Creating and Maintaining Curated View Databases

Susan B. Davidson, Hartmut Liefke
Center for Bioinformatics
Department of Computer and Information Science
University of Pennsylvania
200 South 33rd Street, Philadelphia, PA 19104
Email: susan@cis.upenn.edu, liefke@seas.upenn.edu

Limsoon Wong
Kent Ridge Digital Labs
21 Heng Mui Keng Terrace, Singapore 119613
Email: limsoon@krdl.org.sg

1 Introduction

The process of building a new database relevant to some field of study in biology involves transforming, integrating, and cleansing multiple external data sources, as well as adding new material and annotations. For example, EpoDB is a database created at the University of Pennsylvania Center for Bioinformatics. It was designed to study gene regulation during differentiation and development of vertebrate red blood cells. In building EpoDB, data relevant to red blood cells were *extracted* from GenBank, SWISS-PROT, TRRD (transcriptional regulation data), and GERD (expression levels data). Once extracted, the data were *cleansed* of errors using a semi-automated approach. Cleansed data were then *integrated*, and additional information or *annotations* entered either automatically or manually to the integrated view.

Databases such as EpoDB can be thought of as secondary or derived "value-added" databases. They are proliferating in the biomedical world. In the database community, they are also called *warehouses*, or *materialized views* due to the fact that they extract and integrate data from existing databases. Creating and maintaining these view databases raise a number of problems:

1. How can we specify and implement the transformation and inte-

gration from the underlying source databases to the view database?

2. How can we automate the refresh process?

3. How can we track the origins or "provenance" of data?

The first problem, that of transforming and integrating the source databases to produce the view database, has been the subject of much research in the biomedical as well as database community for the past ten years. The problem is complicated by the fact that the underlying data sources—GenBank, SWISS-PROT, TRRD, GERD, etc.—are stored using different data models or data formats such as ASN.1, EMBL flat file, relational, etc. Solutions to this problem include creating linking databases [12]; relying on commercial relational database technology; adopting a complex or object-oriented view of data, as in the Kleisli [35], Garlic [16], or CORBA integration approaches; or using a semistructured integration strategy, as in Tsimmis [21,13], or more recent proposals with XML [1,28].

Part of the second problem has received a lot of attention by the database community in the context of updating materialized views of relational databases. However, the results have not been generalized to the more complex types of view definitions that can be found when integrating biomedical databases, nor have they taken into consideration the replaying of corrections and annotations that must be applied. The result is that the update process for many secondary databases rely on hand-written scripts, and are therefore quite expensive to write and are difficult to modify as the primary or secondary data source schemas evolve. This affects the periodicity with which refreshing occurs, since the need for information that is as current as possible must be balanced with the expense of keeping the view current. It is therefore quite common for there to be a lag of several months between a change being made to a source database, and it being propagated to the secondary view database.

A second part to the problem of automating the refresh process is determining exactly what changes have occurred to the source databases. In the context of biomedical databases, it is complicated by the fact that updates are typically propagated in one of three ways:

1. Producing periodic new versions which can be uploaded by the user community;
2. Timestamping data entries so that users can infer what changes have occurred from the last time they accessed the data; and
3. Keeping a list of additions and corrections; each element of the list is a complete entry. The list of additions can be uploaded by the user community.

None of these methods precisely describe the minimal changes that have been made to the data. For example, SWISS-PROT adopts the third approach and publishes a list of entries that have been modified since the last release. An entry is rather large, and can consume several pages when printed on paper. The change may be very small, such as adding an extra author to a reference, which consumes only a few characters of the new entry. Depending on the view definition, this change may or may not affect the contents of the secondary database.

The last problem, tracking the origin or provenance of data, is very important but has received relatively little attention by the database community[8,34,29,20]. For example, a secondary database such as EpoDB may make predictions about where genes are located ("hypothetical genes") using sequence information and gene finding algorithms. To understand the origins of such a hypothetical gene, information about the sequence, algorithm, input to the algorithm, and perhaps even the person who supervised the prediction should be maintained.

In this chapter, we discuss the various phases of creating and maintaining curated view databases and contrast solutions where appropriate.

2 Techniques for Data Extraction and Integration

While solutions exist for data transformation and integration when the underlying data sources are all relational, they do not work in the context of biomedical data source due to their variety and complexity. For example, the "database of molecular biology databases"[a] contains in-

[a] See `www.infobiogen.fr/services/dbcat`.

formation about databases of general interest in molecular biology and genetics. Of the more than 400 databases listed, less than 10 appear to use commercial database technology. Over 50% give an ftp site from which source data is available in some structured form, and most provide some sort of web interface. A cursory examination of the ftp sites shows that quite a number of these databases use some variation of the EMBL format; others use ASN.1 or ACeDB format.

```
STANDARD;      PRT;   924 AA. AC   P15711;
DT   01-APR-1990 (REL. 14, CREATED)
DT   01-APR-1990 (REL. 14, LAST SEQUENCE UPDATE)
DT   01-AUG-1992 (REL. 23, LAST ANNOTATION UPDATE)
DE   104 KD MICRONEME-RHOPTRY ANTIGEN.
OS   THEILERIA PARVA.
RN   [1]
RC   STRAIN=MUGUGA;
RX   MEDLINE; 90158697.
RA   IAMS K.P., YOUNG J.R., NENE V.;
RL   MOL. BIOCHEM. PARASITOL. 39:47-60(1990).
DR   EMBL; M29954; G161866; -.
DR   PIR; A44945; A44945.
KW   ANTIGEN; SPOROZOITE.
FT   DOMAIN        1     19       HYDROPHOBIC.
FT   DOMAIN      905    924       HYDROPHOBIC.
```

Figure 1: EMBL-format SWISS-PROT Entry

An example of a SWISS-PROT EMBL format entry can be found in Figure 1.[b] In this example, AC refers to the accession number of the entry and forms a key. DT refers to date; there is always one creation date, and at most one sequence and annotation update. An entry can have many publications associated with it; a publication has a number (RN), several authors (RA), a literature reference (RL), and keywords (KW) among other information. An entry can also have several features (FT) that appear in its region of sequence.

Now suppose that we wish to store this entry in a *normalized* rela-

[b] See `www.ebi.ac.uk/swissprot` for details of what the various tags mean.

tional database, which requires that all fields in a table be single-valued facts about the key of the table [32]. In the resulting relational schema, an entry is split over about 15 tables. For example, the literature in an entry is not a single-valued fact about the key of an entry (AC) since there may be many publications per entry. Publications must therefore be split off as a separate table. Furthermore, since a publication could be related to several different entries, it is not enough to include AC as a *foreign key* in the publication table referencing some tuple in the entry table; a separate table denoting a many-to-many relationship between publications and entries must be created. The same reasoning can be applied to authors of a publication, keywords, features, and so on.

```
[AC: "P15711",
 CreateDate: "01-APR-1990", CreateRelease: 14,
 SeqUpdate: "01-APR-1990", SeqRelease: 14,
 AnnotUpdate: "01-APR-1992", AnnotRelease: 23,
 References: {[RN: 1,
               RA: {"IAMS K.P.", "YOUNG J.R.", "NENE V."}
               RL:"MOL. BIOCHEM. PARASITOL. 39:47-60(1990)"]}
 ... ]
```

Figure 2: Complex value representation of SWISS-PROT Entry

On the other hand, there is a natural translation of the EMBL entry of Figure 1 into a complex value or an object oriented database [10,35] since the fields in a tuple can themselves be sets, lists, bags or tuples of information. Figure 2 shows a portion of this translation into a complex value. In this figure, "[...]" is used to denote a record value; the top-level record has labels AC, CreateDate, SeqUpdate, AnnotUpdate, References. The syntax "{... }" is used to denote a set, as found in the value of References.[c] Once translated into this complex-value model, data can be transformed and integrated using a language appropriate for the model. For example, if SWISS-PROT is a set of entries in the complex object format of Figure 2, then the following Kleisli query

[c] The ellipsis in Figure 2 is used to indicate that there are fields missing in the translation from Figure 1.

derives from it a cross-tabulation of literature reference and the accession number of the SWISS-PROT entry in which the reference is made: {[AC: x.AC, REF: y.RL] | \x <– SWISS-PROT, \y <– x.References}. The syntax of this query is as follows: $[l_1:e_1, ..., l_n:e_n]$ constructs a record with attribute labels $l_1, ..., l_n$, and attribute values $e_1, ..., e_n$ respectively. The synax e.l extracts the attribute value corresponding to the attribute label l in the record e. The syntax {f(x) | \x <– e, c(x)} constructs a set containing those f(x) such that x is in the set e and c(x) is true. Note that each type—set, list, bag, record and variant—has an associated set of constructor and deconstructor operations, which can be combined to form a complex query [5].

The advantage of such an approach is that the source object can retain a complicated but natural structure that allows simpler application-specific alternative views to be derived in a straightforward manner. Furthermore, due to the extensive query optimization that is performed by the Kleisli system, this can be done very efficiently.

A problem with this approach is that the data must conform to the schema or the expected structure. In particular, all fields of a record must be present. If a field is not present, a special "null" value must be used. However, the presence of a null value can be interpreted in several different ways; for instance, the value is not known, or the attribute is inappropriate. When new types of information become available, fields must be added to the record. Such schema changes are expensive, and can proliferate the amount of wasted space or null values in the database.

Due to these problems, data models with less required structure or whose structure is "self-describing" or explicitly represented in the data appear promising. Such formats are called *semistructured* data models [2]. The ACeDB model [31] might loosely be considered an example of semistructured data. Although it has a schema, the schema imposes only weak constraints on the data. For example, it can accommodate missing data. The type system used by ACeDB—that of an edge-labeled graph—is also remarkably close to that adopted for semistructured data. However, this system was intended as a single database system and not an integration system. Another example is the Tsimmis project [21,13], which is an integration system that uses a semistructured model and the Lorel language for manipulating such data.

The newly emerging XML data exchange standard[d] is also an example of a semistructured data model. Unlike ASN.1, there is no separate schema to constrain the structure of the data. What this means in practice is that it will be possible to make ad-hoc changes to the structure of individual components, such as additional fields and missing elements, and collections of these elements may be heterogeneous. Figure 3 shows one possible example of a translation of the SWISS-PROT entry into XML. Note that the type is carried in the labels, which include tags (e.g. Entry, Mod and PrimAC), attributes (e.g. date, Rel and type within the Mod tag), and PCDATA (e.g. "CREATED" and "104 KD MICRONEME-RHOPTRY ANTIGEN").

The semistructured data model can accommodate schema changes because it needs no schema. It can also accomodate complex forms of annotations because it allows for arbitrary forms of nestings. Thus the semistructured data model may appear as a solution to biological data integration problems where the schemas evolve with the progress of experimental techniques and scientific discovery. In fact, data integration may be performed simply by ignoring the structure using one of the query languages developed for semistructured data or XML[11,23].

Nevertheless, we believe this approach should be treated with a great deal of caution. Biological data has a very rich structure which should not be ignored just because that structure evolves. In addition, the semistructured approach has the following serious dangers:

- Queries that would normally fail, in the sense that they would not execute, because of a type error will now succeed. They will probably "succeed" by returning the "empty" answer.

- Users who misunderstand the implicit structure of the data will, as a consequence of the previous point, be further confused by the results of queries.

- Optimization techniques that are essential to efficient data integration are much more difficult to obtain with semistructured data.

[d] See www.w3.org.

```
<Entry mtype="PRT" seqlen="924">
   <PrimAC>P15711</PrimAC>
   <Mod date="01-APR-1990" Rel="14" type="CREATED"></Mod>
   <Mod date="01-APR-1990" Rel="14" type="LAST SEQ UPD"></Mod>
   <Mod date="01-AUG-1992" Rel="23" type="LAST ANNOT UPD"></Mod>
   <Descr>104 KD MICRONEME-RHOPTRY ANTIGEN</Descr>
   <Species>THEILERIA PARVA</Species>
   <Ref num="1">
      <STRAIN>MUGUGA</STRAIN>
      <MedlineID>90158697</MedlineID>
      <Author>IAMS K.P.</Author>
      <Author>YOUNG J.R.</Author>
      <Author>NENE V</Author>
      <Cite>MOL. BIOCHEM. PARASITOL. 39:47-60(1990)</Cite>
   </Ref>
   <EMBL prim_id="M29954" sec_id="G161866" status="-"></EMBL>
   <PIR prim_id="A44945" sec_id="A44945"></PIR>
   <Keyword>ANTIGEN</Keyword>
   <Keyword>SPOROZOITE</Keyword>
   <Features>
      <DOMAIN from="1" to="19">
         <Descr>HYDROPHOBIC</Descr>
      </DOMAIN>
      <DOMAIN from="905" to="924">
         <Descr>HYDROPHOBIC</Descr>
      </DOMAIN>
   </Features>
</Entry>
```

Figure 3: A SWISS-PROT Data Entry in XML

In short, the use of semistructured formalisms and query languages, while they may simplify certain aspects of data integration, will not eliminate the need for biologists to understand their data.

However the ability to incorporate some aspects of semistructured data into integration systems is clearly useful for dealing with evolution and for browsing. One would like, for example, to add a field to some data without having to change a schema and all the applications that depend on it. Equally one would like to ask questions of the database that return some structural information, such as "Where in the database is 'ubiquitin'?" Interestingly, the internal structure of Kleisli and K2 do support this. Unlike many object-oriented and relational systems, the internal representation of data in Kleisli and K2 is dynamic and, if the homogeneous type contraint on sets and other "bulk" data types of Kleisli and K2 are ignored, is a superset of the semistructured data model. This means that it is possible to add semistructured features to the existing query languages of Kleisli and K2, or to develop new query languages for Kleisli and K2. In fact, Kleisli provides a set of functions for semistructure-style navigation and transformation on its data model.

Apart from providing a low-level data format, XML will not do more for us until there is an agreed schema technology. At the time of writing there is a bewildering variety of proposals for adding structure to XML and no standard even for the simple task of representing relational data in XML has yet been defined. It is to be hoped that some form of type system for XML will soon gain widespread acceptance.

The discussion above considers only the data model aspect of data integration. After a common data model—relational, complex object, or XML—is chosen as a representation for the individual data sources, appropriate query languages—SQL, OQL [6], CPL (the query language of Kleisli), XML-QL [11] —for the data model are used to express various transformations to integrate these individual data sources. The actual integration has to consider a second orthogonal aspect that deals with whether there exists a separate physical copy of the integrated database, or whether the integration describes how to translate queries against an integrated view against the underlying data sources [9]. The former is sometimes known as the instantiated or materialized approach. The later is sometimes known as the virtual or links approach. Interestingly, both approaches can use the same data model and even the same queries. In the materialized case, the integration queries are executed in advance to build a precomputed separate physical copy of the integrated database. In the virtual approach, for each set of search parameters supplied by the user at a particular time, the queries are specialized to those particular search parameters and are executed to build just a (hopefully) small relevant part of the integrated database as needed to satisfy the information request of the user for that particular time.

The advantages of the instantiated approach is that system performance tends to be much better than is possible in a distributed environment. First of all, query optimization can be performed locally, assuming a good underlying database management system; second, inter-data source communication latency is eliminated. System reliability should also be higher since there are fewer dependencies on network connectivity and original data sources, assuming the integrated system is reliable. It is also much easier to enforce any inter-database constraints that have been determined in the integration[24,33]. The most important advantage is perhaps the fact many current data sources contain many errors. Moreover, some of these data sources such as the PDB[e] either take a long time to correct errors or do not correct errors, because of practical reason, archival policy, or other reasons.[f] Therefore, the only feasible way for the integrated database to have correct data is to keep a separate cleansed copy.

On the other hand, the advantages of the virtual approach are that it has very low initial cost, very little maintenance cost, and is always up-to-date. The fully materialized integrated database might require a lot of space and computations to set up. For example, a part of the integrated database might be a large set of protein sequences together with results of applying a large number of time-consuming domain prediction algorithms on them. A virtual version of this integrated database

[e] See www.rcsb.org/pdb.

[f] Helen Berman, private communication, May 2000.

might need to perform these domain predictions on just a small number of protein sequences requested by the user. Furthermore, every time an update is made in one of the underlying databases, it must be reflected in the materialized integrated instance. This may involve a cascading of complex queries, instead of just a simple update.

In practice, almost all integrated databases are a hybrid of the instantiated and the virtual. In the rest of this chapter, we focus on two practical problems on maintaining the instantiated part of an integrated database. The problem of detecting changes in the underlying data sources is discussed in section 3. The problem of propagating changes is discussed in section 4.

3 Techniques for Detecting Change

As mentioned earlier, changes in biomedical data sources such as SWISSPROT and EMBL are frequently published as modified entries. That is, an entry typically has an associated create date, modify date—typically referring to the modification of non-sequence information, and sequence modify date, as well as version number information. This is helpful for detecting which entries have been modified or inserted since the materialized view or derived database was last updated. However, outside of sequence updates, the modify date does not help in knowing exactly where the change has occurred.

For example, suppose that an entry is being mapped to a relational database, and recall that this mapping may result in an entry being split over 15 tables. If the change is the addition of a single author to an existing table, only one tuple needs to be inserted to a table in the view. On the other hand, remapping the entire entry will entail updates to all 15 tables in the view.

The issue of change detection has been studied in several different contexts. For example, the UNIX "diff" utility takes two text files, compares them line by line and outputs how one file has changed relative to another; and the UNIX "patch" utility updates the old data file to become the new data file. Calculating the edit distance between two character strings has also been extensively studied in computational biology due to its importance in calculating sequence similarity.

However, none of these techniques are particularly useful in the context of our data since its type is tree-structured—whether represented as a complex value or as an XML value—rather than a flat string or sequence of lines. Work on computing the difference between ordered trees [25,30,36,37,7] is therefore more promising for comparing entries.[g]

In computing the difference between two ordered trees, three basic types of update operations are considered: changing a node, deleting a node, and inserting a node, as depicted in Figure 4. *Changing* a node is straightforward, as shown in Part (a). *Deleting* a node c with parent p entails deleting c and promoting its children to become children of p, as shown in Part (b). *Inserting* a node c at a specified position under p is the inverse of deleting: a specified sub-sequence of p's children become children of c, as shown in Part (c). Several *aggregate* operations are also used—pruning a subtree rooted at a specified node, grafting a subtree at a specified position, and moving a subtree. As with sequence comparison algorithms, a cost is associated with each operation. Note that although each aggregate operation is equivalent to a sequence of node operations the cost of the aggregate operation may be cheaper than the equivalent sequence of node operations. The goal of an ordered tree diff algorithm is to find a minimal cost sequence of operations that maps the old tree to the new tree.

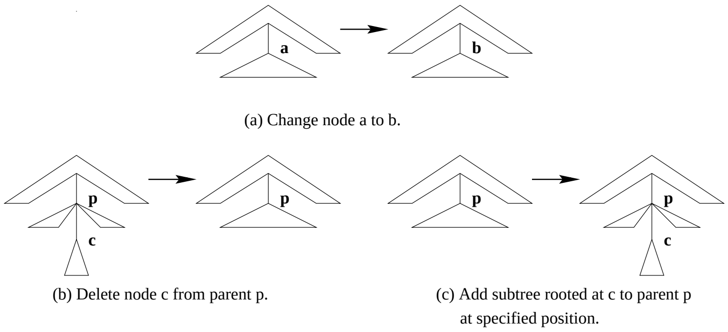

Figure 4: Update operations on trees

Several packages for comparing ordered trees and XML documents

[g] An ordered tree is a tree in which the children are ordered.

are available over the web. One such package is IBM's XMLTreeDiff,[h] which takes an XML document and parses it into a DOM structure[i] which represents the XML data as an ordered tree. The diff Java bean associated with XMLTreeDiff implements the algorithm in [36] and produces an edit script in the XPath notation.[j] In XPath, the "tree address" of a given node is the concatenation of the child position (left-to-right ordering) of each node on the path from the root to the given node. A picture of the tree address form of the XML SWISS-PROT entry of Figure 3 is shown in Figure 5, with the child node number shown in square brackets to the left of each node.

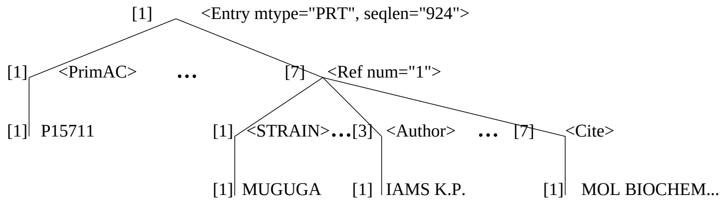

Figure 5: Tree address form of original SWISS-PROT entry

As an example of the output of XMLTreeDiff, suppose the XML SWISS-PROT entry of Figure 3 is modified as shown in Figure 6. The actual changes made were adding a new third author "DOE J.A" and adding a new feature. This also has the side effect of changing the last annotation update date and release, i.e. the attribute information of the third `<Mod>` element.[k] The output of XMLTreeDiff for the old and modified entries is shown in Figure 7. Note that the changed `<Mod>` element's tree address is `=[1][4]`, and the new author is added after node `=[1][7][4]`, and the new feature—which is a tree rather than a single node—is grafted after node `=[1][12][1]`.

Besides the modifications mentioned above, several line breaks have been added to the modified entry of Figure 6. These line breaks do not show up in the TreeDiff output, since they do not modify the XML

[h] See `www.alphaWorks.ibm.com/formula/xmltreediff`.
[i] See `www.w3.org/DOM`.
[j] See `www.w3.org/TR/xpath.html`.
[k] We do not guarantee that these changes are accurate!

```
<Entry mtype="PRT" seqlen="924">
  <PrimAC>
     P15711</PrimAC>
  <Mod
     date="01-APR-1990" Rel="14" type="CREATED"></Mod>
  <Mod date="01-APR-1990" Rel="14" type="LAST SEQ UPD"></Mod>
  <Mod date="01-OCT-1994" Rel="24"
       type="LAST ANNOT UPD"></Mod>
  <Descr>104 KD MICRONEME-RHOPTRY ANTIGEN</Descr>
  <Species>THEILERIA PARVA</Species>
  <Ref num="1">
     <STRAIN>MUGUGA</STRAIN>
     <MedlineID>90158697</MedlineID>
     <Author>IAMS K.P.</Author>
     <Author>YOUNG J.R.</Author>
     <Author>DOE J.A.</Author>
     <Author>NENE V</Author>
     <Cite>MOL. BIOCHEM. PARASITOL. 40:47-60(1990)</Cite>
  </Ref>
  <EMBL prim_id="M29954" sec_id="G161866" status="-"></EMBL>
  <PIR prim_id="A44945" sec_id="A44945"></PIR>
  <Keyword>SPOROZOITE</Keyword>
  <Keyword>ANTIGEN</Keyword>
  <Features>
     <DOMAIN from="1" to="19">
        <Descr>HYDROPHOBIC</Descr>
     </DOMAIN>
     <CDS from="150" to="1090">
        <Gene> "DMRT1" </Gene>
        <product>"doublesex and mab-3 related transcription factor 1"
            </product>
     </CDS>
     <DOMAIN from="905" to="924">
        <Descr>HYDROPHOBIC</Descr>
     </DOMAIN>
  </Features>
</Entry>
```

Figure 6: Modified SWISS-PROT entry

```
<node id="/">
  <node id="/*[1]">
    <node op="replace" type="2" name="date" id="/*[1]/*[4]/@date">
      <value>01-OCT-1994</value>
    </node>
    <node op="replace" type="2" name="Rel" id="/*[1]/*[4]/@Rel">
      <value>24</value>
    </node>
    <node id="/*[1]/*[7]">
      <node id="/*[1]/*[7]/*[4]"/>
      <node op="add" type="1" name="Author">
        <node op="add" type="3">
          <value>DOE J.A.</value>
        </node>
      </node>
    </node>
    <node id="/*[1]/*[12]">
      <node id="/*[1]/*[12]/*[1]"/>
      <node op="graft">
        <CDS from="150" to="1090">
          <Gene> "DMRT1" </Gene>
          <product> "doublesex and mab-3 related transription factor 1"
             </product>
        </CDS>
      </node>
    </node>
</node>
```

Figure 7: XMLTreeDiff Output

value. However, they do show up in the UNIX diff output, as shown in Figure 8, and if enough page layout changes were made, the UNIX diff would perform poorly. Furthermore, if several elements are combined on the same line (for example, all three `<Author>...</Author>`'s), the UNIX diff utility would notice a line change rather than an element change. Thus, while a line-oriented diff may catch changes in files, it does not match the tree structure of XML documents.

```
2,3c2,5
<  <PrimAC>P15711</PrimAC>
<  <Mod date="01-APR-1990" Rel="14" type="CREATED"></Mod>
---
>  <PrimAC>
>     P15711</PrimAC>
>  <Mod
>     date="01-APR-1990" Rel="14" type="CREATED"></Mod>
5c7,8
<  <Mod date="01-AUG-1992" Rel="23" type="LAST ANNOT UPD"></Mod>
---
>  <Mod date="01-OCT-1994" Rel="24"
>       type="LAST ANNOT UPD"></Mod>
```

Figure 8: Partial output of UNIX diff

An interesting question is whether a SWISSPROT entry necessarily corresponds to an ordered or *un*ordered tree. When Kleisli representation of an entry is thought of as a tree, the leaves correspond to base values (integers, strings, etc); and interior nodes are either records, variants or “bulk” types. Record and variant types are unordered, since the labels are distinct. Bulk types like set and bag types are also unordered. Only the list type is ordered. Therefore the Kleisli tree of an entry is most accurately thought of as a *hybrid* tree, with most of the nodes having unordered children. The problem with considering an unordered entry to be ordered is that “equivalent” entries in which attributes (nodes) have simply been moved around rather than actually changed will be considered different, causing unnecessary updates to be produced. Furthermore, it is difficult to see how sequences of updates (e.g. from version 1 to version 2 to version 3) could be represented as a single change (e.g. from version 1 to version 3) since the tree address of nodes will change between versions.

As it currently stands XML is ordered, hence the reliance on tree addresses to identify nodes. However, there is an increasing interest in developing "keyed" versions of XML in which nodes (elements or attributes) are identified by some value rather than by position. For example, in our sample XML value the `PrimAC` tag will never occur more than once, hence the tag itself forms a key within the `<Entry>` element. As a more involved example, even though the `Mod` tag is repeated three times, the value of the `type` attribute will always distinguish them; hence the `Mod` tag together with the `type` attribute could form a key within `<Entry>`. Similarly, the `Ref` tag together with the `num` attribute could form a key for `<Ref>` elements within `<Entry>`. A notion of keys and their syntax is being developed within XML Schema.[l] Figure 9 shows a picture of the SWISS-PROT entry under a keyed version of XML, where the symbol "`@`" is used to distinguish attributes from subelements. In this figure, the order of children is no longer important. Note that if keys are completely specified for an XML tree, then there is a unique path of node labels between the root and any node in the tree; hence any node can be identified completely by a path of values rather than by position. This property characterizes the *deterministic model* of semistructured data recently proposed in [4].

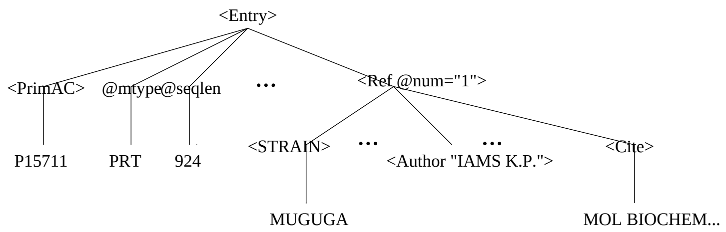

Figure 9: Keyed version of original SWISS-PROT entry

It is well known that calculating the diff of two unordered trees is

[l] see `www.w3.org/TR/xmlschema-1`.

an NP-complete problem [37]. Therefore it is unlikely that efficient algorithms can be found for unordered trees in general. However, if the operations allowed are restricted to inserting a subtree and deleting a subtree (i.e. node labels cannot change), the fact that every child of a node has a distinct keyed label admits a straightforward depth-first linear time algorithm for detecting change. The restriction that node labels are fixed corresponds to the restriction in relational databases that keys cannot be modified, and therefore has some justification. Furthermore, it considerably simplifies the problem of propagating updates through view definitions. This is the topic of the next section.

4 Propagating Updates

Once the diff has been calculated between versions of a file, it must be propagated through the view definition to the materialized view. This is commonly called "view maintenance." To understand why view maintenance may be difficult, consider the following example.

Suppose that we have a view database XREFs which consists of all the Medline accessions that are referenced by some SWISSPROT entry, and that the view contains accession 90158697. Now suppose that the SWISSPROT database is modified by deleting the Medline cross reference 90158697 in some entry. Does this imply that accession 90158697 should be deleted from our database XREFs? Possibly—but only if it is the *last remaining* SWISSPROT entry that references accession 90158697. The same reasoning applies for the insertion of a new entry in SWISSPROT with a Medline cross reference 12345678; the insertion will only carry over to our XREFs database if it is the *first* SWISSPROT entry to reference 12345678.

The example above illustrates a very common class of view definitions in relational databases called "project-select-join" or SPJ views. Such view definitions involve "monotonic" operations, which have the property that the insertion of data in a source database will never entail the deletion of data in the view database. For view definitions involving non-monotonic operations such as set difference (in SQL, "not exists") or uniqueness tests, the insertion of information in a source database may actually involve the *deletion* of a tuple in the view database. For example, suppose the view database NOTXREFs consists of all Medline accessions that are not referenced by any SWISSPROT entry, and that the Medline entry 12345678 is in NOTXREFs. Then the insertion of a new entry in SWISSPROT with a cross reference to 12345678 will entail its deletion in NOTXREFs. Similarly, the deletion of a cross reference to 90158697 in SWISSPROT may entail its insertion in NOTXREFs.

Any view, even the non-monotonic view NOTXREFs, can be maintained by completely re-evaluating the view and replacing the old materialized view with this new result. However, re-evaluation is very expensive, and so *incremental view maintenance* techniques which focus only on the *changes* to the source databases are preferred. In particular, efficient incremental view maintenance techniques exist for SPJ-views such as XREFs, and even slightly more complex view definitions involving aggregate operations such as SUM, MIN, MAX, AVG, etc. For example, given a set of insertions Δ^+ to SWISSPROT, the insertions to XREFs can be calculated by pulling out the Medline entries referenced by the set Δ^+. We will refer to this set of referenced entries as $Medline \propto \Delta^+$, where $\propto$ is the relational semi-join operator [32]. Deletions are slightly more complex, and involve keeping track of the *number* of SWISSPROT entries that reference a Medline accession in XREFs: Given a set of deletions Δ^- to SWISSPROT (or deleted cross references to Medline), the deletions to XREFs can be calculated by computing $Medline \propto \Delta^-$, grouping by common Medline accession numbers, and counting the number of deletions for each Medline accession. For each such accession, the number of deletions is subtracted from the recorded count stored with the entry in XREFs. When the count of a tuple in XREFs becomes zero, the accession is deleted from XREFs. Conceptually, the recorded count for each tuple is the number of derivations for that tuple in the view.

Figure 10 illustrates the counting method for maintaining XREFs. Part (a) shows the initial XREFs relation, with a single attribute Accession; the double line separating this attribute from Count signifies that Count is not visible to users and is only used for view maintenance. Part (b) shows the set of Medline entries referenced by some set of deletions Δ^-. Note that there are two deletions of 90158697, which when

Accession	Count
90158697	2
81761234	5

(a) The view XREFs.

Accession
90158697
90158697
81761234

(b) The set $Medline \propto \Delta^-$.

Accession	Count
81761234	4

(c) The view XREFs after deletions.

Accession
81761234
12333397

(d) The set $Medline \propto \Delta^+$.

Accession	Count
81761234	5
12333397	1

(e) The view XREFs after insertions.

Figure 10: Maintaining view XREFs using counting

subtracted from its count in XREFs gives a count of zero; 90158697 is therefore deleted from the view. There is also one deletion of 81761234, which when subtracted from its count in XREFs gives a count of four; the tuple therefore remains in the view with a reduced count. Parts (c) and (d) illustrate how the counts are maintained under insertion. Note that accession 1233397 is not already in the view; it is therefore inserted, with a count of one.

Incremental view maintenance in relational databases has been studied extensively; see [15] for a survey. Several extensions of conventional SPJ-queries have also been discussed, in particular to include aggregation and group-by operators [22]. As an example of aggregation and group-by, suppose we wished to create a view containing two attributes, keyword and num_entries, where num_entries is the number of SWISS-PROT entries which contain the keyword. The query for this view would involve grouping entries by keyword, counting the number of entries in each group, and projecting out the keyword and count. While such summary queries are very important in business-oriented data mining applications, their importance in biomedical databases is less clear.

View maintenance has also been investigated for object-oriented and complex-value databases [14,17,18], and more recently for semi-structured [38,3] and XML data [27,19]. In [19], a keyed version of XML is used to extend the counting method described above to this hierarchical form of data, with views expressed in a restricted version of a query language for XML called XML-QL [11]. The results in [19] are important for two reasons: First, the view language is rich enough to express a number of common and interesting transformations, and incremental view maintenance can still be efficiently performed. Second, they highlight the advantage of using a keyed approach to hierarchical data.

5 Conclusion

The number of secondary, value-added databases being developed within the biomedical community is increasing rapidly. The advantages of instantiating these databases rather than keeping a "virtual" or on-demand integration were discussed in Section 2, and cannot be minimized: improved system performance and reliability; the ability to correct errors; and the ability to annotate data directly. However, there are a number of problems: the difficulty in knowing how the primary data sources have changed; the difficulty in determining how these changes affect the secondary database; and the difficulty in knowing the origins (or provenance) of information in the secondary databases.

Unfortunately, relatively few developers of secondary databases that the authors are aware of are taking systematic steps to offset the difficulties associated with update and provenance. Since it is likely that secondary databases will continue to be developed, we believe that a number of practical steps should be taken by both the producers of primary data and the developers of secondary databases:

1. *Develop "keyed" XML data interchange formats.* The biomedical community has been looking for a universal data exchange format for several years, and many researchers view XML as the solution: there are an abundance of freely available parsers and other software for XML; there is heavy industry backing of XML; and several of the major relational database vendors (such as Oracle) are developing XML "exporters" and claim to be able to

store XML data, although at present these claims are frequently overstated—the problem of storing XML is still an open research issue. However, to make XML exchange an effective solution, the community must agree upon a vocabulary of terms and possibly some form of schema. In doing so, it would be good to ensure that there are keys at each level of nesting since it simplifies the description of updates. Keys must be chosen carefully (and perhaps can just be unique numbers assigned at each level) as a property of a key is that it cannot be modified.

2. *Publish minimal changes.* Intuitively, rather than just publishing "Entry 90158697 has been modified" it would also be useful to publish more information, such as "Feature X has been added to entry 90158697," where X is the value of the feature added. Given a keyed version of XML (or any form of hierarchical data), the changes to an entry can simply be represented as the set of paths that represent insertions, the set of paths that represent deletions, and the set of paths that represent modifications of values in the old entry. Since not everyone will want this detailed a level of information, users should probably continue to have a choice of obtaining the newest version of a database or of obtaining the entries that have been modified; however, in addition they should be able to obtain the exact modifications to an entry.

3. *Keep track of where the data came from.* Since data in secondary databases is derived from primary sources, it is important to keep track of *where* it came from and *why* it is there. At a minimum, this implies that detailed information should be maintained about the version of the primary data source from which the data was extracted, the date on which the information was extracted, and information about the query that extracted the data. For data that was obtained through extensive analysis packages based on data from primary sources (such as the "hypothetical genes" mentioned in the introduction), even more information should be maintained: What analysis was performed, the input parameters, name of person performing the analysis, etc.

Implicit in the problems encountered in creating and maintaining view databases is the lack of standard and co-operation in the underlying data sources. These kind of problems largely motivated the "standardization" approach based on the Common Object Request Broker Architecture (CORBA) proposed by the Object Management Group (OMG)[m] and described in detail in another chapter of this book[26]. The OMG is a consortium whose mission is to foster interoperability and portability for application integration via co-operative creation and promulgation of object-oriented standards. Underlying all OMG standards is the CORBA, which describes how a network of component systems shoud behave in order to interoperate in a distributed environment. The Life Sciences Research Task Force (LSR)[n] has been formed within the OMG to address requirements and specifications of software components for life sciences using CORBA. Currently, the LSR has reached consensus on specifications for biomolecular sequence analyses and genome maps, and it is now up to individual data source owners or third party to modify their data sources or to provide wrappers to their data sources so that they conform to these specifications. Increased interoperability and standards in biological softwares can be expected over the next few years as a result of the LSR.

Nevertheless, we think the standardization of all biomedical data sources is an unrealistic goal given their diversity, autonomy, rapid changes. Hence the approaches that we emphasized in this chapter, such as Kleisli and XML, do not require standardization of the underlying data sources. Instead, they use simple data models and provide high-level query languages to operate over the data models. The data models are sufficiently expressive and yet sufficiently simple so that any data source can be mapped with minimal effort by any one (who does not need to be the owner of the data source.) The query languages are sufficiently expressive and yet sufficiently high-level so that any typical data transformation query can be expressed with minimal effort by any one (who does not need to be an expert programmer.)

An additional problem that has not been mentioned in this chap-

[m] See `www.omg.org`.
[n] See `www.omg.org/homepages/lsr/mg.html`.

ter but which has implications for creating secondary databases is the ownership of the primary data. Over the past ten years, data which were originally in the public domain has, for a variety of reasons, had increasingly restrictive licenses placed on their use. While databases such as GenBank and MEDLINE are still in the public domain, other databases such as SWISS-PROT, which is itself a valued-added secondary database, are placing restrictions on the use of data. In the case of SWISS-PROT, the restrictions are intended as a funding mechanism aimed at commercial uses rather than at non-profit uses.[o] However, if a non-profit user integrates part of SWISS-PROT into a specialized database which can then be used by commercial users, the non-profit user must provide a list of commercial users so that the licensing can be checked.[p] While such restrictions are quite reasonable given the high cost of producing high-quality data, they may present a significant barrier to instantiating those portions of a secondary database which draws data from primary sources with restrictive licences.

Acknowledgements. This research was supported in part by DOE DE-FG02-94-ER-61923 Sub 1, NSF DBI99-75206, NSF IIS90-17444, ARO DAAG55-98-1-0031, and a grant from SmithKline Beecham.

1. S. Abiteboul. On views and XML. *Proc. ACM Symp. Principles of Database Systems*, 18:1–9, 1999.
2. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web—From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.
3. S. Abiteboul, J. Mc Hugh, M. Rys, V. Vassalos, and J. Wiener. Incremental maintenance for materialized views over semistructured data. *Proc. Intl. Conf. Very Large Data Bases*, 24:38–49, 1998.
4. P. Buneman, A. Deutsch, and W.C. Tan. A deterministic model for semi-structured data. *Proc. Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*,

[o] See `www.isb-sib.ch/announce`.

[p] The SWISS-PROT copyright notice also states for non-profit users: "There are no restrictions on its use by non-profit institutions as long as its content is in no way modified." Does this mean the data cannot be transformed?

Jerusalem, Israel, January 1999.

5. P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, 1994.
6. R. G. G. Cattell and et.al., editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, 1997.
7. S. Chawathe and H. Garcia. Meaningful Change Detection in Structured Data. *SIGMOD Record*, 26(2):26–37, 1997.
8. Y. Cui and J. Widom. Practical lineage tracing in data warehouses. *Proc. Intl. Conf. Data Engineering*, 16:367–378, 2000.
9. S.B. Davidson, C. Overton, and P. Buneman. Challenges in integrating biological data sources. *J. Computational Biology*, 2(4):557–572, 1995.
10. S. Davidson, C. Overton, V. Tannen, and L. Wong. Biokleisli: A digital library for biomedical researchers. *J. Digital Libraries*, 1(1):36–53, 1997.
11. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for XML. *Computer Networks*, 31:1155–1169, 1999.
12. T. Etzold and P. Argos. SRS: An indexing and retrieval tool for flat file data libraries. *Computer Applications of Biosciences*, 9:49–57, 1993.
13. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Proc. Intl. Workshop on Next Generation Information Technologies and Systems*, 2:185–193, 1995.
14. D. Gluche, T. Grust, C. Mainberger, and M. H. Scholl. Incremental updates for materialized OQL views. *Lecture Notes in Computer Science*, 1341:52–66, 1997.
15. A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin*, 18(2):3–18, 1995.
16. L. Haas et al. Starburst mid-flight: As the dust clears. *IEEE Trans. Knowledge and Data Engineering*, 2(1):143–160, 1990.
17. A. Kawaguchi, D.F. Lieuwen, I.S. Mumick, and K.A. Ross. Im-

plementing incremental view maintenance in nested data models. *Lecture Notes in Computer Science*, 1369:202–221, 1997.
18. H.A. Kuno and E.A. Rundensteiner. Incremental maintenance of materialized object-oriented views in multiview: Strategies and performance evaluation. *IEEE Trans. Knowledge and Data Engineering*, 10(5):768–792, 1998.
19. H. Liefke and S.B. Davidson. View maintenance for hierarchical semistructured data. *Proc. 2nd Intl. Conf. Data Warehousing and Knowledge Discovery*, September 2000, to appear.
20. P. A. Bernstein and T. Bergstraesser. Meta-data support for data transformations using Microsoft Repository. *IEEE Data Engineering Bulletin*, 22(1):9–14, 1999.
21. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogenous information sources. *Proc. IEEE Intl. Conf. Data Engineering*, 11:251–260, 1995.
22. D. Quass. Maintenance expressions for views with aggregation. *Proc. Workshop on Materialized Views: Techniques and Applications*, pages 110–118, June 1996.
23. J. Robie, J. Lapp, and D. Schach. XML query language (XQL). *Proc. W3C Query Languages Workshop*, Boston, December 1998. Available at `http://www.w3.org/TandS/QL/QL98/pp/xql.html`.
24. M. Rusinkiewicz, A. Sheth, and G. Karabatis. Specifying interdatabase dependencies in a multidatabase environment. *IEEE Computer*, 24(12):46–53, 1991.
25. S. M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.
26. M. Senger and A. Robinson. Standardization of Biological Data for Knowledge Discovery in Databases. Chapter in this book.
27. D. Suciu. Query decomposition and view maintenance for query languages for unstructured data. *Proc. Intl. Conf. Very Large Data Bases*, 22:227–238, 1996.
28. D. Suciu. Managing Web Data. *SIGMOD Record*, 28(2):510, 1999.
29. T. Lee and S. Bressan and S. Madnick. Source attribution for querying against semi-structured documents. *Proc. CIKM'98*

Workshop on Web Information and Data Management, November 1998.
30. K. C. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.
31. J. Thierry-Mieg and R. Durbin. ACeDB — A C. elegans Database: Syntactic definitions for the ACeDB data base manager, 1992.
32. J. D. Ullman. *Principles of Database and Knowledgebase Systems I*. Computer Science Press, 1989.
33. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.
34. A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. *Proc. Intl. Conf. Data Engineering*, 13:91–102, 1997.
35. L. Wong. Kleisli, a functional query system. *J. Functional Programming*, 10(1):19–56, 2000.
36. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.
37. K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42:133–139, 1992.
38. Y. Zhuge and H. Garcia-Molina. Graph structured views and their incremental maintenance. *Proc. Intl. Conf. Data Engineering*, 14:116–125, 1998.