

Encyclopedia of Database Systems
Springer Science+Business Media, LLC 2009
10.1007/978-0-387-39940-9_847
LING LIU and M. TAMER ÖZSU

Updates through Views

Yannis Velegarakis¹

(1) University of Trento, Trento, Italy

Without Abstract

Definition

Views are windows to a database. They provide access to only a portion of the data that is either of interest or related to an application. Views are relations without independent existence, as is the case of database tables created with the “create table” command. The contents of the instance of a view are determined by the result of a query on a set of database tables, typically referred to as *base tables*. Applications deal with views the same way they deal with any other relation. In fact, an application is rarely aware of whether a relation it is accessing is a base table or a view. This means that the application may issue updates on the view relation as it would have done with any other database table. Since the view instance depends on the instances of the base tables, to execute an update on the view one needs to find a number of base table modifications whose effect on the view instance is the modification described by the update command.

Historical Background

Views are one of the oldest concepts in computer science. They appeared almost at the same time with queries. In the early 1980s, as database vendors were moving towards the relational model, Codd introduced twelve rules that need to be satisfied by a database management system in order to be considered *relational*. Rule number six was referring to the concept of views and in particular, to the ability of the views to be updated when they are *theoretically updatable*, a concept that will be explained below.

The scientific literature contains different definitions (and uses) of the term “view,” with two of them being dominant. The first is that a view is a relation whose instance depends (somehow) on the data of its base tables, and the second is that a view is a short-hand for a query. For query answering, those two definitions are equivalent, but for the purpose of updating, the definition that is considered makes a difference. They have different consequences. Considering the view as a query, implies that at any given point in time the instance of a view is fully specified by the results of the view query over the data, i.e., the set of possible instances of a view is the set of possible relations that can be generated as a result of the view query. A consequence of this is that the view might not be *theoretically updatable*, which means that there are updates on the view that cannot be translated to updates on the base tables.

Recently, Kotidis et al. [8] have relaxed that requirement to accept views whose instance may be different from the result of their view query in order to allow arbitrary updates on the view.

Update propagation is one of the main issues in view updates. When multiple views share the same piece of data from the base tables, all these views should have consistent instances. This means that when some part of the data is updated, this change must propagate accordingly to the base tables and the other views. For the case of materialized views, and updates on the base tables, this is a well-studied problem, namely the view maintenance [10]. View maintenance has considered how to change the instances of the views in response to base table modifications. However, updates on the views have not been studied to the same extend.

Foundations

A *database* $D = \langle S, I \rangle$ is a collection of data along with a description of its structure. The collection of data is referred to as the *database instance* I and the description of its data structures as the *database schema* S . In the relational model, a database instance is a set of relations, and the schema is the set of schemas of these relations. The contents of a relation, i.e., its instance, may be explicitly specified by a user or an application, or they may be derived from the contents of other relations. Relations of the first kind are those constructed using the “create table” command and are often referred to as *base tables*. Relations of the second kind are referred to as *views*. A view is accompanied by a query which is referred to as the *view query* and specifies how its contents will be derived from the contents of the base tables.

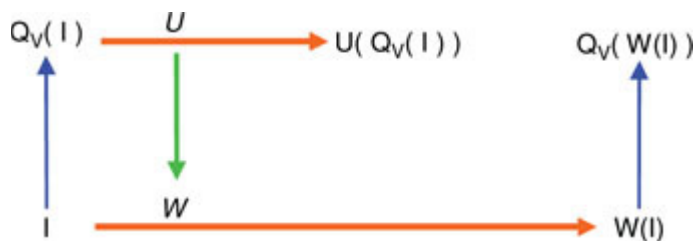
The term “update” on a relation implies an insertion, a deletion, or a value modification of one or more tuples. The literature often chooses to ignore modifications on the basis that a modification can be modeled as a deletion followed by an insertion. The same assumption is followed here.

Let U denote an update command on a relation R , $|R|$ the instance of R and $|U|$ the tuple(s) that need to be inserted in $|R|$ (if U is an insert command) or deleted from $|R|$ (if U is a delete command).

The *implementation* of an update U on a relation R is a new relation R' such that:

- $|R'| = |R| - |U|$ if U is a delete command, and
- $|R'| = |U| \cup |R|$ if U is an insert command

In the case of a base table, the implementation of an update is straight forward (it is done by directly modifying the contents of the base table instance according to the update command). In the case of a view the situation is different. Since the view instance depends on the instances of its base tables an implementation is performed by translating the view update command into a series of updates on the base tables so that the requested update is observed in the view instance. Figure 1 provides a graphical representation of this notion. $Q_V(I)$ denotes the application of the view query Q_V on the database instance I . The result of this query is the instance $|V|$ of the view V . Given an update command U on the view V , the *view update problem* is defined as the problem of finding an update W , referred to as the *translation* of U , on the instance I such that when the view query is applied on the new modified instance $W(I)$, the new view instance $|V'| = Q_V(W(I))$ is an implementation of the update U .



Updates through Views. Figure 1 The updates through views problem.

For a given update U , there may be multiple possible translations. The following example illustrates such a situation. Consider a database instance that consists of a table *Personnel* (*Department*, *Employee*) with tuples $\{[\text{Administration}, \text{Smith}], [\text{HR}, \text{Smith}], [\text{Research}, \text{Kole}]\}$ and the view

V_1 : *select Department from Personnel where Employee = "Smith"*

The instance of view V_1 consists of the two tuples $[\text{Administration}, \text{Smith}]$ and $[\text{HR}, \text{Smith}]$. Let U_1 be an update command that requests the deletion of tuple $[\text{Administration}, \text{Smith}]$ from V_1 . There are many possible changes that can be done on the instance of *Personnel* in order to make tuple $[\text{Administration}, \text{Smith}]$ disappear from the instance of view V_1 . The most obvious one is to simply delete tuple $[\text{Administration}, \text{Smith}]$ from *Personnel*. Alternatively, one could update the value of its *Employee* attribute from "Smith" to *null* or to some value other than "Smith." Even changing the *Department* attribute from "Administration" to some other value "Y" will have the desired effect of removing tuple $[\text{Administration}, \text{Smith}]$ from the view. Thus, each of the aforementioned changes constitutes an implementation of the view update U_1 . They are not, however, equivalent. Each translation may be considered appropriate in different situations. The first translation, for instance, achieves the desired result and only this by deleting only one tuple from the database instance. The second achieves the result without reducing the number of tuples in the database instance, and the latter will cause the appearance of the additional tuple $[\text{Y}, \text{Smith}]$ in the view, which makes the specific translation less appealing.

For the case of a view deletion, the first step in finding a view update translation is to find the provenance of the tuples that are to be deleted from the view, i.e., the tuples in the base tables that are responsible for the appearance in the view of the tuples that are to be deleted [4,5]. Researchers have concentrated their efforts in detecting the minimum number of changes that need to be made on the base tables in order to achieve the view update (*source-side-effect*). These changes, as mentioned previously through the example, may result in additional changes in the view instance apart from the one requested by the view update command. Thus, another important desideratum is to find translations that minimize the number of changes in the view (*view-side-effect*). In certain cases, these two desiderata may conflict.

Finding a translation of a view deletion that minimizes the changes in the base tables or/and the view is in general an NP-hard problem, even with monotone view queries. The complexity remains NP-hard even for the very restrictive classes of views with *select-project-join* or *select-join-union* view queries [4].

A different approach to the view update problem is to deal with it at the semantic level. This approach has been advocated by many early researchers [1]. Their position is that at the moment of view definition, the data administrator needs to specify how each update on the view will be translated to updates on the base tables. This is a safe approach that guarantees a semantically correct translation. However, this approach may be hard to apply in certain practical situations. One reason is that views

may have been defined at a point in which updates were not of interest and the correct translation semantics may be difficult to infer. Furthermore, even if the data designer is aware of the existence of updates on the view, she may not be able to predict at the moment of view definition all the different semantics of the possible view update commands. For instance, the deletion of the [Administration, Smith] tuple in the example above may be simply because Smith stopped working for the Administration department, or because he moved from the Administration to another department. Another reason why this approach is difficult to implement is that views may have been introduced as replacements of tables in cases of physical or logical data reorganization, in which case again, more than one alternative translation may be semantically correct.

Keller [7] has provided a detailed classification of the different kinds of view update translations for projections, selections, select-project-join, and select-project-join-union views. Through an interactive procedure with the data administrator at the time of the view definition, the system tries to infer the right translation rules. At the other end, Dayal and Bernstein [6] have studied the classes of views for which the translation of an update is not ambiguous. For the case of translation ambiguity or the case in which it is possible to implement the update but with more changes in the view instance than those the view command requested, then the update may not be allowed.

A different approach in finding a view update translation is to reduce it to a constraint satisfaction problem [9]. Relational views can be represented as conditional tables and a view update can be translated into a disjunction of a number of constraint satisfaction problems (CSPs). Solutions to the CSPs correspond to possible translations. For the case of multiple candidate translations, semantic information to resolve ambiguity can be modeled as additional constraints.

The problem of updates through views appears also in other models, such as the object views or XML. Since most of the studies have been done for the relational model, one way to deal with the problem of updating object or XML views is to abstract it to the relational case [2,3].

Key Applications

Information Integration Systems. IIS are used to provide a unified view of data that is stored in multiple heterogeneous and physically distributed data sources. The majority of such systems are read-only, since data maintenance is considered to be an exclusive responsibility of the owner of each data source. This situation is gradually changing. Integrated views provide an excellent opportunity for detecting errors and inconsistencies among the different data sources. Once these inconsistencies are detected and corrected at the view level, the data in the sources will have to be accordingly updated.

Corporate Environments. Multiple different systems may operate in different parts of an organization, but centralized data management is crucial for decision making and policy implementations. Views provide the mean to propagate changes from the centralized level to the individual systems of the organization.

P2P. P2P systems use views to describe the relationships between the contents of one peer and its acquaintances. When a modification takes place in one of the peers, the views are used to determine its effects (if any) on the contents of its adjacent peers.

Database Management Systems. View support has been prevalent in Database Management Systems. They provide physical data independence, i.e., decoupling the logical schema from the physical design which is usually driven by performance reasons. By controlling access to the views, one can control access to different parts of the data and implement different security policies. It is not uncommon the

case of databases that provide access to their data exclusively through views. For them the ability to perform arbitrary updates on the views is becoming a necessity since the only way to update the stored information is by issuing updates on the views.

WEB. The World Wide Web has evolved to be the world's largest database where organizations and individuals publish their information and data. Web users are then using applications and integration engines to query and retrieve that information. Since the users have no access to the individual sources, the web in its current form is a read-only system, restricting the potential of its million users. If the Web is to be brought from its read-only status into a full-fledged database, allowing updates on the web views to propagate to the sources is a fundamental requirement.




Cross-references

[Provenance](#)

[Side-Effect-Free View Updates](#)

[View Maintenance](#)

Recommended Reading

1. Aaron B., Jeffrey A. Vaughan, and Benjamin C. Pierce. Relational lenses: a language for updatable views. In Proc. 27th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 338–347.
2. Barsalou T., Siambela N., Keller A.M., and Wiederhold G. 1991. Updating relational databases through object-based views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 248–257.
3. Braganholo V.P., Davidson S.B., and Heuser C.A. On the updatability of XML views over relational databases. In Proc. 6th Int. Workshop on the World Wide Web and Databases, 2003, pp. 31–36.
4. Buneman P., Khanna S., and Tan W.C. On propagation of deletions and annotations through views. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 150–158.
5. Cui Y., Widom J., and Wiener J.L. Tracing the lineage of view data in a data warehousing environment. ACM Trans. Database Syst., 25(2):179–227, 2000.

6. Dayal U. and Bernstein P. On the correct translation of update operations on relational views. ACM Trans. Database Syst., 8(3):381–416, 1982.
 
7. Keller A.M. Choosing a view update translator by dialog at view definition time. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 467–474.
8. Kotidis Y., Srivastava D., and Velegrakis Y. Updates through views: a new hope. In Proc. 21st Int. Conf. on Data Engineering, 2005.

9. Shu H. Using constraint satisfaction for view update translation. In Proc. 21st Int. Conf. on Data Engineering, 2005.
10. Widom J. Research problems in data warehousing. In Proc. Int. Conf. on Information and Knowledge Management, 1995, pp. 25–30.