Curated Databases

Peter Buneman
School of Informatics
University of Edinburgh
Edinburgh EH8 9LE, UK
opb@inf.ed.ac.uk

James Cheney
School of Informatics
University of Edinburgh
Edinburgh EH8 9LE, UK
jcheney@inf.ed.ac.uk

Wang-Chiew Tan
Dept. of Computer Science
University of California, Santa Cruz
Santa Cruz, CA 95064, USA
wctan@cs.ucsc.edu

Stijn Vansummeren
Hasselt University and
Transnational University of Limburg
Diepenbeek, Belgium
stijn.vansummeren@uhasselt.be

ABSTRACT

Curated databases are databases that are populated and updated with a great deal of human effort. Most reference works that one traditionally found on the reference shelves of libraries – dictionaries, encyclopedias, gazetteers etc. – are now curated databases. Since it is now easy to publish databases on the web, there has been an explosion in the number of new curated databases used in scientific research. The value of curated databases lies in the organization and the quality of the data they contain. Like the paper reference works they have replaced, they usually represent the efforts of a dedicated group of people to produce a definitive description of some subject area.

Curated databases present a number of challenges for database research. The topics of annotation, provenance, and citation are central, because curated databases are heavily cross-referenced with, and include data from, other databases, and much of the work of a curator is annotating existing data. Evolution of structure is important because these databases often evolve from semistructured representations, and because they have to accommodate new scientific discoveries. Much of the work in these areas is in its infancy, but it is beginning to provide suggest new research for both theory and practice. We discuss some of this research and emphasize the need to find appropriate *models* of the processes associated with curated databases.

Categories and Subject Descriptors

H.2 [**Database Management**]: General

General Terms

Design, Languages, Theory

Keywords

Curation, annotation, provenance, archiving, citation

PODS'08, June 9–12, 2008, Vancouver, BC, Canada.

1. WHAT IS A CURATED DATABASE?

The term "curated" comes from the Latin *curare* – to care for. Of course, all valuable data sets are cared for in some way or other, but the term *curated database* is normally reserved for those databases whose content (often about a specialized topic) has been collected by a great deal of human effort through the consultation, verification, and aggregation of existing sources, and the interpretation of new (often experimentally obtained) raw data. Many curated databases act as publications and are currently replacing the traditional reference works found in libraries such as dictionaries, encyclopedias, gazetteers, etc. Some notable examples of curated databases include:

- *UniProt* [2] (formerly called SwissProt), which forms the standard reference for protein sequence data in molecular biology. It currently consists of over 300,000 entries. This database has a large team of dedicated curators and is typically regarded as a "gold standard" of protein databases.
- The *CIA World Factbook* [20], which is probably the most widely used source of demographic data. It was first published as a book around 1990 and was thereafter released annually. For the past 8 years or more the Factbook has also been published on-line. Recently, the interval between releases of the on-line version has been reduced to two weeks.
- The *IUPHAR receptor database* [48], which describes the molecules that transmit information across the cell membranes. This database is typical of a very large number of small curated biological databases. For example, unlike UniProt and the World Factbook, it is not backed by a large organization: most of the curation effort is supplied by volunteers, and only two people are involved with its direct maintenance.

Wikipedia and other wikis are also curated in that they are the product of direct human effort, and they provide some of the features we believe are desirable for curated databases. Wikis differ from most curated databases in that there is little or no structure imposed on the entries. Curated databases – whether or not they are supported by relational database technology – are usually designed to be machine

readable and thus amenable to data extraction tools, such as query languages that are common for databases. Socially, curated databases differ from wikis in that the curation process is usually more heavily orchestrated in the latter. However, there is much to be gained by trying to combine the best of wiki and database technology and some of the research we describe in later sections is an attempt to do just that.

The examples above form only the tip of the iceberg. The number of publicly available curated databases is exploding thanks to today's web technology, which makes it relatively easy to publish new databases on-line. Molecular biology alone, for instance, boasts over 1000 on-line databases [36], many of which are curated.

Measured by the work that goes into producing them, the cost of curated databases is huge. For example, there are over 150 people listed as working full-time on UniProt. As publications, curated databases provide academic kudos: the "authors" of databases such as UniProt and OMIM [58] are known primarily for providing comprehensive reference works. Given their importance, good tools to support the creation, maintenance, and evolution of curated databases are required. In this respect, curated databases have some characteristics that may challenge some of our precepts of how a database functions:

Source. The majority of curated data is data that is copied and edited from existing sources, perhaps other curated databases or printed articles. Often, the closest one may get to "source" data is an opinion of the curator that has been added as an annotation. All the other data has been painstakingly extracted from other databases. Since the value of curated databases lies in their quality and organization, knowing the origin of the curated data—its *provenance*—is particularly important. Manually recording provenance is both tedious and error-prone, thus automatic provenance recording support is desirable.

Annotation. In addition to the core data, curated databases also contain *annotations* that carry additional pieces of information such as provenance, the conjectured function of a gene, etc. For the purpose of easy construction of such annotations, it becomes necessary to propagate annotations made on a user's view of the database back to the actual core data, a non-trivial problem as we shall see.

Update. A common practice is to maintain a working database that is updated by a community of curators and periodically to "publish" versions of the database by distributing copies of the working database or by converting it to an appropriate format (XML or HTML) for web publishing. Given the role of curated databases as publications, it becomes important to be able to cite *particular* versions of the database, and to be able to retrieve a *particular* cited version when looking up a reference. In other words, one must make sure that the previous versions of the database are digitally preserved. Since human data entry is slow, curated databases do not grow or change rapidly, and specialized archiving techniques can be used to reduce the amount of storage necessary to record the complete history of the database.

Archiving techniques should also support temporal (sometimes also called longitudinal) queries, which are particularly important for databases like the World Factbook. For example, one might want to query previous versions to retrieve useful information such as the internet penetration of Liechtenstein over the past five years, and perhaps correlate it with economic data in this or another archived database.

```
ID 143F_HUMAN STANDARD; PRT; 245 AA.
AC Q04917;
DT 01-OCT-1993 (REL. 27, CREATED)
DT 01-FEB-1995 (REL. 31, LAST SEQUENCE UPDATE)
DT 01-NOV-1995 (REL. 32, LAST ANNOTATION UPDATE)
DE 14-3-3 PROTEIN ETA (PROTEIN AS1).
GN YWHAH OR YWHA1.
OS HOMO SAPIENS (HUMAN).
OC EUKARYOTA; METAZOA; CHORDATA; VERTEBRATA; TETRAPODA; MAMMALIA;
OC EUTHERIA; PRIMATES.
RN [1]
RP SEQUENCE FROM N.A.
RC TISSUE=BRAIN;
RX MEDLINE; 94032477.
RA SWANSON K.D., DHAR M.S., JOSHI J.G.;
RL BIOCHIM. BIOPHYS. ACTA 1216:145-148(1993).
RN [2]
RP SEQUENCE OF 26-224 FROM N.A.
RC TISSUE=KERATINOCYTES;
RX MEDLINE; 93294871.
RA LEFFERS H., MADSEN P., RASMUSSEN H.H., HONORE B., ANDERSEN A.H.,
RA WALBUM E., VANDEKERCKHOVE J., CELIS J.E.;
RL J. MOL. BIOL. 231:982-998(1993).
CC -!- FUNCTION: ACTIVATES TYROSINE AND TRYPTOPHAN HYDROXYLASES IN THE
CC     PRESENCE OF CA(2+)/CALMODULIN-DEPENDENT PROTEIN KINASE II, AND
...
CC -!- SUBUNIT: HOMODIMER (BY SIMILARITY).
CC -!- TISSUE SPECIFICITY: EXPRESSED MAINLY IN THE BRAIN AND PRESENT IN
CC     OTHER TISSUES ALBEIT AT LOWER LEVELS.
CC -!- SIMILARITY: BELONGS TO THE 14-3-3 FAMILY OF PROTEINS.
DR EMBL; L20422; L20422.
DR EMBL; X57345; X57345.
DR PIR; S29339; S29339.
...
KW BRAIN; NEURONE; PHOSPHORYLATION; MULTIGENE FAMILY.
FT INIT_MET 0 0 BY SIMILARITY.
FT MOD_RES 1 1 ACETYLATION (BY SIMILARITY).
...
FT CONFLICT 156 156 G -> A (IN REF. 2).
SQ SEQUENCE 245 AA; 28058 MW; 289087 CN;
   GDREQLLQRA RLAEQAERYD DMASAMKAVT ELNEPLSNED RNLLSVAYKN VVGARRSSWR
   VISSIEQKTM ADGNEKKLEK VKAYREKIEK ELETVCNDVL SLLDKFLIKN CNDFQYESKV
...  //
```

Figure 1: An abbreviated UniProt entry

Schema and structure. Many curated databases are constructed "on the cheap", starting life with a simple, structured design that allows basic description of the database "entries" (genes, countries, proteins, and so on), which are usually stored in a text file. Almost inevitably, the structure of the entries evolves over time. UniProt, for example, has always been distributed electronically in a specialized data format (see Figure 1 for an early example), but the format has been evolved to include new fields. As they evolve, curated databases may increase their dependence on database technology, but the design is seldom fully normalized; the fields themselves may contain structured text, and there is often some redundancy. In other words: database technology is used more to support persistence, concurrent access, and indexing rather than the data model. In order for database technology to succeed in also providing the data model for future curated databases, it is important to gain an understanding of the processes by which the structure of curated databases evolves.

We will discuss each of these technical topics in more detail in the following sections, but there are economic and social factors that are equally important to the long-term usefulness of curated databases. It is therefore worth taking a brief look at these factors because they may also suggest

some interesting long-term challenges for database technology.

1.1 Sustainability

During the nineteenth and twentieth century, thanks to the development of public, university and other institutional libraries, the function of libraries changed from one of ownership of data to one of distribution. A secondary but useful effect of this is that much more was preserved. Nearly all of our knowledge has been preserved by copying it: by an oral tradition, by copying manuscripts, by the printing press, and now by a plethora of digital media and computer networks such as the Internet. Concentrating it in archival institutions such as the libraries of Alexandria, Baghdad, Cotton, and Louvain, which were all destroyed in various ways, was probably counterproductive. Today – by archiving at remote sites – we have reasonable methods for protecting our data from physical destruction, but this is no guarantee against the economic collapse of the organization that maintains the data. There has been a proliferation of data centers over the past years – many dedicated to the storage of scientific and economic data – but one wonders whether, by analogy with early libraries, we are endangering our data by placing it in such centers. These depend on continued public funding and there are signs [49] that such centers are no more sustainable than early libraries.

A move towards massive systematic distribution of electronic publications is the LOCKSS [55] project in which a large number of university libraries each keep a repository of a set of publications, and a peer-to-peer synchronization process ensures that the repositories are consistent and cannot be corrupted either by bit-rot or deliberate interference. We have likened curated databases to publications, so could one build a LOCKSS system for databases? In addition to the requirements for files, such a system would have to work on incremental updates and would also have to work well with archiving, something we discuss later in this paper.

1.2 Open access

Another transition that was in part due to the proliferation of university libraries was that, in the second half of the twentieth century, academics and other scholars stopped having to pay to get their research into print and disseminated. Unlike textbooks, research articles are, unfortunately, not a highly saleable commodity. The fact that we no longer must pay for their distribution is largely due to libraries who purchase our papers on behalf of the communities that they serve. This economic model of distribution is still very much in use for electronic publications, but it is showing signs of collapse. Although there are costs associated with the production (reviewing, collating and placing in some repository) of an electronic journal, they are mostly initial costs and should be small compared with paper distribution. The idea of open access is that the initial costs should be paid for by the person (or institution or grant) responsible for a publication. After this it should be freely available.

As we have already mentioned, curated databases are unlike research papers in that they are constantly updated. So while many scientific curated databases are open access, it is not clear that this is the right economic model for supporting them. It is almost impossible to obtain indefinite funding to support the development of a database, and unless there is institutional support for a curated database, it is likely to lead a hand-to-mouth existence.

Of course, many curated databases – encyclopedias, dictionaries and other reference works – have evolved from paper publications that were sold, and they continue to charge for their use. But charging policies vary and are generally ill-defined: we give you small amounts of data for free, but you have to pay for the whole database; or, we give you the old data for free but you have to pay for the current stuff. For scientific data, if we draw an analogy between a curated database and a textbook or reference work, it is not unreasonable to expect the user of the database to pay for access, but how does one charge? Could a system of micropayments be designed for these databases? The problem is complicated by the fact that curated databases copy one another and if one database charges for access to some piece of data, it might be required that some of payment goes to the sources of that data. If such a scheme were to be developed, a satisfactory model of provenance is essential.

These are just two areas of curated databases in which the right model of data and data communication could be enormously valuable. Open access impinges on copyright and intellectual property issues, which is are hugely problematic areas even before databases are considered.

Organization. The rest of this paper will cover some of the technical issues that have recently been recognized by the database community as important to the topic of curated databases: propagation of annotations (Section 2), updates (Section 3), why-provenance (Section 4), archiving and citation (Section 5), and evolution of structure (Section 6).

2. PROPAGATION OF ANNOTATIONS

In UniProt [2] the fields are classified into *core data* and *annotation*. Core data includes information like the protein sequence itself, which is extracted from other databases. Annotation is additional information, like the conjectured function of a gene, that has been entered manually or is machine generated by classification and sequence matching programs. In UniProt, the process of annotation is carefully controlled and annotations are placed only in designated annotation fields. By contrast, the Distributed Annotation Server [31] (DAS) is a client-server system that conveys annotations that are external to the underlying database between user interfaces. A an arbitrary user may annotate something such as a segment of sequence data that appears in his or her interface the data, and the annotation will appear to another user using a *different* interface. Each user interface can combine information from a number of annotation servers, each of which acts as an external database for annotations.

2.1 Annotation models

As a mild leap of faith, we liken DAS to the process of annotating a *view* of some data and asking that the annotation be carried backwards to the source data as well as forwards to some other view. Since we are typically annotating a small piece of data, such as a base data value, this requires us to identify where, in the source, that piece came from – its *where-provenance*. The importance of being able to connect elements of the output of a query with those in the input was first recognized in [51, 72] in the context of data integration. The process of annotation immediately complicates traditional notions of query equivalence and hence

query optimization. Consider the two SQL queries:

```
Q1: SELECT R.A, R.B          Q2: SELECT S.A, 50 AS B
    FROM R,S                     FROM R,S
    WHERE R.A=S.A                WHERE R.A = S.A
    AND R.B=50                   AND R.B=50
```

These queries are normally regarded as equivalent, and the simple rewrite of R.B to 50 as B in the select clause is a typical low-level optimization. If one regards the queries as copying base values from the input to the output, however, the queries behave differently: A-values in the output of Q_1 are copied from R, while A-values in the output of Q_2 are copied from S. Moreover, B-values in the output of Q_2 are apparently created by Q_2 itself. It is initially annoying that we need to break the principle of substitution of equals for equals for the purpose of annotation propagation; it calls for a different model of both the underlying data and query evaluation. The model that is developed in [8, 40] and implicitly used in [17,18] is to add unique *colors* to components of the input. These colors may be used to represent annotations or to identify components for provenance tracking. We then formalize how queries operate on colored tables. In the following example there are instances of R and S in which each base value has been annotated with a distinct color, $\flat 1, \flat 2, \ldots,$. (We shall use $\flat 1, \flat 2, \ldots,$ for color values, and $\flat i, \flat j, \ldots$ for color variables.) The output of the two queries Q_1 and Q_2 shows how these colors are propagated.

A	B
$10^{\flat 1}$	$49^{\flat 2}$
$12^{\flat 3}$	$50^{\flat 4}$

R

A	B
$11^{\flat 5}$	$49^{\flat 6}$
$12^{\flat 7}$	$50^{\flat 8}$

S

A	B
$12^{\flat 3}$	$50^{\flat 4}$

Q_1

A	B
$12^{\flat 7}$	$50^{\perp}$

Q_2

(Here, $\perp$ indicates that the value 50 does not originate from the input, but was constructed by Q_2 itself.) One can consider an alternative representation of these annotated databases in which colors are first-class values by pairing each column with another column of colors. An immediate question is whether the transfer of colors that is implicit in our queries can be represented by another query on the alternative representation that explicitly manipulates colors. The problem is that in operations such as union and set-theoretic projection, two base values can be "merged" and the annotation is then a set of colors. For conjunctive queries, it is established in [8,26] that there is indeed an explicit query that can be used to express the annotation transfer. The authors also examine the extension of SQL with explicit statements to "steer" annotations through queries. In addition, they propose the "DEFAULT ALL" propagation scheme in which any two base values that are explicitly found to be equal in a selection or that are implicitly identified in a union or natural join have their annotations merged. Finally, the authors also investigate the efficiency of computing annotation propagation.

Although most work on annotation has considered annotations only on single base values or whole rows, [40] provides a system for attaching annotations to *sets* of base values occurring in the same tuple. The authors point out that an annotation on a base value should be regarded as a curator's opinion of the validity of the value and that this is better modeled as an annotation on the relationship between the base value and the key for the tuple containing that value. In addition, they introduce a query language – a "color algebra" – for querying annotations. Again an explicit relational representation of such block-colored databases is introduced and the color algebra is shown to express exactly all queries that the positive relational algebra can express over the explicit relational representation of block-colored databases. This result was later extended in [41] to full relational algebra.

2.2 Reverse propagation of annotations

We noted earlier that there is a need to propagate annotations from a user's view of the database back to the source data: if an annotation is attached to some base value in the output of a query, to what base value in the input should it be attached? Given the rules proposed in [8,51,72] for propagating input colors to the output of a query, one can in principle answer this question by reversing the propagation rules. In the case that the query contains a union or a projection, however, a given color on an output base value could be produced by more than one possible placing of that color on an input value. Moreover in the case of a join, a color placed on an output base value can only come from a single base value in the source, but that color when carried forward may "spread" to other base values. A source annotation that does not have any such side-effects, i.e., that produces precisely the view annotation, is called side-effect free. In [17] the problem of finding a side-effect free annotation is shown to have NP-hard query complexity for queries containing both a projection and join and was later strengthened to DP-hardness in [69]. In all other cases of positive queries, there is a polynomial time solution.

The problem of finding side-effect free annotation placements is related to the view deletion problem: find a minimal set of tuples in the source to cause the deletion of a tuple in the output [1, 17, 28]. It is yet to be seen whether any of these complexity results form a practical barrier to building general systems for the reverse propagation of annotation. In fact [27] shows that these problems become tractable for classes of key-preserving queries. We should note, however, that these results are for the default propagation rules for relational algebra. There appear to be no results for reversing the more complex forms of annotation suggested in [8, 40].

2.3 Annotation and where-provenance

We have been using colors as a model for annotation, but if we assume that unique colors are assigned to the source values, then the colors that annotate an output value also serve as a description of *where* that element has come from. In the DBNotes system [8,26], a language was developed for the propagation of annotations through queries. As mentioned earlier, in addition to allowing annotations to be propagated based on where elements are copied from, it also allows annotations to be explicitly steered through the query. Thus, a natural question is whether queries that explicitly manipulate provenance is any more expressive than the implicit provenance associated with a query. This question was answered in [14] in a somewhat more general setting. First, colors could be added to arbitrary structures (tuples and tables, as well as base values); second, update languages were also considered. In this section we shall look at query languages.

Figure 2 shows a table in which all values – base values, tuples and tables – have been annotated with a color. The default provenance is shown for a projection and selection. A value that is constructed by a query receives the color $\perp$. Note the assumption that a tuple that is preserved in its entirety (e.g. SQL's **SELECT ***) retains

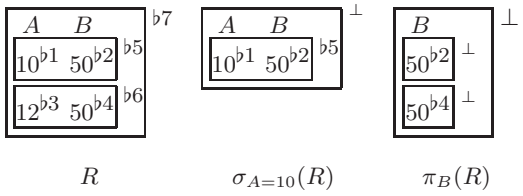

Figure 2: Provenance annotation

its provenance. The output contains two tuples that differ only on their annotation; this is equivalent to one tuple annotated with a set of colors. Because colors can be attached to arbitrary structures, and because of the way explicit provenance is represented, it is more convenient to work in a domain of complex objects [19] or nested relations in which values can be freely constructed out of base values, labeled records $(A{:}e_1, B{:}e_2, \ldots)$ and sets $\{e_1, e_2, \ldots\}$. Thus the un-annotated table in Figure 2 is represented by the expression $\{(A{:}10, B{:}50), (A{:}12, B{:}30)\}$. In order to represent explicit annotations it is a simple matter to pair each value with its color using a record with distinguished labels $\mathcal{V}$ for value and $\mathcal{C}$ for color. For example, $50^{\flat 2}$ is represented by $(\mathcal{V}{:}50, \mathcal{C}{:}\flat 2)$, the tuple $(A{:}10^{\flat 1}, B{:}50^{\flat 2})^{\flat 5}$ by $(\mathcal{V}{:}(A{:}(\mathcal{V}{:}10, \mathcal{C}{:}\flat 1), B{:}(\mathcal{V}{:}50, \mathcal{C}{:}\flat 2)), \mathcal{C}{:}\flat 5)$ etc. These are nested relational structures and can be manipulated by nested relational algebra.

Three properties of implicit where-provenance were identified in [14]:

Copying Since we are dealing with queries rather than updates any value that appears in the output must either have been copied in its entirety from the input or have been constructed by the query. In the latter case the value will be colored with $\perp$. Thus if a value colored with $\flat i$, $(\flat i \neq \perp)$ appears in the output, then the same value with the same color must appear in the input. For example, we cannot have $7^{\flat i}$ in the output and $6^{\flat i}$ in the input. Moreover a value such as $(A : 7^{\perp}, B : 8^{\flat i})^{\flat j}$ that preserves the color of a tuple but changes one of its components cannot be created by a copying transformation.

Bounded inventing A query can generate only a bounded number of base values; that is, only a bounded number of base values in the output can be annotated with $\perp$. Thus a query that transforms $\{1^{\flat 1}, 2^{\flat 2}, \ldots, n^{\flat n}\}^{\flat i}$ into $\{1^{\perp}, 2^{\perp}, \ldots, n^{\perp}\}^{\perp}$ for any n is *not* bounded inventing.

Color propagating Since the purpose of color annotation in describing provenance is to link input and output values, this condition says that the query should not be sensitive to the choice of colors. That is, it should commute with any function – not necessarily injective – that maps colors to colors. In particular, queries that cannot perform equality comparisons on colors is guaranteed to be color-propagating. Moreover, a color-propagating function is characterized by its behavior on *distinctly-colored* inputs, for which the colors can be viewed as addresses.

It is straightforward to show that the implicit provenance associated with a query satisfies these conditions. It is also easy to construct queries that operate on explicit representations of colored complex objects that violate these conditions.

In [14] it is shown that all explicit provenance queries that are copying, bounded-inventing, and never compare colors can also be expressed as implicit provenance queries. In fact, [14] reported a stronger result, but the proof technique relied upon an unchecked assumption that *all color-propagating queries can be expressed by queries that do not compare colors.* In fact, this question is still open, but the proof in [14] does establish expressive completeness with respect to the syntactic restriction that explicit queries do not compare colors.

Although there are some unresolved issues in the characterization of provenance queries, the general message of these results is useful: queries in the nested relational algebra with implicit provenance express all reasonable provenance mappings. Moreover, this result also holds if we are restricted to "flat" tables and relation calculus. Thus the explicit annotation propagation mechanisms of [8] are just a syntactic convenience, if they are used only to record where-provenance.

3. UPDATES IN CURATED DATABASES

We have discussed annotation for queries and its relationship to where-provenance in some detail, but let us return to the larger picture of curated databases. In systems such as DAS, annotations are external to the core database and are superimposed on it [9]. In UniProt, however, an annotation is simply a part of the database that has been entered or modified by a curator. Curated databases are created by copying or abstracting data from other sources and later correcting it. This is a process that is entirely familiar to anyone who has constructed bibliographies for papers such as this one. When doing so, one typically tries to find a bibtex entry on the web, copies and pastes it into one's own bibliography, and then corrects it in accordance with one's own preferences. If this entry were an entry in a scientific database, where-provenance information would include both the source for the entry and a record of what update was made and who made it. A whimsical account of curation is given in [21].

It is important to reiterate that curated databases are constructed by updates to an existing structure; they are not views. So while UniProt contains some parts – such as the taxonomic lineage and the citations – that have probably been imported automatically into the database, many of the the fields are entered by people making individual updates, and it is this activity that gives value to the database. The source of these updates is sometimes information extracted by the curator from a paper; it is also sometimes simply copied from another, possibly curated, database. It is quite common for this copying to be done by the common copy and paste commands that work in most desktop environments. When data is copied between applications or systems, its annotation, context, and especially where-provenance information is lost.

3.1 A copy-paste model of curation

In an attempt to model this process [13] exploits a model in which curated databases are semistructured trees, and the fundamental operation is to copy a data element – as subtree – from one tree to another. This provides a simple model of provenance in which it is possible to ask questions such as

when some data value was first created, by what process did that value arrive in a database, when was a subtree last modified, etc. The cost of storing such provenance information appears to be prohibitive if done naively because some trail of information needs to be kept of each node in the tree. However this can be mitigated by two observations: first that provenance information is hereditary: unless a node in the tree has been modified, its provenance is that of its parent node. Second, one can collect a sequence of basic operations into a transaction, and there is a description of the effects of the transaction that is shorter than recording the log of basic operations. The latter is potentially useful for databases that are published periodically (i.e, every few months or so).

The major unsolved challenge here is the practical goal of building provenance recording into the the tools that a curator normally uses. This is a hard problem involving not only data management, query language design and efficiency issues but also usability and organizational problems. While [13] addressed the problem of recording and storing provenance, the task of formulating queries against the resulting provenance store (and the raw data) was left to users. Can we design a query language that provides good high-level support for writing sophisticated queries against curated databases involving provenance, the raw data, and perhaps previous versions? This is challenging, since reasoning about copy-paste operations on trees is non-trivial [39]. On the organizational side, how can we convince both database, operating system, and other application vendors and independent database curators to adopt common standards for provenance? This is a significant long-term challenge requiring both technical and social development.

In practice, updates to a curated database are performed through the update expressions of SQL. This includes both manual copy-paste operations and bulk loading of data into one database from another. Just as where-provenance has been studied for query languages we may ask how update languages manipulate where-provenance. Figure 3 shows three SQL programs. Although they all have the same "result", the way they carry provenance is different. The first SQL program is a query and is copying (in the sense of Section 2.3) as a provenance-preserving transformation. The second and third program are SQL updates and are not copying as provenance-preserving transformations: they preserve the color of a tuple or table while changing one of its components.

Using a simple complex object update language [52], results paralleling those in Section 2 are also developed in [14]. Updates in this language satisfy a weaker semantic provenance condition than queries, called *kind-preservation*: if a value appears in the output with a given color, then the corresponding value in the input must must have the same type (atom, record, or set); furthermore, if the value is an atom, then the corresponding value in the input must be the *same* atom. Thus a kind preserving transformation can drop or add fields in tuples, or insert and delete elements of sets, while preserving their provenance. The update language with implicit provenance is shown to be complete with regard to the update language in which colors can be explicitly manipulated under the kind preserving, bounded inventing, and no color-equality test conditions. It is still not known how these results for the complex object update language transfer to SQL updates on flat relations.

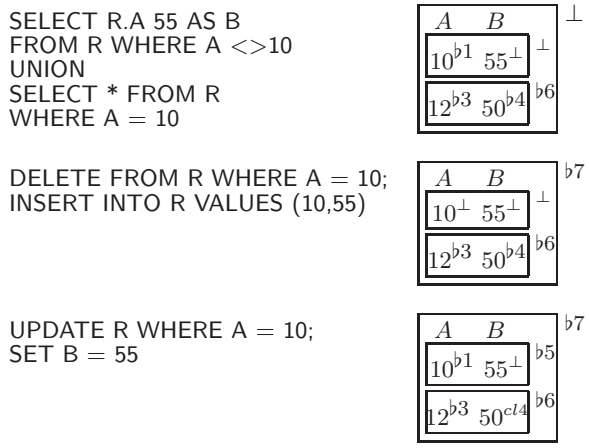

Figure 3: Updates and provenance

When measured by the transformations between input and output domains that they express, update languages such as the update fragment of SQL do not add anything to query languages; in contrast, provenance mappings provide an alternative method of characterizing update languages that highlights differences with queries. But kind-preservation is a very weak condition. It allows one to empty out a tuple and insert completely different fields into it while preserving provenance. Whether this is a useful knowledge is debatable and reminiscent of Theseus' paradox [64]: suppose we build a ship, and then replace each part of it over its lifetime; is it the same ship? There appear to be other possible models of provenance for update languages, for example one could associate provenance with some form of key preservation.

In addition, there has yet to be any work on provenance in update languages for XML and other forms of semistructured data. Although it appears straightforward to extend the approach of [14] to where-provenance for XQuery-style queries, update languages for XML are still in development. Most of them abandon a key design principle of XQuery (and of SQL) by permitting unrestricted side-effecting operations inside queries. Their semantics is sufficiently complex (and the number of competing proposals sufficiently large) that designing provenance-tracking techniques for them is probably premature. See [23] proposes for a different approach to database-style XML updates that is much closer to the complex object update language of [52] and has has been designed with provenance-tracking in mind.

4. WHY-PROVENANCE

So far we have considered situations in which query and update operations are primarily used to *copy and rearrange* data in curated databases, and we have developed a model to describe that process and, in particular, to tell us *where* a particular value has come from. This is a rather restricted view of provenance; one may also want to know *why* a given tuple is in the output. The distinction between why- and where-provenance was introduced and described in [18]. Why-provenance is a more nebulous concept, because we have to decide what counts as an explanation.

A trivial form of why-provenance is simply to give the source database and the query. In some cases, such as a query that counts the number of items in the database, this trivial answer is probably the best. However, in other sit-

uations the database may be large and the query complex, so that tracing the behavior of the query from the output data back to the source is a nontrivial chore. Instead, the reason for the presence of a given value can be explained by a simpler query operating on a small part of the database. For a simple query such as a selection, an explanation why a tuple is in the output can be given by pointing to the tuple in the source that satisfies the selection condition; all other tuples are irrelevant.

In databases, why-provenance has been studied mostly for tuples, and even in this simple case, it has taken some time to arrive at clear and well-motivated definitions. One key issue illustrating this problem is *compositionality*. For example, suppose that, for a query Q, we proffer a candidate definition $P_Q^I(t)$ of the set of tuples in the input I that "influenced" a tuple t in the output $Q(I)$. If we follow the natural intuition that "influence" is a transitive relation, then the provenance of tuples t in the output of of $Q \circ R$, the composition of Q and R should be $P_{Q\circ R}^I(t) = \bigcup\{P_Q^I(t') \mid t' \in P_R^I(t)\}$. Compositionality is desirable because it implies that the provenance behavior of a composite query can be obtained easily from the provenance behavior of its subqueries, just as the ordinary semantics of query composition corresponds to functional composition.

Another problem in this area has been that there have been many definitions of something called "why-provenance", but the definitions have not always been as rigorous as one might like. There have also been several claims that the various definitions are equivalent in some sense, but no proofs. We would like to try to begin to untangle this state of affairs here by describing, naming, and formalizing the different approaches.

In [29], a form of provenance called *lineage* was introduced, as a *set* of input tuples that "influenced" a tuple in the result. A semantic criterion that accorded well with one's intuition was used to define this notion for each relational algebra operation in isolation. Then lineage was defined for arbitrary queries compositionally. The semantic criterion, however, is *not* preserved under composition, and lineage is sensitive to query rewriting.

In [18] why-provenance was defined for a semistructured data model. We will translate the basic idea into relational terms. Let Q be a monotone query. A witness to $t \in Q(I)$ is a subinstance $J \subseteq I$ such that $t \in Q(J)$. The why-provenance of $t \in Q(I)$ in [18] was defined as a particular finite set of witnesses computed by inspecting the syntax of Q. We call these witnesses *proof witnesses* and call this form of provenance *proof why-provenance* to distinguish it from the generic idea of why-provenance and other specific proposed definitions. Provenance in [18] was defined only for queries in a restricted form; the definition was not compositional, and was sensitive to query rewriting.

In [17], another definition of why-provenance was given, which we term *minimal why-provenance*. A minimal witness is simply a witness J such that no subset of J is a witness to $t \in Q(I)$. (In fact, minimal witnesses were also discussed briefly in [18]). Then the minimal why-provenance of $t \in Q(I)$ is just the set of all minimal witnesses to $t \in Q(I)$. Minimal why-provenance is invariant under query rewriting, but it was initially unclear whether, and if so how, minimal witnesses could be obtained compositionally or efficiently. This is answered by a more general approach to provenance, which we now describe.

R :

a	b	c	p
d	b	e	r
f	b	e	s

V :

a	c	$p + (p \cdot p)$
a	e	$p \cdot r$
d	c	$r \cdot p$
d	e	$r + (r \cdot r) + (r \cdot s)$
f	e	$s + (s \cdot s) + (s \cdot r)$

$V(X, Z)$:– $R(X, \_, Z)$
$V(X, Z)$:– $R(X, Y, \_), R(\_, Y, Z)$

Figure 4: Semiring provenance

4.1 Why-Provenance and Semirings

A remarkable paper [44] provides a unifying approach to a number of extensions to the relational algebra using semirings. The starting point is the observation that, in the process of evaluation of a relational algebra expression, two things can happen to tuples: they can be joined together (in a join) or they can be merged together (in a union or projection). Figure 4 shows how this can be exploited. The left sides of R and V are tables, V being constructed from R by the given query. The right side of R shows the identifiers or "abstract quantities" associated with the source tuples. The right side of V shows how the output tuples were derived: (a, c) was formed by unioning $(+)$ p with the result of joining $(\cdot)$ p with itself; (a, e) was formed by joining p and r. We can consider these as abstract polynomials that describe *how* the tuple was formed, and if we look at the properties of the union and join operations on tuples, e.g., "$+$" and "$\cdot$" are both associative and commutative, "$\cdot$" distributes over "$+$", we conclude that these are polynomials in a (commutative) semiring $(K, +, \cdot, 0, 1)$. Various instantiations of this abstract provenance semiring give rise to a number of well-known extensions to positive relational algebra: relational algebra itself, algebra with bag semantics, C-tables [47], and probabilistic event tables [30, 66].

It was claimed in [44] that why-provenance can be obtained by evaluating using the structure $\mathcal{P}(X)$ equipped with $0 = 1 = \emptyset$ and $+ = \cdot = \cup$. This definition actually is closest to lineage. Also, as pointed out by Tannen[1], there is a technical problem: $(\mathcal{P}(X), \cup, \cup, \emptyset, \emptyset)$ is not a semiring since it does not satisfy the multiplicative annihilator law $0 \cdot a = 0$. Instead, the (apparently) intended behavior can be obtained by taking $\mathcal{P}(X) \cup \{\bot\}$ with $0 = \bot$, $1 = \emptyset$, $\bot + S = S + \bot = S$, $\bot \cdot S = S \cdot \bot = \bot$, and $S + T = S \cdot T = S \cup T$ if $S, T \neq \bot$.

A natural definition of proof why-provenance can be given using a different semiring: the set $\mathcal{P}(\mathcal{P}(X))$ of all sets of subsets of X, with $0 = \emptyset$, $1 = \{\emptyset\}$, $S + T = S \cup T$ and $S \cdot T = \{s \cup t \mid s \in S, t \in T\}$. Furthermore, as also pointed out to us by Tannen, minimal why-provenance can be modeled using the semiring of *irreducible elements* of $\mathcal{P}(\mathcal{P}(X))$, denoted $Irr(\mathcal{P}(\mathcal{P}(X)))$, that consists of those elements $S \in \mathcal{P}(\mathcal{P}(X))$ such that for every $s, s' \in S$, if $s \subseteq s'$ then $s = s'$. This again forms a semiring since it is the homomorphic image of the minimization operation $\min(S) = \{s \in S \mid \forall s' \in S. s \subseteq s' \text{ implies } s = s'\}$. Specifically, in $Irr(\mathcal{P}(\mathcal{P}(X)))$ we define $S + T$ as $\min(S \cup T)$ and $S \cdot T$ as $\min\{s \cup t \mid s \in S, t \in T\}$.

Semirings are a useful tool in studying why-provenance. All of the semiring-based definitions are compositional, and clearly illustrate the relationships between lineage, minimal,

[1]Personal communication. Thanks also to Val Tannen for the example.

and proof why-provenance: they are not exactly the same, but they are related by homomorphisms $h : \mathcal{P}(\mathcal{P}(X)) \to Irr(\mathcal{P}(\mathcal{P}(X)))$ and $h' : Irr(\mathcal{P}(\mathcal{P}(X))) \to \mathcal{P} \cup \{\perp\}$. We believe that this motivates abandoning the earlier attempts as false starts. It may also be possible to use semirings to formulate where-provenance. It is less obvious whether semiring-valued relations can be extended to model provenance in update languages. To begin with, the provenance semantics for update operations that combine tuples or sets is not commutative, so presumably some weaker structure than a semiring would be needed.

4.2 Why-Provenance and Program Slicing

If why-provenance is supposed to be a simpler explanation for a small part of the output of a query, then we should consider techniques in other areas that address similar problems. For example, consider program slicing [73], a process used in debugging. If, during a program execution, a variable has an unexpected value, one would like to highlight just that part of the program that was "responsible" for the assignment of the value. One hopes that this part of the program is substantially smaller than the whole program and therefore easier to understand.

This idea leads to a form of provenance for databases, called *dependency provenance* [22, 24]. In contrast to program slicing, dependency provenance identifies a set of parts of the *input* (i.e., the database) on which a given part of the output depends, rather than a subset of program points. In [24], dependency provenance was investigated for a nested relational data model in which each part of the database is annotated with a *set* of colors, and developed a semantic correctness condition called *dependency-correctness*, which ensures that each part of the output is annotated with the colors of all parts of the input on which the output depends, in a rigorous sense. Unfortunately, it is not possible automatically to compute minimal dependency-correct annotations, but it is possible to to approximate fairly well using dynamic or static techniques.

4.3 Why-Provenance and Updates

We have argued earlier in the paper that updates are essential in curated databases. If where-provenance makes sense for updates, what about why-provenance? A curated database is built through a long sequence of updates. It is likely that only a very small number of those updates had any effect on the data of interest, and that a satisfactory answer is to provide the subsequence of relevant updates and their sources. But what does "relevant" mean here, and is it sensible to use the same approach as in why-provenance for queries? Our experience with where-provenance suggests that simply translating sequences of updates to "equivalent" queries may be unsatisfactory from the point of view of provenance.

4.4 Provenance and Workflow

No discussion of provenance would be complete without mention of the large amount of work on workflow provenance. Many scientific data sets are derived from raw data by a complex analysis process involving several programs – a workflow. Scientists need to record the enactment of the workflow for two reasons: first to record precisely how the data set was described and to avoid recomputation; second because it is sometimes cheaper not to keep the derived data, but to recompute it when needed. The recording of a workflow enactment is also called (workflow) provenance. Workflow provenance is sufficiently important that there is a workshop largely devoted to this topic [59]; and workflow systems such as Taverna [62] and Kepler [54] have been developed that record this form of provenance.

To date there has been little interaction between workflow and data provenance research. In the workflow setting, the solution to recording interactions with databases is simply to record the state of the database itself (which requires us also to record the database management system) and the query. Note that this solution presupposes some form of archiving. Quite recently, the importance of combining the two views of provenance has been recognized. In [10] there is a proposal for a stream-based model of workflow equipped with a notion of provenance for elements of a stream. In [45] a more sophisticated model is developed based on Petri nets in which the tokens are complex objects and the transitions can perform certain elementary operations on these objects. While this is a rather low-level description of workflows, it may well lead to a general notion of provenance that embraces both data and workflow provenance. Reconciling these two notions is one of the most immediate and interesting challenges of provenance research.

5. ARCHIVING AND CITATION

Most curated databases ask you to cite them, or provide a link to them, but they differ widely in suggesting how you are to do this. The IUPHAR database, for example, provides links to the whole database or to one of the receptor families (a group of entries). The problem with this is that anyone using this link gets the current version of the database rather than the version that was used when the link was created. Since the database may be expected to change, the usual principles of citation dictate that one should cite, or link to, the appropriate version of the database. This requires that all old versions are recoverable even when the database gets constantly updated. There are numerous ways to do this – from simply keeping all older versions of the database and optionally compressing them with tools such as `zip`, to logging all updates made to a database or keeping differences between versions of the database. Such methods for archiving generally work very well if the objective is simply to ensure that all past versions are recoverable. However, they are not friendly to temporal or longitudinal queries that are common for databases that change. For example, it is frequently the case that temporal queries on demographic and economic data arise when one browses the World Factbook. It would be difficult to answer such queries over the archives constructed with the these methods without at least an attempt to evaluate the query on each version of the database or an analysis of the log or diff files in the archive.

5.1 Database Archiving

Commercial relational database systems have started to provide for "time travel" queries to temporal databases [50, 53]. The traditional method of providing past snapshots of the database have been implemented by undoing the *transaction logs*, which are a critical component of transaction processing. This technique is problematic because detailed transaction logs are usually not retained permanently. Since they can only be used to reconstruct past database states, it is difficult to compare between versions of the database using

the transaction log. An alternative, well-known method [67] is to tag each tuple with the interval during which it remained present and unaltered in the database, i.e., its *transaction time*. An update to a tuple at time t causes the old tuple to be preserved with t as its end time and a new tuple to be generated with t as its start time. Past states of the database are recovered by selecting the subset of tuples that were active at a given time.

An alternative strategy for archiving hierarchical data [16] essentially keeps all database versions intact in a compact *database archive*. By viewing hierarchical data as a tree, each node is associated with a time interval that captures the time during which the node exists in the database (aka the transaction time in [67]) if it is different from the time interval of of its parent node. This is a generalization of the *fat node* method in "persistent" data structures described in [32]. The main technical difficulty in [16] is how to merge a current version of the database with the database archive. The solution in [16] leverages hierarchical key constraints [15] that naturally arise in well-organized curated databases such as UniProt. In the presence of hierarchical key constraints, it becomes possible to identify a node in a tree in a way that is invariant to updates that are performed on the tree. An archive is incrementally created by merging a node in the database version with its identical node in the archive, if present. Otherwise, a new node is created in the database archive. This method has been shown to work well for databases in which updates are mostly additions and when a node tends to persist through many versions of the database. Curated databases such as UniProt possess these two properties and it has been shown that this archiving technique is a space-efficient method for recording all past versions. Since the archive is effectively another database, this archiving technique is also a promising solution for answering a range of temporal queries over hierarchical data by, essentially, executing them directly on the archive [60]. Temporal queries for XML are starting to attract some interest [37, 60, 71].

An open question is whether one could create an archive directly from the transaction log. For hierarchical databases that are actually stored in a native XML store, it may also be possible to use the archive itself as the database.

5.2 Citation

For electronic documents, it is not yet clear whether citations are needed when one has reliable hyperlinking. This is a big "if": it assumes that the hyperlinking mechanism – such as a URL or digital object identifier – is stable and that the target of the hyperlink, the cited document, continues to exist. An electronic citation often carries a hyperlink, but it is more than a hyperlink in that it carries a small amount of extra information such as title and author which may help the reader recognize the cited work and avoid the need to dereference the link. If we assume that citations will continue to be used in electronic documents, then they will also be needed for curated databases, but the citation system needs to be more flexible. It is appropriate to cite the authorship of an entry in the IUPHAR database in a citation, since the authorship is recorded in the database, and the authors want to be cited. This is not the case for other curated databases, but some alternative information might be appropriate instead. Initial proposals on how to generate such citations automatically from the database and how to organize the database so that it is citeable have been reported in [12]. We should emphasize that proper archiving is needed both for accurate citation and hyperlinking.

6. EVOLUTION OF STRUCTURE

The design of a database is almost impossible to get right on the first try. The best one can hope for is to build something that is workable and that can be extended without too much disruption to the applications that were written for the initial design. For example, adding columns to a table in a relational database seldom interferes with existing applications. Evolution of structure is particularly problematic for curated databases. First, curated databases are often constructed "on the cheap", starting from a design that is not well-informed by database design principles. Second, once the database becomes widely used, it is often useful to extend it with additional data imported from other databases. Third, with the advance of scientific theories, the "world model" e may change over time. Finally, the data model itself may not initially be precise. This is especially true of curated databases that evolve from semistructured or text format into a more structured description.

Curated databases almost invariably consist of a sequence of entries, each of some hierarchical form. That biologists needed a flexible method of building databases was first recognized in the development of AceDB [68], based on a simple edge-labeled tree model of data. Like the first proposals for semistructured data [38], an AceDB database could be created without a schema, and a schema could, if needed, be retro-fitted to the data. By encoding lists as binary trees and by allowing pointers to top-level entries, a range of data types could be simulated in AceDB. The fact that it is still in use today and has found applications outside of biology indicates the need for a schema-less, semistructured approach to database construction, at least in the early stages of database construction. Since eventually one will want to retro-fit a schema to the data, it also points to the need of automatic schema inference for semistructured data, a topic that has been investigated for web pages [74] as well as XML [4, 6, 7], and ad-hoc semi-structured data [34].

The evolution of the schema of the UniProt entry in Figure 1 is interesting. Originally, this schema consisted of a simple, line-oriented format which could not be extended because it would break existing parsing tools. As a workaround, extra fields such as `FUNCTION`, `SUBUNIT`, and `SIMILARITY` were embedded in the comment field, something that has been retained in the XML representation of UniProt entries. So here, an extension that is easy for relational representations is complicated by the fact that we are using a serial data format.

The World Factbook shows a different kind of evolution. It appears from past versions that the top-levels of the hierarchical description of each country (describing categories and properties) are reasonably stable, but that there is great variation in the sub-properties. One of the problems here is that for categories such as "government" there is no easy way of describing all possible governmental structures by a uniform list of properties and sub-properties. For example, *Government/Elections/Althing* was a property unique to Iceland that has been deprecated in recent versions of the World Factbook. Only in the past year or so has a schema been explicitly published; before that the schema was implicit in the data (text or HTML).

6.1 XML and the evolution of structure

The hierarchical structure of curated data naturally suggests using XML. The conventional wisdom in the research community is that DTDs or XML Schemas are the most appropriate method for specifying the structure of these databases. We believe this is debatable, if for no other reason than that DTDs or XML Schemas seem to be used rarely in practice, and when they are, only a few of their features are actually used [5].

We believe that one reason for this is that DTDs do not lend themselves to even simple forms of *schema evolution*. In relational databases, adding a column is one of the most harmless operations: it is a reasonable bet that existing applications will continue to work correctly after such a modification. This is an example of record subtyping in programming languages [65], where we can always use a record with fields A, B, C anywhere one with fields A, B is expected.

Suppose, however that we want to do the same with an XML transformation specified against a DTD. For example, suppose we want to insert a new symbol a into a regular expression r. One possibility is to add a at the end of r, thereby obtaining ra. Essentially all work on typechecking and subtyping for XML programming and query languages is based on language inclusion. Thus, a transformation that expects an element of r may break if we provide an element of ra, since the language ra is not a subtype of (that is, contained in) r.

Therefore, it appears worthwhile to investigate alternative definitions of subtyping that provide more flexibility when the schema changes. For example, consider a form of *width* subtyping: r is a subtype of r' if every element of r' is a prefix of some element of r. This would suffice for the simple example above, but it might not be ideal either, because XML's ordered data model forces some arbitrary choices. For example, suppose we add add b to the end of ra get subtype rab, and have a query q that uses r and b but not a. If we later remove a, then this form of subtyping would provide no guarantee that q would still work on rb.

A possible refinement that avoids this order-dependence is to base subtyping on *interleaving*. If we define $r_1 \# r_2$ to be the set of all strings constructed by interleaving some string in r_1 with some string in r_2 we can express both record subtyping and the order constraints imposed by regular expressions. Complexity issues associated with such an operator have been studied in [42, 43, 56]. In particular, removing interleaving can lead to an exponential increase in the size of the regular expression, as is apparent from $a\#b\#c\# \ldots$.

It is not at all clear that either of these approaches is satisfactory; however, until a good solution is found, XML processing languages that are based on regular expression types and inclusion subtyping [3, 46] will continue to have problems with the kinds of extensibility to which we are accustomed in database work. XML Schema tries to address this problem simply by including both unordered record and regular expression types, but it forbids their mixture.

In the face of this complexity one is forced to ask why we are using DTDs to describe the structure of a curated database in the first place? Perhaps we are confusing three things: the underlying data model, the format by which the data is commonly displayed, and the transmission format. A simple complex-object model in which records, sets, and lists can be freely combined, with record subtyping rather than inclusion subtyping, is often adequate as an underlying data model for the curated databases we have encountered. Style-sheets for presentation based on such a model are easy to construct, as is an appropriate variant of XPath. Note that most XPath expressions are insensitive to the addition of new tags, so we would expect them to have the same kinds of guarantees about extensibility as we do for relational databases and SQL. Thus, perhaps the problems involving the evolution of structure are better addressed on the complex-object level, and XML should be used only for presentation and data transmission.

If we focus on these better-motivated aspects of XML, and not on using it to store or manage data within a database, a number of interesting questions arise in connection with optimality. Suppose one wants to move a substantial quantity of data from one database to another. What is the best XML serialization format? Even if the databases' schemas and contents are identical, this may be nontrivial – for example, suppose we have two tables with the same schemas and the same constraints, but the tables are clustered on different keys. There is an immediate problem of constructing the best serial format to optimize some combination of extracting and loading costs; there appears to be little work on this topic.

6.2 Object fission and fusion

There is a form of evolution that is problematic for curated databases in which data entries split or have to be merged. Splitting (fission) is a phenomenon one would expect in the World Factbook over its existence. Merging (fusion) also occurs in genetic databases when it is discovered, for example, that two entries refer to the same gene. Fusion has been studied in the context of data integration [63] but fusion in curated databases is qualitatively different because it is a temporal phenomenon. The problem is acute in connection with archiving and temporal queries over the history of a database. To deal with this phenomenon, UniProt introduces and "retires" object identifiers, but records the retired identifiers along with the new, primary, identifier. This provides some partial evidence of how entries were split and merged, but it is far from complete. Given that fission and fusion are so fundamental to the evolution of databases, they deserve better treatment in data models, which should support, at least, provenance queries of the general form form: "What happened to X?" or "How did Y come about?".

6.3 Ontologies

A number of curated databases are now called "ontologies" rather than "databases". There appear to be two, almost orthogonal, issues at play here. The first is that ontologies involve hierarchical classification or taxonomies. The Gene Ontology [70] has three such hierarchies erected over an underlying database of entries. From a database perspective, the most interesting challenge is the representation and recursive querying of hierarchical data – a challenge that has been largely solved, but has not been realized in practical systems.

The second point is that useful tools [57, 61] have been built for the construction of knowledge representation in the style of KL-One [11]. One of the attractions of these systems is that one can, as with semistructured data, create databases without having first constructed a schema; although a better description of such an ontology would be that it does not distinguish between the construction of schema and

data. As we have noted, in the early stages of construction of a curated database, it is an advantage to be unconstrained by a fixed schema; and this may be the reason for the growing popularity of these tools in curated databases. As far as we are aware the study of many of the topics described in this paper are as much in their infancy for ontologies as they are for databases.

7. OTHER ISSUES AND CONCLUSIONS

Curated databases require a new approach to the construction of databases. While much of the data in curated databases is derived or extracted from other sources, they are not views. Their value lies in the fact that the selection, the annotation and the cleaning of data has all been done by people. We are certainly very far from having a consistent story to tell about them, but there are some threads such as the connections between annotation, provenance, updates, archiving, and evolution, that we have attempted to draw out in this paper.

There are, naturally, many other topics that we have not covered. In every interaction we have had with database curators, data cleaning [33] is a major issue. So is the whole field of data exchange. And of those topics we have investigated there is much more to be done. There has been recent work on provenance in XML and semistructured data [25, 35]. The connection between provenance and program debugging is in its infancy. Moreover, if ontologies are also to be used to build curated databases, surely the issues of provenance, annotation and archiving apply equally well to them.

If finding good theoretical models in which to reason about curated databases is challenging, bringing this theory to practice is even more so. For example, how could we practically capture the copy-paste operation that is built into desktop software in order to capture provenance? Can we build databases that evolve in structure but provide guarantees on the functioning of existing software? Is capturing everything we would like in provenance really feasible? If not, what are the key tradeoffs? To re-iterate our initial claim, the heart of the problem is finding simple models for the processes in curated databases, and this is only going to be solved through combined efforts from theory and practice.

Acknowledgements. The authors thank their colleagues Umut Acar, Amal Ahmed, Adriane Chapman, Laura Chiticariu, Wenfei Fan, Sanjeev Khanna, Leonid Libkin, Heiko Mueller, Chris Rusbridge, Keishi Tajima and Val Tannen for their help and for their collaboration on many of the topics in this paper.

The authors also gratefully acknowledge support from the following sources: Peter Buneman and James Cheney, the United Kingdom Engineering and Physical Sciences Research Council; Wang-Chiew Tan, National Science Foundation (NSF) CAREER Award IIS-0347065 and NSF grant IIS-0430994. Stijn Vansummeren is a Postdoctoral Fellow of the Research Foundation - Flanders.

8. REFERENCES

[1] C. Aravindan and P. Baumgartner. Theorem proving techniques for view deletion in databases. *J. Symb. Comput.*, 29(2):119–147, 2000.

[2] A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence data bank and its supplement trEMBL. *Nucleic Acids Research*, 25(1):31–36, 1997.

[3] V. Benzaken, G. Castagna, and A. Frisch. CDuce: an XML-centric general-purpose language. In *ICFP 2003*, pages 51–63. ACM, 2003.

[4] G. J. Bex, W. Gelada, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. In *WWW 2008*, 2008.

[5] G. J. Bex, F. Neven, and J. V. den Bussche. DTDs versus XML Schema: a practical study. In *WebDB 2004*, pages 79–84, New York, NY, USA, 2004. ACM.

[6] G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise DTDs from XML data. In *VLDB 2006*, pages 115–126, 2006.

[7] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML schema definitions from XML data. In *VLDB 2007*, pages 998–1009, 2007.

[8] D. Bhagwat, L. Chiticariu, G. Vijayvargiya, and W. Tan. An annotation management system for relational databases. *VLDB Journal*, 14(4):373–396, 2005.

[9] S. Bowers, L. Delcambre, and D. Maier. Enriching documents in an information portal using superimposed schematics. In *dg.o '02: Proceedings of the 2002 annual national conference on Digital government research*, pages 1–6. Digital Government Research Center, 2002.

[10] S. Bowers, T. McPhillips, B. Ludaescher, S. Cohen, and S. B. Davidson. A model for user-oriented data provenance in pipelined scientific workflows. In Moreau and Foster [59], pages 133–147.

[11] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[12] P. Buneman. How to cite curated databases and how to make them citable. In *SSDBM 2006*, pages 195–203. IEEE Computer Society, 2006.

[13] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGMOD 2006*, pages 539–550, 2006.

[14] P. Buneman, J. Cheney, and S. Vansummeren. On the expressiveness of implicit provenance in query and update languages. In *Database Theory - ICDT 2007*, volume 4353 of *LNCS*, pages 209–223, 2007.

[15] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39(5):473–487, 2002.

[16] P. Buneman, S. Khanna, K. Tajima, and W. Tan. Archiving scientific data. *ACM Trans. Database Syst.*, 27(1):2–42, 2004.

[17] P. Buneman, S. Khanna, and W. Tan. On the propagation of deletions and annotations through views. In *PODS 2002*, pages 150–158. ACM, 2002.

[18] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Database Theory - ICDT 2001*, volume 1973 of *LNCS*, pages 316–330, 2001.

[19] P. Buneman, S. A. Naqvi, V. Tannen, and L. Wong. Principles of programming with complex objects and collection types. *Theor. Comp. Sci.*, 149(1):3–48, 1995.

[20] Central Intelligence Agency. The world factbook. `http://www.cia.gov/cia/publications/factbook/`.

[21] A. Chapman and H. V. Jagadish. Issues in building practical provenance systems. *IEEE Data Eng. Bull.*, 30(4):38–43, 2007.

[22] J. Cheney. Program slicing and data provenance. *IEEE Data Eng. Bull.*, 30(4):22–28, 2007.

[23] J. Cheney. Lux: A lightweight, statically typed XML update language. In *ACM SIGPLAN Workshop on Programming Language Technology and XML (PLAN-X 2007)*, pages 25–36, 2007.

[24] J. Cheney, A. Ahmed, and U. A. Acar. Provenance as dependency analysis. In *Database Programming Languages - DBPL 2007*, volume 4797 of *LNCS*, pages 139–153. Springer, 2007.

[25] L. Chiticariu and W. Tan. Debugging schema mappings with routes. In *VLDB 2006*, pages 79–90, 2006.

[26] L. Chiticariu, W. Tan, and G. Vijayvargiya. DBNotes: A post-it system for relational databases based on provenance. In *SIGMOD 2005*, pages 942–944, 2005. (Demonstration paper).

[27] G. Cong, W. Fan, and F. Geerts. Annotation propagation revisited for key preserving views. In *CIKM 2006*, pages 632–641. ACM, 2006.

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

[29] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of

view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[30] N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS 2007*, pages 1–12. ACM, 2007.

[31] R. D. Dowell, R. M. Jokerst, A. Day, S. R. Eddy, and L. Stein. The distributed annotation system. *BMC Bioinformatics*, 2:7, 2001.

[32] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making Data Structures Persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

[33] W. Fan. Dependencies revisited for improving data quality. In *PODS 2008*. ACM, June 2008. These proceedings.

[34] K. Fisher, D. Walker, K. Q. Zhu, and P. White. From dirt to shovels: fully automatic tool generation from ad hoc data. In *POPL 2008*, pages 421–434. ACM, 2008.

[35] J. N. Foster, T. Green, and V. Tannen. Annotated XML: Queries and provenance. In *PODS 2008*. ACM, June 2008. These proceedings.

[36] M. Y. Galperin. The molecular biology database collection: 2008 update. *Nucleic Acids Research*, 36, 2008.

[37] D. Gao and R. T. Snodgrass. Temporal slicing in the evaluation of XML queries. In *VLDB 2003*, pages 632–643, 2003.

[38] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *J. Intell. Inf. Syst.*, 8:117–132, 1997.

[39] P. Gardner, G. Smith, M. Wheelhouse, and U. Zarfaty. Local hoare reasoning about DOM. In *PODS 2008*, June 2008. These proceedings.

[40] F. Geerts, A. Kementsietsidis, and D. Milano. MONDRIAN: Annotating and querying databases through colors and blocks. In *ICDE 2006*, page 82. IEEE Computer Society, 2006.

[41] F. Geerts and J. Van den Bussche. Relational completeness of query languages for annotated databases. In *Database Programming Languages - DBPL 2007*, volume 4797 of *LNCS*, pages 127–137, 2007.

[42] W. Gelade, W. Martens, and F. Neven. Optimizing schema languages for XML: Numerical constraints and interleaving. In *Database Theory - ICDT 2007*, volume 4353 of *LNCS*, pages 269–283. Springer, 2007.

[43] G. Ghelli, D. Colazzo, and C. Sartiani. Efficient inclusion for a class of XML types with interleaving and counting. In *Database Programming Languages: DBPL 2007*, volume 4797 of *LNCS*, pages 231–245, 2007.

[44] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS 2007*, pages 31–40. ACM Press, 2007.

[45] J. Hidders, N. Kwasnikowska, J. Sroka, J. Tyszkiewicz, and J. V. den Bussche. DFL: A dataflow language based on petri nets and nested relational calculus. *Inf. Syst.*, 33(3):261–284, 2008.

[46] H. Hosoya and B. C. Pierce. XDuce: A statically typed xml processing language. *ACM Trans. Interet Technol.*, 3(2):117–148, 2003.

[47] T. Imielinski and J. Witold Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[48] IUPHAR receptor database. `http://www.iuphar-db.org`.

[49] S. Jones, D. Abbott, , and S. Ross. Risk Assessment for AHDS Performing Arts Collections: A Response to the Withdrawal of Core Funding. Technical report, Glasgow, December 2007.

[50] S. Kumar and T. Bednar. Oracle9i flashback query. Technical report, Oracle Corporation, 2001.

[51] T. Lee, S. Bressan, and S. E. Madnick. Source attribution for querying against semi-structured documents. In *First Workshop on Web Information and Data Management*, pages 33–39. ACM, 1998.

[52] H. Liefke and S. B. Davidson. Specifying updates in biomedical databases. In *SSDBM 1999*, pages 44–53. IEEE, 1999.

[53] D. Lomet, R. Barga, M. F. Mokbel, G. Shegalov, R. Wang, and Y. Zhu. Immortal DB: transaction time support for SQL server. In *SIGMOD 2005*, pages 939–941. ACM, 2005.

[54] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice & Experience*, 18(10):1039–1065, 2006.

[55] P. Maniatis, M. Roussopoulos, T. J. Giuli, D. S. H. Rosenthal, and M. Baker. The LOCKSS peer-to-peer digital preservation system. *ACM Trans. Comput. Syst.*, 23(1):2–50, 2005.

[56] A. J. Mayer and L. J. Stockmeyer. Word problems-this time with interleaving. *Inf. Comput.*, 115(2):293–311, 1994.

[57] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera ontology environment. In *Proceedings of Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 1123–1124. AAAI Press, 2000.

[58] V. A. McKusick. OMIM - online mendelian inheritance in man. `www.ncbi.nlm.nih.gov/omim/`.

[59] L. Moreau and I. T. Foster, editors. *Provenance and Annotation of Data - IPAW 2006*, volume 4145 of *LNCS*. Springer, 2006.

[60] H. Müller, P. Buneman, and I. Koltsidas. XArch: Archiving scientific and reference data. In *SIGMOD 2008*, June 2008. Demonstration Paper. To appear.

[61] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, and M. A. Musen. Creating semantic web contents with Protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[62] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[63] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *VLDB 1996*, pages 413–424. Morgan Kaufmann, 1996.

[64] Plutarch. Vita Thesei 22-23.

[65] D. Rémy. Type inference for records in a natural extension of ML. In *Theoretical aspects of object-oriented programming*. MIT Press, 1994.

[66] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE 2006*, page 7. IEEE Computer Society, 2006.

[67] R. T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, July 1999.

[68] L. D. Stein and J. Thierry-Mieg. AceDB: A genome database management system. *Computing in Science and Engg.*, 1(3):44–52, 1999.

[69] W. Tan. Containment of relational queries with annotation propagation. In *Database Programming Languages - DBPL 2003*, volume 2921 of *LNCS*, pages 37–53. Springer, 2003.

[70] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, May 2000.

[71] F. Wang and C. Zaniolo. Temporal queries in XML document archives and web warehouses. In *TIME*, pages 47–55. IEEE Computer Society, 2003.

[72] Y. R. Wang and S. E. Madnick. A polygen model for heterogeneous database systems: The source tagging perspective. In *VLDB 1990*, pages 519–538. Morgan Kaufmann, 1990.

[73] M. Weiser. Program slicing. In *ICSE*, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.

[74] G. Yang, I. V. Ramakrishnan, and M. Kifer. On the complexity of schema inference from web pages in the presence of nullable data attributes. In *CIKM 2003*, pages 224–231. ACM, 2003.