Bulletin of the Technical Committee on

Data Engineering

September 2002 Vol. 25 No. 3

IEEE Computer Society

Letters

Special Issue on Integration Management

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

The Data Engineering Bulletin web page is http://www.research.microsoft.com/research/db/debull.

Letter from the Editor-in-Chief

ICDE and its Steering Committe

The Interational Conference on Data Engineering (ICDE) is the flagship conference of the IEEE Computer Society in the area of databases (data engineering). This conference is held annually during the late winter or early spring at locations around the world, whereever there is a community of database researchers and/or database industry. For example, ICDE'01 was in Heidelberg, Germany, while ICDE'02 was held in San Jose. ICDE'03 will be held this coming February in Bangalore, India. A "call for participation" for ICDE'03 is on the back inside cover of this issue of the Bulletin.

I suspect that most individuals do not know the details of how the ICDE conference is organized. While the conference is sponsored by the Technical Committee on Data Engineering (TCDE), the conference is, in fact, organized and run by the ICDE Steering Committee. By coincidence, Erich Neuhold is the chair of both the Technical Committee and the ICDE Steering Committee, though having different chairs is more common. Erich was just re-elected to a second two year Steering Committee term. Members of the Steering Committee are limited to serving two two-year terms. Other current members of the ICDE Steering Committee are Elisa Bertino, Dimitrios Georgakopoulos, Masaru Kitsuregawa, Paul Larson, David Lomet, Calton Pu, and Philip Yu. They meet at least once a year, usually twice, and select from among the proposals that are presented to them for conference venues. To be eligible to serve on the steering committee, one must have first served in a significant capacity in an ICDE conference.

The Current Issue

"Integrating" data from multiple sources is a major problem of our industry now and it will continue to be a problem in the future. We as a community have become increasingly aware of this because of the emergence of the web. But, indeed, enterprises faced this problem earlier, within their own organizations. There is too much foresight and too much coordination needed to avoid having local centers emerge, each with its information stored in a manner ideosyncratic to the local center. Hence, this is likely to be a permanent problem. Clearly, the more that we can know about the data involved, the better able we will be to integrate it effectively. Hence "metadata" (the information about data) and how we treat it becomes a very big part both of the integration problem and of any solution to it.

"Integration Management" is the subject of the current issue. Once again, in the Bulletin tradition, the issue reports on work across a wide swath of our community, academia, industrial research, and industrial development and product. Renée Miller is an academic who has worked in this area herself, *and* with two industrial research labs working in the area. She knows the area well and has assembled a fine issue that provides a broad snapshot of the latest technology in information integration. Renée had a very busy summer, and was originally reluctant to commit to doing the September issue. But the result speaks for itself. I want to thank René both for the fine job and for her cooperative attitude, which surely makes my job easier.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

In this special issue, we take a look at a number of new trends in the management of heterogeneous data. I have titled this issue "Integration Management" because the development of robust, comprehensive management and maintenance solutions is a central theme underlying these trends.

We begin with two papers from integration practitioners that present several important research challenges. The first by Seligman *et al* presents a survey of how effort is distributed among different integration tasks. This survey highlights several time-consuming tasks that are ripe for enhanced automation. A few of these tasks that are considered within this issue include: obtaining source knowledge (including the understanding and representation of structure and data semantics), the creation of schema correspondences (or schema matches), the creation of data combination rules, the creation of logical mappings, and the creation of data translation programs. Narayanan and Subramanian present a different look at integration, but again with a practitioner's eye. They identify several open problems including the management of partial or incomplete data. In addition, some important issues are raised with respect to materializing or replicating integrated data. Finally, Narayanan and Subramanian consider the need to manage and integrate not only data, but also the queries (and query capabilities) and the context or environment of the data. In effect, the authors are proposing more robust and extensive knowledge management facilities that manage the full integration problem, not just the data.

After the practitioners' call-to-arms, we move on to three specific systems designed to manage heterogeneous data. Buneman *et al* present a system for integrating vertically partitioned databases. The main goal of the system is to provide fast, flexible data transformation operators. The emphasis is on data integration in environments where the integrated data may be materialized and this materialized data must be kept updated. The system is used in the very dynamic environment of Wall Street trading. Our next two systems are the result of strong industry-university collaborations. The first is a collaboration between the University of Washington and Microsoft Research on *generic model management*. In the paper presented here, Pottinger and Bernstein consider a system for creating integrated schemas that is *generic*. The emphasis is on abstracting out the data-model-independent portions of the schema integration task. The algorithm makes use of schema correspondences to produce an integrated schema. While these correspondences do not contain data semantics, this work suggests that the merge result may be used to constrain the set of possible mappings. The next system is a collaboration between the University of Toronto and IBM Almaden Research Center on *schema management*. In the paper presented in this issue, Andritsos *et al* focus on the management of schemas and mappings. They describe solutions for creating and managing: schemas; correspondences (matches) between schemas; mappings between schemas (which provide a semantics for transforming data); and translation programs or queries. The authors consider the use of mappings for both data integration and data exchange.

We conclude with two papers related to the management of incompleteness and inconsistency that may arise from the integration or transformation of data. These two papers lay a foundation for identifying, representing, and reasoning about incompleteness and inconsistency within the context of data transformation. Calì *et al* address the problem of data integration where a unified (virtual) view is provided over a set of heterogeneous sources. They argue that integrity constraints on the view are required to model the semantics of the integrated data and to correctly query the integrated data. In the presence of such constraints, they show that query answering in data integration raises the issue of dealing with both incompleteness and inconsistency. Grahne also considers the relationship of querying integrated data and querying incomplete information. He surveys recent result on query rewriting for incomplete information and shows how they may be applied in data integration.

It is notable that the importance of *data semantics* in managing heterogeneous data is highlighted throughout the issue.

Renée J. Miller
Department of Computer Science
University of Toronto

Data Integration: Where Does the Time Go?

Len Seligman, Arnon Rosenthal, Paul Lehner, Angela Smith
{seligman, arnie, plehner, asmith}@mitre.org

Abstract

We present a modular breakdown of data integration tasks and the results of a survey on the distribution of effort among those tasks. The modularization aids in project planning and enables portions of the work to be allocated flexibly among various human specialists, and also to automated tools. The survey results are useful for determining: (1) what are the highest value research problems to tackle? (2) where should large enterprises focus their investments in data integration tools?, and (3) how should a data integration project manager allocate people and schedule?

1 Introduction

Data integration–i.e., meaningful information exchange among systems not originally designed to work together–is a difficult challenge for large enterprises. Despite substantial progress, data integration remains expensive and labor-intensive.

In prior work, we described organizational factors–e.g., staff specialization, training costs, and participants' incentives–that are too rarely considered in data integration research. [9] proposed a repackaging of familiar data integration tasks and research problems to better fit organizational needs, describing the skills needed for each step. The goal is to move toward a more industrial process of data integration.

This paper summarizes the categorization of data integration tasks from [9] and presents the results of a survey on the distribution of effort among those tasks. Section 2 describes the task breakdown, Section 3 explains the survey procedure, and Section 4 presents its results. We close with a discussion and summary.

2 A Task Breakdown for Data Integration

The survey required a breakdown of integration tasks that was independent of any particular process or tools, instead describing products that must (at least implicitly) be part of any process. We emphasized tasks that must be performed by user organizations. Thus, we considered translation of schemas into a common data model to be out of scope, since this is increasingly provided by database and middleware vendors. The tasks are:

1. *Gather knowledge about sources* – Prior to integrating multiple data sources, one must understand the schemas, representation, and semantics of each data source.

2. *Gather knowledge about desired consumer (target) view(s)* – Similar to Task 1, but this time for the interface(s) to be used by consumer users and systems.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

3. *Identify semantic correspondences among sources and from sources to the consumer views [6]* – In this step, one determines entities and attributes in the different systems that refer to the same (or similar) real world concepts–e.g., that Emp.Seniority in a source system can be used for Worker.YearsOfService in the consumer view.

4. *Create needed attribute transformations* – This step produces executable functions that transform attributes in the sources to properly feed the consumer views – e.g., that one must multiply HourlyWage by HoursWorked to produce Salary.

5. *Specify data combination rules* – When multiple source rows each contribute values to a single target row, how should the combination work?

   - Join or union? If a join, on what fields? Inner or outer join?
   - What result to produce when sources differ on the same fact (e.g., what is Joe's salary, really). For example, one might specify "use the mean" or "use Source1 if non-Null".

6. *Create logical mappings from sources to consumer* – Given the above information, produce an explicit mapping from sources to the target (e.g., expressed as SQL views).

7. *Data cleaning* – Discovering and correcting incorrect data values.

8. *Create and optimize an executable connection for the specific run-time environment* – e.g., a replication tool that supports an SQL subset or an extract-transform-load (ETL) tool that uses Visual Basic for transformations.

There are several advantages of this task breakdown. First, it is composed of modular, single-skill tasks. Such a breakdown aids in human resource planning, and enables portions of the work to be allocated flexibly among various human specialists (e.g., DBAs, domain experts, distributed systems programmers), and also to automated tools. Second, we believe this categorization applies to diverse integration scenarios, for example:

- Developing a data warehouse and populating it with commercial ETL tools
- XML-based information exchange among loosely coupled systems with divergent schemas
- Creating a database federation

Third, this breakdown is finer grained than previous ones. In particular, we make the knowledge capture tasks (1 and 2) first class, instead of subordinating them to others (e.g., "Schematic interschema relationship generation" and "Integrated schema generation" in the breakdown of [8]). In order to discover correspondences and discrepancies across schemas, one must learn a lot about the systems involved. We believe it is important to capture this knowledge for use in subsequent integration efforts.

Finally, the categorization identifies natural tool/expertise niches. Commercial data profilers (e.g., from Ab Initio, Ascential) support capturing source knowledge. Cupid [4] and other matchers address correspondence identification. The Context Interchange project [1] addresses creation of attribute transforms. Clio [5] supports several task categories including correspondence identification, data combination rules, producing logical mappings, and executable ones for various environments (e.g., XSL, XQuery, and SQL).

Response Type	Helpful	Not Helpful	% Helpful
All Responses (n=93)	76	17	82%
Non-MITRE (n=61)	57	4	93%
Anonymous (n=16)	14	2	88%

Table 1: Percent that found the Integration Task Categories Helpful

3 Survey Procedure

A web accessible survey was made available. Requests for participation were solicited from the following sources (the number of responses received from each is shown in parentheses):

- "All MITRE technical staff" email list, consisting primarily of systems engineers supporting U.S. government applications (33 responses)
- Referrals from MITRE technical staff, consisting of U.S. Government personnel and their contractors with data integration experience (18)
- DBWORLD email list, consisting mostly of data management researchers (23)
- DBRSRCH - A list (compiled by the authors) of selected data integration researchers (2)
- ODTUG - Oracle Development Tools Users Group email list (15)
- DB2 International Users Group email list (1)
- Comp.databases newsgroup (2)
- Participants at the Data Warehousing Institute Spring '02 World Conference (1)
- SQL Server World Users Group Development (SSWUG-DEV) email list (0)

The survey (see Appendix) contained six background questions on research or tool development experience, and whether they had experience on real integration efforts of the following types:[1] small (but nontrivial) point-to-point, small multipoint, large point-to-point, and large multipoint. Respondents were then asked to allocate percent of effort devoted to the eight categories of tasks described above. They were asked to do this for both small and large data integration efforts.

4 Results

We received 95 responses from a wide variety of organizations. Not all respondents answered all questions, but most questions had at least 75% coverage. Detailed procedures and definitions appear in the Appendix.

Normalization and Outliers. For the analyses below, the effort distribution percentages were normalized to 100. As with any questionnaire some respondents may answer randomly or dishonestly. To test for this, we excluded two "outliers" (see Appendix) and analyzed the remaining 93 responses.

Perception of Survey's Task Categorization. Overall, as indicated in Table 1, most respondents indicated that they found the task categorization described above to be helpful in thinking about data integration problems. A few respondents suggested that "integration testing" be included explicitly.

[1] In the survey, "real" meant the goal was to provide data to real users and not merely to demonstrate integration technology. "Small but nontrivial" meant 8-20 entities, while "large" was > 20.

	source knowl	user view knowl	smntc crspnc	atrbt xforms	data combo rules	logical map-pings	data clng	opti-mize exctbl
Small (n=80)	17.3	11.8	13.4	12.2	10.0	11.7	14.1	9.6
Large (n=70)	18.2	11.6	14.7	11.8	11.3	9.8	14.9	7.8
Diff	(0.9)	0.2	(1.4)	0.4	(1.3)	1.9	(0.8)	1.8

Table 2: Time Allocation on Small vs Large Efforts

	source knowl	user view knowl	smntc crspnc	atrbt xforms	data combo rules	logical map-pings	data clng	opti-mize exctbl
Yes (n=55)	17.9	11.3	13.8	11.0	11.4	10.0	16.6	8.0
No (n=15)	19.4	12.5	18.1	14.6	10.6	8.9	8.7	7.3
Diff	(1.6)	(1.2)	(4.3)	(3.6)	0.8	1.1	**7.9****	0.7

Significance levels with two tailed t-test: **=p<.01 (no others attained p<.05)

Table 3: Large-scale Data Integration Experience Yes/No

Small vs. Large-scale data integration. Table 2 compares the effort estimates for small and large-scale data integration efforts (across all respondents, regardless of experience). As can be seen, there is little discrimination between the two distributions, with no statistically significant differences.

The lack of differentiation between small vs. large data integration tasks also holds true in all of the analyses we discuss below. Consequently, the rest of this section shows results only for large data integration projects. Within that set, the results are categorized by the respondent's apparent experience.

Applied data integration experience. Table 3 shows the difference between 55 respondents who claim to have large-scale data integration experience from the 15 who do not have such experience. Here there is a substantial difference on data cleaning. Specifically, experienced integrators view data cleaning as the second most effort consuming task (16.6%), while those without this experience view data cleaning as the second least effort consuming task (8.7%).

To broaden this observation, we hypothesized a correlation between data integration experience and perceived effort devoted to data cleaning. While the questionnaire did not directly ask about quantity of experience, we are able to approximate experience level by sorting respondents by:

- **2 Lrg** = Large multipoint and Large point-to-point experience
- **1 Lrg** = Either Large multipoint or Large point-to-point experience (but not both)
- **Sml MP** = Not Large experience but small multipoint experience
- **Sml PtP** = Small point-to-point experience only
- **None** = No experience at all on real integration efforts

As can be seen in Table 4, there is a clear relationship between perceived effort for data cleaning as experience increases. There also appears to be an inverse relationship between experience and estimates for specifying semantic correspondences, although the effect is not statistically significant.

Research Experience. The results shown in Table 5 reveal substantial and statistically significant differences between apparent researchers (see Appendix) and practitioners. Respondents from research sources significantly

Experience Level (see above)	source knowl	user view knowl	smntc crspnc	atrbt xforms	data combo rules	logical map-pings	**data clng**	opti-mize exctbl
2 Lrg (n=40)	14.2	10.5	13.0	11.8	11.5	11.5	**18.1**	9.3
1 Lrg (n=15)	27.6	13.4	15.9	8.8	11.2	6.1	**12.4**	4.5
Sml MP (n=9)	19.0	15.5	17.4	12.8	10.6	9.3	**9.5**	6.0
Sml PtP (n=5)	17.0	8.6	19.0	16.6	11.8	9.0	**8.0**	10.0
None (n=1)	35.0	5.0	20.0	20.0	5.0	5.0	**5.0**	5.0

Table 4: Level of Experience

	source knowl	user view knowl	smntc crspnc	atrbt xforms	data combo rules	logical map-pings	data clng	opti-mize exctbl
Research (n=16)	22.7	13.0	20.5	10.0	7.8	8.7	9.7	7.6
Other (n=54)	16.9	11.2	13.0	12.3	12.3	10.1	16.4	7.9
Diff	**5.9***	1.8	**7.4***	(2.3)	**(4.4)****	(1.4)	**(6.7)****	(0.3)

Significance levels with two tailed t-tests: **p<.01, *p<.05

Table 5: Research Sources vs Other Sources

underestimated the effort required for data cleaning and specifying data combination rules compared to respondents from other sources (p<.01). In addition, researchers overestimated the effort to gather knowledge from sources and specify semantic correspondences (p<.05).

Magnitude of Effort for Different Integration Subtasks. The bar chart in Figure 1 repeats the estimates (from Table 3) of respondents who have done large-scale data integration, and also shows standard deviations. As can be seen, the most time-consuming tasks were gathering knowledge of sources, data cleaning, and identifying semantic correspondences. The least time consuming tasks were creating logical mappings and creating and optimizing an executable connection. We discuss these results below.

5 Discussion

Retrospective estimates of time allocations on typical tasks are clearly prone to error. As shown in Figure 1, estimates for all integration task categories had high standard deviations.[2] The time estimates seemed suspiciously uniform across categories (not more than ~2:1 ratio from largest to smallest) and between estimates for large and small projects. The regularity could be an artifact of the survey procedure.

The estimates can provide a first draft for staff and cost estimates for an integration project, and ensure that none of our identified steps is forgotten. To refine the schedule, one must apply judgements about the current project, especially where estimates' variability is greatest (e.g., data cleaning). Another use is for judging progress. If one has already expended 30% of allocated project resources on (say) gathering knowledge of sources, there are problems either with the process or with the schedule.

If one averages over many projects, randomness decreases. Therefore, the estimates (if not systematically biased) can give hints to CIOs and vendors who prioritize different types of integration tools, based on potential savings of effort and of scarce categories of personnel. Data profiling tools (useful for gathering knowledge of

[2]Given the high standard deviation, measurements for only a few real projects would seem equally misleading. In addition, it was difficult to gain access to current projects.

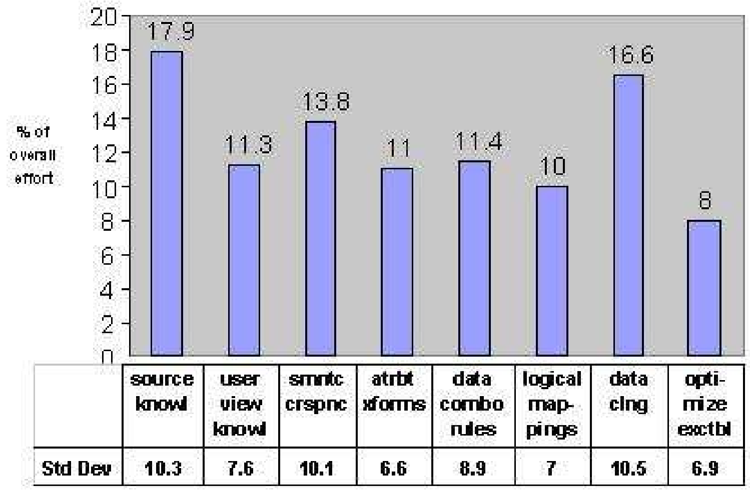


Figure 1: Percent distribution on large efforts, where respondents had done large-scale integration (n=55)

sources) and data cleaning tools (both areas where there has been commercial progress) are especially worth considering.[3]

Table 5 suggests that researchers should increase their attention to data cleaning and to specifying data combination rules. Today's tools are compendiums of simple techniques, but researchers are now seeking more power (e.g., through a combination of machine learning and human assistance) [2, 7, 10].

We were encouraged by the total number of responses received – it enabled results to be significant under 2-tailed t tests. The tactic of distinguishing users by experience was less successful. Most users claimed multiple kinds, and this was incompatible with our tactic of assuming that user estimates described the sort of project they had worked on (e.g., multi-point).

We are considering doing additional surveys. A larger response set would give larger populations for combinations of categories. However, few of our hypotheses just missed being significant, so a modest increase in sample size (e.g., by emailed requests for respondents to bring in their colleagues) seemed of marginal use. Still, a revised survey could be illuminating, with project-specific estimates tied to project characteristics, such as point-to-point vs. multipoint, types of tools used, and political environment (e.g., does your organization control the sources?). Finally, a collaboration with a contractor who does many integration efforts might give better task estimates than those obtained from respondents' memories.

The survey data is available to the community, for testing other hypotheses.

6 Summary

We adapted conventional steps of schema integration to be more friendly to project management. Each has a defined result, requires a narrow set of skills, and (in many cases) corresponds to a natural product capability.

Our survey produced interesting results, both negative and positive, though it is clearly not definitive. Most respondents (82%) found the task breakdown helpful. The reported time allocations for small and large integration problems were nearly identical. Researchers produced higher estimates for analyzing sources and for semantic correspondence than practitioners. Practitioners assigned greater time for data combination rules, and much greater weight for data cleaning; the latter effect increased with experience.

[3] In addition to data cleaning tools (which apply fixes to already bad data), managers should look seriously at improving data quality management at the sources whenever possible [3].

References

[1] C. Goh, S. Bressan, S. Madnick, M. Siegel. Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. TOIS, 17(3), 1999

[2] M. Hernndez, S. Stolfo. The Merge/Purge Problem for Large Databases. SIGMOD Conference 1995

[3] D. Loshin. Enterprise Knowledge Management: The Data Quality Approach, Morgan Kaufmann, 2001

[4] J. Madhavan, P. Bernstein, E. Rahm. Generic Schema Matching with Cupid. VLDB 2001

[5] R.J. Miller, M. Hernndez, L. Haas, L. Yan, H. Ho, R. Fagin, L. Popa. The Clio Project: Managing Heterogeneity. SIGMOD Record, 30(1), 2001

[6] E. Rahm, P. Bernstein: A survey of approaches to automatic schema matching. VLDB Journal 10(4)

[7] V. Raman, J. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. VLDB 2001

[8] S. Ram, V. Ramesh. Schema Integration: Past, Present, and Future, in A. Elmagarmid, M. Rusinkiewicz, A. Sheth, Management of Heterogeneous and Autonomous Database Systems, Morgan Kaufmann, 1999

[9] A. Rosenthal, L. Seligman, S. Renner, F. Manola. Data Integration Needs an Industrial Revolution. International Workshop on Foundations of Models for Information Integration (FMII-2001), Viterbo, Italy, September 2001, http://www.mitre.org/resources/centers/it/staffpages/arnie/pubs/Foundations01.pdf

[10] L.-L. Yan, R.J. Miller, L. Haas, R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. SIGMOD Conf., 2001.

Appendix: Methodology Details

This section provides details and rationales for our analysis, to help make the effort repeatable. The survey text and the response data are at http://www.mitre.org/resources/centers/it/staffpages/arnie/SurveyPaper-DataEng

Definitions

***"Researchers"*:** Self-rating of research experience was of little use. 70% of respondents claimed research experience, including respondents from sources consisting primarily of practitioners (e.g., 53% of the ODTUG respondents). From this, we concluded that interpretations varied considerably on what it means to "perform research in data integration"– probably from researching alternate products for a warehouse implementation, to performing original research aimed at peer-reviewed conferences or journals. In the future, we suggest that surveys make the latter definition explicit.

Given this problem, we instead distinguished researcher vs. nonresearcher by comparing the two primarily research-oriented sources of respondents (i.e., dbworld and "dbrsrch," our list of selected data integration researchers) against responses from the other sources, which consisted primarily of practitioners. This gave statistically significant results, though self-identification did not.

Outliers: If a respondent provided a task percentage that was more than five standard deviations away from the mean response, then that respondent's data was dropped from the analysis. Two respondents were rejected using this test. Upon inspection, one of these was an obvious coding error. Perhaps the other had an idiosyncratic interpretations of the definitions.

Procedures and Rationales

We wanted a short questionnaire, since longer ones get fewer responses. We felt that we could ask for estimates of only two situations, and opted for long versus short projects. Beyond this, we used self-described experience or source of the response to differentiate kinds of efforts.

For the multipoint/point-to-point dichotomy, there was substantial overlap in the populations. Of those that gave estimates for large projects, 49 indicated they had done both multipoint and point-to-point, 13 said they had done only multipoint, while only 7 said they had just done point-to-point. There were no substantial or statistically significant differences between those who had only done point-to-point vs. those who had only done multipoint.

Since large and small gave similar behavior, and we were especially interested in large-scale integration problems, most of the data presented was for large-scale efforts.

The hardcopy questionnaire indicated that responses would be normalized to 100, while the Web version indicated that the total should equal 100 and displayed a running total as respondents completed the questionnaire. Of the 95 responses, only 4 used the hardcopy version.

While participant selection was not random, we have no reason to believe that self-selected participation was based on reasons other than curiosity and a willingness to help.

Integration Through a Practitioner's Eye

Srinivasa Narayanan, Subbu N. Subramanian, and the Tavant Team
Tavant Technologies
3114 Scott Blvd., Santa Clara, CA 95054, USA
{srinivas.narayanan, subbu}@tavant.com

1 Introduction

Publications in the area of data integration address several important data model, query processing, and data semantics issues. This paper presents integration from a practitioner's perspective – those practical pain points that are not discussed in typical research literature, but big enough to warrant careful study in order to realize practically useful integration systems. The paper discusses several challenges that arise in real-life integration scenarios, some techniques for addressing them, and identifies several key areas that require further study.

This paper is based on our experiences with building Tavant InterConnect, the crucial integration component of the Tavant Solutions Framework [NST01]. In Section 2, we set the stage for our discussions by providing an overview of the Tavant solution with special emphasis on its integration component. In Section 3, we discuss the nature of integrated data in practice and issues related to its representation. We discuss the use and manipulation of integrated data in Section 4, present several practical challenges that arise in real-life integration in Section 5, and finally conclude in Section 6.

2 An Overview of Tavant Integration System

The Tavant system is a framework for collaborative commerce solutions that help companies bring their sales and distribution processes to the Web. It leverages the internet as a medium to enable collaboration among various partners (such as manufacturers, dealers, distributors, and end customers) in the sales and distribution channels of large corporations. The systems, data, and business processes of the channel partners are usually autonomous and heterogeneous. Achieving a seamless integration towards providing the end user with a unified "single point of access" user experience is a key feature of the Tavant system.

Tavant InterConnect, the integration component of the Tavant system, provides a hub-and-spoke model for integrating the data and business processes of the channel partner systems. As illustrated in Figure 1, the Tavant integration hub integrates the various partner spoke systems and enables existing business processes on the spoke systems to be accessed via the hub. In addition, it provides new collaborative business processes enabled by the integration of the partner systems.

Invoking a business process that is exposed on the integration hub involves accessing data and business logic on the hub and from one or more spoke systems. The hub stores meta-data about the various systems and acts as an intelligent orchestrator by pulling data local to the hub as well as data resident on the spoke systems, invoking

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

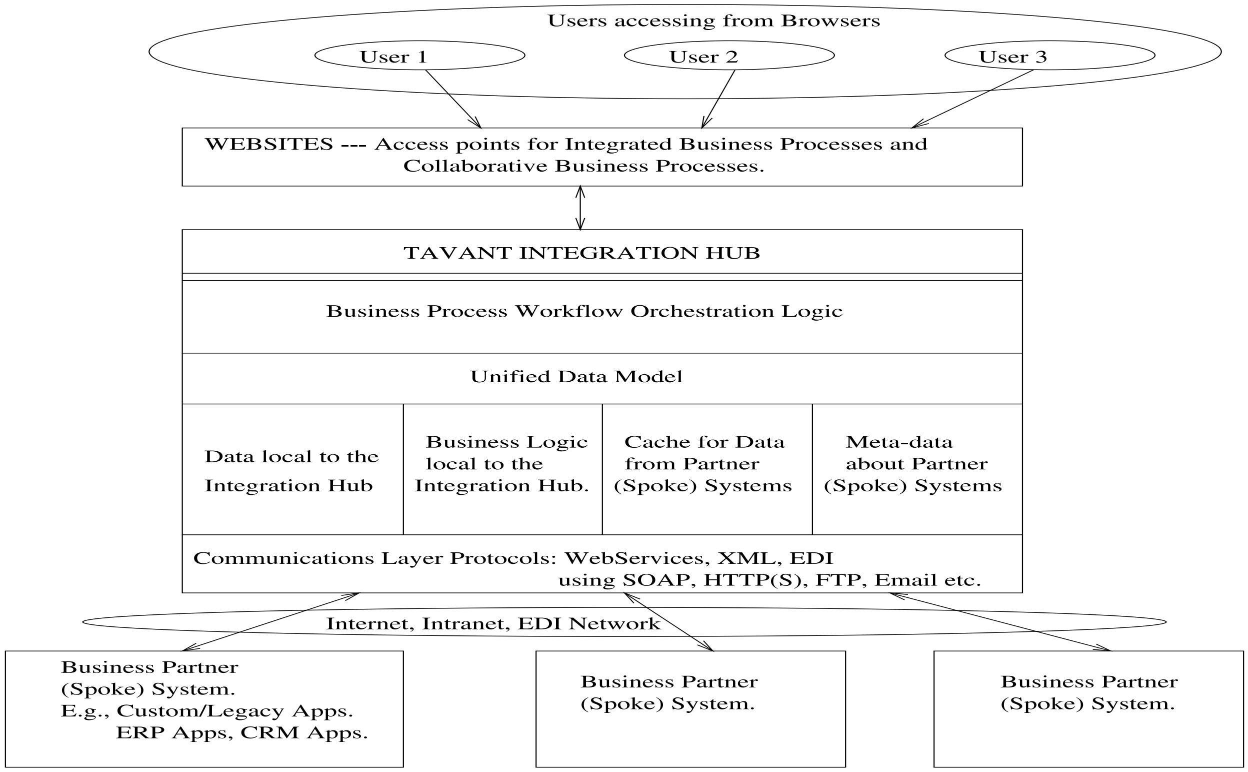


Figure 1: The Tavant Integration Architecture

appropriate business logic on the various systems, and composing the result of all these transactions into a result that is meaningful for the end user.

3 Integration in Real-life – Nature and Representation

In this section, we discuss the nature of integration as well as its influence on the representation of integrated data.

As mentioned in Section 2, the key goal of many real-life integration systems, including Tavant InterConnect, is to bring together related, but isolated business processes belonging to multiple component systems in such a manner that they can be seamlessly accessed over a common medium (such as the web). Thus, the ultimate focus is on *process integration* as opposed to data integration alone. We elaborate on this distinction below (Section 3.1). We discuss how business needs and real-life requirements motivate the need for integration in the presence of partial data and the challenges they impose on data representation (Section 3.2). We also motivate the need for extending the canonical data model to handle component system-specific needs and present some techniques to address this challenge (Section 3.3).

3.1 Process Integration *vs* Data Integration

The key goal of the Tavant system is to Web-enable existing business processes and to provide new collaborative business process among business partners. To achieve this goal, the main focus of the system is to integrate the business processes of the partners rather than achieve deep semantic integration of business data belonging to the component systems.

Data integration in the Tavant system involves mapping the data model of the component systems to Tavant hub's *canonical data model*. The canonical data model is powerful and allows for achieving a well-defined mapping among concepts belonging to the component data models whenever these concepts are precise and unambiguous in all systems. However, in real-life, there exist concepts that are unique to component systems or whose semantics vary among the component systems. In order to account for these, the canonical data model has the flexibility to allow for concepts to be modeled without necessarily associating any deep semantics with them. Though the model does not "interpret" these concepts, the hub system is still aware of them. This allows for the hub system to act as a conduit for these uninterpreted data to be passed (as part of a business process flow) to the specific component systems. The component systems, which are aware of the semantics of the data, individually interpret the data for further processing.

We now illustrate this with an example. The Tavant integration hub has a generic *Order Management module* that models the order processing functionality of various partners in the sell-side channel. The canonical data model for this module includes concepts such as *buyer, seller, shipping address,* and *billing address*. These concepts are usually precise and unambiguous across all the channel partners. However, there exist concepts that are unique to a channel partner. For example, consider a channel partner that has a proprietary set of *order processing codes* – codes such as urgent, normal, or ground that determine the shipping priority of the order – that the buyer can associate with an order. Since these order processing codes are unique to that channel partner, they are not modeled as concepts in the canonical data model. However, the model allows for capturing this information as part of the order processing workflow so that the codes can be sent to the channel partner component system for interpretation and further processing. As can be seen, the Tavant hub system focuses on integrating the *Order Submission process* across the various channel partners while respecting differences among the component systems. A simplified view of this integrated order submission process consists of the following steps – (1) *Product Selection*, (2) *Buyer Information and Preferences Entry* (that includes entering the priority code in the case of the channel partner discussed above), (3) *Shipping Information Entry*, (4) *Billing Information Entry*, and finally (5) *Order Submission*. Thus, the Tavant integration layer focuses on integrating these workflow processes to allow the channel partners to submit and receive orders, than on attaining deep data integration.

Another crucial component of the integration hub is a canonical business process flow model that acts as a master workflow template from which the component systems' workflows can be derived. The canonical workflow accounts for the minor differences in the component workflows. For example, the order submission process described earlier may differ slightly across the various channel partners. Some channel partners may not need the "Billing Information Entry" step of the workflow because their existing accounts and invoicing procedures do not require this information. Similarly, some channel partners may not want a specific section of the "Shipping Information Entry" (such as the one that allows the buyer to choose a "Shipment Carrier") because this information is not applicable to them. To address these needs, Tavant's Order Management module defines a master workflow template for the Order Submission process consisting of a set of components that can be customized to the individual channel partner needs as outlined above.

In practice, to facilitate collaboration, business partners maybe willing to make limited changes to their existing workflows. The definition of the canonical workflow must take into account the current workflows of the partner systems and the changes that are permissible on these workflows. If business processes can be represented using workflow definition languages, an interesting challenge is to automatically create the canonical workflow from the existing workflows and the changes permissible on these workflows.

3.2 Partial Integration

Many businesses are web-enabling their business processes in a phased manner by doing them one by one instead of doing it all at once. The business reasons are obvious – minimize risk, see early return on investment, and synchronize technology changes with user adoption rates. Hence, the integration hub system needs to deal with integrating partial business data and processes belonging to the component systems.

Consider again the example of the order management hub system from the previous section. Recall that the hub system's canonical model comprises of concepts such as Orders, Invoices, and Shipping as well as the business processes involving these concepts. Now, consider a scenario where a business wants to achieve integration and collaboration involving its invoicing system first and wants to do the same for its shipping system in a later phase. To address this requirement, the hub system needs to represent Invoice data in the canonical model *when the corresponding data for Orders and Shipments is not available*. This causes interesting technical challenges: (1) Data modeling becomes a more complicated task – *e.g.*, foreign key restrictions in the canonical model need to be relaxed. (2) Business logic needs to be made inherently aware that only parts of the data in the data model graph may be genuinely available. (3) With no clear models for specifying unavailability of data in relational model, queries on the data model must be careful not to return incorrect answers because of the partial availability of data. We discuss these issues further in Section 4.2 as well.

3.3 Representing Component System Specific Information In A Canonical Data Model

In section 3.1, we motivated the need for information that is specific to a component system to be represented in the canonical model without necessarily associating any deep data semantics in the model. However, the hub system needs to be able to process such "custom" information in some ways to help towards the goal of process integration. This includes providing the ability to *pass-through* this information in workflows, allowing for simple querying in the business logic, and performing UI manipulations in the presentation layer. Tavant system uses a *Custom Attribute* framework to represent information unique to a component system, that need not be interpreted in the canonical data model.

The Tavant integration hub caters to the needs of many component systems (belonging to several partners) each of which can potentially have a large number of custom concepts. In order to scale easily across a number of such partners, it is impractical to represent the custom concepts as first-class attributes (i.e. columns) in the relational model. In the Tavant custom attribute framework, such pieces of information are stored as $< attr\text{-}name, value >$ pairs in relations. For example, in the case of the order processing code example of Section 3.1, the "Order Code" is not explicitly modeled as an attribute of the Order relation. Instead, the code in the context of a specific order O_1 flowing to the component system C_1 is represented as a tuple $< C_1, O_1$, "$OrderCode$", "$Urgent$"$>$ in the "Order Custom Attribute" relation.

There are several challenges that arise in the context of querying and manipulation of information stored as $< name, value >$ pairs. Lack of support in commercial databases for SchemaSQL-style querying ([LSS01]) makes even simple querying of custom attribute tables a complicated task. For example simple selection queries involving custom concepts need to be expressed as join queries on the custom attribute table. Another challenge is the creation of indexes on the custom attributes for performance reasons. It is not possible to build indexes on custom concepts due to the fact that the information is modeled as data and not schema. There is a need for commercial database products to address these issues.

4 Use and Manipulation of Integrated Data

In this section, we study the various contexts in which integrated data is used in real-life integration systems. We also discuss how integrated data gets further manipulated to provide for "function integration" and other post-integration requirements.

4.1 Integration Hub as a Cache

The Tavant integration hub provides seamless access to information from multiple component systems and offers a single-system view to its users. In most cases, the system hides from the user the fact that the integrated information and processes actually belong to multiple component systems. However, the overhead of pulling information from and orchestrating workflows among a multitude of component systems for every user request can cause slow response time and may not be acceptable from a system performance standpoint.

To address this performance issue, the Tavant integration hub has the ability to cache data locally from the component systems. This is useful especially in cases where the data on the component systems do not change frequently and when the timeliness of the data is not crucial for the end user. In this scenario, the cache acts a "transparent cache" where the user is unaware of the existence of the cache. The cache on the hub is maintained by explicit update requests from the component system (push model) or via an automatic refresh initiated by the hub system on a periodic basis (pull model).

Besides helping with performance, the cache on the integration hub allows for *function integration* by enabling the creation of completely new features or processes that do not exist in the component systems. For example, consider a product catalog from a component system that is cached on the integration hub. Let us say one of the component systems does not have support for an operation such as keyword search, while all other component systems are capable of performing this operation. If the integration hub has support for the keyword search operation, the users of the integration hub can seamlessly perform keyword search over *all* integrated data, including over data belonging to the "keyword search challenged" component system.

The above technique of compensating for the component systems' data and functions at the integration hub is particularly applicable in the context of integrating legacy systems. We revisit this issue in Section 5.2.

The Tavant integration hub also allows for scenarios where data changes occur frequently on the component system and/or timely and accurate information is crucial. In these cases, user requests are answered by pulling data dynamically from the component systems.

4.2 Leveraging Integrated Data to Handle Connectivity Challenges

Query processing in a distributed systems environment is a well-studied subject [OV99]. While many of the ideas are relevant in the context of a data integration setting as well, there are key differences due to the fact that these two environment differ in some fundamental ways.

While the distributed systems environment is tightly coupled, the data integration environment comprises of loosely coupled, autonomous component systems. In particular, the reliability of the component systems as well as reliable communication among them cannot be guaranteed. Thus in practice, the integration system should be able to cope with unavailable component systems and serve useful purpose even when some of the component systems are unavailable.

A technique for addressing the above problem makes use of the integration hub cache discussed in the previous section. This cache can be used to substitute for the component system when the component system is unavailable. This approach is based on the belief that it is better to present a possibly stale result to the user rather than not present any results at all. In these instances, the hub system may want to inform the user about the existence of the cache and warn the user that the data may not be current.

To continue with the example from previous section, let us say the product catalog in the integration hub cache contains pricing information for the various products. Consider a scenario where, while answering a user query regarding product price and availability, the hub fails to obtain the latest pricing information from the component system (say due to connectivity issues). The integration system may still want to present the user with the pricing information from the cache along with an indication to the user that the information may not be accurate.

In scenarios where the cache substitution technique may not be applicable (because the component system information is not cached or because security considerations prohibit the availability of pieces of information required for query processing), techniques of query relaxation ([BT98, GGM92]) can be applied to answer queries. The challenges here are that the hub system should be able to deal with missing and/or approximate information (at both the data and schema level) every step of the way till the results are presented to the user.

The hub's data integration model needs to be cognizant of and be able to deal with unavailable data and schema. Lack of proper models in commercial systems for dealing with unavailable or approximate information makes this a challenging task.

It is important to note that the above techniques for dealing with unavailable component systems may not be applicable in the context of *all* integration scenarios. The integration system should allow for declaratively specifying the scenarios where the cached information from the integration hub can be substituted and/or query relaxation techniques can be applied. The system should make use of these specifications to apply the appropriate technique for a given scenario. Also the integration system should allow for an option where the end user is made aware of the fact that in order to generate the results to her query, one or more of these scenarios have been applied.

4.3 Data Semantics as a Function of User Expectation

This section discusses yet another scenario that leverages the existence of the integration hub cache. It exploits the fact that in real-life, semantics is context sensitive. For example, when a user is browsing the product catalog, it can be acceptable to present a possibly stale and hence approximate price information. But when the user is placing an order for the same item, it is important that the user is presented the current accurate price of that item. A smart integration system can take advantage of these semantical distinctions that arise in real-life. For example, in the Tavant integration system, the catalog prices are cached on the hub on a nightly basis. Thus, prices for items can be more efficiently obtained for the more frequent catalog browsing activity. For the less frequent ordering activity, current prices are obtained from the component system using real-time webservice queries.

4.4 Integration and OLAP

Integration allows for related data in multiple, disparate systems and data formats to be accessible via one common data model. Such an integrated model naturally serves as a basis for business analysis and reporting. While integration facilitates business analysis, in practice there are several differences between an online transaction processing (OLTP) based integration system and an online analytical processing (OLAP) system ([KRRT98]). The integration system should be cognizant of this distinction. It should facilitate OLTP queries across component systems, while allowing for building an efficient OLAP system that leverages the integration model as much as possible. Better still, the integration system should be able to infer the optimal model for the OLAP system based on the business analytics workload.

Currently, this problem is solved in practice by creating an explicit datawarehousing server that is distinct from the integration server. The datawarehousing server is managed by data replication (often involving data transformations) from component systems as well as the integration hub. Finally, for seamless OLTP and OLAP querying via the integration server, the datawarehousing system is added as a component system in the integrated environment.

5 Challenges and Opportunities

In this section we identify several challenges that arise in real-life integration scenarios that require further research. We also outline some practical approaches that help address these challenges in the absence of elegant solutions.

5.1 Fault Tolerance and Error Recovery

Fault tolerance and error recovery in the context of a transaction that spans multiple distributed systems where each component system is transactional is a well-studied subject ([OV99]). However, work flow integration in our context involves performing transactions over loosely coupled component systems that do not necessarily support transactional semantics. Currently, no models exist to handle this scenario. Thus, when a component system involved in a workflow encounters a failure (unexpected errors or server crashes), it is very difficult to even detect what happened during the accesses to non-transactional resources, let alone recovering from the errors. Offering very strict transactional semantics to the entire application layer may be too prohibitive and even impossible in most cases. However, it would be useful to have models that offer a range of capabilities from making it possible to detect what happened, to even recovering from the error situations.

The above mentioned scenarios arise frequently in Webservice calls where an application communicates with a partner system over the Web. Note that the interfaces between these systems are typically not transactional. For example, let us say the user initiated an action on system A that causes an order to be submitted (as a webservice request) from system A to system B. Let the order be accepted by system B. However, before system B could respond to system A acknowledging the successful completion of the order request, let us say that system B crashes (in the course of some post-processing task on system B) that prevents the successful delivery of the acknowledgement to system A. In this scenario, the user should ideally be informed of the error that happened in system B *as well as* the fact that the order was successfully placed before the failure occurred. To program the application to remember this fact in case of any error that happens is a very difficult task. Due to the lack of proper models and transaction systems to deal with these situations, Tavant uses an application programming model where the whole sequence of operations on system A can be defined as a workflow involving a sequence of transactional operations. In such a model, the information about any completed transaction will be automatically stored persistently by the engine executing the workflow. In addition, to deal with these scenarios in a practical way, the Tavant system also keeps the user informed about such situations. For example, in the above scenario, system A would inform the user that an error occurred while system B was processing the order, the order *may* have been successfully processed, and that the user should check later to see if the order has indeed been accepted by system B.

5.2 Legacy Systems Integration

Webservices is increasingly being touted as the technology for dealing with legacy systems integration. In theory, Webservices promise increased interoperability and lower cost of application integration and data sharing regardless of operating systems, languages, or hardware platform. This is achieved by the availability of tools, techniques, and standards to easily "wrap" existing applications and enable them to communicate among each other. However, for reasons that include lack of tools in a variety of platforms, scarcity of skilled personnel, strict adherence to established standards and procedures, and businesses' inherent resistance to change, there exist legacy systems that cannot be webservice enabled and only support legacy protocols. Towards this end, The Tavant system supports a variety of legacy protocols including FTP, EDI, and flat file upload.

Several challenges arise even in scenarios where webservice enabling of legacy systems is possible. The legacy system's participation in the integration environment will naturally lead to increased workload on the system and greater demand for system resources. However, the application may not have been built in anticipation of the new workload. For example, consider a PC-based application in a corporate office that provides information on field inventory for select corporate users. Let us say this application needs to be webservice enabled so that dealers and distributors in the sales channel will have access to the global inventory information. Simply wrapping the application as a webservice is not sufficient to address this need because the PC-based application does not have the wherewithal to handle the huge volume of transactions initiated by the channel partners.

Scaling a webservice enabled legacy application to handle increased workload poses interesting challenges. In certain cases, clustering techniques can be used to address this issue. For example, the PC-based application of the above example can be scaled by means of a cluster of PCs running the application and a load balancer that handles transaction requests. However this assumes that the application is stateless – i.e. it processes each transaction request without any knowledge of previous requests. For the cases where this assumption is not valid, a technique used in practice is to cache both data and functions (or function results) on the integration hub. Thus, in these scenarios, the hub provides for function integration in addition to data and process integration.

5.3 Streaming Data – Query Processing Issues

Integrating data from multiple systems frequently involves several data transformations as the data flows through various levels of protocols among the different systems. When dealing with large data, the ability to stream data as it undergoes transformations (without the need for temporarily materializing the data) is a crucial factor in improving performance. While this concept of "pipelining" is well understood and implemented within database systems, many systems and protocols related to data transformation at the middleware or application level do not lay emphasis on transformations on streams and deal only with fully materialized representations of the data they need to convert. Important examples include commercially available tools for conversion between EDI and XML, and tools for conversion between XML and Java. While the XML SAX processor has the ability to deal with XML data streams, parsing at the SAX level has several overheads that are cumbersome for application development. The challenges in this area require more work at a level of academic research as well as commercial products.

6 Conclusion

In this paper, we discussed the practically useful problem of integrating disparate business data and processes and making them available on the Web. We presented the Tavant integration framework, discussed several challenges that arise in real-life integration scenarios, and our approach to addressing them. In particular, we discussed the nature of integration needs in real-life, issues related to representation, and the use and manipulation of integrated data to solve practical problems. We also identified several challenges that do not currently have elegant solutions and hence provide opportunities for further research.

References

[BT98] Philippe Bonnet and Anthony Tomasic. Partial answers for unavailable data sources. *Lecture Notes in Computer Science*, 1495, 1998.

[GGM92] Terry Gaasterland, Parke Godfrey, and Jack Minker. Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1(3/4):293–321, 1992.

[KRRT98] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses.* John Wiley & Sons, 1998.

[LSS01] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. SchemaSQL: An extension to SQL for multidatabase interoperability. *ACM Transactions on Database Systems*, 26(4):476–519, 2001.

[NST01] Srinivasa Narayanan, Subbu N. Subramanian, and The Tavant Team. Tavant system architecture for sell-side channel management. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, Roma, Italy*. Morgan Kaufmann, 2001.

[OV99] Ozsu, M.T. and Valduriez, P. *Principles of Distributed Database Systems.* Prentice Hall, 1999.

Data Integration in Vector (Vertically Partitioned) Databases

Peter Buneman Jon G. Riecke Eric Sandler Vladimir Seroff
University of Edinburgh * Aleri Inc. †

Abstract

Data integration requires not only the absorption of data, but the transformation of that data. For instance, manufacturing companies must combine customer order information for ordering supplies as well as for financial planning; large banks may need to produce consolidated profit-and-loss statements daily, or even more frequently, to manage liquidity and risk. The data may be spread across many heterogeneous databases located in different countries, with different standards of cleanliness, described in different currencies or different units of measure. The transformation rules may also change over time when, for instance, companies merge or split, or new rules of accounting are imposed. Clearly it is best if the data integration tool is flexible enough not only to accomodate new sources of data, but also to change the rules of combining that data. This paper describes one data integration tool, Aleri Inc.'s Modeler. We focus primarily on one aspect of Modeler: the use of vectorization both as an implementation technique and as the fundamental unit of computation. Vectorization improves both the efficiency and the ease-of-use of Modeler.

1 Vectorization

1.1 Vertical Partitioning

The widely accepted storage technique for relational databases is to store each tuple contiguously. An alternative, which is almost as old as relational databases [4], is *vertical partitioning*, in which one stores the columns contiguously. The benefit is that queries that only involve a small number of columns require less i/o. Moreover, there are dramatic performance improvements to be made if the columns can compactly represented and manipulated as vectors in main memory.

The idea of vertical partitioning has re-emerged for implementing object-oriented databases [3, 2] and to speed up transfer between main memory caches and disk in relational databases [1]. It has also been used commercially in Sybase IQ and recently by Aleri Inc. where it is combined with vector processing language technology and has been successfully deployed in a number of financial database applications to support data integration and OLAP. We shall use the term "vectorization" for that form of vertical partitioning which stores the columns of a table as vectors and, in particular, preserves the *order* in each column, which is an added bonus for applications in which the vertical order of a table is needed; for example the table may represent a time-series

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*http://www.informatics.ed.ac.uk/people/staff/Peter_Buneman.html
†http://www.Aleri.com

thereby departing from the relational (SQL) model in which a table is defined as a *set* (*multiset*) of tuples. There are, of course, a few drawbacks to vectorization. Row deletion is prohibitively expensive if naively implemented. Also, there is a sense in which vertical partitioning is used in conventional database implementations: an index typically stores information about one column or a small subset of columns contiguously, and it is occasionally possible to answer a query on the basis of an index alone.

1.2 Vector Languages

The language APL [5] is based on a number of array manipulation primitives. It shares with relational algebra the principle that languages based on "bulk" operations are more concise and lend themselves to optimization both through efficient implementation of those operations and through code rewriting.

VCL, the main implementation language used by Aleri Inc., is one of a number of derivatives of APL in use today. Apart from the superficial, but useful distinction that VCL uses a conventional character set, VCL is a higher-order language in the sense that functions are first-class values, and functions can take other functions as arguments and return other functions as results.

For readers who have not seen APL or its derivatives, it may help to give the flavor of VCL through a few examples on vectors (one-dimensional arrays). The operator "+" is overloaded to add a scalar to an array:

$$5 + (\,4\;7\;6\,) \models\; 9\;12\;11$$

(we shall use the symbol $\models$ to mean "evaluates to"). This operator is further overloaded to work pointwise on vectors of equal length:

$$(\,3\;2\;1\,) + (\,4\;7\;6\,) \models\; 7\;9\;7$$

Indexing is also overloaded in that one can index one vector by another to produce a vector:

$$(\,5\;7\;2\;4\;1\,)[\,3\;0\;4\,] \models\; 4\;5\;1$$

An operation that produces an indexing vector from a vector is "&" which operates on a boolean (0 or 1) vector to produce the positions in which that vector is 1

$$\&(\,0\;0\;1\;0\;1\;1\,) \models\; 2\;4\;5$$

Putting these operations together, suppose we have a relational table R with attributes A and B and we wish to evaluate a simple select-project query SELECT A FROM R WHERE B = 123. In a vector representation of this table we label the columns R.A and R.B and evaluate the expression

$$\mathrm{R.A}[\,\&(\mathrm{R.B} = 123)] \tag{a}$$

To explain what is happening here, the boolean operator "=" is, like "+", overloaded to produce a boolean vector. From this an indexing vector is produced by &, which is used to index into R.A. Note that, like SQL, this operation does not eliminate duplicates, but there is a further VCL operation that produces unique values within a vector.

1.3 Using Vectorization to Implement Relational Databases

Example (a) illustrates the overwelming advantage of vectorization, namely that one only needs to "read" the vectors R.A and R.B in order to evaluate the query. Contrast this with the conventional implementation of a relational database system in which (unless one could capitalize on information stored in an index) one would read all the tuples. This, taken with the fact that a relational database system typically does not fill disk pages to capacity, leads to a dramatic reduction in i/o.

At the same time, we do not want to take the VCL evaluation of a query literally. Example (a) appears to call for the creation of two intermediate vectors, but this is clearly not neccessary. Another case in point is join evaluation. If one represents columns of a table as vectors, one is tempted to think that replacing a foreign key by an index into the table referenced by the foreign key will speed up a join. However consider the problem of evaluating $A[V]$ where both A and V are too large for main memory. The naive evaluation requires approximately $|V|$ reads of A; but we know we can do much better using a hash or sorted join. Such a join may be useful even in main memory evaluation where it could be used to reduce transfer into caches. The point to be made is that use of *both* vector processing techniques and database technology is needed for implementation of vectorized databases.

In addition to implementing a query engine for vectorized databases, Aleri has developed (a) transaction processing software (b) a declarative system, Modeler, for data integration and (c) efficient view maintenance technology that depends both on the vector organization of the database and on (a) and (b). In this paper we shall focus on (b) and remark briefly on (c).

2 Data Integration using Modeler

Modeler is a declarative language for data integration, view/query definition and OLAP. It is based on the use of rules and structural assertions that guide the formation of views. In the following account we use a surface syntax that is somewhat different than that used in Modeler, which is partly controlled by a graphical interface. Let us start with a simple example, in which we are given the source tables:

order:

oid*	cid	date
17	Jones	07/21/01
18	Smith	09/22/01

olist:

oid*	pid*	qty	priceq
17	A4	100	145.00
17	B7	15	32.15
18	A4	10	16.00

The order table gives order details: customer and date; the olist table gives the parts associated with each order, the quantity, and the quoted price for that order. The keys for each table are flagged with an asterisk; oid and pid form a joint key for the olist table.

The simplest form of "rule" is based on one table:

olist.unitprice := olist.priceq / olist.qty

This provides the price per part associated with each order by augmenting the olist table with a new column:

olist:

oid*	pid*	qty	priceq	*unitprice*
17	A4	100	195.00	1.95
17	B7	15	32.15	2.14
18	A4	10	16.00	1.60

We have italicized the unitprice header to indicate that it is a derived column. This operation is unconventional in two ways:

- Because of the vectorized organization of the database, no internal reorganization is required, and the computation only requires access to the priceq and qty columns.
- This is more than a conventional SQL "add column" operation. Updates to either the priceq or qty columns will be immediately reflected in the derived unitprice column. In effect we are changing the olist table so that part of it is the source table and part is a materalized view.

One of the most important features of vectorized databases is the ability to do efficient and general materialized view maintenance. By contrast the construction of materialized views in conventional database systems is subject to a host of restrictions, the most serious being that they cannot be composed. In Modeler, even though one can declare new tables and think of a view as a new table, the fundamental operation is creating a new column or vector.

Continuing with our example, suppose we want to extract information from the combined tables. Our first task is to say how the tables are to be combined. This is performed by a *connecting* rule:

connect(olist.oid = order.oid)

Although this looks like a join instruction, its function is not to perform an immediate join but to connect the tables for the purpose of deriving further data – by means of a join – at a later stage. For example:

olist.date	:=	order.date
order.total	:=	sum(olist.priceq)

The first of these, rather trivially, adds date information to the olist table; the second performs an aggregation. The key structure of the two tables and the form of connect declaration tells us that there is a *many-one* relationship between the two tables. Thus aggregation is appropriate in one direction but not in the other.

We have used two kinds of rules in this process: *derivation rules* which generate columns and *connecting rules* which describe how tables are to be connected. Either kind of rule may be accompanied by a condition such as olist.date $>$ 08/17/02 which, in the case of a connecting rule, selects a subset of the data for consideration.

So far our examples have only added data to existing columns. In general we shall want to construct new tables which, in conventional database parlance are *materialized views*. To do this we first declare the structure of the new table:

part(pid*, total)

Any table declaration must include a key field. We also make the connecting declaration connect(part.pid = olist.pid). This table may now be populated by the derivation rules:

part.pid	:=	olist.pid
part.total	:=	sum(olist.qty)

This generates the derived table

part:

*pid**	*total*
A4	110
B7	15

Of these derivation rules, the derivation of the key, pid is pivotal. It is evaluated prior to any other rule for columns in the part table. Moreover, it generates a set of distinct values, because pid is a key. The non-key columns in the table can be constructed by the process outlined above. In fact, the only thing we need to do is to establish a new table is to provide a key and a derivation rule for that key. We can continue to define further columns independently, e.g.

part.aveprice	:=	sum(olist.qty $*$ olist.priceq) / part.total
part.delay	:=	min(order.date)

Note that the first of these rules uses columns from different tables, and the second derives data via two connecting rules.

At this point we should discuss under what circumstances a set of rules is meaningful. Suppose the connecting rule is connect$(a_1 = a_2)$ where a_1, a_2 are attributes of R_1, R_2 respectively. If a_1 and a_2 are both keys, the connection induces a *one-one* relationship between R_1 and R_2; if one of them is a key, the relationship is *one-many*; and if neither is a key the relationship is *many-many*. This analysis can be extended, in certain cases, to more general connecting rules of the form connect$(e_1 = e_2)$ where e_1 and e_2 are respectively expressions on the attributes of R_1 and R_2. For example we can connect on the basis of multiple attributes, connect$((a_1, b_1) = (a_2, b_2))$. It is also possible to extend connecting assertions to certain other expressions, such as connect(r.agegroup = trunc(s.age / 10)).

From this analysis we obtain a graph, whose nodes are the tables in the database and whose edges are marked to indicate the kind of relationship (many-one, one-one, etc) to indicate the kind of relatyionship. We can reasonably refer to this graph as the *schema*, for it has a close connection to the entity-relationship diagram that one might design in order to describe this integration process. Our first requirement is that the schema, viewed as an undirected graph, should be a *tree*. This guarantees that there is at most one path between any two

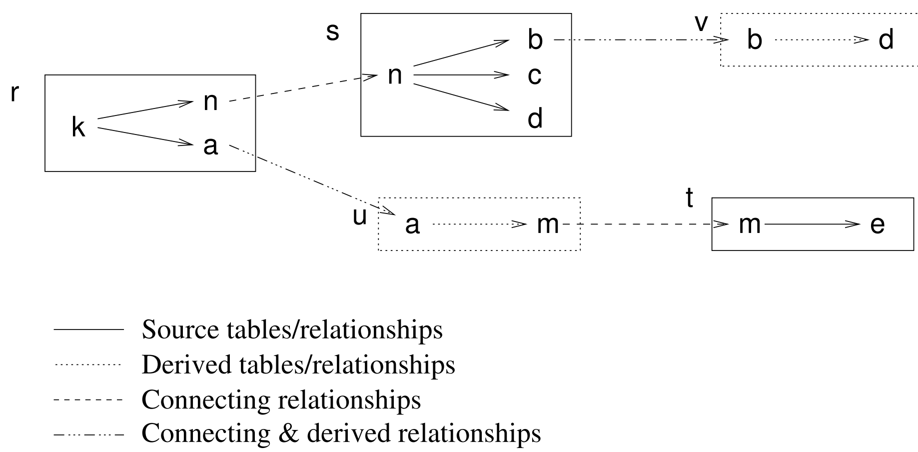


Figure 1: A schema graph generated by rules

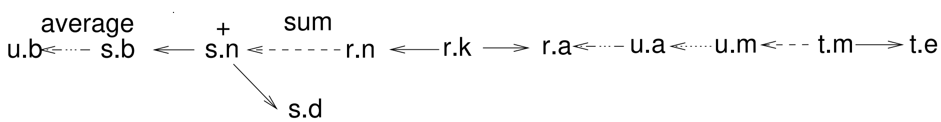


Figure 2: The tree used in the computation of v.d

attributes, which is needed to ensure that the derivation of one attribute from other attributes is unambiguous.

Our second requirement has to do with the derivation rules. It ensures that there is no circularity in the derivation of attributes and that there are enough rules to ensure that every attribute gets derived – directly or indirectly – from source data. The schema is also used in the compilation of rules into executable code. The details are not given here, but the following example illustrates the issues involved.

Source tables:	r(k*,n,a), s(n*,b,c,d), t(m*,e)
Derived tables:	u(a*,m), v(b*,d)
Connecting rules:	connect(r.n=s.n), connect(t.m=u.m), connect(r.a=u.a), connect(s.b=v.b)
Derivation rules:	u.a:=r.a, u.m:=min(s.c), v.b:=s.b, v.d:=average(s.d + sum(t.e))

Figure 1 shows the schema that is derived from this set of rules. Each box describes one table, the arrows within each box show the many-one relationship between the key and the other attributes. A corresponding entity-relationship (ER) diagram can be derived from this figure simply by replacing each edge between tables with an ER relationship node and creating the obvious ER attribute nodes.

Figure 2 shows the tree of dependencies used in the computation of the column v.d. Computing this column is tantamount to computing the values of a function on the key v.b. The details are not given here, but to sketch what happens: First, the expressions s.d and sum(t.e) "converge" at s.n. The arrows between s.n and t.e give rise to a many-many relationship between these columns, and with a group-by and sum one can obtain a further derived column of r. Now add this column to s.d to obtain another derived column of r. Between this column and r.b we have a many-many relationship and again we can perform a group-by and an average.

It is also worth observing that, if the purpose of the rules is to derive the v table, there is an equivalent set of rules (corresponding to an equivalent ER diagram) in which the u table has been replaced by an additional column in the r table.

Source tables: r(k*,n,a), s(n*,b,c,d), t(m*,e)
Derived tables: v(b*,d)
Connecting rules: connect(r.n=s.n), connect(t.m=r.m), connect(s.b=v.b)
Derivation rules: r.m:=min(s.c), v.b:=s.b, v.d:=average(s.d + sum(t.e))

The ability to perform unions is an important part of data integration. In Modeler this is achieved (as in datalog) by providing two derivation rules for keys. Continuing our parts/orders example, suppose there is an additional base table supplies(supplier*,pid*,unitprice) we may add the derivation rule for part:

part.pid := supplies.pid

Other attributes in the derived part table must be changed appropriately, e.g.:

part.total := ifnull(olist.pid) then 0 else sum(olist.qty)

This also illustrates the use of a conditional in a derivation rule. Using conditions and aggregates, it is possible to express set membership, hence negation and subtraction. This enables us to express all operations of the relational algebra.

3 View maintenance in Modeler

We noted before that Modeler does more than "one-shot" data integration; the derived data is updated in response to updates to source tables. This is materialized view maintenance in conventional databases, but in part because of the vector representation, views can be more efficiently maintained, and they are fully compositional.

There are two reasons for the gain in efficiency. The first derives from the fact that the queries involved, even complex aggregate queries, typically involve only a small number of source columns, therefore – as we have already noted – recomputation of queries is inherently more efficient. The second reason is more subtle. Recall that certain updates such as deletion are expensive in vectorized databases, but if we are in an environment in which the majority of updates are modifications we can again capitalize on the vectorized representation. Modifications to a table are not random: in most situations one or two columns in a table are highly volatile, while the remaining columns will remain relatively stable. Thus we can tune the "refresh" algorithms to try to keep memory-resident those vectors that derive from volatile columns.

At the core of the view-maintenance algorithm is an "indices table" that keeps track of what source attribute values can affect derived attribute values. Consider an update to s.d in the example above. If the sum(t.e) vector is materialized, then we need to do one addition and recompute a new average in order to maintain the view v. The indices table keeps track of which elements are needed in the recomputation of the average. Note how, once again, vectorization helps in this process. View maintenance requires the computation of a few vectors rather than updates to a number of tables.

4 Conclusions

We have attempted to describe how a vectorized representation of relational databases is useful in data integration. It is useful at the conceptual level because the fundamental unit of construction is the column; it is useful in implementation because not only the initial computation of derived data, but also refreshing that data in response to modification of the source – materialized view maintenance – is much more efficient.

In basing the declarative formulation of views on relationships (many-one, one-one, etc.) derived from keys, Modeler is close in spirit to the Clio project [6] at IBM. The idea of using an tree-like structure to ensure unique derivation paths without specifying the paths bears some relationship to the acyclic hypergraph schemes of the universal relation assumption [7] – though in that model there was no method of deriving new columns.

Compared with other methods of data integration that exploit high-level knowledge representation formalisms, Modeler may appear somewhat low-level. Nevertheless it is simple, efficient and may serve as supporting technology for a higher-level formalism, should one ever become popular.

Acknowledgements

We are grateful to Renée Miller for her comments on a draft of this paper.

References

[1] A. Ailamaki, D.J. DeWitt, M.D. Hill, and M. Skounakis. Weaving Relations for Cache Performance. In *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001.

[2] Peter A. Boncz, Annita N. Wilschut, and Martin L. Kersten. Flattening an Object Algebra to Provide Performance. In *International Conference on Data Engineering*, pages 568–577, 1998.

[3] George P. Copeland and Setrag Khoshafian. A Decomposition Storage Model. In Shamkant B. Navathe, editor, *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 28-31, 1985*, pages 268–279. ACM Press, 1985.

[4] Don S. Batory. On Searching Transposed Files. *ACM Transactions on Database Systems*, 4(4):531–544, 1979.

[5] Kenneth E. Iverson. *A Programming Language*. John Wiley & Sons, 1962.

[6] Renée J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 77–88, Cairo, Egypt, Sept 2000.

[7] Jeffrey D. Ullman. On Kent's "Consequences of Assuming a Universal Relation". *ACM Transactions on Database Systems*, 8(4):637–643, 1983.

Creating a Mediated Schema Based on Initial Correspondences

Rachel A. Pottinger
University of Washington
Seattle, WA, 98195
rap@cs.washington.edu

Philip A. Bernstein
Microsoft Research
Redmond, WA 98052-6399
philbe@microsoft.com

1 Introduction

Video Place, a company that sells videos through its website, has entered a partnership with Movie Reviews, a website that lists movie facts and reviews. Video Place and Movie Reviews wish to set up a data integration system to allow both databases to be accessed at once. Much schema level work is required to do this: a mediated schema must be developed, mappings must be created between the source schemas and the mediated schema, and a mechanism must be provided for translating queries over the mediated schema into queries over source schemas. In addition, if Video Place and Movie Reviews want to change their previous queries to retrieve data from both sources, queries must be translated from the source schemas to the mediated schema.

First, we must create the mediated schema. We can reduce the problem of creating the mediated schema to three tasks: (1) creating a mapping between the source schemas by matching the two schemas to determine where they overlap, (2) merging the two schemas to create a data-model-independent mediated schema, and (3) applying a translation operator to transform the schema into a specific data model (e.g., relational).

Consider the first task of creating a mapping between the two schemas [DMDH02, MHH00, MBR01, RB01]. Suppose each database's actor information is structured as an XML-tree-like representation where each edge is a sub-element relationship, as shown in Figure 1. In this example, the matching task is trivial since elements that correspond to each another have the same name. However, even with such a perfect mapping, differences can remain. For example, the Video Place database (VPDB) represents an actor's name by a single element while the Movie Reviews database (MRDB) represents it by two elements, FirstName and LastName.

Given this difference, should the mediated schema represent Name as one element, two elements (FirstName and LastName), three elements (Name with FirstName and LastName as children), or some other way? The answer affects the behavior of Merge, the amount of guidance about the merge result that should be encoded in the mapping, and the semantics of the mapping needed between the source schemas and mediated schema. The first two issues are largely data-model-independent. Often the third is too, as in our example where the semantics can be expressed by concatenation, an operator present in most view definition languages. Yet most work on merging schemas concentrates on performing the task in a data-model-specific way, e.g., for XML [BM99], or relational and object-oriented databases [LNE89, BC86]. Our goal here is to show how to abstract out the data-model-independent part of the task so it can be reused for any data model. The data-model-dependent part can be performed as a separate subsequent step.

Section 2 describes how to merge two source schemas to create a mediated schema, given an initial mapping between the source schemas. Section 3 describes the kinds of conflicts encountered in creating the mediated

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

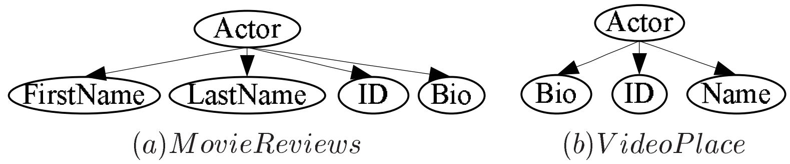


Figure 1: The actor information schema from (a) MRDB and (b) VPDB stored in a generic graph representation

schema. Section 4 describes issues that occur when the semantics of a specific data model is desired. Section 5 describes model management, a data-model-independent approach to a broader collection of schema-related problems similar to the one discussed here. Section 6 concludes.

2 Creating the Mediated Schema

Our goal is to design a generic (i.e., data-model-independent) merge operator that, given two schemas and a mapping between them, returns a merged (i.e., mediated) schema that describes all of the information in the input schemas. To be generic, Merge must use a schema representation that is rich enough to encode features of most commonly used data models. We refer to a schema in this data-model-independent representation as a *model*. Due to space limitations, we'll use an oversimplified representation consisting of elements, properties, and relationships. An *element* is a first-class object that can have single-valued scalar *properties*, including name and (unique) ID, and *relationships* that connect it to other elements. There are two kinds of relationships: Contains that connects an element to its components, and Type-Of that connects an element to its type (which is an element). A model is represented by a set of elements that are connected by Contains and Type-Of relationships. A *meta-model* is a set of elements that represents the types in a given data model, e.g., for SQL schemas or ODMG schemas [CBB$^+$00]. Usually all elements of a model have types taken from one meta-model (e.g., a relational schema). For simplicity, types are omitted from the figures. The *meta-meta-model* is the representation in which models and meta-models are expressed; in our case it consists of the elements, properties, and relationships described above.

The mapping that is input to Merge is a model some of whose elements are *mapping elements*, each of which tells how some elements in the two source schemas are related to each other. A mapping element has:

- *Mapping relationships* (a third kind of relationship) to corresponding elements of the source schemas.
- Figure 2 specifies a potential mapping between MRDB and VPDB where all mapping elements have HowRelated = "equal". If corresponding source elements are tagged as equal, then Merge will collapse them into a single element in the merged schema. By contrast, a different mapping between MRDB and VPDB might indicate that Name corresponds to (but is not equal to) FirstName and LastName. This would be represented by replacing elements 1, 2, and 3 in Figure 2 by a single mapping element with HowRelated = "similar" and with mapping relationships to Name, FirstName and LastName. Merge preserves each similarity mapping element along with the elements it relates to, so that later operations that understand the semantics of the similarity can resolve them.
- An optional property, called Expression, that provides semantics for the correspondence. For example, Expression might say that Name equals FirstName concatenated with LastName. An expression may be meta-model specific. For example, it might be expressed as a conjunctive query.

The mapping need not (and usually does not) refer to all elements of the source models. As well, mappings between elements in the models do not have to be one-to-one.

Reifying a mapping as a model allows us to represent more complex relationships between the input schemas than could be represented by direct correspondences between the input schemas. For example, the mapping in

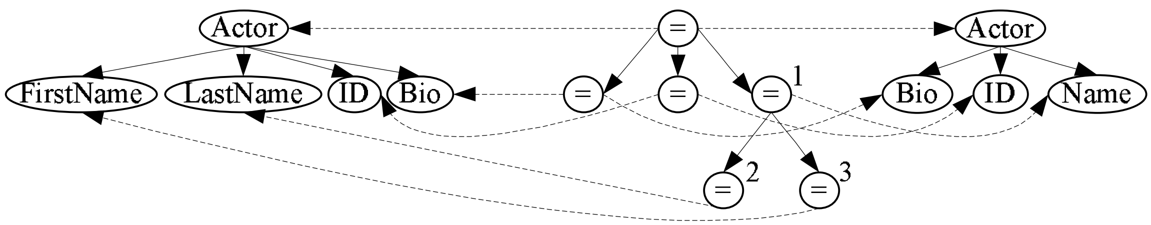


Figure 2: This mapping relates the actor information in MRDB to the actor information in the VPDB. The dashed lines represent mapping relationships

Figure 2 shows that FirstName and LastName in MRDB are effectively sub-elements of Name in VPDB. This is stylistically similar to mappings in [CL93, MBDH02], where a mapping is a logical expression over elements in the two source schemas. We encapsulate such logical expressions in a mapping element's Expression property.

The merge operator produces a mediated schema with the following properties:

- Elements in the two source schemas that are declared to be equal by the mapping are collapsed into one element.
- Elements in the two source schemas that are declared to be similar by the mapping are retained as distinct elements in the mediated schema, along with a description of the similarity relationship between them. We do not specify the detailed semantics of the similarity relationship since that is data-model-dependent.
- Elements in a source schema that are not related to the mapping are retained as distinct elements in the mediated schema.
- For each relationship between two elements in a source schema, if the two elements remain distinct in the mediated schema, then the relationship is copied to the mediated schema
- Each element in the mediated schema includes the disjoint union of all the properties of the input elements from which it was derived.
- No additional elements or relationships are in the result beyond those implied by the previous five bullets unless needed to satisfy constraints (see Section 3).

For example, using the mapping in Figure 2, Merge would return the mediated schema shown in Figure 3.

Merge also returns a mapping between the mediated schema and each source schema. Each mapping relates each element in the mediated schema to the source schema elements that correspond to it. These mappings are not shown in the figure.

3 Conflict Resolution

A schema returned by the previously described merge operator may be semantically flawed. In particular, the schema may be internally inconsistent (i.e., no non-empty database satisfies its constraints) or ambiguous (e.g., incompatibilities between the source schemas have not been reconciled). Most published schema merging algorithms address such flaws by so-called conflict resolution rules.

We classify these flaws into three kinds of schema conflicts: representation (model) conflicts, meta-model conflicts, and fundamental (meta-meta-model) conflicts. Some of them are meta-model-specific and are therefore not appropriate to solve in a generic Merge. Others are generic. We describe here the different classes of conflicts and where they should be resolved.

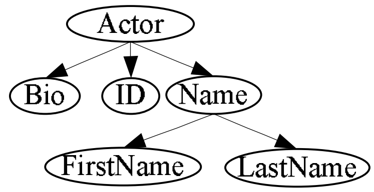


Figure 3: The result of merging the source schemas using the mapping in Figure 2.

Representation Conflicts A representation conflict arises when two source schemas describe the same concept in different ways. Our running example has one such representation conflict: an actor's name is represented by two elements in MRDB (FirstName and LastName) but by one element in VPDB (Name).

We solve a representation conflict by requiring a mapping between the conflicting representations. The mapping describes how the elements in one representation are related to those of the other. This explicitly determines the representation in the mediated schema, including how many elements must be present in the mediated schema and their relationship to one another. If the mapping specifies that the conflicting representations are equal, then Merge can simply collapse the representations into a single element that includes all relationships incident to the elements in the source schemas. If the mapping says that the conflicting representations are similar, then the mediated schema retains the correspondence. In either case, Merge resolves the representation conflict simply by following the course that the mapping has laid out.

For example, the mapping in Figure 2 specifies one possible resolution to the name conflict. The result of that resolution is shown in Figure 3. In this case Name has FirstName and LastName as sub-elements.

Meta-model Conflicts Meta-model conflicts occur when the merge result violates schema constraints specific to a given meta-model (e.g., SQL). For example, suppose that MRDB is a relational database, VPDB is an XML database, and we want the mediated schema to be relational. If we model Actor as a table and use the mapping in Figure 2 to obtain the result in Figure 3, there will be a meta-model conflict because SQL (a particular meta-model) has no concept of sub-column.

By definition, meta-model conflicts must be solved with respect to a specific meta-model, i.e., data model. That is, they are inherently non-generic, so we leave them to be solved after the generic Merge. This adds another requirement to Merge: It must retain enough information in the merge result to enable a post-processing step to repair meta-model conflicts.

Fundamental Conflicts A fundamental conflict arises when the merge result is not a well-formed schema according to the rules of the meta-meta-model. One example is that the meta-meta-model may require an element to have at most one type. So an element with two types manifests a fundamental conflict. For example, VPDB might say Actor ID is of type integer while MRDB says it is of type string. According to the properties of Merge, Actor ID should be both of type date and string, which is a fundamental conflict.

Since Merge must return a well-formed instance of the meta-meta-model, it must resolve fundamental conflicts. Resolution rules for some fundamental conflicts have been proposed, such as [BDK92] for the above one-type restriction. We have identified some other types of fundamental conflicts and resolution rules for them. We incorporate all of these rules into our generic Merge.

4 Semantics

While creating the generic mediated schema can be largely done without considering semantics, semantic issues are unavoidable to make the mediated schema useful. One example is enforcing constraints by resolving

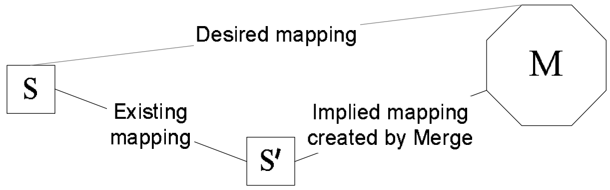


Figure 4: The desired mapping can be created by composing the previously existing mapping between S and S' and the implied mapping between M and S'

representation and meta-model conflicts, as discussed in Section 3.

We must be able to translate user queries posed over the mediated schema into queries over the source schemas. To do so requires semantic mappings that describe how the mediated schema is populated with tuples based on tuples in the source relations. Merge only provides generic mappings from the mediated schema back to the source schemas. While these mappings lack the necessary semantics, they constrain the problem of determining the semantic mappings.

Clio [MHH00] allows users to create semantic mappings based on *value correspondences* which provide information that items are related. Using the additional structure that our mappings offer over their value correspondences, we suspect that some of the mapping semantics can be generated automatically. We plan to explore this, particularly how to exploit the Expression properties of the mapping elements of the mapping that is input to Merge.

5 Model Management and Data Integration

The merge operator described here addresses only one of the schema manipulation problems that arise in data integration. In [BHP00], we proposed several other generic schema manipulation operators as part of a general-purpose model management system. (There we exclusively use the term model instead of schema to emphasize it genericity.) The other main operators are:

- Match - create a mapping between two given models
- Apply - apply a function to all of the elements in a given model
- Compose - given a mapping between models A and B and a mapping between models B and C, compose them to return a mapping between A and C
- Difference - given two models and a mapping between them, return a model consisting of all the items in the first model that are not mapped to the second model.

These operators can help us solve other data integration problems. For example, Apply can be used to encapsulate resolution rules for meta-model conflicts and, therefore, to do the necessary post-processing on the result of Merge. As another example, suppose a mediated schema M was constructed by a series of merges, and we want to add another source S. First, we use Match to create a mapping between S and M, followed by human review to polish the mapping. Then we use Merge to return a schema that subsumes S and M.

Suppose that instead of using Match to create a new mapping between M and S we can use a previously existing mapping between S and a source S' that has already been incorporated into M, as shown in Figure 4. We can compose the S–S' mapping with the S'–M mapping that was returned by Merge, to create a mapping between M and S.

The mapping returned by Merge between a source schema S and a mediated schema M can also be used to identify which elements to remove from M should S leave. The difference operator can tell which elements of M correspond to those in S. Such elements that are not in the range of a mapping between M and some other data source should be removed.

6 Summary

We presented a generic merge operator for creating a mediated schema, discussed various conflicts that can arise, and showed how we solve them in generic model management framework.

Acknowledgements

We thank Alon Halevy and Renée Miller for many helpful comments throughout this ongoing research.

References

[BC86] J. Biskup and B. Convent. A formal view integration method. In *SIGMOD*, pages 398–407, 1986.

[BDK92] Peter Buneman, Susan B. Davidson, and Anthony Kosky. Theoretical aspects of schema merging. In *EDBT*, pages 152–167, 1992.

[BHP00] Philip A. Bernstein, Alon Y. Halevy, and Rachel A. Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

[BM99] C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In *ICDT*, pages 296–313, 1999.

[CBB+00] R. G. G. Cattell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, and Fernando Velez, editors. *The Object Database Standard: ODMG 3.0*. Morgan Kaufmann Publishers, 2000.

[CL93] Tiziana Catarci and Maurizio Lenzerini. Interschema knowledge in cooperative information systems. In *CoopIS*, pages 55–62, 1993.

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *WWW*, 2002.

[LNE89] James A. Larson, Shamkant B. Navathe, and Ramez Elmasri. A theory of attribute equivalence in databases with application to schema integration. *Transactions on Software Engineering*, 15(4):449–463, April 1989.

[MBDH02] Jayant Madhavan, Philip A. Bernstein, Pedro Domingos, and Alon Halevy. Representing and reasoning about mappings between domain models. In *AAAI*, pages 80–86, 2002.

[MBR01] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with Cupid. In *VLDB*, pages 49–58, 2001.

[MHH00] Renée J. Miller, Laura M. Haas, and Mauricio A. Hernández. Schema mapping as query discovery. In *VLDB*, pages 77–88, 2000.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal.*, 10(4):334–350, 2001.

[SJJ96] William W. Song, Paul Johannesson, and Janis A. Bubenko Jr. Semantic similarity relations in schema integration. *Data Knowledge and Engineering*, 19(1):65–97, 1996.

Schema Management

Periklis Andritsos* Ronald Fagin† Ariel Fuxman* Laura M. Haas†
Mauricio A. Hernández† Ching-Tien Ho† Anastasios Kementsietsidis*
Renée J. Miller* Felix Naumann† Lucian Popa† Yannis Velegrakis*
Charlotte Vilarem* Ling-Ling Yan†

Abstract

Clio is a management system for heterogeneous data that couples a traditional database management engine with additional tools for managing schemas (models of data) and mappings between schemas. In this article, we provide a brief overview of Clio and place our solutions in the context of the rich research literature on data integration and transformation. Clio is the result of an on-going collaboration between the University of Toronto and IBM Almaden Research Center in which we are addressing both foundational and systems issues related to heterogeneous data, schema, and integration management.

1 Introduction

Heterogeneous data sets contain data that have been represented using different data models, different structuring primitives, or different modeling assumptions. Such sets often have been developed and modeled with different requirements in mind. As a consequence, different schemas may have been used to represent the same or related data. To manage heterogeneous data, we must be able to manage these schemas and mappings between the schemas.

The management of heterogeneous data is a problem as old as the data management field itself. However, its importance and the variety of contexts in which it is required has increased tremendously as the architectures for exchanging and integrating data have increased in sophistication. Traditionally, heterogeneous data sets were managed within a *federated* architecture where a virtual, global schema (or view) is created over a set of heterogeneous sources (or local schemas) [HM85]. A mapping (in this case a set of view definitions) is established between each source and the global view. The mapping may define each component of the global schema in terms of the source schemas (*global-as-view* or GAV), or alternatively, the mapping may define the components of each source schema in terms of the global (*local-as-view* or LAV) [LRO96]. A federated architecture is appropriate for closely-coupled applications where queries on the virtual global view can be answered using data stored in the sources [Len02]. Notice that in federated systems, data is only exchanged at run-time (that is, query-time). However, the Web has inspired a myriad of more loosely-coupled, autonomous applications where

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

* University of Toronto, Computer Science, www.cs.toronto.edu/db/clio
† IBM Almaden Research Center, www.almaden.ibm.com/software/km/clio/index.shtml

data must be shared, but where run-time coupling of the applications is not possible. These applications include data grids and peer-to-peer applications [BGK+02]. In such applications, data must be exchanged, replicated, or migrated. To accomplish this, mappings between pairs of schemas are established (between a source and target schema). However, unlike in more integrated federated environments, the target data must be materialized so that target queries can be answered locally (even when the source data is unavailable). Furthermore, this target data must reflect the source data as accurately as possible. In federated environments, no such exchange of data is required. In general, we would like to build and manage schema mappings that permit both virtual data integration (as in federated environments) and the exchange of data between independent cooperating peers.

The architecture of Clio is described elsewhere [MHH+01]. In this work, we focus on the specific solutions employed within Clio as they relate to schema and mapping management. We consider our solutions to old, well-established research problems such as generating *schema correspondences* [BLN86] (more recently called *schema matching* [RB01]) and data integration [Len02]. We also identify new foundational and systems issues related to the problem of exchanging data between heterogeneous schemas. In Section 2, we describe our solutions for creating and managing schemas. We consider the creation and management of correspondences between schemas (Section 3) and how correspondences can be used to create mappings (Section 4). Finally, we present our solutions for using mappings in both data integration and data exchange settings.

2 Creating and Managing Schemas

For the purposes of this overview, a schema is a set of structuring primitives (predicates) and a set of constraints (logical statements) over these primitives. Within Clio, we use a nested-relational model over which a powerful set of constraints, including nested referential constraints, can be expressed.

A schema is a model of a data set and it is the primary vehicle we use for both querying the data and understanding the data. To manage heterogeneous data sets, each modeled by a different schema, we create mappings between their schemas. Within Clio, schemas are the primary vehicle for understanding and creating this mapping. Many data management solutions assume that we are given a schema that accurately reflects the data. Yet, often, this is not the case. We may be using data from a legacy database management system with little support for representing schemas, particularly constraints. Alternatively, we may be using data from a system where the data and schema are managed by different loosely-coupled, or perhaps uncoupled, software tools. So the data may be dirty and the schema may not be an accurate model of the data.

To overcome these challenges, we provide data mining tools for discovering constraints (and approximate constraints) that hold on a specific data set [Vil02]. Given a schema, we mine the data (within the given structure) for constraints. For example, in the expenseDB schema depicted on the left side of Figure 1, we could mine the data to discover the two inclusions r_1 and r_2. These inclusions represent the fact that in the given database instance, all `grant.grantee` values are a subset of `company.cid`. An inclusion is a candidate inclusion dependency. However, from a single database instance, we cannot determine whether the inclusion will continue to hold as the data changes. In addition, we may discover that an approximate inclusion holds from `grant.sponsor` to `company.cid` (not shown in the Figure) indicating that most sponsors are companies. Constraint mining looks to find a set of constraints (logical statements) that hold on a given data set and structure. The set of predicates used in these constraints is fixed – that is, the set of predicates corresponds exactly to the set of tables (or nested tables) defined in the schema.

Schemas are the result of a data design process (performed by a human designer or by an automated tool). The choices made in data design are known to be highly subjective. A schema is inherently one out of many possible choices for modeling a data set. To understand and reconcile heterogeneous data, we may need to understand (and explicitly represent) some of the alternative design choices. Our current research focuses on the development of schema discovery techniques for finding such alternative designs.

Consider again the expenseDB schema of Figure 1. Suppose that some but not all values in `grant.grantee`

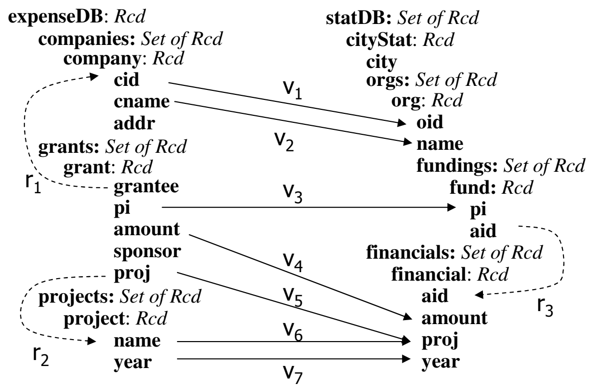


Figure 1: Two example schemas: expenseDB (left) & statDB (right).

are in `company.cid`, so that no inclusion holds between the attributes of `grants` and `companies`. Instead, `grant.grantee` may be a combination of companies, non-profit foundations and government agencies. (To keep our example simple, assume that foundations and government agencies are each modeled by a separate relational table.) Furthermore, suppose the `grants` given by `companies` (almost) always have a PI, while those given by both `foundations` and `governmentAgencies` never have a PI (the grants are given to an institution, not to an individual). These facts suggest that an alternative design for this information would be to horizontally decompose the `grants` table into three tables: `companyGrants`, `foundationGrants`, and `governmentGrants`. Such a decomposition is suggested because each subgroup of tuples shares structural characteristics (the presence or lack of a PI along with three distinct inclusion relationships into the three tables `companies`, `foundations` and `governmentAgencies`) that the group as whole does not share. We are developing mining algorithms to help find such decompositions by identifying groups of data that share common structural characteristics.

Our work can be distinguished from other structure mining techniques, including Bellman [DJMS02], in that we are searching for models that best fit the data. However, like Bellman we are performing a form of schema and data analysis. Such analysis can be very valuable in practice for large schemas that are hard for a DBA to fully understand. The tool can help a user abstract away the specific (given) design choices and better understand the schemas and the data they structure.

3 Creating and Managing Correspondences

To integrate or reconcile schemas we must understand how they correspond. If the schemas are to be integrated, their corresponding information should be reconciled and modeled in one consistent way. If data is to be exchanged (or translated) between two schemas, we must first understand *where* two schemas overlap or correspond. Methods for automating the discovery of correspondences use linguistic reasoning on schema labels [BHP94] (including ontologies and thesauri) and the syntactic structure of the schema [RB01]. Such methods have come to be referred to as *schema match operators*. The output of a schema match is an uninterpreted *correspondence relation* between elements of two schemas [RB01]. As such, a match can be viewed as a similarity join [Coh00] between two relations containing the names or descriptions (types, annotations, properties, etc.) of two separate schemas [RB01].

Within Clio, we make use of correspondences between attributes or atomic elements [NHT+02]. Our work uses not only attribute labels but also syntactic clues in the data itself to classify attributes and suggest attribute correspondences. We analyze data values and derive a set of characteristic features. We then use a classifier to group attributes with similar features. Through experimentation, we have derived a small set of domain-independent features that capture most of the structural information embedded in the attribute data. For the two example schemas of Figure 1, our techniques will, for example, suggest the attribute correspondences v_1 through v_7. Correspondence v_4 is suggested not solely based on the name similarity between attribute labels, but rather because the data values within these two attributes have similar characteristics (for example, the range of the amounts is similar and they perhaps have many 0's in them). The values themselves do not need to overlap, rather they simply need to have similar characteristics.

Note that all of the correspondence techniques we have described make use of little, if any, semantic knowledge about the data. Constraint annotations (for example, the declaration of an attribute as a key) may be used as syntactic clues to suggest a correspondence (two key attributes are more likely to correspond than a key and non-key attribute). However, these correspondence or matching approaches do not, in general, attempt to attach a semantics or meaning to specific correspondences [RB01]. Many of the matching approaches return not only attribute correspondences, but also correspondences between other schema components including relations. These correspondences may be suggested by name similarity or by correspondences among attributes of the relations. For example, the correspondences v_1 and v_2 suggest that `company` and `org` correspond. Similarly, v_3 and v_4 suggest that `grant` corresponds to both `fund` and `financial`. Note however, that the techniques for finding such "structure" correspondences are inherently dependent on the specific logical design choices made in a schema (for example, the grouping of attributes into relations or the nesting of relations) [RB01]. Hence, applying such a technique on a schema and an equivalent normalized version of the schema may yield different results. Rather than relying on such syntactic structure correspondences, we use only attribute correspondences and then apply semantic reasoning to derive a mapping [MHH00].

4 Creating and Managing Mappings

To be able to use correspondences to map data, we must associate a meaning to the correspondence relation. Typically, this semantics is expressed as inter-schema constraints. Following this convention, we express the mapping semantics as referential constraints. Since we are using a nested relational model, the mappings are then *nested referential constraints* [PVM$^+$02]. Each constraint in the mapping asserts that a query q_S over the source is associated with a query q_T over the target. In Clio, we consider *sound* mappings [Len02] where the association is a containment relation: $q_S \subseteq q_T$. Our mappings are generalizations of two common types of mappings: GAV (global-as-view) and LAV (local-as-view) mappings [Len02]. In GAV, a query over the source is asserted to be associated with a single relation of the target (global) schema. In other words, q_T is restricted to be a single relation. In LAV, a single source is asserted to be associated with a query over the target. So in LAV, q_S is restricted to be a single relation. The mappings produce by Clio have been referred to as GLAV since they have neither the GAV nor the LAV restriction [FLM99].

To create a mapping, we must take a correspondence relation R over schemas S and T and create a set of constraints, each of which has the form $q_S \subseteq q_T$. We have referred to this process as *query discovery* in that we must discover the queries used in the mapping [MHH00]. Our approach is to use the semantics of the data model to determine a set of source and target queries that produce sets of semantically related attributes. We illustrate our approach with an example.

Consider the correspondences depicted in Figure 1. In creating a mapping for the `financial` relation, we must determine which combinations of values from `grant.amount`, `project.name` and `project.year` to use in creating `financial` tuples. It is unlikely that all combinations of values are semantically meaningful. Rather, we can use the schema to determine that by taking an equi-join of `grant` and `project` on `grant.proj = project.name`, we can create triples of `grant.amount`, `project.name` and `project.year` values that are semantically related. The queries we use in our mappings are based on this idea. We use the nested structure of the schema and constraint inference to determine a set of queries that each return a set of semantically related attributes.

Often, we will find alternative semantic relationships. For example, in addition to the foreign key join between `grants` and `projects` on r_2, it may also be possible to associate `grants` with `projects` that are led by the `grant.pi`. We use carefully selected data values to illustrate and explain to a user the various mapping alternatives [YMHF01].

5 Using Mappings

The mappings produced by Clio can be used for both *data integration* (where the target schema is virtual) and for *data exchange* (where the target schema is materialized).

5.1 Data Integration

In data integration, we are given one (or more) source schemas and a single target (or global) schema. Mappings are established between each of the source schemas and the target schema. The target schema is a virtual view and as such is usually limited to have no constraints [Len02].[1] Queries posed on the target are answered by reformulating the query as a set of queries over the sources. Notice that the sound mappings that we produce do not determine a unique target instance that satisfies the mappings for a given set of source instances. Hence, it is not clear what the "right" answer to a query on the target is. The query semantics adopted most often in data integration is that of computing the *certain answers*. An instance I of the sources will produce a set of possible instances of the target J such that I and J collectively satisfy the mappings. The answer to a target query q is then the intersection of $q(J)$ as J varies over all of these possible target instances. In data integration, a target query is rewritten as a source query and the certain answers are computed *using the source instances*.

5.2 Data Exchange

A data exchange setting is very similar to that of data integration [FKMP03]. We are given a source schema (or a set of source schemas), a target schema and a mapping (or mappings) between each source and the target. As in data integration, mappings may not determine a unique target instance for a given instance I of the sources. The problem in data exchange is to materialize a single target instance (among the set of all possible target instances) that reflects the source data as accurately as possible. Furthermore, in data exchange, the target often has constraints, so we are not guaranteed that any target instance exists that satisfies the mappings and the target constraints. Given an instance I of the sources, a solution to the data exchange problem is a single target instance J that satisfies the target constraints and such that I and J collectively satisfy the mapping. In data exchange, we are faced with the problem of both determining if there is a solution to the data exchange problem and determining which solution, that is, which target instance, is the best one to exchange.

To perform data exchange, we have given an algebraic specification of a special class of *universal* solutions [FKMP03]. A universal solution has no more and no less data than needed in data exchange. In particular, for relational schemas and for target queries q that are unions of conjunctive queries, then the certain answers for q are exactly the tuples in $q(J)$ if and only if J is a universal solution. We are able to compute a canonical universal solution efficiently for a large and practical set of target constraints and mappings. We have also investigated the computational complexity of computing the certain answers in data exchange [FKMP03].

[1] Note that recent work has begun to consider constraints expressed on the target view [CGL+01].

6 The Clio System

In our system, we provide support for creating mappings between combinations of relational or XML schemas (including DTDs) [PHVM02]. In addition to mappings (the constraints described in Section 4), Clio also produces data translation queries. These source queries produce target data satisfying the constraints and (possibly nested) structure of the target schema. If the source and target have different data content, then the translation queries may also invent values for non-null target attributes that have no correspondence to the source [PVM$^+$02]. For XML sources, these queries are produced in XQuery or XSLT. For relational sources, these queries are produced in SQL (if the target is relation) or SQL with XML extensions (if the target is XML). These translation queries are used for data exchange to produce a materialized target instance. They may also be used for query composition in data integration settings.

Our technology is already being transferred into the DB2 product line. Clio has been used to create a sophisticated query builder (SQL Assist) for relational schemas. In SQL Assist, a user specifies the schema of the desired query (the attributes) and the tool suggests the query structure. Constraint reasoning is used to suggest queries that are semantically meaningful. In addition, Clio is being used in an XML to relational translator. The tool provides a default translation of an XML document into relations. A user may then modify this relational schema to meet her own requirements. Clio automatically produces and maintains a translation query as the target schema is modified.

References

[BGK$^+$02] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In *Proc. of the Int'l Workshop on the WEB and Databases (WebDB)*, 2002.

[BHP94] M. W. Bright, A. R. Hurson, and S. Pakzad. Automated Resolution of Semantic Heterogeneity in Multidatabases. *ACM Trans. on Database Sys. (TODS)*, 19(2):212–253, June 1994.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[CGL$^+$01] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Data integration in data warehousing. *Int'l Journal of Cooperative Information Systems*, 10(3):237–271, 2001.

[Coh00] W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Office Information Systems*, 18(3):288–321, 2000.

[DJMS02] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. How to Build a Data Quality Browser. In *ACM SIGMOD Int'l Conf. on the Management of Data*, 2002.

[FKMP03] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. To appear, 2003.

[FLM99] M. Friedman, A. Levy, and T. Millstein. Navigational Plans for Data Integration. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 67–73. AAAI Press/The MIT Press, 1999.

[HM85] D. M. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Office Information Systems*, 3:253–278, 1985.

[Len02] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 251–262, Bombay, India, 1996.

[MHH00] R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 77–88, Cairo, Egypt, September 2000.

[MHH$^+$01] R. J. Miller, M. A. Hernández, L. M. Haas, L.-L. Yan, C. T. H. Ho, R. Fagin, and L. Popa. The Clio Project: Managing Heterogeneity. *SIGMOD Record*, 30(1):78–83, March 2001.

[NHT$^+$02] Felix Naumann, Ching-Tien Ho, Xuqing Tian, Laura Haas, and Nimrod Megiddo. Attribute classification using feature analysis. In *Proc. of the Int'l Conf. on Data Eng.*, San Jose, CA, 2002. Poster.

[PHVM02] L. Popa, M. A. Hernández, Y. Velegrakis, and R. J. Miller. Mapping XML and Relational Schemas with CLIO, *System Demonstration*. In *Proc. of the Int'l Conf. on Data Eng.*, San Jose, CA, February 2002.

[PVM$^+$02] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 598–609, Hong Kong SAR, China, August 2002.

[RB01] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The Int'l Journal on Very Large Data Bases*, 10(4):334–350, 2001.

[Vil02] C. Vilarem. Approximate Key and Foreign Key Discovery in Relational Databases. Master's thesis, Department of Computer Science, University of Toronto, 2002.

[YMHF01] L.-L. Yan, R. J. Miller, L. Haas, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. *ACM SIGMOD Int'l Conf. on the Management of Data*, 30(2):485–496, May 2001.

On the Role of Integrity Constraints in Data Integration

Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
{lastname}@dis.uniroma1.it

Abstract

We discuss the issue of dealing with integrity constraints over the global schema in data integration. On the one hand, integrity constraints can be used to extract more information from incomplete sources, similarly to the case of databases with incomplete information. On the other hand, integrity constraints raise the problem of dealing with the inconsistency of the whole system, due to contradictory data at the sources. We also present a data integration system developed by taking into account such issues.

1 Introduction

Integrating heterogeneous data sources is a fundamental problem in databases, which has been studied extensively in the last two decades both from a formal and from a practical point of view, cf. [1]. Recently, also driven by the need to integrate data sources on the Web, much of the research on integration has focussed on so called *data integration* [2, 1]. Data integration is the problem of combining the data residing at different sources, and providing the user with a unified view of these data, called global (or mediated) schema, over which queries to the data integration system are expressed. A data integration system has therefore to free the user from the knowledge on which sources contain the data of interest, how such data are structured at the sources, and how such data are to be merged and reconciled to answer her/his queries. A crucial issue in data integration is how elements of the global schema and element at the sources are mapped to each other. In particular, two basic approaches have been proposed: *global-as-view* (GAV) and *local-as-view* (LAV) [2, 1]. The first one requires that the elements of the global schema are expressed as a view (a query) over the elements at the sources [3, 4, 5]. The second one requires that the elements at the sources are expressed as a view over the elements in the global schema [6, 7, 8]. Commonly, it is assumed that the GAV approach leads to simpler query answering mechanisms, while the LAV approach makes it easier to add or remove sources from the system. On the other hand, the LAV approach requires more sophisticated query answering techniques that are related to query answering in the presence of incomplete information [7, 9, 10, 1]. This distinction however becomes blurred as soon as we introduce integrity constraints in the global schema (see later).

A fundamental assumption related to mapping global schema and sources to each other is whether the data at the sources are considered *exact* or just *sound* in the mapping. That is, do the views defining the mapping capture exactly the relation between the data at the sources and those in the global schema (i.e., is the mapping

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

exact), or are the data provided by the sources only part of the data that should populate a given element of the global schema (i.e., is the mapping sound)?

Since the global schema acts as the interface to the user for query formulation, it should incorporate flexible and powerful representation mechanisms to relate the various elements of the global schema according to the semantics of the domain of interest. *Integrity constraints* play exactly this role by asserting the interrelations among the elements in a schema. The importance of allowing for integrity constraints in the global schema has been stressed in several works on data integration [1, 8, 11].

Introducing integrity constraints on the global schema has a deep impact on the data integration system and the query answering mechanisms. Indeed, the data retrieved from the sources may or may not fulfill the constraints in the global schema. Moreover, if the retrieved data do not fully satisfy the constraints, in principle, it may still be possible to complete the retrieved data (provided the mapping is sound) so as to satisfy the constraints. Or, vice-versa, it may not be possible to do so. In this case one may still be interested in the part of the data that does not violate the constraints. In other words, introducing integrity constraints raises issues related to query answering in the presence of incomplete information [12] (just as in the LAV approach) and to query answering in the presence of inconsistent information [13, 14, 15].

As a result of this observation, we have a spectrum of possible data integration systems, which can be classified according to the direction of the mapping (LAV/GAV), the type of the mapping (exact/sound), and the presence and type of integrity constraints allowed in the global schema. In this paper we concentrate on the GAV approach only, and we survey the general issues raised in the various cases of the above classification. To make the discussion concrete we also present an implemented data integration system based on the relational model, that follows the GAV approach, assumes sources to be sound, and allows for key and foreign key constraints in the global schema. Notably, the system takes into account such constraints in the query answering process.

2 Framework for Data Integration

We illustrate a framework for data integration that accounts for the presence of integrity constraints in the global schema. We formalize a *data integration system* $\mathcal{I}$ in terms of a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where

- $\mathcal{G}$ is the *global schema*, which provides a reconciled, integrated, and virtual view of the underlying sources, in terms of which users query the integration system. We assume in the discussion that such a global schema is expressed as a relational schema, possibly including integrity constraints, such as key constraints, foreign key constraints, and general inclusion dependencies.

- $\mathcal{S}$ is the *source schema*, which describes the structure of the set of sources accessible by the integration system. We assume that the sources are wrapped so as to be viewed as relations.

- $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$, which establishes the connection between the elements of the global schema and those of the sources. We restrict our attention to mappings of type GAV, which associate to each relation g in $\mathcal{G}$ a query $\mathcal{M}(g)$ over $\mathcal{S}$. For each element g of the global schema, a further specification (which may be either $exact$ or $sound$, see below) is associated to $\mathcal{M}(g)$ [7, 9, 10].

Given the data at the sources, we specify which data satisfy the global schema, compatibly with the data at the sources and the mapping. Let us denote by $q^{\mathcal{DB}}$ the answer set of a query q over a database $\mathcal{DB}$. A *source database* $\mathcal{D}$ for a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ is constituted by one relation $r^{\mathcal{D}}$ for each source r in $\mathcal{S}$. We call *(global) database for* $\mathcal{I}$ any database $\mathcal{B}$ for $\mathcal{G}$, and such a database is said to be *legal* for $\mathcal{I}$ wrt to $\mathcal{D}$ if:

- $\mathcal{B}$ satisfies the integrity constraints of $\mathcal{G}$, and
- $\mathcal{B}$ satisfies $\mathcal{M}$ with respect to $\mathcal{D}$, i.e., for each relation g in $\mathcal{G}$ we have that: if $\mathcal{M}(g)$ is *exact*, then $\mathcal{M}(g)^{\mathcal{D}} = g^{\mathcal{B}}$; if $\mathcal{M}(g)$ is *sound*, then $\mathcal{M}(g)^{\mathcal{D}} \subseteq g^{\mathcal{B}}$.

Note that in general there is more than one database that is legal for $\mathcal{I}$ wrt $\mathcal{D}$, hence the semantics must be specified in terms of a set of databases rather than a single one. The consequences of this will be discussed in the next section.

Queries to a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ are posed in terms of the relations in $\mathcal{G}$, and are intended to provide the specification of which data to extract from the virtual database represented by $\mathcal{I}$. The task of specifying which tuples are in the answer to a query is complicated by the existence of several legal global databases, and this requires to introduce the notion of certain answers. A tuple t is a *certain answer* to a query q wrt a source database $\mathcal{D}$, if $t \in q^{\mathcal{B}}$ for *all* global databases $\mathcal{B}$ that are legal for $\mathcal{I}$ wrt $\mathcal{D}$.

Example 1: Let $\mathcal{I}^1 = \langle \mathcal{G}^1, \mathcal{S}^1, \mathcal{M}^1 \rangle$ be a data integration system where $\mathcal{G}^1$ has the relation schemas $\mathsf{student}(Scode, Sname, Scity)$, $\mathsf{university}(Ucode, Uname)$ and $\mathsf{enrolled}(Scode, Ucode)$, and the constraints

$$\begin{array}{rcl} key(\mathsf{student}) & = & \{Scode\} \\ key(\mathsf{university}) & = & \{Ucode\} \\ key(\mathsf{enrolled}) & = & \{Scode, Ucode\} \end{array} \qquad \begin{array}{rcl} \mathsf{enrolled}[Scode] & \subseteq & \mathsf{student}[Scode] \\ \mathsf{enrolled}[Ucode] & \subseteq & \mathsf{university}[Ucode] \end{array}$$

$\mathcal{S}^1$ consists of three sources. Source stu, of arity 4, contains information about students with their code, name, city, and date of birth. Source univ, of arity 2, contains codes and names of universities. Finally, Source enr, of arity 2, contains information about enrollment of students in universities. The mapping $\mathcal{M}^1$ is defined by

$$\begin{array}{rl} \mathcal{M}^1(\mathsf{student}): & \mathsf{student}(x,y,z) \leftarrow \mathsf{stu}(x,y,z,w) \\ \mathcal{M}^1(\mathsf{university}): & \mathsf{university}(x,y) \leftarrow \mathsf{univ}(x,y) \\ \mathcal{M}^1(\mathsf{enrolled}): & \mathsf{enrolled}(x,w) \leftarrow \mathsf{enr}(x,w) \end{array}$$

3 The Role of Integrity Constraints in Query Answering

The ultimate goal of a data integration system is to answer queries posed by the user in terms of the global schema. Since data are stored at the sources, query answering requires to reformulate the query in terms of the source schema, taking into account the mapping. It follows that the nature of the mapping has a great influence on the whole process. We now discuss this issue, by pointing out the impact of integrity constraints in the reformulation step. We focus on the GAV approach only. We distinguish among four cases, depending on whether the mapping is sound or exact, and whether the global schema contains integrity constraints or not.

The first case we consider is the one with *exact mapping*, and with *no integrity constraints* in the global schema. The exact mapping provides a unique way to determine which are the data of the global database on the basis of the data at the sources. Since such data cannot contradict the global schema, in this case, the semantics of the data integration system is characterized by a single global database, namely the one that is obtained by associating to each element g the set of tuples computed by the query $\mathcal{M}(g)$ over the sources. It follows that query answering can be based on a simple *unfolding* strategy: when processing a query q over the global schema, every element in q is substituted with the corresponding query over the sources, and the resulting query is evaluated at the sources. Most GAV data integration systems, such as TSIMMIS [3], Garlic [16], Squirrel [17], and MOMIS [18], follows this approach.

The second case is the one with *exact mapping*, and *with integrity constraints* in the global schema. Again, the exact mapping precisely specifies which are the data of the global database in terms of the data at the sources. If such data are coherent with respect to the integrity constraints of the global schema, query answering can be carried out as if there were no constraints. However, differently from the previous case, it might be that the data retrieved from the sources do not satisfy the integrity constraints in the global schema. This situation is similar to the case where we aim at answering queries over a single inconsistent database. This problem is studied in several papers, including [13, 14, 15, 19]. Obviously, all the above mentioned papers start from the observation that the usual first-order semantics is not suited for this case, and tackle the fundamental problem of devising a meaningful semantics for query answering. In general, the role of such a semantics is to single out which are the reasonable "repairs" of the global database. Intuitively, a *repair* of a database $\mathcal{B}$ is a global database that satisfies the integrity constraints of the global schema, and can be obtained from $\mathcal{B}$ by a minimum number of insertions and deletions [20]. One of the basic problems is that, depending on the types of integrity constraints, there might be more than one repair. It follows that a simple unfolding strategy is likely to be inappropriate for the new semantics, and, indeed, recent papers aim at defining new query answering methods that are sound and complete with respect to the new formal framework [19].

The third case is the one with *sound mapping*, and with *no integrity constraints*. In this case, the mapping does not determine a unique global database on the basis of the source data, and indeed, we must accept all global databases that are supersets of the data retrieved from the sources. Formally, this is similar to the case where we have a database with incomplete information (or, a partially specified database). In principle, query answering can be affected by such an incompleteness. However, when the queries posed to the global schema are monotone (in particular, when they do not contain negation), in computing certain answers one can exploit the fact that there is a unique "minimal" global database, which is the one obtained by associating to each element of the global schema the set of tuples computed by the corresponding query. In other words, a simple unfolding strategy is again sufficient for this case.

The last case is the one with *sound mapping*, and *with integrity constraints* in the global schema. In this case, given the nature of the mapping, the semantics of the data integration system is specified in terms of a set of global databases, namely, those databases that are supersets of the data retrieved from the sources. However, among all the elements of this set, we must select only those databases that satisfy the integrity constraints in the global schema. If there are no such global databases, we basically fall into the second case, in the sense that we have to consider the notion of repair to make query answering meaningful. If there is at least one global database that is coherent with the mapping, and satisfies the integrity constraints of the global schema, then the query processing technique should be able to select only those answers that are true in all such global databases. Obviously, the characteristic of query processing will strongly depend upon the types of integrity constraints specified in the global schema. In the next section, a query processing strategy that is able to deal with the class of foreign key constraints is shown. Notice that in such a case the unfolding strategy proves to be too naïve, as shown by the following example.

Example 2: Referring to Example 1, where key and foreign key constraints are present in the global schema, suppose to have the following source database $\mathcal{D}$:

$\mathsf{stu}^{\mathcal{D}}$:

12	*anne*	*florence*	21
15	*bill*	*oslo*	24

$\mathsf{univ}^{\mathcal{D}}$:

AF	*bocconi*
BN	*ucla*

$\mathsf{enr}^{\mathcal{D}}$:

12	*AF*
16	*BN*

Due to the integrity constraints in $\mathcal{G}^1$, 16 is the code of some student. Observe, however, that nothing is said by $\mathcal{D}$ about the name and the city of such a student. Therefore, we must accept as legal all databases that differ in such attributes of the student with code 16. Note that this is a consequence of the assumption of having sound views. If we had exact or complete views, this situation would have lead to an inconsistency of the data integration system. Instead, when dealing with sound views, we can think of extending the data contained in the sources in order to satisfy the integrity constraint over the global schema. The fact that, in general, there are several possible ways to carry out such an extension implies that there are several legal databases for the data integration systems. Now, consider the query

$$\mathsf{q}(x) \leftarrow \mathsf{student}(x, y, z) \wedge \mathsf{enrolled}(x, w)$$

The correct answer to the query is $\{12, 16\}$, because, due to the integrity constraints in $\mathcal{G}^1$, we know that 16 appears in the first attribute of $\mathsf{student}$ in all the databases for $\mathcal{I}^1$ that are legal wrt $\mathcal{D}$. However, we do not get this information from $\mathsf{stu}^{\mathcal{D}}$, and, therefore, a simple unfolding strategy retrieves only the answer $\{12\}$ from $\mathcal{D}$, thus proving insufficient for query answering in this case. Notice that, if the query asked for the student name instead of the student code (i.e., the head is $\mathsf{q}(y)$ instead of $\mathsf{q}(x)$), then one could *not* make use of the dependencies to infer additional answers.

4 The IBIS Data Integration System

To make the discussion concrete, we now present how the above concepts have been realized in a data integration system called IBIS [21, 22], which was developed by CM Sistemi in collaboration with the University of Rome "La Sapienza". IBIS is based on the relational model, and allows for key and foreign key constraints in the global schema. IBIS adopts the GAV approach for defining the mapping between the global schema relations and the sources. It deals with heterogeneous data sources, such as relational databases, legacy data sources, and web pages. Non-relational data sources are suitably wrapped so as to present themselves to the query precessing subsystem as relational sources (possibly with binding pattern, but we will ignore this aspect in the following).

The GAV mapping between the relations in the global schema and the tables at the sources is realized through arbitrary queries. This generality allows us to incorporate in such queries data cleaning mechanisms [23] to resolve conflicts on keys. Indeed it has been shown that, if key conflicts are resolved by the extraction process from the sources, then the other constraints in the global schema, namely the foreign key constraints, although tuple generating, will not cause further violation of key constraints. Observe that, by adopting this solution, IBIS is essentially delegating to the designer of the GAV mapping the responsibility of dealing with key conflicts.

The IBIS query processing algorithm fully takes into account foreign key constrains, and this allows the system to return answers to queries that depend on joins on attribute values that are not stored in the sources, but whose existence is guaranteed by the foreign key constraints (cf. Example 2). Differently from key constraints, the system deals with foreign key constraints in a completely automated way. It is possible to show that the IBIS query processing algorithm produces exactly the set of certain answers to queries, i.e., it is sound and complete with respect to the semantics of the data integration system. The IBIS query processing algorithm is constituted by three conceptually separate phases (in fact, the system is highly optimized and does not necessarily keep such phases distinct in performing its computations):

1. the query is *expanded* to take into account foreign key constraints in the global schema;
2. the atoms in the expanded query are *unfolded* according to their definition in terms of the mapping, obtaining a query expressed over the sources;
3. the expanded and unfolded query is *executed* over the sources, to produce the answer to the original query.

Unfolding and execution are standard steps in GAV query processing (although in IBIS they are more involved than usual because they perform data cleaning to resolve conflicts on keys). The expansion phase is the distinguishing feature of query processing in IBIS. The expansion is performed by viewing each foreign key constraint $r_1[\vec{x}] \subseteq r_2[\vec{y}]$, where $\vec{x}$ and $\vec{y}$ are sets of h attributes and $\vec{y}$ is a key for r_2, as a logic programming rule

$$r_2'(\vec{x}, f_{h+1}(\vec{x}), \ldots, f_n(\vec{x})) \;\leftarrow\; r_1'(\vec{x}, x_{h+1}, \ldots, x_m)$$

where each f_i is a Skolem function, $\vec{x}$ is a vector of h variables, and we have assumed for simplicity that the attributes involved in the foreign key are the first h ones. Each r_i' is a predicate, corresponding to the global relation r_i, defined by the above rules for foreign key constraints, together with the rule

$$r_i'(x_1, \ldots, x_n) \;\leftarrow\; r_i(x_1, \ldots, x_n)$$

Once such a logic program $\Pi_{\mathcal{G}}$ has been defined, it can be used to generate the expanded query associated to the original query q. This is done by performing a partial evaluation [24] wrt $\Pi_{\mathcal{G}}$ of the body of q', which is the query obtained by substituting in q each predicate r_i with r_i'. In the partial evaluation tree, a node is not expanded anymore either when: (i) no atom in the node unifies with a head of a rule; or (ii) when the Skolem functions have appeared in the substitutions for the distinguish variables, which makes the current answer unsuitable for being returned; or (iii) when the node is subsumed by (i.e., is more specific than) one of its predecessors, since the node cannot provide any answer that is not already provided by its more general predecessor [21]. Variants of the IBIS query processing mechanism have also been studied for other kinds of integrity constraints [25].

View-based query answering is known to be tightly related to query containment [7, 26]. Query containment in the relational setting in presence of inclusion dependencies alone, and in presence of key-based constraints and inclusion dependencies of a special form has been studied in [27], using chase-based techniques. However, more work has to be done to understand whether such techniques can be adapted for doing query answering in data integration systems, since a straightforward use of such techniques would require to materialize the whole global database, which may not feasible in practice.

Finally, as mentioned, the presence of integrity constraints in the global schema blurs the distinctions between GAV and LAV. In particular, [28] shows that a LAV system based on the relational model where the LAV mapping is defined in terms of conjunctive queries over the global schema, can be rephrased into a (query) equivalent GAV system that includes, in the global schema, inclusion dependencies together with a simple class of equality-generating dependencies.

5 Conclusions

In this paper we have investigated how in data integration the presence of integrity constraints on the global schema has a significant impact on the semantics of a data integration system. The presence of such constraints, even of a simple form, raises the need of dealing with incomplete information and possibly with inconsistencies. These issues confirm that data integration is a rich and challenging research topic [1].

References

[1] Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of PODS 2002. (2002) 233–246

[2] Halevy, A.Y.: Answering queries using views: A survey. VLDB Journal **10** (2001) 270–294

[3] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J.D., Vassalos, V., Widom, J.: The TSIMMIS approach to mediation: Data models and languages. J. of Intelligent Information Systems **8** (1997) 117–132

[4] Tomasic, A., Raschid, L., Valduriez, P.: Scaling access to heterogeneous data sources with DISCO. IEEE Trans. on Knowledge and Data Engineering **10** (1998) 808–823

[5] Goh, C.H., Bressan, S., Madnick, S.E., Siegel, M.D.: Context interchange: New features and formalisms for the intelligent integration of information. ACM Trans. on Information Systems **17** (1999) 270–293

[6] Kirk, T., Levy, A.Y., Sagiv, Y., Srivastava, D.: The Information Manifold. In: Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments. (1995) 85–91

[7] Abiteboul, S., Duschka, O.: Complexity of answering queries using materialized views. In: Proc. of PODS'98. (1998) 254–265

[8] Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Data integration in data warehousing. Int. J. of Cooperative Information Systems **10** (2001) 237–271

[9] Grahne, G., Mendelzon, A.O.: Tableau techniques for querying information sources through global schemas. In: Proc. of ICDT'99. Volume 1540 of LNCS., Springer (1999) 332–347

[10] Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Query processing using views for regular path queries with inverse. In: Proc. of PODS 2000. (2000) 58–66

[11] Fernandez, M.F., Florescu, D., Levy, A., Suciu, D.: Verifying integrity constraints on web-sites. In: Proc. of IJCAI'99. (1999) 614–619

[12] van der Meyden, R.: Logical approaches to incomplete information. In Chomicki, J., Saake, G., eds.: Logics for Databases and Information Systems. Kluwer Academic Publisher (1998) 307–356

[13] Lin, J., Mendelzon, A.O.: Merging databases under constraints. Int. J. of Cooperative Information Systems **7** (1998) 55–76

[14] Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: Proc. of PODS'99. (1999) 68–79

[15] Greco, G., Greco, S., Zumpano, E.: A logic programming approach to the integration, repairing and querying of inconsistent databases. In: Proc. of ICLP'01. Volume 2237 of LNAI., Springer (2001) 348–364

[16] Tork Roth, M., Arya, M., Haas, L.M., Carey, M.J., Cody, W.F., Fagin, R., Schwarz, P.M., II, J.T., Wimmers, E.L.: The Garlic project. In: Proc. of ACM SIGMOD. (1996) 557

[17] Zhou, G., Hull, R., King, R., Franchitti, J.C.: Using object matching and materialization to integrate heterogeneous databases. In: Proc. of CoopIS'95. (1995) 4–18

[18] Beneventano, D., Bergamaschi, S., Castano, S., Corni, A., Guidetti, R., Malvezzi, G., Melchiori, M., Vincini, M.: Information integration: the MOMIS project demonstration. In: Proc. of VLDB 2000. (2000)

[19] Lembo, D., Lenzerini, M., Rosati, R.: Source inconsistency and incompleteness in data integration. In: Proc. of KRDB 2002. (2002)

[20] Fagin, R., Ullman, J.D., Vardi, M.Y.: On the semantics of updates in databases. In: Proc. of PODS'83. (1983) 352–365

[21] Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: Data integration under integrity constraints. In: Proc. of CAiSE 2002. Volume 2348 of LNCS., Springer (2002) 262–279

[22] Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M., Naggar, P., Vernacotola, F.: IBIS: Data integration at work. In: Proc. of SEBD 2002. (2002) 291–298

[23] Bouzeghoub, M., Lenzerini, M.: Introduction to the special issue on data extraction, cleaning, and reconciliation. Information Systems **26** (2001) 535–536

[24] De Giacomo, G.: Intensional query answering by partial evaluation. J. of Intelligent Information Systems **7** (1996) 205–233

[25] Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: Accessing data integration systems through conceptual schemas. In: Proc. of ER 2001. (2001) 270–284

[26] Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: View-based query answering and query containment over semistructured data. In: Proc. of DBPL 2001. (2001)

[27] Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. of Computer and System Sciences **28** (1984) 167–189

[28] Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: On the expressive power of data integration systems. In: Proc. of ER 2002. (2002)

Information Integration and Incomplete Information

Dedicated to Witold Lipski Jr., 1949–1985

Gösta Grahne
Department of Computer Science, Concordia University
Montreal, Quebec, Canada H3G 1M8
grahne@cs.concordia.ca

Abstract

We first briefly review relational databases and incomplete information. We then describe how querying in Information Integration Systems amounts to querying incomplete databases. We also discuss query rewriting, which essentially is a (compile-time) query optimization technique used in processing queries in Information Integration systems. Finally, we review some recent advances in query rewriting.

1 Basic Definitions

Relational Databases. The Relational Model structures information in the form of tables. There are various ways to formalize this notion. The first is a direct adaptation of the mathematical concept of a relation: Let the atomic values come from a countably infinite set $\mathbf{dom}$.[1] Let k be a positive integer. A k-ary relation is then a finite subset of $\mathbf{dom} \times \mathbf{dom} \times \cdots \times \mathbf{dom}$, k-times, sometimes denoted $\mathbf{dom}^k$. In order have names for relations, such as R, R_1, R_2, etc, we assume a function I that when applied to a relation name R gives the *instance* $I(R)$ of the actual tuples from $\mathbf{dom}^k$. For a relational "schema" $\{R_1, R_2, \ldots, R_n\}$, the database instance consists of an instance $I(R_i)$ for each relation name R_i, $i \in \{1, \ldots, n\}$. Note that there are no column names in this formalization, only positions (first column, second column, etc).

In this paper we adopt a variation of this modeling influenced by the logic programming approach to databases. From the domain $\mathbf{dom}$ and the relation names we build up a (Herbrand) universe consisting of all expressions of the form $R(a_1, a_2, \ldots, a_k)$, where R is a relation name and the a_i's are values in $\mathbf{dom}$. Such expressions are called *facts*. A database instance is then simply a finite set of facts, i.e. a finite subset of the universe, such as $\{R_1(a_1, a_3), R_1(a_2, a_3), R_2(a_3, a_4)\}$. Since relation names are part of the facts, we do not need to list facts from different relations separately.

For more information on the various ways to formalize the relational model, see [AHV95].

A *query* is an expression that defines a function from all database to (usually) a single relation. In this paper we focus on a simple class of queries, namely the conjunctive queries. For this we need the concept of an *atom*. An atom is like a fact, except that we allow *variables*, taken from and infinite set $\mathbf{var}$, as well as constants. For

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] We assume for simplicity and clarity of exposition that there is only one domain from which all values are drawn.

instance, $R(a, X)$ is an atom, whereas $R(a, b)$ is a fact.[2] The variables in the atoms are placeholders, and they stand for any domain value. Variables will be denoted by uppercase letters, such as X and Y.

A *conjunctive query* φ is an expression of the form

$$head(\varphi) \leftarrow body(\varphi), \tag{1}$$

where $body(\varphi)$ is a set of atoms over relation names in the database schema, and $head(\varphi)$ is an atom over an answer relation name not used in the database. We assume that all variables occurring in $head(\varphi)$ also occur in $body(\varphi)$, i. e. that the query φ is *safe*.

A conjunctive query φ can be applied to a database $\mathbf{db}$ resulting in a set of facts

$$\varphi(\mathbf{db}) = \{\sigma(head(\varphi)) : \sigma(body(\varphi)) \subseteq \mathbf{db} \text{ for some valuation } \sigma\}. \tag{2}$$

A *valuation* σ, is formally a finite partial mapping from $\mathbf{var} \cup \mathbf{dom}$ to $\mathbf{dom}$ that is the identity on $\mathbf{dom}$.

Example 1: If the database $\mathbf{db}$ is $\{R_1(a_1, a_3), R_1(a_2, a_3), R_2(a_3, a_4)\}$, and φ is $Q(X) \leftarrow R_1(X, Y), R_2(Y, Z)$, then $\varphi(\mathbf{db}) = \{Q(a_1), Q(a_2)\}$. On the other hand, if φ is $Q(X) \leftarrow R_1(a_1, Y), R_2(Y, X)$, then $\varphi(\mathbf{db}) = \{Q(a_4)\}$. □

Incomplete Databases. Usually a database is a complete description of the state of the world modeled by it. This is actually true only if we adopt the *Closed World Assumption*, CWA [Rei78], according to which facts not explicitly stored in the database are false. For example, in the database $\{R_1(a_1, a_3), R_1(a_2, a_3), R_2(a_3, a_4)\}$, the fact $R_1(a_3, a_1)$ is false. The closed world assumption is convenient since there in general is an infinitude of false facts. The CWA is however not appropriate for modeling all situations. We might for instance have a database $\mathbf{db} = \{$*Owns(Sue,Mazda),Owns(Joe,Chevrolet)*$\}$. If we consider the fact *Owns(Mary,Toyota)* we might not want to draw the conclusion that Mary does not own a Toyota, we just don't have evidence recorded about it. We then adopt the *Open World Assumption*, OWA, which regards the database as an *incomplete* description of the world: All facts stored in the database are true, the truth value of any other fact is *unknown*. Semantically this means that the stored database $\mathbf{db}$ actually is a finite description of a *set of possible worlds*, defined as

$$\{\mathbf{db}' : \mathbf{db} \subseteq \mathbf{db}'\}. \tag{3}$$

Each $\mathbf{db}'$ represents one possible complete state of affairs (or world). Thus for instance the fact *Owns(Mary, Toyota)* will be true in some possible worlds, and false in others. Facts that are true in *some* possible worlds are called *possible* facts, and facts true in *all* possible worlds are called *certain* facts.

A database under the OWA is the simplest example of an incomplete database. To go a step further, suppose we know that Mary owns a car, but we don't know the make of it. This we could represent by the atom *Owns(Mary,*X*)*. Here again, we have an incomplete description of the world. The world could correspond to *Owns(Mary,Toyota)*, or to *Owns(Mary,Mazda)*, or to *Owns(Mary,Drabant)*, and so on.

A very simple (and efficient) way to represent possible worlds is to allow the database to consist of atoms, as opposed to pure facts only. A set of facts is called a *tableau* [Men84]. A tableau is denoted T, as opposed to $\mathbf{db}$ which stands for a finite set of facts (not atoms).

Example 2: Let $T = \{$*Owns(Sue,Mazda),Owns(Mary,*X*)*$\}$. This tableau contains two atoms, the first of which actually is a fact (a special case of an atom). □

A tableau T represents a set of databases. This set is denoted $rep(T)$, and it is defined by

$$rep(T) = \{\mathbf{db} \ : \text{ there is a valuation } \sigma \text{ such that } \sigma(T) \subseteq \mathbf{db}\}. \tag{4}$$

The definition says that a database $\mathbf{db}$ is represented by a tableau T, if there is a valuation σ such that when all variables in T are replaced by their image under σ, the set of facts thus obtained is a subset of $\mathbf{db}$.

[2] $R(a, b)$ is of course also an atom, since a fact is a special case of an atom.

Example 3: In our tableau $T = \{$*Owns(Sue,Mazda), Owns(Mary,X)*$\}$, we will have $\{$*Owns(Sue,Mazda), Owns(Mary,Mazda)* $\} \in \mathit{rep}(T)$, $\{$*Owns(Sue,Mazda),Owns(Mary,Chevrolet)*$\} \in \mathit{rep}(T)$, etc. In this database *Owns(Sue,Mazda)* is a certain fact since it is true in all possible worlds in $\mathit{rep}(T)$, and *Owns(Mary, Mazda)* is a possible fact since it is true in some possible worlds represented by T. □

There are many other ways to represent sets of possible worlds, see [Gra84, IL84], and [Gra91, AHV95] for an overview.

Now what does it mean to query an incomplete database? Since an incomplete database is a set, each element representing a possible world, the natural answer to a query is the *set* of answers obtained by applying the query on each possible world. This set of answers is called the *exact answer*.

Suppose now we have a query φ on an incomplete database represented by a tableau T. The exact answer would be $\varphi(\mathit{poss}(T))$. But just as the original database was represented by a tableau T we would like to represent the exact answer by another tableau U, such that $\mathit{rep}(U) = \varphi(\mathit{rep}(T))$. More practically thinking, given a query φ and a tableau T, we would like to have an algorithm, or evaluation mechanism, call it $\widehat{\varphi}$, applicable on tableau, such that

$$\mathit{rep}(\widehat{\varphi}(T)) = \varphi(\mathit{rep}(T)). \tag{5}$$

This requirement can be illustrated by the following commutative diagram.

$$\begin{tikzcd}
\text{Tableau } T \arrow[r, "\mathit{rep}"] \arrow[d, "\widehat{\varphi}"'] & \text{Possible worlds } \mathit{rep}(T) \arrow[d, "\varphi"'] \\
\text{Answer } \widehat{\varphi}(T) \arrow[r, "\mathit{rep}"] & \text{Exact answer } \varphi(\mathit{rep}(T))
\end{tikzcd}$$

Is the requirement in Equation (5) achievable? The answer is yes, but only if we use a representation mechanism more complicated than tableaux (see [IL84]). If we wish to use tableaux we have to contend with a weaker form of (5), where we use coinitiality, instead of equality. Let $\mathcal{X}$ and $\mathcal{Y}$ be two enumerable sets of possible databases. We say that $\mathcal{X}$ and $\mathcal{Y}$ are *coinitial* if they have the same $\subseteq$-minimal elements. Coinitiality is denoted $\mathcal{X} \approx \mathcal{Y}$.[3]

For tableaux, it turns out that if we for evaluation purposes treat the variables as pairwise distinct constants, and distinct from all "true" constants, we can apply standard evaluation and obtain an representation of the result that is coinitial with the exact answer. To formally explain the evaluation, we need the concept of a substitution. A *substitution* is a valuation, except that we allow variables to be mapped into variables, not only constants. Thus, a substitution θ is a function from (a subset of) $\mathbf{dom} \cup \mathbf{var}$ to $\mathbf{dom} \cup \mathbf{var}$, keeping in mind that constants have to be mapped to themselves. Then

$$\widehat{\varphi}(T) = \{\theta(\mathit{head}(\varphi)) : \theta(\mathit{body}(\varphi)) \subseteq T \text{ for some substitution } \theta\}. \tag{6}$$

The fundamental result that

$$\mathit{rep}(\widehat{\varphi}(T)) \approx \varphi(\mathit{rep}(T)) \tag{7}$$

was established in [IL84, Var86], and adapted to tableaux in [GM99, GK02]. Furthermore, in Eq. (7) the certain facts of $\varphi(\mathit{rep}(T))$, that is $\cap(\varphi(\mathit{rep}(T)))$, can be obtained from $\widehat{\varphi}(T)$ by retaining only the pure variable-free atoms.

[3] For a deeper treatment of coinitiality, see [IL84].

Example 4: Let $T = \{R_1(a,b), R_1(d,X), R_2(b,c), R_2(X,e), R_2(Y,f)\}$, and $\varphi = Q(X,Y,Z) \leftarrow R_1(X,Y), R_2(Y,Z)$. Then $\widehat{\varphi}(T) = \{Q(a,b,c), Q(d,X,e)\}$. The only certain fact in the answer is $Q(a,b,c)$. If $\varphi = Q(X,b) \leftarrow R_1(X,b)$ we have $\widehat{\varphi}(T) = \{Q(a,b)\}$. This fact is also certain. □

2 Global databases: The meaning of integrated information

Information Integration systems [Ull97, Hal00, Hal01, Len02] aim to provide a uniform query interface to multiple heterogeneous sources. One particular and useful way of viewing these systems, first proposed within the Information Manifold project [LRO96], is to postulate a *global schema* (called a world view) that provides a unifying data model for all the information sources. A query processor is in charge of accepting queries written in terms of this global schema, translating them to queries on the appropriate sources, and assembling the answers into a global answer. Each source is modeled as a *materialized view* defined in terms of the global relations, which are virtual. Note the reversal of the classical model: instead of thinking of views as virtual artifacts defined on stored relations, we think of the views as stored, and the relations that the views are defined on as virtual. [4]

A question of semantics now arises: what is the meaning of a query? Since a query is expressed in terms of the global schema, and the sources implicitly represent an instance of this global schema, it would be natural –at least conceptually– to reconstruct the global database represented by the views and apply the query to this global database.

Well, it turns out that the global database actually is incomplete. In other words, there might be *several* (usually infinitely many) global databases that are consistent with the definition of, and the data in the sources.

Example 5: For a simple example, suppose we have a global relations *Owns(Person,Car)*, and two sources: Source S_1 has definition $S_1(X,Y) \leftarrow$ *Owns(X,Y)* and extension $\mathbf{s}_1 = \{S_1(\textit{Sue, Mazda})\}$. Source S_2 has definition $S_2(X) \leftarrow$ *Owns(X,Y)* and extension $\mathbf{s}_2 = \{S_2(\textit{Mary})\}$. Then it is natural to think that any database that contains at least the facts *Owns(Sue,Mazda)*, and *Owns(Mary,a)*, for some $a \in \mathbf{dom}$, is a possible global database for $\mathcal{S} = \{S_1, S_2\}$. □

Formally, for a source collection $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, where each S_i has definition ψ_i and extension $\mathbf{s}_i$, we call its set of possible databases $\mathit{poss}(\mathcal{S})$, and define it as

$$\mathit{poss}(\mathcal{S}) = \{\mathbf{db} \ : \ \mathbf{s}_i \subseteq \psi_i(\mathbf{db}), i \in \{1, \ldots, m\}\}. \tag{8}$$

Note that $\mathit{poss}(\mathcal{S})$ is infinite. Now the set $\mathit{poss}(\mathcal{S})$ can be conveniently represented by a tableau denoted $T(\mathcal{S})$, such that

$$\mathit{rep}(T(\mathcal{S})) = \mathit{poss}(\mathcal{S}). \tag{9}$$

Example 6: For $\mathcal{S} = \{S_1, S_2\}$ in Example 5, we have $T(\mathcal{S}) = \{\textit{Owns(Sue,Mazda), Owns(Mary,}X)\}$. □

The details of how to construct $T(\mathcal{S})$ can be found in [GM99, GK02].

In definition 8 we make a version of the OWA. The facts in the sources are considered to be *sound*, in the sense that they all are generated by facts in the global database. An alternative view would be to state that a source can contain (spurious) facts that do not stem from facts in the global database. In this case the source would be considered *complete*, but not necessarily sound. If the sources are required to be both sound and complete, they are called *exact*.[5] The situations where sources are allowed to be both sound and complete

[4] Whether the views are actually stored at each source or materialized by other means, and whether they consist of relations or semi-structured objects or files are issues irrelevant to our discussion and hidden from the query processor by appropriate wrappers.

[5] If all sources are complete, the definition of poss would be $\mathit{poss}(\mathcal{S}) = \{\mathbf{db} \ : \ \mathbf{s}_i \supseteq \psi_i(\mathbf{db}), i \in \{1, \ldots, m\}\}$, and for exact sources we would have $\mathit{poss}(\mathcal{S}) = \{\mathbf{db} \ : \ \mathbf{s}_i = \psi_i(\mathbf{db}), i \in \{1, \ldots, m\}\}$.

are treated in [AD98, GM99]. The presence of complete sources however usually increase the computational complexity of decision problems related to integration, see [AD98, GM99]. In [MM01] the authors consider a generalization of the sound/complete view assumptions. Each source is is considered to be both *partially sound*, and *partially complete*. A soundness of say 80% would say that the source contains 80% facts that are derived from facts in the global database, and 20% of "spurious" facts. A completeness of 80% says that 80% of the facts derivable from the global database are in the source.

Now that we have seen that a source collection $\mathcal{S}$ actually defines an incomplete database $poss(\mathcal{S})$ that we can represent by a tableau $T(\mathcal{S})$, and that an incomplete database represented as a tableau can be queried using the $\widehat{\varphi}$-evaluation of a query φ, we have a method for computing a representation of the exact answer to a query φ posed on the global database: First compute the tableau $T(\mathcal{S})$, then apply $\widehat{\varphi}$ to obtain $\widehat{\varphi}(T(\mathcal{S}))$. Equation (7) gave us $rep(\widehat{\varphi}(T)) \approx \varphi(rep(T))$, for any tableau T, and since Equation (9) gives $rep(T(\mathcal{S})) = poss(\mathcal{S})$, we get

$$rep(\widehat{\varphi}(T\mathcal{S})) \approx \varphi(poss(\mathcal{S})). \tag{10}$$

Example 7: In examples 5 and 6, if the query φ were the identity, i.e. it asked for the facts in the *Owns*-relation, a representation of the exact answer would be the tableau $\{$*Owns(Sue,Mazda)*, *Owns(Mary,*X*)*$\}$. The certain fact in this answer is *Owns(Sue,Mazda)*. □

3 Using rewritings to answer queries on global databases

In the previous section we saw how to answer a query φ by first constructing the tableau representing all possible global databases. However, computing $T(\mathcal{S})$ might involve a lot of redundant work, since it amounts to constructing the tableau corresponding to the entire source collection $\mathcal{S}$, whereas the global relations that are in the body of φ might be mentioned in only few source definitions. Furthermore, the query might have selections and joins that could be computed directly at the sources.

A *rewriting* φ' of a global query φ is a query formulated directly in terms of the source relations S_i. It appears that rewritings were first proposed for a restricted setting in [YL87]. They were generalized in [LMSS95] and [LRO96]. The correctness criteria for rewritings were defined in terms of a certain containment between the rewriting and the original query.

Example 8: Let $\mathcal{S} = \{S_1, S_2\}$, with definitions $S_1(X,Z) \leftarrow R_1(X,Y), R_2(Y,Z)$, and $S_2(X,Z) \leftarrow R_1(X,Y)$, $R_3(Y,Z)$. Let the query be $\varphi = Q(X,Y) \leftarrow R_1(X,Y)$. According to the correctness criteria in [LRO96], the desired rewriting φ' would be the union of $Q(X,Y) \leftarrow S_1(X,Y)$, and $Q(X,Y) \leftarrow S_2(X,Y)$. □

The correctness criteria did not however state what the answer produced by the rewriting meant. It was subsequently shown in [DG98, GM99] that the rewritings actually produced the *certain answer* $\cap(\varphi(poss(\mathcal{S})))$. This is easy to see in Example 8. Source S_1 contains facts of R_1 that join with a fact in R_2, and S_2 contains facts of R_1 that join with a fact in R_3. Since the sources provide us no information about R_2 and R_3, there is a possible database where R_2 and R_3 are empty. Therefore, the facts of R_1 that are in every possible database is the union of the facts in S_1 and S_2, which is exactly what the "correct" rewriting would produce in Example 8.

The first algorithms for rewritings essentially explored the whole (finite) solution space in order to find φ'. Later more efficient algorithms were developed. These include the set-covering based algorithm in [GM99], the MiniCon algorithm of [PL00], the CoreCover algorithm [ALU01], and its extension in [ALM02]. Other extensions of the basic technique are surveyed in [Hal00, Hal01]. The cases where the queries defining the sources, as well as the user query, are allowed to be more general than conjunctive ones are analyzed in [AD98].

However, perhaps since the relationship between information integration and incomplete information had not been clearly articulated, there were no algorithms for computing the exact answer, until the tableau techniques were applied in [GM99], and [GK02] showed how to come up with a rewriting that actually produces a representation of the exact answer.

Example 9: Suppose the global schema contains two relations, *Owns(Person,Car)*, and *Lives(Person, City)*. A fact *Owns(Sue,Mazda)* means that *Sue* owns a *Mazda*, and e.g. *Lives(Sue,Edmonton)* means that *Sue* lives in *Edmonton*. We have two sources, $S_1(X,Y) \leftarrow$*Owns*(X,Y), *Lives*(X,Z), and $S_2(X,Z) \leftarrow$*Owns*(X,Y), *Lives*(X,Z). The user issues the query $\varphi = Q(X,Y,Z) \leftarrow$*Owns*$(X,Y)$,*Lives*$(X,Z)$. Using the correctness criteria in [LRO96], we get an empty rewriting, and thus an empty query result for the user. However, suppose $\mathbf{s}_1 = \{S_1(\textit{Sue,Mazda})\}$, and $\mathbf{s}_2 = \{S_2(\textit{Mary,Prague})\}$. Then $T(\{S_1,S_2\}) = \{\textit{Owns(Sue,Mazda)},\textit{Owns}(\textit{Mary},X_1),$ *Lives(Mary,Prague)*, *Lives*$(\textit{Sue},X_2)\}$, and $\widehat{\varphi}(T(\{S_1,S_2\})) = \{Q(\textit{Sue, Mazda}, Y_1), Q(\textit{Mary},Y_2,\textit{Prague})\}$. This exact answer tells the user that *Sue* owns a *Mazda* and lives in an unknown city, and that *Mary* lives in *Prague* and owns a car of an unknown make. This is all the information we can deduce based on incomplete information about the the global relations provided by the sources. □

In [GK02] a methodology is given for constructing a rewriting $\widetilde{\varphi}$ of a query φ, applicable directly on the sources $\mathcal{S}$, such that

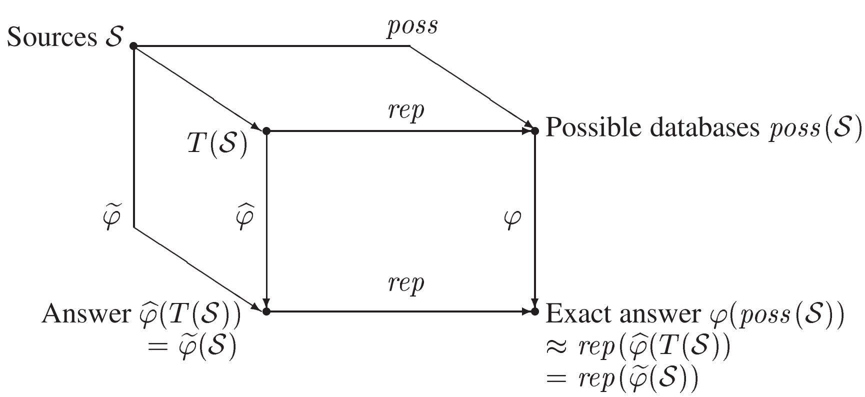
$$\widetilde{\varphi}(\mathcal{S}) = \widehat{\varphi}(T(\mathcal{S})). \tag{11}$$

Since Equation (10) says that $\mathit{rep}(\widehat{\varphi}(T(\mathcal{S}))) \approx \varphi(\mathit{poss}(\mathcal{S}))$, Equation (11) now gives

$$\mathit{rep}(\widetilde{\varphi}(\mathcal{S})) \approx \varphi(\mathit{poss}(\mathcal{S})). \tag{12}$$

Example 10: In Example 9 the method in [GK02] would produce as rewriting the union of $Q(X,Y,Y_1) \leftarrow S_1(X,Y)$ and $Q(X,Y_2,Z) \leftarrow S_2(X,Z)$. The unsafe variables Y_1 and Y_2 represent unknown values. When evaluating such unsafe queries, an unsafe variable in the head of the query essentially generates a new variable in the answer. The answer is then a tableau, with the variables representing missing values. Applying the rewriting to the sources in Example 9 will indeed produce the result $\{Q(\textit{Sue, Mazda}, Y_1), Q(\textit{Mary},Y_2, \textit{Prague})\}$. □

Query answering in Information Integration systems can now be summarized by the following diagram.



Acknowledgments

Many thanks to Victoria Kiricenko and Alberto Mendelzon for comments on an earlier version of this paper.

References

[AD98] Abiteboul S. and O. M. Duschka: Complexity of Answering Queries Using Materialized Views. In *Proc. 17th Annual ACM Symp. Principles of Databases (PODS '98)*, Seattle, Washington 1998, pp. 254–263.

[AHV95] Abiteboul S., R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading Ma. 1995.

[ALM02] Afrati F., C. Li and P. Mitra Answering Queries Using Views with Arithmetic Comparisons. In *Proc. 21st ACM Symp. on Principles of Database Systems (PODS '02)*, Madison, Wisconsin 2002, pp. 209–220.

[ALU01] Afrati F., C. Li and J. D. Ullman. Generating Efficient Plans for Queries Using Views. In *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD '01)* Dallas, Texas 2001.

[DG98] Dushka O. M. and M. R. Genesereth. Query Planning with Disjunctive Sources. In *Proceedings of the AAAI Workshop on AI and Information Integration*, Madison, Wisconsin 1998.

[Gra84] Grahne G. Dependency satisfaction in databases with incomplete information. *Proc. of the 10th International Conference on Very Large Databases (VLDB '84).* Singapore 1984, pp. 37–45.

[Gra91] Grahne G. *The Problem of Incomplete Information in Relational Databases*. Lecture Notes in Computer Science, vol. 554. Springer-Verlag, Berlin 1991.

[GM99] Grahne G. and A. O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. In *Proc. 7th International Conference on Database Theory (ICDT '99)*. Delphi, Greece 1999, pp. 332–347.

[GK02] Grahne G. and V. Kiricenko. Obtaining more answers from Information Integration Systems. *Proc. Fifth International Workshop on the Web and Databases (WebDB '02)*. Madison, Wisconsin 2002, pp. 67–76.

[Hal00] Halevy A. Y. Theory of Answering Queries Using Views. *SIGMOD Record* 29(4): 40-47 (2000).

[Hal01] Halevy A. Y. Answering queries using views: A survey. *VLDB Journal* 10(4): 270-294 (2001).

[IL84] Imielinski T. and W. Lipski Jr. Incomplete Information in Relational Databases. *J. ACM* **31**:4, 1984, pp. 761–791.

[Len02] Lenzerini M. Data Integration: A Theoretical Perspective. Invited tutorial in *Proc. 21st ACM Symp. on Principles of Database Systems (PODS '02)*. Madison, Wisconsin 2002, pp. 233–246.

[LMSS95] Levy A. Y., A. O. Mendelzon, Y. Sagiv and D. Srivastava. Answering Queries Using Views. In *Proc. 14th ACM Symp. on Principles of Database Systems (PODS '95)*. San Jose, California 1995, pp. 95–104.

[LRO96] Levy A. Y, A. Rajaraman, J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. *Proc. 22nd Int'l. Conf. on Very Large Databases (VLDB '96)*, pp. 251–262, Mumbai (Bombay), India 1996.

[Men84] Mendelzon A. O. Database States and Their Tableaux. In *ACM Trans. on Databases Systems* **9**:2, 1984, pp. 264–282.

[MM01] Mendelzon A. O. and G. Mihaila. Querying Partially Sound and Complete Data Sources. In *Proc. 20th ACM Symp. on Principles of Database Systems (PODS '01)*, Santa Barbara, California 2001, pp. 162–170

[PL00] Pottinger R. and A. Y. Levy. A Scalable Algorithm for Answering Queries Using Views. In *Proc. 26th Int'l. Conf. on Very Large Databases (VLDB '00)*, Cairo, Egypt 2000, pp.484–495.

[Rei78] Reiter R. On Closed World Databases. In *Logic and Databases*, H. Gallaire and J. Minker (Eds.), pp. 56–76, Plenum Press, New York, 1978.

[Ull97] Ullman J. D. Information Integration Using Logical Views. In *Proc. 6th International Conference on Database Theory (ICDT '97)*. Delphi, Greece 1997, pp. 19–40.

[Var86] Vardi M. Y. Querying Logical Databases. *Journal of Computer and System Sciences* 33, pp. 142–160, 1986.

[YL87] Yang H. Z. and P. A. Larson. Query Transformation for PSJ Queries. In *Proc. 13th Int'l. Conf. on Very Large Data Bases (VLDB '87)* Brighton, England, pp. 245–254.

ICDE 2003

CALL FOR PARTICIPATION

March 5 - 8th 2003

Bangalore, Karnataka, India

http://www.aztecsoft.com/icde2003

The **ICDE 2003 International Conference on Data Engineering** will be held in Bangalore, Karnataka, India, the center of Indian software activity with a variety of historical and natural attractions within striking distance.

ICDE 2003 aims at providing a premier forum for:

- presenting new research results
- exposing practicing engineers to evolving research, tools, and practices and providing them with an early opportunity to evaluate these
- exposing the research community to the problems of practical applications of data engineering
- promoting the exchange of data engineering technologies and experience among researchers and practicing engineers

Keynote Speakers

- Bruce Croft, UMass, USA
  Language Modeling for Information Retrieval
- Ramesh Jain, Georgia Tech, USA
  Event Based Management of Multimedia Data

Program

- 50 Technical Papers
- 30 Technical Posters
- 7 Tutorials
- 2 Panels
- Industry/Application Papers

Workshop

13th RIDE – Multi Lingual Information Management, March 10-11th, 2003, Hyderabad, INDIA.
http://www.iiit.net/conferences/ride2003.html

Tutorials

Application Servers and Associated Technologies
C. Mohan, IBM Almaden
Business Process Management Systems,
Anil K. Nori, Microsoft Corp., Former CTO (and founder) Asera Inc.
Database Technologies for eCommerce
Rakesh Agrawal, IBM Almaden
Data Engineering for Mobile and Wireless Access
Panos K. Chrysanthis, Univ. of Pittsburgh, Vijay Kumar, Univ. of Missouri- KC
Evaggelia Pitoura, University of Ioannina, Greece
Sequence Data Mining Techniques and Applications
Sunita Sarawagi, IIT Bombay
Storage and Retrieval of XML Data using Relational Databases
Surajit Chaudhuri, Microsoft Research, Kyuseok Shim, Seoul National University
Approximate Matching in XML
Sihem Amer-Yahia, Nick Koudas and Divesh Srivastava, AT&T Labs--Research

Visit Karnataka
A Land of
Heritage,
Monuments,
Temples,
Wildlife,
Sea coast,
and
Hill stations.

http://kstdc.nic.in/

Panels

Cyberinfrastructure for Bioinformatics
Databases for ambient Intelligence

General Co-chairs:
Deepak Phatak, IIT Bombay, India
Marek Rusinkiewicz, Telcordia, USA

Program Co-chairs:
Umesh Dayal, HP Laboratories, USA
Krithi Ramamritham, IIT Bombay, India

Industry Program Chairs:
C. Mohan, IBM Almaden Research Center
N. Ramani, HP Laboratories, India

Advanced Technology Seminars Chars:
Alex Buchmann, Tech. Univ. Darmstadt, Germany
S. Sudarshan, IIT Bombay, India
Vijay Kumar, Univ. Missouri, Kansas City, USA

Local Arrangements:
V. R. Govindarajan, Aztec Software, India
Jayant Haritsa, IISc. India

Publicity Chairs:
Fabio Casati, HP Laboratories, USA
Kamal Karlapalem, Intl.IIT, India
Kam-Yiu Lam, City Univ. HKSAR

Panels Chairs:
Sham Navathe, Georgia Tech., USA
Hongjun Lu, HKUST, HKSAR
Gerhard Weikum, Univ. Saarbruecken, USA

Demonstration Chairs:
Panos Chrysanthis, Univ. of Pittsburgh, USA
Srinath Srinivasa, IIIT, Bangalore, India

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903