots,\ \mathsf{clr}(R_m)\ r_m \text{ where } \varphi;\end{aligned}$$

- $\mathcal{P}[Q_1 \text{ union } Q_2] := \mathcal{P}[Q_1] \text{ union } \mathcal{P}[Q_2]$.

Example 1: To illustrate, $\mathcal{P}[Q]$ with $Q =$ select s.beer, 'stout' as kind, s.origin from S s where s.kind $=$ 'stout' as in example (5) from the Introduction yields

$$\begin{aligned}&\text{select s.beer, 'stout' as kind, s.origin, s.beer}^c \text{ as beer}^c, \perp \text{ as kind}^c, \text{ s.origin}^c \text{ as origin}^c, \perp \text{ as tup}^c\\ &\text{from } \mathsf{clr}(S) \text{ s where s.kind} = \text{'stout'}.\end{aligned} \tag{9}$$

Also, $\mathcal{P}[Q]$ with Q as in query (7) yields query (8).

Inherent to definition of $\mathcal{P}[Q]$ is that queries that are equivalent under the normal semantics need not be equivalent under the provenance semantics. For example $Q_1 :=$ select $r.*$ from R r is equivalent to $Q_2 :=$ select $r.A$ as $A, r.B$ as B from R r when R consists only of attributes A and B, but $\mathcal{P}[Q_1]$ is not equivalent to $\mathcal{P}[Q_2]$ as the former retains the tuple colors from the input, while the latter colors all tuples $\perp$.

Expressive completeness Let us now see how this provenance semantics compares with the manual approach to recording provenance. In this respect, note that queries mapping colored databases to colored tables, such as (8) above, can also be viewed as being *manually constructed* to record provenance. In other words, we want to compare the class of queries $\{\mathcal{P}[Q] \mid Q \text{ a query on uncolored databases}\}$ with the class of queries P mapping colored databases to colored tables. Since we are interested in *recording* provenance, however, we will exclude from our discussion queries P such as

$$\mathsf{select\ t.}* \mathsf{\ from\ clr}(T)\ \mathsf{t\ where\ t.beer}^c\ = c_1 \tag{10}$$

that *query* provnenance rather than record it.

Definition 2: A *provenance recording* query is a query P mapping colored databases to colored tables in which every where-clause φ mentions only normal attributes.

Clearly, (10) is hence not provenance recording. In contrast, $\mathcal{P}[Q]$ is always provenance-recording since the where-clause of a query Q operating on uncolored tables only mentions normal attributes and since $\mathcal{P}[Q]$ does not affect where-clauses.

Can every provenance-recording query be defined in terms of $\mathcal{P}[Q]$ for some Q? The answer is no, for two reasons. First, due to our view of queries as either constructing new items or copying whole items, it is impossible for $\mathcal{P}[Q]$ to yield something like

$$\mathsf{select\ t.beer\ as\ beer,\ t.beer}^c\ \mathsf{as\ beer}^c,\ \mathsf{t.tup}^c\ \mathsf{as\ tup}^c\ \mathsf{from\ clr}(T)\ \mathsf{t} \tag{11}$$

that returns tuples which do not literally occur in T, yet have the same colors as tuples in T. Similarly, it is impossible for $\mathcal{P}[Q]$ to yield something like

$$\mathsf{select\ 'USA'\ as\ origin,\ t.origin}^c\ \mathsf{as\ origin}^c,\ \bot\ \mathsf{as\ tup}^c\ \mathsf{from\ clr}(T)\ \mathsf{t} \tag{12}$$

in which data values are given the provenance of data values from T although the data value itself need not occur in T. We refer to provenance recording queries that only color output items i by color c if i also occurs in the input as '*copying*'. (See [5, 6] for a full formal definition of this concept).

Second, due to our view that only data values constructed by a constant expression are colored $\bot$, it is impossible for $\mathcal{P}[Q]$ to yield something like

$$\mathsf{select\ t.beer\ as\ beer,}\ \bot\ \mathsf{as\ beer}^c,\ \bot\ \mathsf{as\ tup}^c\ \mathsf{from\ clr}(T)\ \mathsf{t} \tag{13}$$

that colors every possible beer data value by $\bot$. Indeed, to simulate (13) by means of $\mathcal{P}[Q]$, Q would have to mention every possible beer value as a constant, of which there are unboundedly many. We call queries that can color only a finite, bounded number of atoms by $\bot$ '*bounded inventing*'.

We view the fact that $\mathcal{P}[Q]$ can only define provenance recording queries that are 'copying' and 'bounded inventing' as a desirable property: it ensures that the provenance relationship between input and output described by $\mathcal{P}[Q]$ is not arbitrary, but meaningful. After all, one could hardly argue that the provenance relationships described by (11) and (12) above correspond to the intuitive notion "is copied from T". Similarly, queries without aggregation are typically considered as "domain-preserving" in database theory, with limited ability to create new data values. The bounded inventing property merely ensures that the provenance semantics respects this view. With regard to the copying and bounded inventing queries, our provenance semantics can be shown expressively complete:

Proposition 3 (Buneman et al. [5, 6]): For every provenance recording query P that is copying and bounded inventing there exists a query Q mapping uncolored databases to uncolored tables such that $P \equiv \mathcal{P}[Q]$.

As such, we are ensured that we do not lose flexibility when using $\mathcal{P}[Q]$ to record provenance automatically instead of recording provenance manually. Of course, in practice we also would like to record provenance for queries involving aggregation such as select $A, \mathsf{sum}(B)$ from R group by A that fall outside the current framework. Indeed, although we could in principle simply define the provenance of all atomic data values resulting from sum to be created by the query itself, this causes the provenance semantics to become *un*bounded inventing. A more satisfying approach than simply defining the results of an aggregation operator to be created by the query itself could be to record *how* this result was computed. For example, we could color a data value resulting from the above sum aggregation by $\mathsf{sum}(c_1, c_2, c_3)$ indicating that it was obtained by applying sum to the set of data values from the input colored by c_1, c_2, and c_3, respectively. This use of *expressions* as provenance is similar to the approach of Green et al. [**?**], who use semi-ring expressions to describe the provenance of relational algebra queries without aggregation. It is also analogous to certain techniques for *workflow* provenance, as known from the geospatial and Grid computing communities [3, 10, 15]. It is not clear, however, whether and how our expressive completeness results transfer to this setting.

3 Provenance Recording for Updates

Our discussion of provenance for queries is straightforwardly extended to updates like (2) and (3) from the Introduction of the form insert into $R\,Q$: the provenance of the inserted items is simply given by $\mathcal{P}[Q]$.

Definition 4 (Provenance of query insertion): $\mathcal{P}[\mathsf{insert\ into}\ R\,Q] := \mathsf{insert\ into}\ \mathsf{clr}(R)\ \mathcal{P}[Q]$.

For instance, for update (3) from the Introduction this yields

$$\begin{aligned}\mathsf{insert\ into}\ \mathsf{clr}(R)\ \mathsf{select\ t.beer,\ 'Lager'\ as\ kind,\ t.origin,}\\ \mathsf{t.beer}^c\ \mathsf{as\ beer}^c,\ \bot\ \mathsf{as\ kind}^c, \mathsf{t.origin}^c\ \mathsf{as\ origin}^c, \bot\ \mathsf{as\ tup}^c\ \mathsf{from\ clr}(T)\ \mathsf{t}.\end{aligned} \tag{14}$$

Updates of the form insert into $R(A, \ldots, B)$ values $(d, \ldots, d')$ like (1) from the Introduction clearly add newly constructed items to R. We hence define:

Definition 5 (Provenance of value insertion):

$$\begin{aligned}\mathcal{P}[\mathsf{insert\ into}\ R(A, \ldots, B)\ \mathsf{values}\ (d, \ldots, d')] :=& \mathsf{insert\ into}\ \mathsf{clr}(R)(A, \ldots, B, A^c, \ldots, B^c, \mathsf{tup}^c)\\ & \mathsf{values}\ (d, \ldots, d', \bot, \ldots, \bot),\end{aligned}$$

where we assume for ease of exposition that $A, \ldots, B$ comprise all attributes of R.

The provenance semantics of delete statements is also straight-forward, as deleting a tuple also deletes its provenance.

Definition 6 (Provenance of deletion): $\mathcal{P}[\mathsf{delete\ from}\ R\ \mathsf{where}\ \varphi] := \mathsf{delete\ from}\ \mathsf{clr}(R)\ \mathsf{where}\ \varphi$.

Observe that for the updates considered so far, $\mathcal{P}[U]$ is still 'copying' and 'bounded inventing'. Moreover, the provenance semantics is still expressively complete with regard to the class of updates from colored databases to colored databases that manually record provenance, in the following sense. Similar to the case for queries, we exclude from our discussion updates such as delete from $\mathsf{clr}(T)$ t where $\mathsf{t.beer}^c = c_1$ that *query* provenance rather than record it.

Definition 7: A *provenance recording* update is an update mapping colored databases to colored tables in which every where-clause φ mentions only normal attributes.

beer	origin	beer^c	origin^c	tup^c
Leinenkugel	USA	c_1	c_2	c_3
Stella Artois	USA	c_4	c_5	c_6

beer	origin	beer^c	origin^c	tup^c
Leinenkugel	USA	c_1	c_2	c_3
Stella Artois	Belgium	c_4	$\bot$	c_6

Figure 2: Color propagation for update (15). On the left is $\mathsf{clr}(T)$, the colored version of table T, and on the right is the colored result.

Proposition 8: Let V be a provenance recording update of the following form.

$$U \quad ::= \quad \mathsf{insert\ into}\ R\ Q \mid \mathsf{insert\ into}\ R(A,\ldots,B)\ \mathsf{values}\ (d,\ldots,d') \mid \mathsf{delete\ from}\ R\ \mathsf{where}\ \varphi.$$

If V is copying and bounded inventing, then there exists an update U operating on uncolored databases, also of the above form, such that $V \equiv \mathcal{P}[U]$.

Let us now consider update statements. In this respect, note that updates such as the following intuitively do not construct new tuples but modify existing ones "in-place".

$$\mathsf{update}\ T\ \mathsf{set\ origin} = \text{'Belgium'}\ \mathsf{where\ beer} = \text{'Stella Artois'} \tag{15}$$

It hence seems reasonable to define their provenance semantics in a way that agrees with how system identifiers are preserved in practical database management systems.

Definition 9 (Provenance of updates): $\mathcal{P}[\mathsf{update}\ R\ \mathsf{set}\ (A,\ldots,B) = (d,\ldots,e)\ \mathsf{where}\ \varphi] := \mathsf{update}\ \mathsf{clr}(R)\ \mathsf{set}\ (A,\ldots,B,A^c,\ldots,B^c) = (d,\ldots,e,\bot,\ldots,\bot)\ \mathsf{where}\ \varphi$.

For instance, for update (15) above this yields

$$\mathsf{update}\ \mathsf{clr}(T)\ \mathsf{set}\ (\mathsf{origin},\mathsf{origin}^c) = (\text{'Belgium'},\ \bot)\ \mathsf{where\ beer} = \text{'Stella Artois'}. \tag{16}$$

Note that although update (15) and query (7) essentially express the same database transformation on uncolored tables, their provenance semantics differs significantly. Indeed the query maps the colored table at the left of Fig. 1 to the colored table at the right of Fig. 1, while the update maps that same table to the colored table at the right of Fig. 2. In particular, the provenance semantics of the update is *not* copying, as the first output tuple is not identical to the first input tuple, although they are colored the same. The provenance semantics of the update statement is '*kind preserving*' however: it will only color an output atom by color c if the atom also occurs in the input with color c; and it will only color an output tuple by color c if there is a tuple in the input with color c. We refer to Buneman et al. [5, 6] for a full definition of this concept.

Every copying V is also kind preserving. As such, the provenance semantics $\mathcal{P}[U]$ of all updates U considered in this article is kind preserving. The provenance semantics is *not* expressively complete with regard to the class of kind preserving and bounded inventing provenance recording updates, however. To see why, suppose that R consists only of the attribute origin, and further suppose that we want to simulate the update

$$\mathsf{insert\ into}\ \mathsf{clr}(R)\ (\mathsf{select}\ \text{'Belgium'}\ \mathsf{as\ origin},\ \mathsf{t.origin^c}\ \mathsf{as\ origin}^c,\ \bot\ \mathsf{as\ tup}^c\ \mathsf{from}\ \mathsf{clr}(T)\ \mathsf{t}), \tag{17}$$

which is kind preserving, but not copying. (The inserted tuples are colored the same as tuples of T, but are not identical.) To simulate this update in terms of $\mathcal{P}[U]$ for some U, U clearly needs to be an insert statement itself. We already know, however, that this implies that $\mathcal{P}[U]$ is copying; as such it cannot express (17). Nevertheless, by adding extra update operators it is possible to regain expressive completeness; see Buneman et al [5, 6].

4 Conclusion

In order to assess the trustworthiness of a database it is vital to know the provenance of its data. Since manually recording such provenance quickly becomes very tedious, time-consuming, and error-prone, is preferable to let the user write queries and updates as before, and to let the database system record provenance automatically. In this respect, it is imperative to have a good explanation of the meaning of queries and updates with regard to provenance. Fortunately, the intuitive view of queries as either constructing new items or copying whole items from the input, as well as the intuitive view of updates as modifying items in-place, yields an automatic provenance recording semantics that is guaranteed to be as flexible as recording provenance manually. We conclude this article by remarking that, while the full provenance semantics presented here remains to be implemented in practice, preliminary experiments by Bhagwat et al. [2] and Buneman et al. [4] suggest that the overhead incurred by recording provenance as opposed to not recording it is reasonable.

Acknowledgement We are grateful to Peter Buneman for introducing us to provenance in databases, and for the very enjoyable collaboration that led to the expressive completeness results presented here.

References

[1] Marcelo Arenas and Michael I. Schwartzbach, editors. *Database Programming Languages, 11th International Symposium, DBPL 2007, Vienna, Austria, September 23-24, 2007, Revised Selected Papers*, volume 4797 of *Lecture Notes in Computer Science*. Springer, 2007.

[2] Deepavali Bhagwat, Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. *VLDB Journal*, 14(4):373–396, 2005.

[3] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, 2005.

[4] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *SIGMOD 2006: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 539–550, Chicago, IL, 2006. ACM.

[5] Peter Buneman, James Cheney, and Stijn Vansummeren. On the expressiveness of implicit provenance in query and update languages. In Thomas Schwentick and Dan Suciu, editors, *ICDT 2007: Proceedings of the 11th International Conference on Database Theory*, volume 4353 of *Lecture Notes in Computer Science*, pages 209–223, Barcelona, Spain, 2007. Springer.

[6] Peter Buneman, James Cheney, and Stijn Vansummeren. On the expressiveness of implicit provenance in query and update languages. Technical report, 2007.

[7] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *ICDT 2001: Proceedings of the 8th International Conference, on Database Theory*, volume 1973 of *LNCS*, pages 316–330, London, UK, 2001. Springer.

[8] James Cheney, Amal Ahmed, and Umut A. Acar. Provenance as dependency analysis. In Arenas and Schwartzbach [1], pages 138–152.

[9] Yingwei Cui, Jennifer Widom, and Janet L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[10] Ian Foster and Luc Moreau, editors. *Proceedings of the 2006 International Provenance and Annotation Workshop (IPAW 2006)*, number 4145 in LNCS. Springer-Verlag, 2006.

[11] Floris Geerts and Jan Van den Bussche. Relational completeness of query languages for annotated databases. In Arenas and Schwartzbach [1], pages 127–137.

[12] Floris Geerts, Anastasios Kementsietsidis, and Diego Milano. MONDRIAN: Annotating and querying databases through colors and blocks. In *ICDE 2006: Proceedings of the 22nd International Conference on Data Engineering*, page 82, Atlanta, Georgia, 2006. IEEE Computer Society.

[13] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS 2007: Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 31–40, New York, NY, USA, 2007. ACM Press.

[14] European Molecular Biology Laboratory. Swiss-prot database.

[15] Yogesh Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.

[16] Universal Protein Resource. `http://www.ebi.uniprot.org/`.

[17] Y. Richard Wang and Stuart E. Madnick. A polygen model for heterogeneous database systems: The source tagging perspective. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 519–538, Brisbane, Queensland, Australia, 1990. Morgan Kaufmann.