The Database Wiki Project: A General-Purpose Platform for Data Curation and Collaboration

Peter Buneman, James Cheney,
Sam Lindley
School of Informatics
University of Edinburgh
Edinburgh, United Kingdom
{opb, jcheney}@inf.ed.ac.uk,
Sam.Lindley@ed.ac.uk

Heiko Mueller
Tasmanian ICT Centre
CSIRO
Hobart, Australia
heiko.mueller@csiro.au

ABSTRACT

Databases and wikis have complementary strengths and weaknesses for use in collaborative data management and data curation. Relational databases, for example, offer advantages such as scalability, query optimization and concurrency control, but are not easy to use and lack other features needed for collaboration. Wikis have proved enormously successful as a means to collaborate because they are easy to use, encourage sharing, and provide built-in support for archiving, history-tracking and annotation. However, wikis lack support for structured data, efficiently querying data at scale, and localized provenance and annotation. To achieve the best of both worlds, we are developing a general-purpose platform for collaborative data management, called DBWiki. Our system not only facilitates the collaborative creation of structured data; it also provides features not usually provided by database technology such as annotation, citability, versioning, and provenance tracking. This paper describes the technical details behind DBWiki that make it easy to create, correct, discuss, and query structured data, placing more power in the hands of users while managing tedious details of data curation automatically.

1. INTRODUCTION

Curated databases are finding use in all branches of science and scholarship. Most curated databases are created and maintained in a collaborative effort by a dedicated group of people – the curators – who produce a definitive reference work for some subject area. Common examples include UniProt, a resource of protein sequence and functional information [8], and IUPHAR-DB, the official database of the IUPHAR Committee on Receptor Nomenclature and Drug Classification [7], which contains contributions of a large community of experts in the field. Some curated databases are also being developed in support of "citizen science", where the public at large can contribute to the database (see [3] for examples). A system that maintains curated databases faces several technical and usability challenges [20, 13]:

1. Past versions of data need to be archived and easy to retrieve. The archiving system should also support temporal queries over the history of data.
2. Much curated data is copied and edited from existing sources. Since the value of curated databases lies in their quality and organization, knowing the origin of the curated data — its provenance — is particularly important.
3. In addition to the actual data, curated databases carry additional valuable annotations consisting of opinions of curators about the quality of data or suggested changes.
4. Curators should receive credit for their contributions. Thus, the system needs to make data items citable and attributable to their contributors.
5. Curated databases are collections of entries that predominantly follow a common structure. This common structure (or database schema) may need to change over time as the subject area evolves.
6. Many data curation projects rely on their web presence to distinguish themselves from other projects; in fact, journals such as *Nucleic Acids Research* require databases to maintain Web interfaces in order to be considered for publication.

It can be seen from these requirements that we are asking for a mixture of functionalities provided by databases and wikis. The ability to handle structured data is to some extent already present in wikis through the use of infoboxes, but they fall far short of the functionality of a database; in fact they are not even intended as the primary representation of the data they contain [6]. Relational databases, on the other hand, provide little in the way of generic support for these features. However, while it is always possible to add them for individual applications, the ability to do this largely remains the preserve of professional programmers and database administrators and requires substantial coding effort. The expense of doing this is unrealistic for many small projects that lack the resources to employ this expertise.

We believe that the needs of database curation projects could be met more reliably and cost-effectively by developing new general-purpose systems that combine the advantages of databases and wikis. We call such systems *Database Wikis*. Much of the basic research on curated databases needed to implement database wikis, such as archiving, citation, provenance, and annotation management, has already been conducted [10, 11, 13, 21]. However, there as yet is no single system that draws these techniques together.

1.1 Contributions

We are developing DBWIKI, a Database Wiki that aims to combine the ease of use and flexibility of a wiki with the robustness and scalability of a database; furthermore, DBWIKI provides unified generic techniques for database curation that have previously been prototyped in separate systems.

DBWIKI provides the ability to create, populate and browse curated databases using a standard web browser. Data entry and modification is done either using system-generated web forms or by import from other data sources such as XML files. Each piece of information has a provenance record. All changes to the data and the database schema are logged in the database. The provenance and prior versions of each individual data item are browsable through the user interface. Moreover, each piece of information within the database can be annotated (including annotations themselves). Annotations are free-form text that allow curators to share and discuss their opinions, much like comments on blogs or forums; however, annotations can be attached to any part of the data, not just pages. To demonstrate the full capabilities of DBWIKI we have used it on data from several existing curated databases, including the CIA World Factbook [2], DBLP [4], and IUPHAR-DB [7].

In addition to the database capabilities, DBWIKI includes a declarative "markdown" language for defining wiki pages. We extended a markdown syntax parser with syntax for embedded queries for viewing data. This extension enables querying and aggregating information from the curated database, *e.g., list the population of countries in Europe*. Similar proposals have been made for embedding SPARQL queries in other wiki extension projects such as Semantic MediaWiki [1]. DBWIKI's query capabilities, however, go beyond querying the current state of the data, as we also allow embedded queries over the history and the provenance of data, *e.g., list all changes made to the population of Greece*. Such queries are not currently supported by other wiki extension projects.

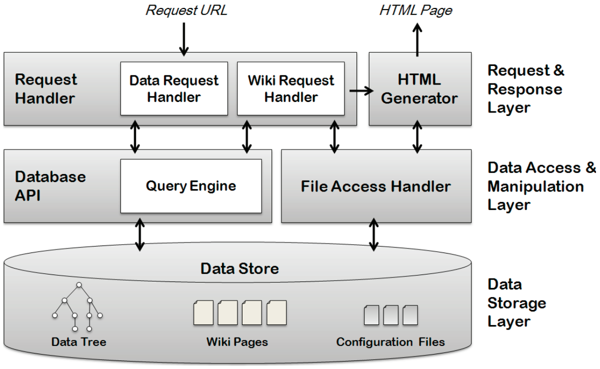


Figure 1: DBWIKI **System Architecture**

2. SYSTEM OVERVIEW

We implemented DBWIKI as a stand-alone Java application [12]. The architecture, divided in three layers, is shown in Figure 1. The bottom layer is responsible for storing the data and any additional information. The middle layer is responsible for querying and manipulating the data. The top layer of the architecture handles incoming HTTP requests and generates HTML pages in response. In parallel to the Java-based prototype presented here, we used a high-level Web programming language called Links [17] to develop another prototype, where we experimented with wiki-embedded query language design. Our experiences with the Links prototype are presented in a companion paper [16]; the lessons have been incorporated into the Java system.

2.1 Data Model and Storage

DBWIKI extends the XML archive management system XARCH [21]. XARCH is an archiving system based on a simplified XML data and schema model, as illustrated in Figure 2. The schema describes the set of possible paths in the data tree and distinguishes internal ($) from text (@) nodes. In XARCH, the edges of the data tree are annotated with time intervals indicating the range of times (or version numbers) during which a given subtree was present in the database. Timestamps are similar to those proposed in [14] and implemented in XARCH. For example, a timestamp $[1-5, 10-12]$ indicates that the associated node was present in database versions $1-5$ and $10-12$. A special value "now" indicates an open interval. XARCH supports an efficient sorting-based *merge* operation that identifies the differences between the current version of the database and a new version. XARCH has been used to archive real examples such as the CIA World Factbook [2] and IUPHAR-DB [7].

DBWIKI currently supports a common set of data and schema modification operations including insert, delete, and update. It is also easy to copy-and-paste

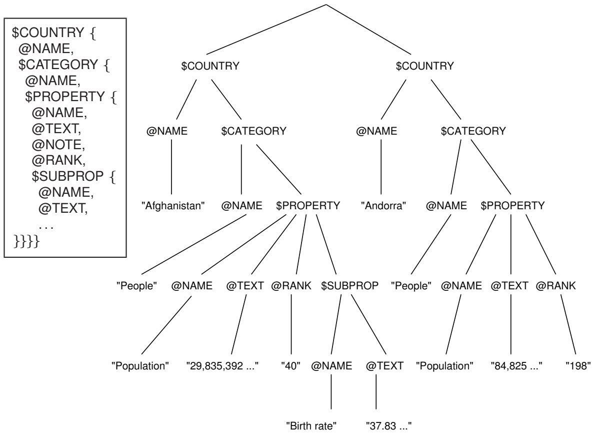


Figure 2: A schema and data tree

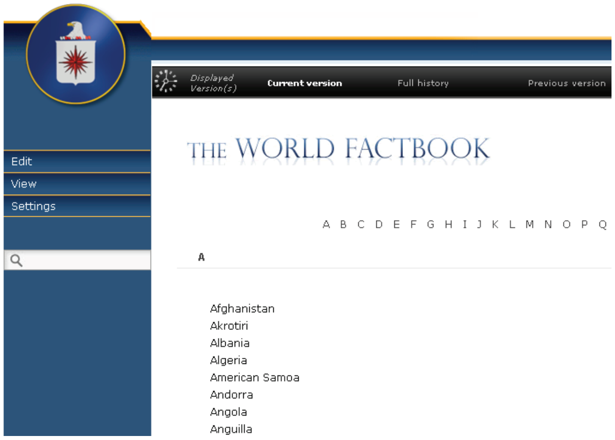


Figure 3: CIA World Factbook web interface in DBWiki with time machine and edit menu

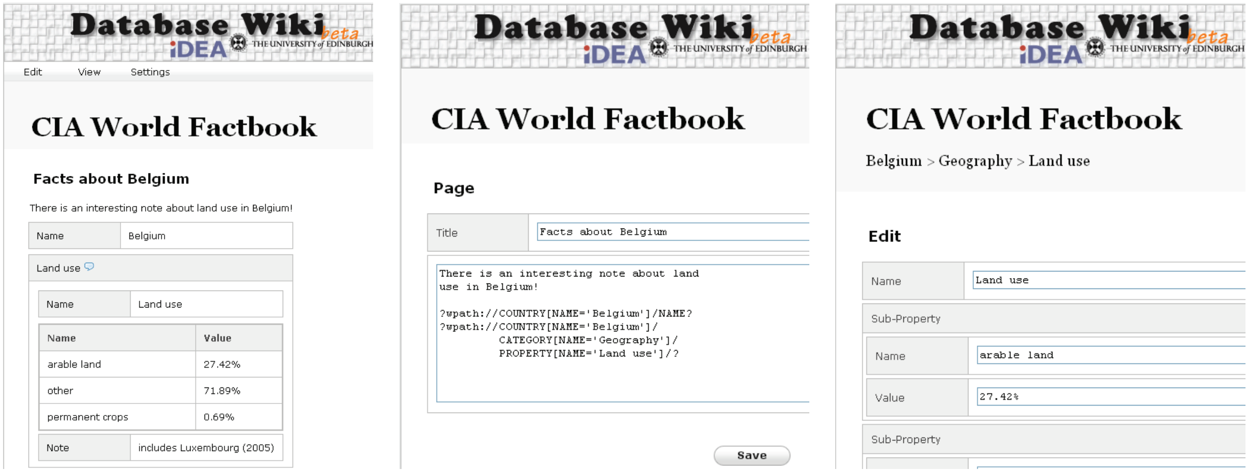
nodes and subtrees within or among different curated databases. Each operation creates a new version of the database (efficiently using the archiving approach from XArch). With each node we associate a timestamp that describes those database versions in which the node was present, using the same interval annotation approach as in XArch. Based on the timestamp and information about the action that created each database version we derive provenance information for data nodes following the provenance model defined in [11]. Each node may also be associated with a list of annotations. Annotations are timestamped but not versioned, and creating an annotation does not create a new version of the data.

We use a relational database back-end to store the data tree, annotations, and version information, *i.e.,* we shred the data tree, schema, and other metadata into relations. The data is stored relationally using the edge relation. Each tree is given its own entry number which identifies all the rows corresponding to that tree. Moreover, each row is associated with a timestamp indicating the versions when that edge was present in the tree. The main difference between the storage model of DBWiki compared to XArch is that instead of using XML files with special string-valued attributes to store timestamps, we store both the tree data and the temporal information in a relational database. This is useful for supporting annotation and provenance-tracking since we do not have to serialize this extra data into a textual XML form. DBWiki supports different RDBMS back-ends using the Java JDBC interface. The relational database also stores the wiki page markup sources and configuration files used for web page layout (see below). Each of these files is also versioned.

2.2 User interface

Users interact with DBWiki through a web browser, making requests encoded using URLs for either browsing or modifying the data, or for viewing or editing wiki pages. The URLs for wiki pages are similar to those in Wikipedia, *i.e.,* the page title is used as the page identifier. When browsing the data tree, we allow URLs similar to the query formats (see Section 3.1 for details). Once requested data has been retrieved, it is passed to the HTML generator that generates the response page. One of the design criteria of DBWiki was to keep HTML generation separate from the rest of the system, hence highly customizable, as in typical wikis or content management systems. HTML generation is guided by three configuration files.

The first file is a HTML template. The template specifies the basic HTML to be used for all web pages of a data collection. Besides standard HTML tags, the template contains placeholders for predefined user-interface components. These components take the data node to be displayed and database version information as parameters and generate standard HTML snippets. One such component is the “time-machine”. The time-machine provides links to different views of the data, *i.e.,* the current version, each previous version, highlights of changes since a given previous version, and the full history of the data. Other examples for predefined user-interface components are HTML and JavaScript code for displaying and editing annotations and provenance information. The second configuration file is a layout definition that specifies how to map the tree-structured data to HTML pages, tables or lists. The simplicity of the schema language makes it easy to specify different layouts for the data. The layout system allows configuring which attribute names are used in paths, whether groups are rendered as tables or lists, and how subtrees are grouped, *i.e.,* all on one page or split into several pages. The third configuration file is a cascading style-sheet (CSS) file used to format the HTML output produced by the tem-



CIA World Factbook

Facts about Belgium

There is an interesting note about land use in Belgium!

Name	Belgium

Land use

Name	Land use

Name	Value
arable land	27.42%
other	71.89%
permanent crops	0.69%

Note	includes Luxembourg (2005)

CIA World Factbook

Page

Title: Facts about Belgium

```
There is an interesting note about land
use in Belgium!

?wpath://COUNTRY[NAME='Belgium']/NAME?
?wpath://COUNTRY[NAME='Belgium']/
        CATEGORY[NAME='Geography']/
        PROPERTY[NAME='Land use']/?
```

CIA World Factbook

Belgium > Geography > Land use

Edit

Name: Land use

Sub-Property

Name: arable land

Value: 27.42%

Sub-Property

Figure 4: Wiki pages (from left to right) embedded query result, page source, and data update form.

plate and layout engine.

The wiki interface also allows editing all configuration files on-the-fly. Together, these configuration files give the user a great amount of flexibility in customizing the look-and-feel of the web pages. An example is shown in Figure 3 which shows a DBWiki version of the CIA World Factbook using some of the features of the original Web site. Figure 4 shows another three pages from the Factbook wiki using the default layout. However, while DBWiki supports automatic browsing and editing of Factbook data, it cannot at present exactly duplicate all aspects of the existing interface. There is a trade-off between simplicity and expressiveness of the template and layout language, and we are interested in exploring richer stylesheet languages.

2.3 Database Queries

With DBWiki one can query the data tree and embed the results in wiki pages. Thus, DBWiki's wiki pages are dynamic, combining hypertext with views of the structured data. Wiki page queries are translated to SQL queries against the relational data store. We currently support three different query formats. The first format uses the node identifier to retrieve a node (and its subtree) from the database.

The second query format is a special form of path expression, *i.e.,* sequences of node labels with optional constraints. Path expressions allow positional references as well as constraints on values of a node's children. For example, the query /COUNTRY:2 returns the second country in the CIA World Factbook [2]. Note that the order of nodes is defined by the order of their node identifiers which in turn reflects the order in which the nodes have been inserted into the database. The following query returns the population for Chile in our version of the Factbook.

```
/COUNTRY[NAME='Chile']/
  CATEGORY[NAME='People']/
    PROPERTY[NAME='Population']
```

The third query format (currently in development) is an adaptation of the XAQL query language that was implemented with XArch. This format allows to select multiple sub-trees from a node as well as posing constraints on the timestamps and provenance information of nodes. For example, the following query returns the name and GDP of all countries that were modified by user admin since 2010:

```
SELECT $c/NAME, $p/TEXT FROM $c IN /COUNTRY,
$p IN $c/CATEGORY[NAME='Economy']/PROPERTY[NAME='GDP']
WHERE $c WAS MODIFIED SINCE 2010-01-01 BY admin
```

The path expressions in our queries correspond to a simple fragment of XPath, and we can use a wide variety of known techniques to evaluate them efficiently. We currently use a simple two-step approach: Based on the constraints in the path expression we first generate a SQL query that retrieves all candidate entries that potentially satisfy the constraints. We then load each candidate entry into memory and evaluate the path expression on it. For XAQL queries there is a third step to filter the results using the path expressions in the SELECT-clause.

3. TECHNICAL HIGHLIGHTS

In this section we give further details of the various components of the system. In this short paper we cannot give full details of algorithms but we refer when possible to similar techniques in the literature.

3.1 URLs

DBWiki provides unique URLs for each node in the data tree. These URLs are essential for displaying and editing individual nodes in separate web pages, and for node annotation. The URLs and the provenance information that we store, furthermore, provide a mechanism that can be used to cite (parts of) database entries and give credit to users for their contributions.

URLs in DBWiki are based on node identifiers and they take the form *http://server/database/node-id*. While this URL scheme fulfills the need for persistent and unique identifiers, it is not very intuitive for human users, nor is it robust if data is exported and re-imported. In addition, DBWiki is able to decode URLs that reflect the current path under which a node is located based on predefined node identification rules. For each group node in the database schema one can define a descendant attribute node whose value is to be used as node identifier. These identifiers are similar to keys for XML as defined in [15]. An example specification for the CIA World Factbook is:

```
IDENTIFY /COUNTRY BY NAME,
IDENTIFY /COUNTRY/CATEGORY BY NAME,
IDENTIFY /COUNTRY/CATEGORY/PROPERTY BY NAME
```

Based on this specification the following URL refers to the the population of Chile in our Factbook collection: *http://server/CIAWFB/Chile/People/Population*. To avoid ambiguities each path component can be prefixed with the element label, *e.g., /COUNTRY:Chile*. These URLs are not stable as the data and schema are likely to change over time; we plan to extend them with time information to make them more useful as stable citations [10].

3.2 Update operations

Updating the data. DBWiki supports atomic insertion and deletion of subtrees and editing of attribute nodes. These operations were not previously supported efficiently in XArch, which focuses on merging whole database versions instead. Each update operation is currently implemented by first loading the entry to which the update applies (including all past version information) into memory. We then perform consistency checks and translate the update operation into a set of INSERT and UPDATE statements to the relational database. Details for each type of operation are given below. We further record the system time of the update and the operation itself in the version table. We refer to the new database version by t in the following. In many cases it would be more efficient to translate tree updates to SQL queries instead of loading and storing the whole entry. We regard this as an opportunity for future work.

An insertion is handled in several steps. First, we ensure that the parent node is alive and check whether the subtree being inserted matches the schema. If not, depending on the database's policy we either reject the update, ignore non-matching data, or extend the schema. Next, we insert the subtree as a child of the parent. Each node in the subtree results in an additional tuple in the relational database. Only for the root of the inserted subtree we also explicitly store a timestamp $[t - now]$. Similar to XArch, all nodes that do not have an explicit timestamp inherit the timestamp of their parent.

A deletion of a node in a given entry is handled by adjusting the timestamps of the deleted node and any live descendants that have explicit timestamps. Depending on whether the deleted node has an explicit timestamp we either replace "now" with $t - 1$ in that timestamp or insert a new timestamp for the node that ends at $t - 1$. We then replace "now" with $t - 1$ for all explicit timestamps of live descendants. Note that we do not delete any tuples from the relational database to ensure that all past versions of the data are available.

Each attribute can have multiple text nodes as children. We refer to these nodes as attribute values. A live attribute can only have one live value. When editing an attribute a we first terminate the current value of a. Let v denote the new attribute value. If a has had value v sometime in the past, then we extend the timestamp of the corresponding text node with $[t - now]$. If not, then we create a new child of a with timestamp $[t - now]$. We avoid performing (and recording) spurious updates that do not actually change the value.

Copy and paste. DBWiki supports a simple form of copy and paste for data subtrees, following the design given in [11]. Copy and paste is implemented as an insert with the data to be inserted being retrieved using a DBWiki URL. That is, one first selects the target node for the paste operation and then specifies the URL that points to the root of the subtree that is to be copied. This URL can either point to the current or a previous version of the subtree; in either case the value of the subtree at a single time instant is copied and pasted as a child of the target node. DBWiki sends a copy of the subtree that is to be copied in XML format in response to a parameter "?cpxml" on the URL. The source URL of a copy and paste operation is recorded in the version table. Using URLs enables copy and paste between different servers.

Schema updates. DBWiki currently supports only insertion and deletion of schema elements. The schema is versioned, so that past versions of the data can be understood. Insertion happens automatically when inserting data that require schema extensions. Deletion is implemented by deleting from the schema in the same way as for data deletion, and then deleting the corresponding data subtrees. We are currently investigating more efficient implementations of insertion and deletion as well as considering additional operations such as renaming.

3.3 Provenance and annotation

The initial provenance record for any database is the record of when it was imported, by whom, and (optionally) a URL pointing to the source. Of course, we have no way of ensuring that the source URL is stable; it is just for documentation purposes. We also store one provenance record per insert, delete, copy-paste or update operation. Some of this information can be inferred from the change history. We also support annotations (as textual comments) to any part of the database.

4. EXTENSIONS

The choice of a very simple data model already pays some dividends, and suggests that we can select among a wide variety of techniques for querying and manipulating XML or tree-structured data, as well as for publishing relational data as XML. It should also be straightforward to export DBWIKI data as RDF/Linked Data, an approach to data exchange that is becoming popular in scientific data communities. Other techniques for XML such as security views [18] should also be easy to incorporate into DBWIKI.

A more sophisticated approach to path querying based on Grust et al.'s XPath Accelerator [19] has been implemented in a student project. However, the interaction between XML indexing and temporal issues remains largely unexplored. Another student has added facilities to query data and plot query results using charts, graphs, or maps, much as Google Fusion Tables allows plotting table data using Google APIs. For DBWIKI, this problem is more difficult because the data is nested. A third student has developed ways to visualize and query the provenance information, such as showing the number of edits or annotations per user over a given period as a chart or graph.

5. CONCLUSION

This paper presents DBWIKI, a system that combines the insights of wiki-style user interfaces with the capabilities to structure and query data efficiently characteristic of database management systems. The system is still under development, but we believe it represents solid progress towards the goal of making curated database technology available to those who need it the most: namely, scientific database curators and consumers of scientific data, where provenance, annotation, citation, and versioning are key requirements that currently need to be revisited for each new database. We also hope that DBWIKI will help us evaluate the effectiveness of research so far on these topics, and identify new research directions or unmet needs. The DBWIKI source code has been made publicly available under open-source terms [9], and we invite other researchers or projects to experiment with it, extend it or critique it.

Acknowledgments. This work has been supported by an EPSRC platform grant, by Google and by the University of Edinburgh IDEA Lab. Hui Li, Haoli Qu and Snehal Waychal have contributed code as part of their MSc projects. We are grateful to Tony Harmar and his colleagues for giving us access to the IUPHAR [7] source data.

6. REFERENCES

[1] http://semantic-mediawiki.org.

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

[3] http://www.citizensciencealliance.org/.

[4] http://www.informatik.uni-trier.de/∼ley/db.

[5] http://www.geneontology.org.

[6] http://en.wikipedia.org/wiki/Help:Infobox

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

[8] http://www.uniprot.org.

[9] http://code.google.com/p/database-wiki/

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

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

[12] P. Buneman, J. Cheney, S. Lindley and H. Müller DBWiki: a structured wiki for curated data and collaborative data management In *SIGMOD*, demo, pages 1335–1338. ACM, 2011.

[13] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren. Curated databases. In *PODS*, pages 1–12. ACM, 2008.

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

[15] P. Buneman, S. Davidson, W. Fan, C. Hara, and W.-C. Tan. Keys for XML. In *WWW*, pages 201–210, 2001.

[16] J. Cheney, S. Lindley, and H. Müller. Using Links to prototype a Database Wiki. In *DBPL*, 2011.

[17] E. Cooper, S. Lindley, P. Wadler, and J. Yallop. Links: web programming without tiers. In *FMCO*, pages 266–296. Springer-Verlag, 2007.

[18] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure XML querying with security views. In *SIGMOD*, pages 587–598, 2004.

[19] T. Grust, M. Van Keulen, and J. Teubner. Accelerating XPath evaluation in any RDBMS. *ACM Trans. Database Syst.* 29(1):91–131, 2004.

[20] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. in *SIGMOD*, pages 13-24. ACM, 2007.

[21] H. Müller, P. Buneman, and I. Koltsidas. XArch: archiving scientific and reference data. In *SIGMOD*, pages 1295–1298. ACM, 2008.