

On the Importance of Updates in Information Integration and Data Exchange Systems

Yannis Velegarakis

Department of Information Engineering and Computer Science
University of Trento
Via Sommarive 14, 38100 Trento, Italy
velgias@disi.unitn.eu

Abstract. During the last two decades we have witnessed great advances in the area of information integration and data sharing. We already know how to efficiently query distributed data through global integrated schemas and through the use of P2P architectures. Despite these developments, updates have not received considerable attention. By updates, we mean the ability to modify the data the same way we query it. Updates are an important aspect of every integration system since integration offers a great opportunity to discover inconsistencies that could not be detected in the individual sources. They are also important to P2P systems for keeping data located in different sources synchronized. Furthermore, in DBMSs that only expose their data through views, updates on the views is the only way for a person or an application to modify the data. In this work we are arguing on the importance of updates, we illustrate the different dimensions of the problem, highlight the main challenges and describe solutions that we have developed.

1 Introduction

The advent of the web allowed thousands of data providers to make their data sources available on-line. These sources are highly heterogeneous. The heterogeneity is due to the fact that these sources have been developed at different times, by different organizations and with different requirements in mind. On the other hand, for organizations and individuals alike, it is critically important to be able to locate, retrieve and integrate this data in order to present it in a uniform way. Integration systems have been developed for that purpose. Traditionally, integration systems were managed using a federated architecture where a virtual (global) schema is created over a set of (local) source schemas [1]. Mappings are the logical expressions that are used to describe the relationships between the local and the global schemas [2–5]. This relationship explain how data conforming to the structure of the local schema can be transformed into a structure conforming to the global schema. Depending on whether the global schema is expressed in terms of the local schemas or vice-versa, the mappings are referred to as *global-as-view (GAV)* [6] or *local-as-view (LAV)* [7], respectively. The first favors query answering, while the second favors run-time introduction

of new sources. A more expressive form of mappings has been recently introduced under the term *global-local-as-views (GLAV)* [8]. A GLAV mapping is of the form $Q^S \rightsquigarrow Q^T$ where Q^S is a query on the source schema and Q^T is a query on the global schema.

Data integration through the federated architecture became inconvenient in the presence of the myriad loosely-coupled autonomous sources that exist on the web, since run-time coupling of data sources is a hard task. The grid and peer-to-peer (P2P) systems [9] were introduced as an alternative to the federated architecture. In a P2P architecture each source (peer) is associated to only a number of other sources (peers), referred to as *acquaintances*, forming a network of interlinked sources. A query sent to a source is then propagated through these links to other sources that produce partial results that are returned back to the origin, merged and presented to the entity (person, program or organization) that initiated the query [10]. Similar to federated systems, the relationship between data in two different peers is also expressed through mappings.

Most of the research in federated or P2P systems has concentrated on developing advanced query mechanisms [3, 6, 5]. The goal was to shield the user from the heterogeneity and allow the management of the different data source as one centralized database. In this effort, updates were to a large degree ignored. This was mainly due to the fact that data ownership, and consequently, data updates, were considered to be a sole responsibility of the individual data owner and not of the integrated system user. However, updates are important for many reasons. They allow the correction of inconsistencies that can only become visible when the data is integrated, and they can propagate changes between sources to keep their data instances synchronized. What makes the problem of updates even more challenging is the fact that updates can occur not only on the data values, but on their schemas and also on their semantics of the values. We elaborate next in more details on the benefits of updates in real applications so that we better realize their importance.

Information Integration Systems. Data Integration offers a great opportunity to detect data inconsistencies that are hard to find otherwise, because they occur among different data sources. Once these inconsistencies are detected, updates may be issued on the integrated data to eliminate them. As happens also with the queries, these updates will have to be propagated to the individual sources in order to be applied to the actual materialized instances. If the relationships between the global and the local schemas are expressed as LAVs, then view update techniques [11] can be employed for that purpose. On the other hand, if the relationship is expressed as GAV then the problem of update propagation is becoming equivalent to the problem of updates through views [12]. Most of the existing integration systems nowadays do not support such updates, which keeps them far from becoming full-fledged data management solutions.

Schema Integration. Schema integration is the process through which the global schema of an information integration system is constructed along with its relationships to the local schemas of the individual sources. Numerous algorithms and tools have been proposed to automate or semi-automate schema

integration [13]. However, at its core, schema integration is a schema design problem. Some integration choices will necessarily be subjective and different users or designers may wish to make different choices or alter a heuristic choice made by a tool. Some tools anticipate this and for a limited set of alternative designs, will still produce a correct mapping between the source and the integrated schema [13]. Others will permit users to use a set of composable schema transformation operators to produce an integrated (transformed) schema (with a composed mapping) [14]. However, these approaches, in general, do not permit arbitrary changes to the integrated schema. Even a simple horizontal decomposition of an integrated table based on a user-defined predicate will typically require the designer to manually edit the mapping. Furthermore, changes in the source schema (even modest ones) are not supported. Such changes require the schema integration algorithm to be rerun.

Data Exchange and P2P Systems. In data exchange, mappings are used to transform an instance of a source schema into an instance of a different target schema [15, 2] or to keep the data of two peers in coordination. Designers may wish to modify either the source or target schema or the data instance. If the change happens on the schema, then some of the existing mappings may become inconsistent and will have to be modified. If the change happens on the data, then through the mapping it will have to propagate to the instance of the other source. Thus efficient and effective methods of adapting the existing mappings and propagating data updates are required [16].

Physical data design. Physical storage wizards, which permit the customization of physical schemas and storage structures, must maintain a mapping between the physical and logical schemas. A common example of such wizards are tools for customizing the relational storage of XML data. Such tools evaluate (or help a designer to evaluate) the relative cost of different physical relational designs. However, they consider only a fixed large set of physical schemas, each with a built-in mapping to the given logical (XML) schema. To permit a designer to suggest schema designs outside of this limited set, the tool would have to be able to adapt the XML to relational mapping to the *ad-hoc* user-proposed schema changes.

System consolidation. Large organizations may often operate on many independent data management systems with outdated hardware. This costs millions to the companies or the organizations since it creates inconsistencies and performance problems. For that reason, it is not rare the case that an organization consolidates the data management infrastructures of many of its independent systems into a modern centralized repository. Then the various applications throughout the company need to be modified to access that new repository. Since the repository is designed with new requirements in mind and contains many different kinds of information, the way the company applications were interacting with their repositories will also have to be modified as well. Such modifications are also costly. One way to overcome this obstacle is to create views on the centralized repository that have the same schema as the one of the individual data sources previously used by the applications. Then the ap-

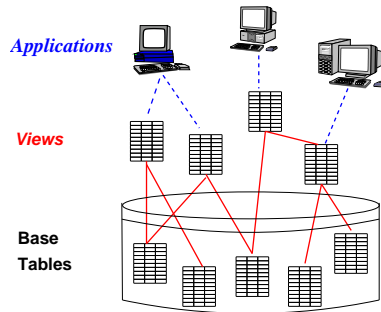


Fig. 1. Accessing Information Through Views

plications can continue to operate as before, without realizing that instead of accessing their initial data repository they are accessing a view on the centralized one. Figure 1 illustrates this situation. Of course, the applications typically send not only queries but also updates. Thus, in order for the sources to continue to operate as before, the system needs to support updates on the defined view that will have to propagate down to the tables in the repository.

Having argued on the importance of updates and the urgent need to be taken into consideration in information integration and P2P systems, we make in Section 2 a brief illustration of the research areas related to updates. In the sequel, we present three main challenges in supporting updates, namely, updates on the views (Section 4), updates on the schemas (Section 3) and updates on the semantics of the data values (Section 5), and we illustrate solutions we have developed for each one respectively. We conclude with a short review and the main messages that we aim to deliver through this work.

2 Related Work

The notion of updates is as old as the notion of queries and data management itself. Over the years a number of researchers have studied different aspects of updates in different contexts. We briefly explain here the different aspects that have been studied and the areas for which further studies are required. Figure 2 provides an illustration of the different parts of an integration architecture that each area has studied. The circle indicates where the updates occur and the question mark the part that needs to be modified as a consequence of the update.

When a database schema is evolving, its instance data needs to also be modified in order to conform to the evolved schema. Schema evolution deals with the problem of minimize the cost of updating the instance data when the schema is modified. Most of the solutions to this problem consider a set of primitive changes and provide an implementation of each such change on the instance [17–19].

Changes may occur not only at the schema level, but also at the instance data. In case of a materialized view, the materialization needs to be modified

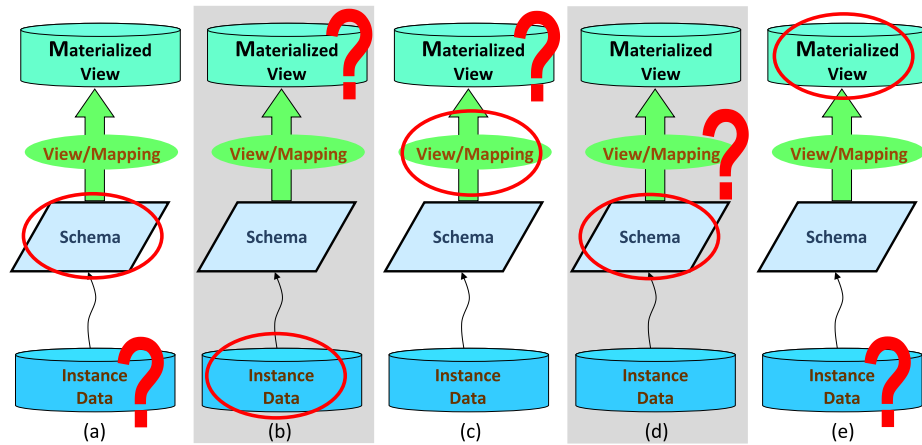


Fig. 2. Research areas related to updates: (a) Schema evolution, (b) Incremental View Maintenance, (c) View Adaptation, (d) Mapping Adaptation, (e) Updates Through Views

in order to remain consistent to the view definition and the instance data. One can always drop the materialization and recreate it from scratch, but this is not a desired solution. Maintaining the materialized view incrementally, known as the problem of *view maintenance*, is a hot topic that has been studied for many years [20, 11, 21].

The opposite of view maintenance is referred to as *updates through views* [22]. It is the field that studies ways to update the instance data when an updates is issued on the view. Existing literature [12, 23] has shown that for many practical cases there may be no solution, which may be one of the main reasons that updates in integration systems have not been studied as much as the queries.

View adaptation [24, 25] is a variant of view maintenance and studies ways of keeping the data of a materialized view up-to-date in response to changes to the view definition itself.

When the schema of a data source is modified, some of the mappings that have been defined on the schema may become invalid. When this happens, a data administrator that has a good knowledge of the semantics of the schema and of the mappings can redefine them in order to work for the new schema. *Mapping adaptation* [26] (also found in the literature under the term *view synchronization*) is the field that studies techniques and methodologies to automate this task. Existing techniques [27] require the system administrator to specify at mapping definition time the way the mapping will have to respond to each schema change. The limitation of this is that the administrator cannot predict at design time the semantics of the updates that occur on the schema, and based on these semantics there may be many alternative mapping modifications.

3 Schema Updates

To deal with the evolving nature of schemas, it is important to be able to modify the mappings defined over the schemas, as the schemas evolve. One way to approach this problem is to have a predefined set of interesting changes. For each such change, there is a, typically “hard-coded”, method specifying how the mappings should be modified in response to the specific change. The advantage of this approach is that it is known in advance how exactly the schema will react to each change. The disadvantage is that it requires every schema modification to be expressed as a sequence of these modification. Furthermore, since the modifications are all hard coded, the system is not flexible to the semantics of the actual modification.

A second alternative is to allow schemas to evolve and then find the changes that took place by comparing the modified schema S' to the original version S . For example, one could use a matching tool to find corresponding portions of the two schema versions [28] and then use a mapping creation tool to add semantics to these correspondences [29]. This will produce a mapping from S' to S which can be composed with the original mapping. One issue that needs to be considered in that case, is the situation in which there are more than one ways that lead from the schema S to S' . Each one of them may imply different modifications to the mappings, thus selecting the right one is a challenge. This approach is inevitable, of course, in cases in which the sequence of individual changes that has taken place is not available but only the initial and final version of the schema [30].

ToMAS [31] is a system designed to maintain the consistency of the mappings while the schemas are modified. It is based on the first approach but avoids some of its drawbacks. It first detects mappings that are becoming inconsistent due to the schema changes and then suggests modifications. The modifications can be structural changes like element insertion, deletion or renaming, or modification on the schema constraints. ToMAS bases its decisions on the notion of *association*. An association is a set of semantically related schema elements. Naturally, all the attributes of a relational table belong to the same association, since the fact that the schema designer has chosen to put them all together under one table (or element in the case of XML schemas), means that they are somehow semantically related. ToMAS extends this notion for nested schemas, and also for attributes in different tables. The latter is based on the fact that attributes in two different tables are related if and only if there is a join path between them. Such associations are formed by starting with the associations formed by sets of attributes that are in the same table (or element) and extending them by using the traditional chase technique [32]. This extension allows ToMAS to deal with changes that are not only structural, but also semantic, and in particular, addition or deletion of foreign key constraints.

One of the unique advantages of ToMAS is that it can track semantic decisions made by the mapping designer during the definition of the mapping and reuse them. These semantic decisions are important to be preserved because schemas are often ambiguous (or semantically impoverished) and may not con-

tain sufficient information for a mapping modification algorithm to make the right mapping choices. ToMAS adapts the mappings by adapting first the affected associations, by performing the minimum required computations to bring them to the new schema structure and semantics. The affected mappings are those that are using affected associations. Based on how these associations have changed, a number of candidate rewritings are generated. To achieve the preservation of the designer choices, among these rewritings, ToMAS chooses the one that is semantically closer to the original one. And by semantically closer, it means the one that differs less in terms of structure and schema constraints.

The following example illustrate how ToMAS works. Assume the three tables $R(A, B, C)$, $S(D, E, F)$ and $T(G, H)$ of a schema with the foreign key constraints $D \rightarrow C$, $E \rightarrow C$ and $G \rightarrow F$. Let the following mapping M be one that has been defined on the specific schema. The mapping joins the three tables and maps the attributes A and H into a table $U(A, H)$ which can be a table in the global schema of an integration system. The mapping expression is the following:

$$U(A, H) : -R(A, B, C), S(D, E, F), T(G, H), G = F \wedge D = C \quad (1)$$

Notice that, although tables R and S can join either on attribute D (due to the foreign key $D \rightarrow C$), or E (due to the foreign key $E \rightarrow C$), the mapping specifies that the join be on D . Now assume that the schema is modified by the deletion of attribute F . Naturally the above mapping becomes inconsistent since the join between S and T cannot be performed anymore. ToMAS detects this and generates one mapping that involves only T and one that involves R and S (the minimal modifications principle instructs that there is no reason for breaking up this later join). Furthermore, despite the two ways that R and S can join, ToMAS selects the one that is based on attribute D in order to be semantically closer to the original mapping which was also joining R and S based on attribute D . Thus, the original mapping that became inconsistent will be replaced by the following two mappings:

$$U(A, H) : -T(G, H) \quad (2)$$

$$U(A, H) : -R(A, B, C), S(D, E, F), D = C \quad (3)$$

One would expect from a mapping adaptation tool like ToMAS to generate mapping rewritings equivalent to the affected mapping. Unfortunately, this may not always be possible, which is natural to happen, since when a schema is changing, inevitably, its semantics are becoming different and the initial semantics may not apply any more. In such a case, it may make no sense to look for equivalent rewritings.

4 Data Updates

The problem of propagating changes through the mappings, highly depends on the direction of the mappings. Changes at the source data propagated to the vir-

tual view of a GAV mapping require view maintenance techniques. The opposite direction is harder and requires techniques for updating views.

Given an instance I , a view $V(I)$ defined over I and an update U , the problem of updates through views is defined as the problem of finding an update W on the instance I such that when the update is applied on the instance I , the result of applying the view V on the modifying instance is the same one would have resulted if the update U had been applied on a materialized instance of $V(I)$. In other words, $U(V(I)) = V(W(I))$. If this result is not achieved, then the update W is said to generate side-effects.

There are many kinds of views for which side-effects are unavoidable for certain updates [33, 34, 22]. Side-effects are highly undesired if updates are to be used in an integration system to propagate updates issued on the views to the data sources. Consider, for instance, a view that joins a table **Person** and a table **House** and a delete request for a tuple t_{ph} in the view that has been formed by the pairing of the tuples t_p and t_h of the table **Person** and **House**, respectively. The deletion of t_{ph} from the view can be achieved by removing either tuple t_p from **Person** or t_h from **House**. If t_p joins with many other tuples of **House**, then its removal will result to additional deletions from the view. The same applies for tuple t_h . It will be very strange for a user of an integration system who is accessing the data through the integrated schema, to send the update on the integrated schema/instance and then see additional tuples being removed from it, apart from those specified by her update.

A solution that has typically been used in practice is to allow side-effects to happen, to develop algorithms to detect them prior of happening and warn the user [35], or to restrict the kind of updates that can be performed on a view [22]. We believe that either of these solutions is acceptable for applications such as integration or data exchange systems, due to the fact that the main purpose of an integration system is to shield the user from the individual sources and allow her to interact with the global schema as if she would have interacted with any regular database system.

The above have led us to explore new ways of supporting updates on the views. One observation that can be made is that the reason of the side-effects is the fact that the instance of a view is required to be equal to the result of the evaluation of the view query on the source data. In other words, the view query is a function that determines the view instance. The side-effects occur when an update issued on the view instance brings the view into a state for which there is no source instance data that when the view query is applied to it, the result is the updated view instance. Given this, we have introduced the notion of cTables [36]. A cTable is like a view, with the only difference that its instance may not be exactly equal to the instance of its query on the instance data. The relationship between the cTable instance and the evaluation of its query, is governed by the property of well-foundedness. According to this, no tuple should appear in the cTable unless its appearance is justified by the appearance of certain tuples in the instance data. This translates to:

$$I_{cTable} \subseteq Q_{cTable}(I) \tag{4}$$

where I_{cTable} is the instance of the cTable and Q_{cTable} is its query.

The full semantics of cTables can be found in [36]. There are two ways one can implement these semantics. One is the use of deltas. Since $I_{cTable} \subseteq Q_{cTable}(I)$, the instance of a cTable at any given time can be determined from by evaluating the cTable query over the instance data and then removing a numbers of tuples that are recorded in the delta. Thus, in an information integration system one can associate to every view (relation) of the global schema, a delta relation that will serve for that purpose.

The second implementation approach is to use a specialized physical schema. This is possible due to one of the main principles of database systems, that there should be a clear separation between the physical and the logical level. The basic logical relations are defined as queries on the physical instances as usual. The cTables, on the other hand, instead of being defined as queries on the logical base relations (which is what happens with the traditional views), are also defined as queries directly on the physical level. Figure 3 provides a graphical illustration of this difference, but the details of the implementation can be found elsewhere [36].

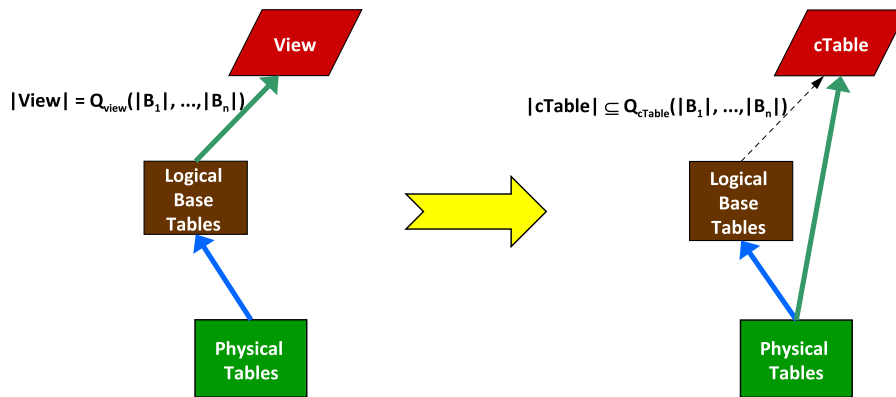


Fig. 3. Difference between cTables and traditional views.

5 Semantic Updates

A first step in the design of every (relational) database schema is the design of an ER schema. The ER schema is a conceptual schema that models the real world data stored in the database and is used to communicate its semantics to users and applications. Apart from the ER diagrams, numerous other formalisms have been developed over time to model the semantics of the relational data and not only. This semantic information is important for query formulation and query answering. However, once the query is formulated, no further interpretation is

done on the data values. Everything is based on value equivalence. Consider, for instance, an application looking for a person named John. It issues a query on a table “Owner” with the condition that the value of attribute “Name” is “John”. Only tuples that satisfy that condition will be returned. Other tuples that have the name “Gianni”, “Giovanni” or “Jose” will never be returned, although semantically they all mean “John”. Furthermore, there may be companies named “John” and will also be returned in the result set. So, it becomes more and more important to be able to identify the actual meaning of the data values, in addition to the one provided by the database schema. To tackle that issue, ad-hoc approaches may be used, schemas may be accompanied by ontologies [37], or data may be annotated with additional meta-information [38, 39].

What these solutions have mainly ignored is the fact that concepts are also evolving over time. Words may lose their meaning, concepts may be merged, new concepts may be generated to mean the same or different thing, etc. Consider, for instance, a query that asks about “Germany”. Entries that refer to the country in the 60s, may not have the string Germany per-se since at that moment the country was known as East Germany and West Germany. However, if we issue a query about the history of Germany today, we would like to also receive information about the East and West Germany. This is becoming an important issue in the current web environments where data is contributed by different people and organizations that have different knowledge, goals, and expertise.

The OKKAM project (www.okkam.org) is a multidimensional project that aims at developing solutions for identifying entities on the web [40]. Since there are no global identifiers on the web, an entity is characterized by a number of descriptions (attributes). Over time, these descriptions may change, new may be added and old may be removed. This means that the meaning of the entity may change over time. Different entities may be proven to represent the same thing, or one entity may be found to represent two different real world objects and may have to be split in two. This kind of semantic changes that may occur to an entity have serious consequences to the applications that are using them and to queries that may be posed on the system. To deal with that, OKKAM provide explicit constructs to model semantic evolution in a way that is transparent to the actual users and in a way that does not affect the operation of the existing applications.

6 Conclusions

In this work we have elaborated on the importance of updates. We have identified the limited research that has been performed for updates within information integration and data exchange systems. We argued that updates are critically important for such systems, either in order to be able to keep data instances synchronized, or to allow users and applications to treat integration systems as one monolithic data management solution, which is actually the main goal of such systems. We showed that updates can happen at three different levels: at

the data (i.e., instance), at the schema and also at the semantics of the actual values. For each such case we presented solutions we have developed or we are currently developing.

Acknowledgements: This work has been partially funded from the EU grants GA-215032, ICT-215874 and MIRG-CT-2006-046548.

References

1. Heimbigner, D., McLeod, D.: A federated architecture for information management. *ACM Trans. Inf. Syst.* **3** (1985) 253–278
2. Alexe, B., Tan, W., Velegarakis, Y.: Stbenchmark: Towards a benchmark for mapping systems. In: *VLDB*. (2008)
3. Beneventano, D., Bergamaschi, S., Guerra, F., Vincini, M.: The momis approach to information integration. In: *ICEIS* (1). (2001) 194–198
4. Hernandez, M., Papotti, P., Tan, W.: Data exchange with data-metadata translations. In: *VLDB*. (2008)
5. Tatarinov, I., Ives, Z., Madhavan, J., Halevy, A., Suciu, D., Dalvi, N., Dong, X., Kadiyska, Y., Miklau, G., Mork, P.: The Piazza peer data management project. *SIGMOD Record* **32** (2003)
6. 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. *Journal of Intelligent Information Systems* **8** (1997) 117–132
7. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying Heterogeneous Information Sources Using Source Descriptions. In: *VLDB*. (1996) 251–262
8. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: *PODS*. (2002) 233–246
9. Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing : A vision. In: *WebDB*. (2002) 89–94
10. Kementsietsidis, A., Arenas, M., Miller, R.J.: Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In: *SIGMOD Conference*. (2003) 325–336
11. Ceri, S., Widom, J.: Deriving Production Rules for Incremental View Maintenance. In: *VLDB*. (1991) 277–289
12. Dayal, U., Bernstein, P.A.: On the Updatability of Relational Views. In: *VLDB*. (1978) 368–377
13. Spaccapietra, S., Parent, C.: View Integration : A Step Forward in Solving Structural Conflicts. *TKDE* **6** (1994) 258–274
14. Gyssens, M., Lakshmanam, L., Subramanian, I.N.: Tables as a Paradigm for Querying and Restructuring. In: *PODS*. (1995) 93–103
15. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data Exchange: Semantics and Query Answering. In: *ICDT*. (2003) 207–224
16. Andritsos, P., Fuxman, A., Kementsietsidis, A., Miller, R.J., Velegarakis, Y.: Kanata: Adaptation and evolution in data sharing systems. *SIGMOD Record* **33** (2004) 32–37
17. Banerjee, J., Kim, W., Kim, H., Korth, H.F.: Semantics and implementation of schema evolution in object-oriented databases. In: *SIGMOD*. (1987) 311–322

18. Lerner, B.S.: A Model for Compound Type Changes Encountered in Schema Evolution. *ACM TODS* **25** (2000) 83–127
19. McBrien, P., Poulouvasilis, A.: Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach. In: *CAiSE*. (2002) 484–499
20. Blakeley, J.A., Larson, P.A., Tompa, F.W.: Efficiently Updating Materialized Views. In: *SIGMOD*. (1986) 61–71
21. Mumick, I.S., Quass, D., Mumick, B.S.: Maintenance of Data Cubes and Summary Tables in a Warehouse. In: *SIGMOD*. (1997) 100–111
22. Keller, A.M.: Algorithms for Translating View Updates to Database Updates for Views Involving Selections, Projections, and Joins. *SIGMOD* (1985)
23. Cosmadakis, S., Papadimitriou, C.: Updates of Relational Views. In: *PODS*. (1983) 317
24. Gupta, A., Mumick, I., Ross, K.: Adapting Materialized Views After Redefinition. In: *SIGMOD*. (1995) 211–222
25. Mohania, M.K., Dong, G.: Algorithms for Adapting Materialised Views in Data Warehouses. In: *CODAS*. (1996) 309–316
26. Velegrakis, Y., Miller, R.J., Popa, L.: Mapping Adaptation under Evolving Schemas. In: *VLDB*. (2003) 584–595
27. Lee, A.J., Nica, A., Rundensteiner, E.A.: The EVE Approach: View Synchronization in Dynamic Distributed Environments. *TKDE* **14** (2002) 931–954
28. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. *VLDB Journal* **10** (2001) 334–350
29. Popa, L., Velegrakis, Y., Miller, R.J., Hernandez, M.A., Fagin, R.: Translating Web Data. In: *VLDB*. (2002) 598–609
30. Yu, C., Popa, L.: Semantic adaptation of schema mappings when schemas evolve. In: *VLDB*. (2005) 1006–1017
31. Velegrakis, Y., Miller, R.J., Popa, L., Mylopoulos, J.: ToMAS: A System for Adapting Mappings while Schemas Evolve, (System Demonstration). In: *ICDE*. (2004)
32. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing Implications of Data Dependencies. *ACM TODS* **4** (1979)
33. Dayal, U., Bernstein, P.: On the Correct Translation of Update Operations on Relational Views. *ACM TODS* **8** (1982) 381–416
34. Tomasic, A.: Correct View Update Translations via Containment (1993)
35. C. Medeiros and F. Tompa: Understanding The Implications Of View Update Policies. In: *VLDB*. (1985)
36. Kotidis, Y., Srivastava, D., Velegrakis, Y.: Updates Through Views: A New Hope. In: *ICDE*. (2005) 2
37. An, Y., Borgida, A., Miller, R.J., Mylopoulos, J.: A semantic approach to discovering schema mapping expressions. In: *ICDE*. (2007) 206–215
38. Srivastava, D., Velegrakis, Y.: Intensional associations between data and metadata. In: *SIGMOD Conference*. (2007) 401–412
39. Velegrakis, Y., Miller, R.J., Mylopoulos, J.: Representing and Querying Schema Transformations. In: *ICDE*. (2005) 81–92
40. Palpanas, T., Chaudhry, J., Andritsos, P., Velegrakis, Y.: Entity data management in okkam. In: *DEXA Workshops*. (2008) 729–733