PRIMA: Archiving and Querying Historical Data with Evolving Schemas

Hyun J. Moon, Carlo A. Curino†, Myungwon Ham, Carlo Zaniolo
UCLA, †Politecnico di Milano
{hjmoon, ham, zaniolo}@cs.ucla.edu, †carlo.curino@polimi.it

ABSTRACT

Schema evolution poses serious challenges in historical data management. Traditionally, historical data have been archived either by (i) migrating them into the current schema version that is well-understood by users but compromising archival quality, or (ii) by maintaining them under the original schema version in which the data was originally created, leading to perfect archival quality, but forcing users to formulate queries against complex histories of evolving schemas. In the PRIMA system, we achieve the best of both approaches, by (i) archiving historical data under the schema version under which they were originally created, and (ii) letting users express temporal queries using the current schema version. Thus, in PRIMA, the system rewrites the queries to the (potentially many) pertinent versions of the evolving schema. Moreover, the system offers automatic documentation of the schema history, and allows the users to pose temporal queries over the metadata history itself. The proposed demonstration highlights the system features exploiting both a synthetic-educational running example and the real-life evolution histories (schemas and data), which include hundreds of schema versions from Wikipedia and Ensembl. The demonstration offers a thorough walk-through of the system features and a hands-on system testing phase, where the audiences are invited to directly interact with the advanced query interface of PRIMA.

Categories and Subject Descriptors

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

General Terms

Management

1. INTRODUCTION

The ability of archiving past database information and supporting temporal queries over historical databases has long been recognized as highly demanded in Information Systems [9]. This objective, which has provided a long standing motivation for temporal database research, is becoming more and more pressing [3], due to the accountability obligations of organizations such as financial institutions and web portals, e.g., Wikipedia.

Schema evolution, which represented a serious problem already for traditional information systems [10, 7], is even more critical for web information systems [3], and scientific databases [1].

To address the preservation and querying of transaction-time databases with evolving schemas, $\mathcal{PRIMA}$ system [8] has been designed and implemented, based on the following key concepts: (i) a language of atomic schema modification operators (SMOs), exploited by the users to design complex evolution steps, (ii) an XML-based temporal data model for archiving historical data with evolving schemas, (iii) the corresponding temporal query interface based on XQuery, and (iv) a query answering semantics and algorithms, by which users can issue complex temporal queries spanning over multiple schema versions in an easy way. Furthermore, the system allows to pose temporal queries over metadata histories (records of the schema history), similarly to what is done for regular data. We describe the architecture of $\mathcal{PRIMA}$ and a demonstration which (i) guides the audience through the system functionalities, and (ii) allows the participants to directly interact with the system query interface to issue complex temporal queries over transaction-time data archives under schema evolution. The combination of synthetic-eductional and real-life case studies exploited in the demonstration provides an ideal balance between introductory illustrative examples and actual evolution histories from the domain of genetic scientific databases and web information systems, which include the genuine evolution histories of the Wikipedia[1] the free encyclopedia and Ensembl[2] genome database.

2. DEMONSTRATION SCENARIO

The demonstration begins with a brief introduction of the $\mathcal{PRIMA}$ system architecture, as discussed in the Section 3. It then proceeds by highlighting the system features through four simple interrogation scenarios, based on the synthetic evolution history summarized in Table 1 and the evolution histories of Wikipedia and the Ensembl databases. We present the system features in the order of increasing level of complexity as follows:

1. **Historical schema navigation:** This first scenario shows how the $\mathcal{PRIMA}$ users can inspect the schema evolution history itself, by posing temporal queries on it. This functionality is based on the $\mathcal{PRIMA}$ extension we presented in [5]. The users are allowed to ask

SIGMOD'09, June 29–July 2, 2009, Providence, Rhode Island, USA.
ACM 978-1-60558-551-2/09/06.

[1] `http://www.wikipedia.org`
[2] `http://www.ensembl.org`

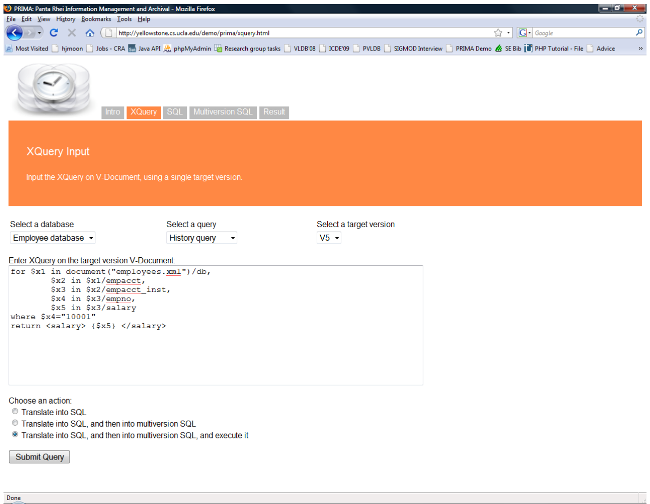

Figure 1: *PRIMA* Interface Screenshot

queries such as *"What was the Wikipedia schema valid at time T_1?"* or *"What is the evolution history of the* `gene` *table in the Ensembl database?"* The relevance of such features is illustrated with the interesting findings from the real-world evolution histories.

2. **Snapshot queries:** This step allows user to issue queries directly on the historical data. Users will select a schema version (by exploiting the above-introduced functionality) and manually pose a snapshot query such as *"Find the salary of employee* 1337 *at time T_1 as of 2001-07-01."* on the data archive. This experience will illustrate the value of a complete archive of the database for flashback or auditing purposes. At the same time it shows the difficulty of manual querying on an archive under schema evolution, even for the simplest snapshot queries. This motivates the *PRIMA* research effort, whose contribution is presented in the next step.

3. **Snapshot queries in the past via the current schema:** This scenario presents one of the main advantages of exploiting *PRIMA* . Users can access the same historical information of the previous example, without even being aware that the schema has ever evolved: past snapshot queries are naturally posed through a schema of choice (typically the current one), and automatically rewritten by the efficient query rewriting engine of *PRIMA* into the equivalent ones valid under the correct past schema version. The system is run open-hood to illustrate the internal mechanics involved.

4. **General temporal queries via the current schema:** Lastly, users are invited to explore the full power of *PRIMA* temporal query engine. The potential of XQuery as temporal language becomes clear when we present several complex temporal queries and their natural XQuery rendering. Users are allowed to ask general temporal queries (e.g., history, range, and temporal-join queries) without the need to deal with the underlying schema evolution. Synthetic and real-life examples are exploited to present some of the optimizations implemented in *PRIMA* . This allows the audience to fully understand how *PRIMA* performance are built from several individual optimizations.

Table 1: Schema evolution in Employee DB

	Schema Versions	T_s	T_e
V_1	engineerpersonnel (empno, name, hiredate, title, deptname) otherpersonnel (empno, name, hiredate, title, deptname) job (title, salary)	T_1	T_2
V_2	empacct (empno, name, hiredate, title, deptname) job (title, salary)	T_2	T_3
V_3	empacct (empno, name, hiredate, title, deptno) job (title, salary) dept (deptno, deptname, managerno)	T_3	T_4
V_4	empacct (empno, hiredate, title, deptno) job (title, salary) dept (deptno, deptname, managerno) empbio (empno, sex, birthdate, name)	T_4	T_5
V_5	empacct (empno, hiredate, title, deptno, salary) dept (deptno, deptname, managerno) empbio (empno, sex, birthdate, firstname, lastname)	T_5	*now*

Table 2: Schema Modification Operators (SMOs)

SMO Syntax
CREATE TABLE R($\mathbf{A}$)
DROP TABLE R
RENAME TABLE R INTO T
COPY TABLE R INTO T
MERGE TABLE R, S INTO T
PARTITION TABLE R INTO S WITH *cond*, T
DECOMPOSE TABLE R INTO S($\bar{\mathbf{A}}$,$\bar{\mathbf{B}}$), T($\bar{\mathbf{A}}$,$\bar{\mathbf{C}}$)
JOIN TABLE R, S INTO T WHERE *cond*
ADD COLUMN C [AS *const*$\vert$*func*($\mathbf{A}$)] INTO R
DROP COLUMN C FROM R
RENAME COLUMN B IN R TO C

A first prototype of the demo (work in progress) is currently on-line at: `http://yellowstone.cs.ucla.edu/demo/prima`. A short video tutorial (i.e. screencast), presenting some of the core system features, can also be found at the same address. A richer and more stable version of the interface, which will be the one presented, is about to be released.

3. PRIMA ARCHITECTURE

Here we discuss how schema evolution are described using schema modification operators and how historical data are archived under schema evolution, based on XML. Then we briefly discuss the algorithm for query rewriting between schema versions. Interested readers are referred to [8] for further details.

3.1 SMOs

Schema modification operators (SMOs) are a set of operators capable of representing schema changes. We summarize SMOs supported in *PRIMA* in Table 2, each of which perform an atomic action on both the schema and the underlying data. The SQL-inspired syntax should be self-explanatory to the purpose of this paper, while the interested readers are referred to [4] for the detailed and formal presentation of the SMOs and their capabilities.

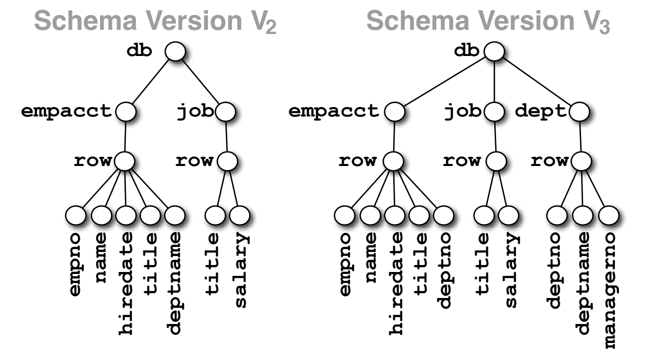

Figure 2: Two schema versions of Employee DB in V-document (V_2 and V_3)

3.2 XML-Based Transaction-time Databases

We archive relational data based on XML, which provides temporally grouped representation (or attribute-level timestamping)[3].

3.2.1 V-Document

V-Document models the history of relational data using XML, as in Figure 2, where versions V_2 and V_3 from our running example are captured as a V-document schema. Its intuitive structure can be represented with an XPath notation as ***/db/table-name/row/column-name***. Each of the nodes, representing respectively database, tables, tuples and attributes, has two attributes, start-time, (**ts**), and end-time, (**te**), representing respectively the (transaction-) time in which the element was added to the database and the time in which was removed. A special value *"now"* is used to represent the end time, which means that the associated value is part of the current DB.

XQuery is used, without any extension, as a temporal language over this representation [11]. This is possible due to the expressive power of XQuery, which is Turing-complete.

3.2.2 V-Document with Evolving Schemas

In order to represent the history of a relational database where the schema evolves along with the content, we extend V-document. Consider the example in Figure 2: the two-table schema version V_2 evolved into the three-table schema version V_3. This change is represented in XML by simply appending new columns and tables after the old ones. The timestamp values guarantee an unambiguous association among tuples, tables and schema versions.

Therefore, we have a general representation, named MV-Document (Multi-schema-version V-Document), capable of representing both the content and the history of our databases using a standard XML representation. Note that all historical data are stored under their original schema version, satisfying our archival requirement.

3.3 Query Rewriting

We rewrite queries between schema versions. The semantics of query rewriting is described in Figure 3: we answer the queries as if all historical data are first migrated into the queried schema version and the the query is executed. Instead of literally implementing this semantics, we take an efficient approach where we rewrite the input query into the relevant historical schema versions. For query rewriting, we use MARS [6] that performs a series of chase and backchase. MARS uses XML Integrity Constraints (XICs) to infer the mappings between schema versions, which is generated based on SMOs.

[3]It has been shown that temporally grouped representation is better than the ungrouped one, due to redundancy and coalescing problems [2].

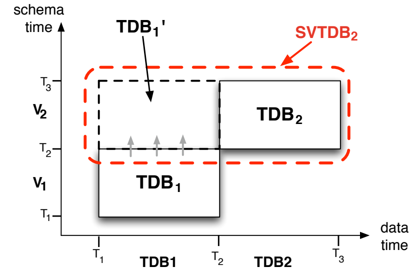

Figure 3: Transaction-time DB under V_1 and V_2

4. REFERENCES

[1] Schema evolution benchmark [on-line]: `http://yellowstone.cs.ucla.edu/schema-evolution/index.php/Benchmark_Extension`.

[2] J. Clifford, A. Croker, F. Grandi, and A. Tuzhilin. On Temporal Grouping. In *Recent Advances in Temporal Databases*, pages 194–213. Springer Verlag, 1995.

[3] C. A. Curino, H. J. Moon, L. Tanca, and C. Zaniolo. Schema evolution in wikipedia: toward a web information system benchmark. In *International Conference on Enterprise Information Systems (ICEIS)*, 2008.

[4] C. A. Curino, H. J. Moon, and C. Zaniolo. Graceful database schema evolution: the prism workbench. *Proc. of VLDB*, 1(1), 2008.

[5] C. A. Curino, H. J. Moon, and C. Zaniolo. Managing the history of metadata in support for db archiving and schema evolution. In *ECDM 2008*, 2008.

[6] A. Deutsch and V. Tannen. Mars: A system for publishing XML from mixed and redundant storage. In *VLDB*, 2003.

[7] S. Marche. Measuring the stability of data models. *European Journal of Information Systems*, 2(1):37–47, 1993.

[8] H. J. Moon, C. A. Curino, A. Deutsch, C.-Y. Hou, and C. Zaniolo. Managing and querying transaction-time databases under schema evolution. *Proc. of VLDB*, 1(1), 2008.

[9] G. Ozsoyoglu and R. Snodgrass. Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):513–532, 1995.

[10] D. I. Sjoberg. Quantifying schema evolution. *Information and Software Technology*, 35(1):35–44, 1993.

[11] F. Wang, C. Zaniolo, and X. Zhou. Archis: An xml-based approach to transaction-time temporal database systems. *The International Journal of Very Large Databases*, 17(6):1445–1463, 2008.