Automating Database Schema Evolution in Information System Upgrades

Carlo Curino[†] Hyun J. Moon[‡] Carlo Zaniolo
University of California at Los Angeles
zaniolo@cs.ucla.edu

Abstract

The complexity, cost, and down-time currently created by the database schema evolution process is the source of incessant problems in the life of information systems and a major stumbling block that prevent graceful upgrades. Furthermore, our studies shows that the serious problems encountered by traditional information systems are now further exacerbated in web information systems and cooperative scientific databases where the frequency of schema changes has increased while tolerance for downtimes has nearly disappeared. The $\mathcal{PRISM}$ project seeks to develop the methods and tools that turn this error-prone and time-consuming process into one that is controllable, predictable and avoids down-time. Toward this goal, we have assembled a large testbed of schema evolution histories, and developed a language of Schema Modification Operators (SMO) to express concisely these histories. Using this language, the database administrator can specify new schema changes, and then rely on $\mathcal{PRISM}$ to (i) predict the effect of these changes on current applications, (ii) translate old queries and updates to work on the new schema version, (iii) perform data migration, and (iv) generate full documentation of intervened changes. Furthermore, $\mathcal{PRISM}$ achieves good usability and scalability by incorporating recent advances on mapping composition and invertibility in the implementation of (ii). The progress in automating schema evolution so achieved provides the enabling technology for other advances, such as light-weight database design methodologies that embrace changes as the regular state of software. While these topics remain largely unexplored, and thus provide rich opportunities for future research, an important area which we have been investigated is that of archival information systems, where $\mathcal{PRISM}$ query mapping techniques were used to support flashback and historical queries for database archives under schema evolution.

Keywords Schema Evolution, Database Design Methodologies, Software Upgrades.

1. Introduction

The difficulty of supporting schema evolution has long been recognized as one the major obstacles hampering software upgrades in information systems [22, 32, 33]. Indeed, there has been no significant progress on this problem since the introduction of relational databases which shield information system applications from changes in the underlying organization of stored data far better than previous database systems such as Codasyl and IMS. After the great progress in physical data independence achieved by the relational model forty years ago, little or no progress has been made toward enabling information system applications to cope with the evolution of the logical structure of the underlying database schema. Indeed, as of today, schema evolution remains an error-prone and time-consuming undertaking, because the DB Administrator (DBA) lacks the methods and automatic tools needed to manage and automate this endeavor by (i) predicting and evaluating the effects of the proposed schema changes, (ii) rewriting queries and applications to operate on the new schema, and (iii) migrating the database. Furthermore, the problem is becoming more serious with the burgeoning spread of web information systems and the databases of large cooperative 'big science' projects, where fast-advancing domain knowledge and active participation by many stakeholders have accelerated the pace of schema evolution. For instance, the MediaWiki software supporting Wikipedia has seen, during its first 4.5 years of development and maintenance, 171 different DB schema versions [8], and the Ensembl Genetic DB over 400 schema versions in nine years of life [7]. Furthermore, similar evolution statistics hold for most of the many information systems we have examined in our Schema Evolution Benchmark (SEB)[1]. Using the rich set of examples provided by the SEB [7], we have have extracted a collection of Schema Modification Operators (SMOs) that can express all (or nearly all) the transformations experienced by the information systems in the benchmark. While the practical effectiveness of the SMOs is supported by the SEB benchmark, their theoretical soundness follows from recent results on database mappings [16], which have produced concepts such as pseudo-inverse mappings [17] and query chase and backchase [14], that proved invaluable in transforming the queries and updates on a schema into equivalent ones [9, 11] on the transformed schema.

The technical advances just outlined are integrated in the design and implementation of $\mathcal{PRISM}$ which through a schema transformation language based on these SMOs and a user-friendly interface enables the DBA to:

(i) predict and evaluate the effects of schema changes upon current applications, and
(ii) once the DBA decides to move forward with these changes, $\mathcal{PRISM}$ performs all the required transformations upon data, queries and updates, and then
(iii) records and documents all these changes on behalf of the DBA.

[†] Current address: MIT CSAIL, curino@mit.edu

[‡] Current address: NEC Lab America, hjmoon@sv.nec-labs.com

[1] While still in an early phase of construction, SEB contains the history of more than a dozen information systems, including those from Wikipedia, Ensembl Genetic DB, and various CERN physics databases [7]. SEB also makes available an assortment of tools for extracting, analyzing, and summarizing such histories available online at `http://www.schemaevolution.org`.

HotSWUp'09, October 25, 2009, Orlando, Florida, USA.

Table 1. New Schema Versions in the lifetime of popular IS

System Name	System type	# of schema versions	lifetime (years)
ATutor	Educational CMS	216	5.7
CERN DQ2	Scientific DB	51	1.3
Dekiwiki	CRM, ERP	11	1.11
E107	CMS	16	5.4
Ensembl	Scientific DB	346	9.8
KT-DMS	CMS	105	4
Nucleus CMS	CMS	51	6.7
PHPWiki	Wiki	18	4.11
SlashCode (slashdot.org)	News Website	256	8.10
Tikiwiki	Wiki	99	0.9
Mediawiki (wikipedia.org)	Wiki	242	6.2
Zabbix	Monitoring solution	196	8.3

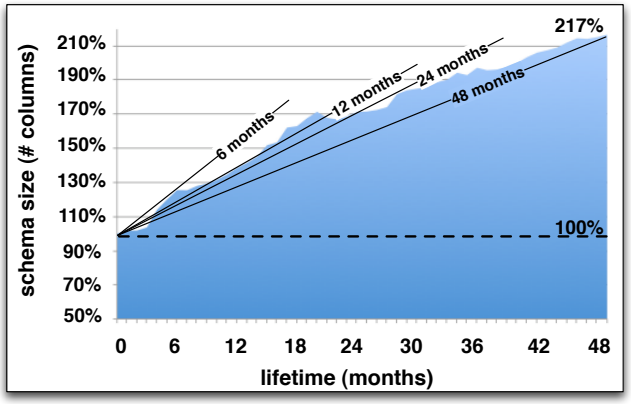

Figure 1. Average growth in the number of tables and columns for the systems in Table 1.

The improvements delivered by $\mathcal{PRISM}$ include significant savings in labor and system downtime, and several other benefits discussed next. A first advantage is that the database schema history is automatically preserved by $\mathcal{PRISM}$ in a standard digital form, which provides the basis for a variety of documentation and automation tools. For instance, the History Metadata Manager (HMM) uses $\mathcal{PRISM}$'s output to represent the history of the database information schema in XML, whereby queries such as *"How many schema changes occurred in the last six months, and what were the tables affected?"* can be answered via simple XQuery statements [10]. Facilities such as this are critical for accountability, data provenance, documentation and training; they also bring a major improvement over the ad-hoc approaches currently used in implementing schema changes—a startling inconsistency with respect to the great efforts, formal methods, and tools used in designing an initial schema that, at the high rates of change we observed, soon becomes obsolete, departing from the original well documented, and well engineered version.

$\mathcal{PRISM}$ has paved the way to other technical advances, including transaction-time databases, and many others. A transaction-time database is simply an append-only history of the database content, kept by archiving and time-stamping subsequent version of records upon creation, modification, and deletion. Alternative database structures and query language extensions were proposed as sophisticated techniques for storing timestamped data, and supporting flash-back and advanced historical queries [34]. However, past approaches did not deal effectively with the ramifications of schema evolution, and in particular with the problem that, in order to achieve archival quality, database information must be maintained according to the schema version under which it was originally created. This "original-schema" storing policy, while achieving perfect quality, generates a serious usability problem, since it requires the formulation queries on a history that is often spread over hundreds of schema versions. To address this problem, we developed the PRIMA system that uses the information stored by $\mathcal{PRISM}$ and similar query rewriting technology to allow querying of the current schema version, whereby the queries are automatically adapted to match every involved historical schema version [25, 26]. For each query Q, PRIMA performs the following operations: (i) determines the applicable schemas according to the temporal conditions in Q, (ii) translates Q into a set of equivalent queries against each applicable schema version, and (iii) combines these queries into an optimized execution plan, that is then executed to produce results conforming to the current schema. The techniques used to to turn this design into an efficient and scalable system are discussed in [25, 26].

Many other promising applications of $\mathcal{PRISM}$ remain largely unexplored. A quite natural one is in support of novel agile methodologies for database design and refactoring, inspired to the corresponding software engineering literature [2, 23]. $\mathcal{PRISM}$ allows in fact to improve and extend the database design by retaining both its behavioral and its information semantics. By starting with a minimalists design and iterating through the steps of (i) refactoring without adding functionality and (ii) adding or revising the functionality of the information system, this Agile Data Method has shown the potential of safely fixing legacy databases, and support evolutionary development. Clearly a system such as $\mathcal{PRISM}$ represents the *sinequanon* for turning database refactoring into a practical low-cost methodology.

A final prospective application of $\mathcal{PRISM}$ is in on-line upgrades of information systems, which seek to support the database schema evolution without down time.

2. Schema Evolution: a reality check

To answer the question: *"How pressing is the problem of schema evolution?"* we collected and analyzed the schema evolution histories of a set of open-source, and scientific, popular information systems (listed in Table 1), that include well-known website such as Wikipedia, Big-Science projects such as the Ensembl genetic DB, and open-source software systems such as Zabbix. This schema evolution benchmark (SEB) offered us a unique vantage point from which to explore, understand, and quantify the characteristics of the evolution of real systems. Our initial hunches about the severity of the problem were more than confirmed by the intense evolution pace we observed. On the average, the systems we analyzed had over 26 schema versions a year; moreover, each evolution step can impact up to 70% of the queries in the applications. In other words, the unsupported evolution of the database schema is a main culprit in maintenance cost blowup.

For the purpose of scaling our analysis to include many information systems, we designed and developed a tool suite automating the process of downloading and analyzing the schema evolution histories. Figures 1 and 2 show two examples of the many statistics collected: the first figure shows the averaged schema growth registered (for the systems listed in Table 1), and the second shows the impact of evolution on the query success rate, i.e., the measure of how many query survive each evolution step (for the first 170 evolution steps of Wikipedia).

More information on the tool-suite and the resulting analysis are available at `http://www.schemaevolution.org`.

3. PRISM: graceful schema evolution

Motivated and validated on the testbed we collected, the $\mathcal{PRISM}$ framework builds on top of the solid theoretical background [13,

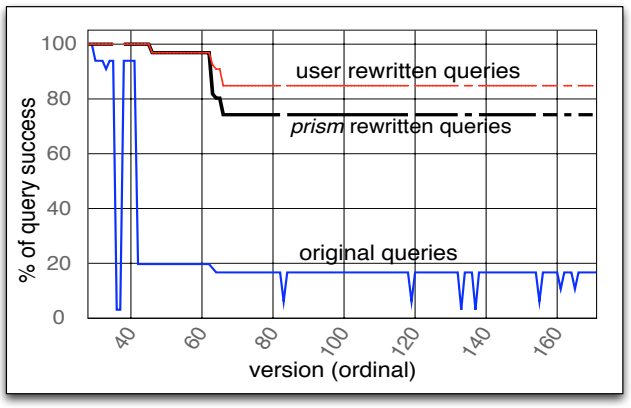

Figure 2. Query success rate in the Wikipedia schema evolution history.

15, 17], offering advanced functionalities through an intuitive, operational interface[2], thus, appealing both researchers and practitioners. The larger systems of SEB proved also invaluable in validating the scalability and performance of $\mathcal{PRISM}$.

Data Migration and Query Rewriting The $\mathcal{PRISM}$ framework addresses the issue of evolving the schema of a database by fully automating the management of data migration, query, and views adaptation upon structural schema changes [9]. The system offers an SQL-inspired, operational language of *Schema Modification Operators (SMO)* to concisely represent the desired schema changes. SMOs represent an intuitive and powerful conceptual tool, that allows DataBase Administrators (DBA)s, to model evolution as a series of atomic changes to the schema. The set of operators and the frequency in which they occur in SEB is shown in Table 2.

Starting from a sequence of SMOs issued by the DBA, the system will generate the SQL scripts needed to evolve the schema and migrate the data (and inverse scripts to migrate the data back if necessary). The system is capable of inverting SMOs sequences and to derive and compose logical mappings between subsequent schema versions. Thus $\mathcal{PRISM}$ system harness recent advances in the theory of schema mapping and query rewriting under constraints, into an effective rewriting engine, that based on the logical mapping between schemas is capable of adapting legacy queries to operate on the current schema version, 's see Figure 3 for an example. $\mathcal{PRISM}$'s solid theoretical foundations provide strong formal guarantees about the correctness of the rewritten queries. Such functionality effectively shields applications from the effects of schema evolution, which thus becomes completely transparent, as we showed in [9, 11]. Finally, $\mathcal{PRISM}$'s graphical interface provides continuous, real-time feedback about the impact on schema, data and queries of the evolution being designed. This significantly increases the predictability of the evolution process itself.

During the generation of the data migration scripts the system performs sanity checks of the existing indexes, verifying whether the evolution would compromise them. If so, the system automatically suggests possible repairs of the indexes, based on a conservative adaptation of the existing indexes given the evolution being proposed. While this solution does not guarantee optimality of the produced set of indexes, it proved nonetheless effective in the most typical scenarios we analyzed.

$\mathcal{PRISM}$ provides an enabling technology to support *agile methodologies for database design*, in ways that are similar to code-refactoring for software design. In fact, the strong tool for migrating data and rewriting queries, and the automatic documentation features of our system, are conducive to *inexpensive, incremental schema design*.

[2] A screencast of the main system interface is available at: `http://videos.schemaevolution.org/PRISM++.mov`

Table 2. SMO Frequencies in the evolutions of systems in Table 1

Operator type	# of usages	% of usage
`ADD COLUMN`	930	34.26%
`DROP COLUMN`	248	9.13%
`RENAME COLUMN`	214	7.88%
`CREATE TABLE`	449	16.54%
`DROP TABLE`	99	3.64%
`RENAME TABLE`	33	1.21%
`COPY TABLE`	6	0.22%
`JOIN TABLE`	7	0.25%
`DECOMPOSE TABLE (vertical)`	4	0.14%
`MERGE TABLE (union)`	4	0.14%
`PARTITION TABLE (horizontal)`	12	0.44%
`ADD PRIMARY KEY`	465	17.13%
`DROP PRIMARY KEY`	38	1.40%
`ADD FOREIGN KEY`	176	6.48%
`DROP FOREIGN KEY`	29	1.06%
TOTAL	2723	100%

Recording and Querying Schema Evolution Histories Continuous evolution leads to a schema that is much changed with respect to its original layout, and where the changes are (typically) not as carefully documented as the original design. In fact, due to constant pressure of deadlines, personnel turnover, and operational interrupts, the documentation about the evolution of the schema fall short of best engineering practices. In this situation, $\mathcal{PRISM}$'s completely automatic approach of systematically recording every change of the schema, achieves timeliness and completeness, in recording crucial *schema provenance*. As shown in Table 1, evolution histories can easily get to hundreds of schema versions, calling for an archival system to store and manage the schema information history, and provide sophisticated temporal access to evolution data to fully empower users. For this very reason the $\mathcal{PRISM}$ offers: *recording and temporal querying of the schema evolution history*. This functionality is provided by the History Metadata Manager (HMM) [10], a component that completely automates the temporal archive of the `information_schema`[3] Therefore, the HMM is a special-purpose transaction-time DB, which supports complex temporal queries on schema evolution histories, such as *"What columns were present in table* `test` *on June, 2003?"*. HMM also records extra information about the operations performed on the schema allowing schema provenance queries such as: *"How were the data currently stored in table* `test` *represented in May, 1999?"*. At the best of our knowledge no other tool provides such advanced querying capabilities.

Managing Historical Data under Schema Evolution Modern organizations are often faced with archival requirements, due to accountability obligations, legal compliance, and the sheer importance of past DB information. The two main problems that occur in managing this archival information are as follows:

- The problem of finding effective temporal representations and query languages for archived information. To this purpose, our system *ArchIS* [36] provides a powerful temporal data model based on XML that captures and support querying on

[3] The `information_schema` is the SQL standard representation of the schema, currently implemented by most DBMS vendors.

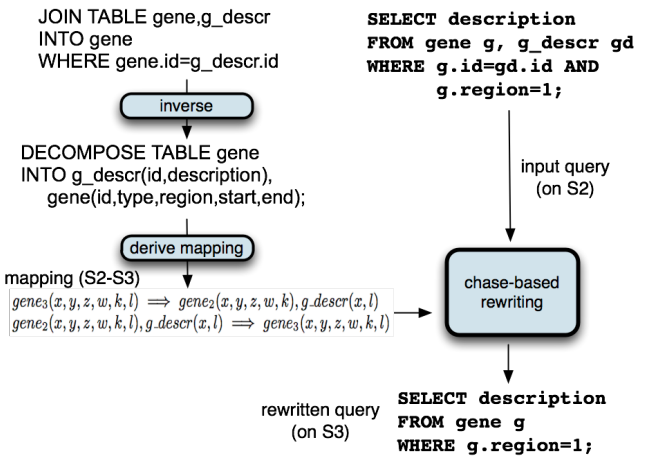

Figure 3. An example of query rewriting in PRISM.

an attribute-level-timestamped history of the snapshot DB. To overcome the current performance limitations of XML, *ArchIS* exploits mature RDBMS technology, by storing data in a efficient column-oriented relational storage.

- The problem of managing and querying information archived under multiple schemas. *PRIMA* [25, 26] solves this problem by extending the functionalities of *ArchIS* to allow the schema to evolve. At every schema change, the snapshot data are archived using the schema version under which they first appeared, achieving perfect archival quality. This produces a transaction-time database that archives the data history under multiple schema versions as needed to archive the content of a database undergoing schema evolution. Query rewriting technology derived from the $\mathcal{PRISM}$ system is then exploited to ease the querying of data history over multiple schema versions.

Practitioners-oriented features The current implementation of $\mathcal{PRISM}$ supports the following functions:

1. transparent recording and documentation of schema evolution,
2. support for complex temporal queries on the metadata history,
3. automatic generation of SQL data migration scripts,
4. automatic rewriting of legacy queries and views.

An extension of the system to support the automatic rewriting of updates (besides queries and views) has been completed, and future extensions to re-write stored procedures are being investigated.

The SEB benchmark proved invaluable in improving the system usability by forcing the addition of several practical improvements, including the following: (i) compatibility with DBMS from different vendors, (ii) ability to automatically rewrite views (based on the same query rewriting technology mentioned above), (iii) the capacity to support queries in the old schema version by means of composed, optimized views (avoiding run-time query rewriting), (iv) a supportive browser-based AJAX interface, (v) support to mimic existing evolution histories (with automatic validation of the designed evolution), and (vi) automatic visualization of schema evolution histories and statistics.

Finally, the SEB benchmark has enabled an extensive side-by-side comparison of our system against several existing commercial tools, open source tools and academic world prototypes.

4. An enabling technology

The set of tools and methods developed within the $\mathcal{PRISM}$ project enables: i) on-line upgrades and graceful evolution of the schema, and ii) a new generation of light-weight database design methodologies.

The recently advances in software engineering involving light-weight development methodologies based on code refactoring, and other approaches embracing evolution as an inevitable portion of an information system development have significantly reduced the costs of evolving and upgrading an Information System. On the contrary the database community has been more slow in providing tools to support the evolution of a database schema as an unavoidable reality. The technological advances we presented in this papers are therefore to be considered an enabling technology for the graceful upgrade of complex, database-centric Information Systems. This also reduce system down time and can eventually lead to online schema evolution.

Moreover, by reducing the costs of evolving the schema of a database the tools we developed paved the road for light-weight database design methodologies. In this context, the rigid upfront schema design methodologies, currently adopted, can be easily substituted by more incremental approaches where the database designer and the software designer works side by side during the information system design and development. The reduced cost of revisioning of the schema of the database, allows one to start with a minimal schema design, directly associated at the initial core of the system being designed, followed by an incremental and evolutionary design process. Furthermore, such approach would allow early validation and inexpensive re-design of the database schema layout, enabling a close-loop of design and development with potentially ground-braking consequences on the performance of the DB.

Such a co-evolution development methodology and the set of tools we described also nicely fit multi-tenant scenarios like the one currently adopted by SalesForce [4], where tenant-specific customization spawn from a basic initial schema. In such a scenario, ease of upgrade is guaranteed by a unified codebase, and common schema portions. Schema customization remains one of the main obstacles that prevent fast/inexpensive upgrades, and it is now effectively addressed by $\mathcal{PRISM}$. Furthermore, similar schema mapping technologies and ad-hoc query rewriting are already employed to guarantee scalability in the numbers of tenants, and thus by the schema size produce by a high cardinality of customizations. A unified approach to deal with schema evolution and multi-tenancy management would increase performance and uniformity all at once.

5. Related Works

Among the many approaches and techniques that are relevant for the general problem of schema evolution we have the impact-minimizing methodology of [29], the unified approach to application and database evolution of [20], the application-code generation of [6] and the framework for metadata model management of [24] and the further contributions available in [3, 4, 30, 35, 37]. While these and other interesting attempts provide solid theoretical foundations and interesting methodological approaches, the lack of operational tools for graceful schema evolution observed by Roddick in [32] remains largely unsolved twelve years later. $\mathcal{PRISM}$ represents, at the best of our knowledge, the most advanced attempt in this direction available to date.

The big players in the world of commercial DBMSs have been mainly focusing on reducing the downtime when the schema is updated [28] and on assistive design tools [12], but lack the automatic query rewriting features provided in $\mathcal{PRISM}$.

Further related works include the results on mapping information preservation by Barbosa et al. [1], the ontology-based reposi-

[4] See `http://www.salesforce.com`

tory of [5], the schema versioning approaches of [21]. XML schema evolution has been addressed in [27] by means of a guideline-driven approach. Object-oriented schema evolution has been investigated in [18]. In the context of data warehouse X-TIME represent an interesting step toward schema versioning by means of the notion of augmenting schema [19, 31]. $\mathcal{PRISM}$ differs form all the above both in terms of goals and techniques.

6. Conclusion

In this paper, we summarized a large research effort aiming at supporting graceful evolution of databases. The products of this effort are a set of tools and methods that effectively reduce the cost evolving the schema of a database. This allows us to consider new lightweight methodologies for database design, that would elegantly fit in the panorama of agile software development methodologies. Furthermore the machinery we designed and developed largely automates the data migration and query adaptation problems, which have been generally been considered one of the main problems in software upgrades. Future directions for this research will focus on effects of schema evolution on data provenance, design of agile methodologies for DB design, and schema evolution workflows.

Acknowledgments

The authors would like to thank Alin Deutsch for the numerous in-depth discussions. This work was supported in part by NSF grant IIS 0917333, "Information Systems Under Schema Evolution: Analyzing Change Histories and Management Tools".

References

[1] D. Barbosa, J. Freire, and A. O. Mendelzon. Designing information-preserving mapping schemes for xml. In *VLDB*, pages 109–120, 2005.

[2] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1st edition, October 1999. ISBN 0201616416.

[3] P. A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.

[4] P. A. Bernstein and E. Rahm. Data warehouse scenarios for model management. In *ER*, 2003.

[5] H. Bounif and R. Pottinger. Schema repository for database schema evolution. *DEXA*, 0:647–651, 2006.

[6] A. Cleve and J.-L. Hainaut. Co-transformations in database applications evolution. *LNCS: Generative and Transformational Techniques in Software Engineering*, pages 409–421, 2006.

[7] C. Curino and C. Zaniolo. Pantha rei data set : `http://dataset.schemaevolution.org/`. 2009.

[8] C. Curino, H. J. Moon, L. Tanca, and C. Zaniolo. Schema Evolution in Wikipedia: toward a Web Information System Benchmark. *ICEIS*, 2008.

[9] C. Curino, H. J. Moon, and C. Zaniolo. Graceful database schema evolution: the prism workbench. *Very Large DataBases (VLDB)*, 1, 2008.

[10] C. Curino, H. J. Moon, and C. Zaniolo. Managing the history of metadata in support for db archiving and schema evolution. In *ER Workshop on Evolution and Change in Data Management (ECDM)*, 2008.

[11] C. Curino, H. J. Moon, M. Ham, and C. Zaniolo. The prism workwench: Database schema evolution without tears. In *ICDE*, pages 1523–1526, 2009.

[12] DB2 development team. DB2 Change Management Expert. 2006.

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

[14] A. Deutsch, A. Nash, and J. Remmel. The chase revisited. In *Principles of database systems (PODS)*, pages 149–158, 2008.

[15] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Composing schema mappings: Second-order dependencies to the rescue. In *PODS*, 2004.

[16] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.

[17] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Quasi-inverses of schema mappings. In *PODS '07*, pages 123–132, 2007.

[18] R. d. M. Galante, C. S. dos Santos, N. Edelweiss, and A. F. Moreira. Temporal and versioning model for schema evolution in object-oriented databases. *Data & Knowledge Engineering*, 53(2):99–128, 2005.

[19] M. Golfarelli, J. Lechtenbörger, S. Rizzi, and G. Vossen. Schema versioning in data warehouses. In *ER (Workshops)*, pages 415–428, 2004.

[20] J.-M. Hick and J.-L. Hainaut. Database application evolution: a transformational approach. *Data Knowl. Eng.*, 59(3):534–558, 2006.

[21] H. V. Jagadish, I. S. Mumick, and M. Rabinovich. Scalable versioning in distributed databases with commuting updates. In *Conference on Data Engineering*, pages 520–531, 1997. ISBN 0-8186-7807-0.

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

[23] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003. ISBN 0135974445.

[24] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A programming platform for generic model management. In *SIGMOD*, 2003.

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

[26] H. J. Moon, C. Curino, M. Ham, and C. Zaniolo. Prima: archiving and querying historical data with evolving schemas. In *SIGMOD Conference*, pages 1019–1022, 2009.

[27] M. M. Moro, S. Malaika, and L. Lim. Preserving XML Queries during Schema Evolution. In *WWW*, pages 1341–1342, 2007.

[28] Oracle development team. Oracle database 10g online data reorganization & redefinition. *Oracle White Paper*, 2005. URL `\url{http://www.oracle.com/technology/deploy/availability/pdf/ha_10gR2_online_reorg_twp.pdf}`.

[29] Y.-G. Ra. Relational schema evolution for program independency. *Intelligent Information Technology*, pages 273–281, 2005.

[30] S. Ram and G. Shankaranarayanan. Research issues in database schema evolution: the road not taken. In *Boston University School of Management, Paper No: 2003-15*, 2003.

[31] S. Rizzi and M. Golfarelli. X-time: Schema versioning and cross-version querying in data warehouses. In *ICDE*, pages 1471–1472, 2007.

[32] J. Roddick. A Survey of Schema Versioning Issues for Database Systems. *Information and Software Technology*, 37(7):383–393, 1995.

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

[34] V. J. Tsotras and A. Kumar. Temporal database bibliography update. *SIGMOD Record*, 25(1):41–51, 1996.

[35] Y. Velegrakis, R. J. Miller, and L. Popa. Mapping adaptation under evolving schemas. In *VLDB*, 2003.

[36] 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*, 2008.

[37] C. Yu and L. Popa. Semantic adaptation of schema mappings when schemas evolve. In *VLDB*, 2005.