Archiving and Maintaining Curated Databases

Heiko Müller

University of Edinburgh, UK

hmueller@inf.ed.ac.uk

Abstract

Curated databases represent a substantial amount of effort by a dedicated group of people to produce a definitive description of some subject area. The value of curated databases lies in the quality of the data that has been manually collected, corrected, and annotated by human curators. Many curated databases are continuously modified and new releases being published on the Web. Given that curated databases act as publications, archiving them becomes a necessity to enable retrieval of particular database versions. A system trying to archive evolving databases on the Web faces several challenges. First and foremost, the systems needs to be able to efficiently maintain and query multiple snapshots of ever growing databases. Second, the system needs to be flexible enough to account for changes to the database structure and to handle data of varying quality. Third, the system needs to be robust and invulnerable to local failure to allow reliable long-term preservation of archived information. Our archive management system XArch addresses the first challenge by providing the functionality to maintain, populate, and query archives of database snapshots in hierarchical format. This presentation intends to give an overview of our ongoing efforts of improving XArch regarding (i) archiving evolving databases, (ii) supporting distributed archives, and (iii) using our archives and XArch as the basis of a system to create, maintain, and publish curated databases.

1 Introduction

Curated databases are databases that are populated and updated with a great deal of human effort [5]. The content of curated databases is usually collected through (i) consultation, verification, aggregation, and annotation of existing sources, or through (ii) processing, and interpretation of new raw data. A typical example of a curated database is the *Universal Protein Resource* (*UniProt*), the central repository for information of protein sequence and functional annotation [12]. The *UniProt consortium* has a large team of curators that extract biological information from the literature and perform numerous computational analyses to provide all relevant information about a particular protein. Another example of a curated database is the *CIA World Factbook* [1], a comprehensive resource of demographic data. The first unclassified Factbook was published in 1971. Since 1981 the Factbook has been published annually as a book. Recently, the *CIA* decided to exclusively publish the Factbook on the Web. Just as the Factbook, many curated databases act as publications on the Web that are currently replacing the traditional reference works found in libraries. For many database providers it has become common practice to overwrite existing database states when changes occur and publish new releases of the data on the Web. For example, the *CIA* publishes a new version of the Factbook approximately every two weeks. Thus, archiving becomes a necessity to (i) enable retrieval of particular cited database versions, and (ii) to support temporal queries to allow analysis of how certain objects changed over time. Such queries are especially interesting for the *CIA World Factbook*. For example, one might want to find out how the population of European countries changed over the past 30 years and how these changes correlate with energy consumption, or *Gross Domestic Product* change.

Curated databases are predominantly kept in well-organised hierarchical data formats having a key structure that provides a canonical identification for each element by the path in which it occurs and the values of some of its sub-elements. In [6] a nested merge approach to archiving is developed that efficiently stores multiple versions of hierarchical data in a compact archive by "pushing down" time and introducing timestamps as an extra attribute of the data. Archives of multiple database versions are generated by merging the versions into a single hierarchical data structure. Corresponding elements in different versions are identified based on their key values. The archiver stores each element only once in the merged hierarchy to reduce storage overhead. Archived elements are annotated with timestamps representing the sequence of version numbers in which the element appears. This approach has several advantages regarding storage space, retrieval of database versions, and tracking of object history. Based on the algorithms described in [6], we implemented the archive management system XArch [10]. The system allows one to create new archives, to merge new versions of data into existing archives, and execute both snapshot and temporal queries using a declarative query language.

Outline. Our initial approach to archiving makes the critical assumption that the structure of an archived database remains unchanged. This restriction, however, is almost certainly to become an issue when archiving a database over a long period of time. The first part of this presentation discusses our ongoing efforts of archiving and querying evolving databases. Creating archives in a stand-alone manner, however, appears infeasible and undesirable for long-term preservation. The natural solution to enhance reliability is to follow the model of traditional libraries to share the burden of archiving by employing a set of independent, distributed archivers. The second part of this presentation proposes a low-cost solution to distributed archiving. We are currently developing algorithms to synchronize and query distributed archives without requiring any central authority. The presented techniques cover some of the challenges for curated databases discussed in [5]. The last part of this presentation outlines how our archives and XArch may form the basis of a system to create, maintain, and publish curated databases.

2 Archiving Evolving Databases

We start with a brief description of our data model. Archives and database states (called *database snapshots*) are modeled as trees. We consider two types of nodes: (i) *element nodes* having a label and a key value, and (ii) *text nodes* having a value. Only element nodes may occur as internal nodes. Nodes in an archive additionally carry a timestamp that represents the sequence of version numbers in which they appear. Element keys are defined using key constraints. A *key specification* K is a set of *key definitions* $k = (q, s)$, where q is an absolute path of element labels and s is a *key value expression*. Each key definition (q, s), specifies a set of elements reachable by path q and defines how the key value is derived from the elements subtree. We distinguish between three types of key value expressions: (i) *existence*, (ii) *subtree*, and (iii) *values*. All element keys are relative keys, *i.e.,* the key value uniquely identifies an element among its siblings. Elements that are keyed by existence are keyed by their label. Elements that are keyed by subtree are uniquely identified by the value of their whole subtree (see [6] for definitions of subtree values). For elements e that are keyed by values an additional set of relative path expressions $\{p_1, \ldots, p_k\}$ is given. Each p_i specifies an element in the subtree of e whose value is used as part of the key value for e. These values are referred to as *key path values*. In XArch we currently use *XML* as the storage format of archives. Element nodes and timestamps are represented as *XML* elements. Text nodes are *XML* strings. *XML*, however, is not the only possible storage format and we are currently considering other formats like relational databases.

Schema Evolution. When archiving databases over a long period of time, schema changes almost certainly become an issue. Here, we distinguish between changes to (i) key path values, and (ii) to the key specification. If key path values are modified between different database

snapshots the archiver treats the corresponding elements as distinct elements and does not merge them. These changes are particularly common in the *CIA World Factbook* and they limit our ability of tracking an objects history. We are currently developing methods to detect such key path value changes based on complementary timestamps and similarity of element subtrees. Once we are able to detect these changes, we can merge the corresponding elements in a post-processing step and annotate their key values with appropriate timestamps. Thus, we need to extend our data model to allow timestamped key values for element nodes. Problems arise, however, when elements are being moved around between different parents. In order to handle these cases, we would have to extend our data model and consider archives as graphs instead of trees making it even more questionable whether *XML* is the appropriate storage format for our archives.

Changes to the key specification are currently handled as follows: A key specification itself is a tree whose nodes are elements that have a label, a type, and (for value keys) a set of key path expressions. Such a tree is easily transformed into a hierarchical dataset as described above. We are thereby enabled to maintain different key specifications within a separate archive. We recently extended XArch to handle database snapshots with different key specifications within a single archive. Instead of defining a single key specification for all database snapshots, each snapshot may have its own key specification. Before merging a snapshots into an archive we first merge its key specification into the *key specification archive*. When annotating key values for elements in the archive we find the appropriate key definition based on an elements timestamp. This approach models changes to the key specification as insertion and deletion of key definitions. In order to allow a more meaningful and compact description and representation of object histories additional operations are required. For example, when representing the renaming of an element by a delete and insert operation the problem again is to identify elements that are keyed by different key definitions but represent the same real-world object. In [9] a set of *schema modification operators* (*SMOs*) is defined that are capable of describing evolution of relational schemas. A *SMO* takes a schema version as input and produces a new schema version. Apart from creating and deleting tables and columns, *SMOs* allow for example to rename tables and columns, merge tables, and copy or move columns. These operations can easily be adopted to describe key specification modifications. In [7] a functional update language for *XML* (*FLUX*) is presented that also contains structure modifying operators like renaming and nesting of elements. We are looking into *SMOs* and *FLUX* to derive a set *key specification modification operations* capable to describe evolution key specifications. Similar to the approach in [9] we intend to use the information about structure evolution to rewrite queries that are written against the current key specification to retrieve data from snapshots structured under previous key specification.

3 Synchronizing Distributed Archives

Creating archives for long-term preservation in a stand-alone manner appears infeasible and undesirable for several reasons. First of all, having a stand-alone archive creates a single point of failure, making the preservation effort depending on usually unreliable hardware and storage media. Furthermore, the number of online available curated databases is large and only expected to increase in future. The exponential growth of many curated databases is well documented. Thus, maintaining archives for each of these databases requires high-performance systems with high network bandwidth and huge amounts of storage space. The natural solution to enable reliable and low-cost preservation is to share the burden of archiving. A viable approach is to employ a large network of independent, low-cost computers that cooperate to populate and query archives of online databases. Such an approach has for example being taken by *LOCKSS* (*Lots of Copies Keep Stuff Safe*), a system that aims at preserving access to journals and other archival information published on the Web [8]. The main assumption of *LOCKSS*, however, is that archived information like journal articles do not change.

We envision a scenario similar to *LOCKSS* of independent archivers that continuously archive snapshots of curated databases. The problem we are focusing on in our current research is to reconstruct the database history from a given set of individual archives that are maintained without central control. Our goal is to reconstruct the database history by defining a chronological order for all database snapshots in the archives. This problem is complicated for two reasons: First, data sources on the Web rarely provide unique version identifiers with each of their published snapshots. Second, the distributed scenario does not allow us to assume global unique timestamps for database snapshots in different archivers. While internally each archiver may maintain unique timestamps for each of its snapshots, timestamps provided by individual archivers may not necessarily be comparable. Common examples are unsynchronized clocks or incomparable timestamp formats. To define a chronological order in the absence of timestamps, we have to rely on the order of snapshots in individual archives and on the data itself.

The problem of estimating a chronological order for a given set of databases has been referred to as *seriation* [11]. Existing approaches for database seriation focus on small databases of Boolean data values. We are currently developing algorithms to allow seriation of large hierarchical datasets like those in our archives of curated databases. The main idea of our approach is to use a distance measure for pairs of snapshots that reflects their edit distance using insert and delete operations. Given a set of independent archives for the same database, we simply merge these archives into a single archive. The order in which we merge the archives is irrelevant. The edit distance for pairs of snapshots can then be computed efficiently by evaluating the timestamps in the resulting archive. We then compute a total order on all snapshots that (i) adheres to the partial order of snapshots defined by the individual archives, and (ii) minimizes the sum of edit distances of adjacent snapshots in the total order. The problem of seriation for a given set of partially ordered databases is shown to be of high complexity. We develop algorithms and heuristics that solve the problem efficiently and with high accuracy.

4 Maintaining Curated Databases

Maintaining curated databases presents a number of challenges for database research [5]. Archiving and evolution of structure are two of them that are already tackled by XArch. Others topics are propagation of annotations, citation, and provenance.

Citation. Initial proposals on how to make curated databases citable are given in [3]. Archives based on a keyed hierarchical data model satisfy some of the desiderata. First, each element within a hierarchical dataset is uniquely identified by its path and the key values of the elements in the path. Second, the archive records all database versions. Thus, the combination of element path, element keys, and timestamps allows to uniquely identify any element in a particular version of the database and to retrieve the data from the archive. We are currently working with the *IUPHAR receptor database* [2] on methods to publish current and past versions of the database on the Web and allow people to cite and retrieve any particular version of the data. One of the open research questions here is to develop generic and easy-to-use methods for generating appropriate Web pages from an existing archive.

Updates and Provenance. The majority of data in curated databases is derived or copied from other sources. Since the value of curated databases lies in the quality of the data that has been manually collected, corrected, and annotated by human curators, knowing the origin of curated data - its *provenance* - is particular important. In [4] general-purpose techniques for modeling and recording provenance for data that is copied among databases is presented. The techniques are based on a keyed hierarchical data model and a basic update language that allows to insert, and delete nodes, and copy nodes within or between curated databases. User actions while modifying a database are recorded in a convenient form that allows to ask questions such as when some data was first created, by what process did a value arrive in a database, or when

was a node or its subtree last modified. A insert and delete operator have recently been included in XArch, the copy operator remains future work. Each operator modifies the last snapshot in an archive and adds a new database snapshot to the archive. The provenance, *i.e.,* the operator that created a snapshot, is maintained in an appropriate way as described in [4]. Given the presented extensions we believe that XArch forms a viable tool to support the creation, manipulation, and evolution of curated databases.

Acknowledgments

The presented work is based on collaborations with Peter Buneman, James Cheney, and Floris Geerts from the *University of Edinburgh*.

References

[1] https://www.cia.gov/library/publications/the-world-factbook/index.html.

[2] http://www.iuphar-db.org/.

[3] P. Buneman. How to cite curated databases and how to make them citable. In *SSDBM '06: Proceedings of the 18th International Conference on Scientific and Statistical Database Management*, pages 195–203, Washington, DC, USA, 2006. IEEE Computer Society.

[4] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 539–550, New York, NY, USA, 2006. ACM.

[5] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren. Curated databases. In *PODS '08: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–12, New York, NY, USA, 2008. ACM.

[6] P. Buneman, S. Khanna, K. Tajima, and W.-C. Tan. Archiving scientific data. *ACM Trans. Database Syst.*, 29(1):2–42, 2004.

[7] J. Cheney. Flux: functional updates for xml. In *ICFP '08: Proceeding of the 13th ACM SIGPLAN international conference on Functional programming*, pages 3–14, New York, NY, USA, 2008. ACM.

[8] 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.

[9] H. J. Moon, C. A. Curino, A. Deutsch, C.-Y. Hou, and C. Zaniolo. Managing and querying transaction-time databases under schema evolution. *Proc. VLDB Endow.*, 1(1):882–895, 2008.

[10] H. Müller, P. Buneman, and I. Koltsidas. Xarch: archiving scientific and reference data. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1295–1298, New York, NY, USA, 2008. ACM.

[11] A. Ukkonen, M. Fortelius, and H. Mannila. Finding partial orders from unordered 0-1 data. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 285–293, New York, NY, USA, 2005. ACM.

[12] Uniprot-Consortium. The universal protein resource (uniprot) 2009. *Nucleic acids research*, October 2008.