Data Integration using Self-Maintainable Views

Ashish Gupta
Oracle Corporation
ashgupta@us.oracle.com

H. V. Jagadish
AT&T Bell Laboratories
jag@research.att.com

Inderpal S. Mumick
AT&T Bell Laboratories
mumick@research.att.com

Abstract

In this paper we define the concept of *self-maintainable* views – these are views that can be maintained using only the contents of the view and the database modifications, but without accessing any of the underlying databases. We derive tight conditions under which a view is self-maintainable, and we give algorithms to self-maintain several types of select-project-join views upon insertions, deletions and updates. Self-Maintainability is a desirable property for efficiently maintaining large views in applications where fast response and high availability are important. One example of such an environment is Data warehousing wherein views are used for integrating data from multiple databases.

1 Introduction

Most large organizations have related data in distinct databases. Many of these databases may be legacy systems, or systems separated for organizational reasons like funding and ownership. Integrating data from such distinct databases is a pressing business need. A common approach for integration is to define an integrated view and then map queries on the integrated view onto queries on the individual systems. This model is not good for applications where response time is critical, or for decision support applications with complex queries. Further, in this model, the availability of the integrated view is the lowest denominator of the availabilities of all databases touched by a query. Finally, such query transaltion can become almost impossible when some of the underlying "databases" are actually flat files, or other non-relational systems.

We propose an alternative model, good for applications where fast response to queries and high availability are important: The integrated view is materialized and stored in a database. Queries on the view are then answered directly from the stored view. This approach involves the additional storage expense for the materialized integrated view. Also, we need to maintain the integrated view current as the underlying databases change. Maintaining the integrated database current requires efficient incremental view maintenance techniques.

Often, It may be expensive or impractical to obtain much more than just a periodic report of local updates from each underlying database. Even if an underlying database does not provide relational query support, one can require it to provide a log of changes it makes. Under such circumstances, it becomes crucial that the integrated view be *self-maintainable* (Self-M), meaning that view maintenance should be possible without requiring access to any underlying database, and without access to any information beyond the view itself and the log of the changes. Self-M can be used in data warehousing environments not only to efficiently maintain views but also to vaoid concurrency control problems faced by generic view maintenance strategies [Z+94].

We study the syntactic restrictions for SPJ view to be self-M, and gives the corresponding maintenance algorithms. Using the syntactic restrictions it is possible for a data integrator to write self-M views that are efficiently maintainable. Thus, even if a view itself is not self-M, a super-set of the view may often be self-M. The full version of this paper appears as [GJM94].

Results

- We define the self-maintenance problem as the problem of maintaining a view in response to insertions, deletions, or update (collectively referred to as modifications) using only the view and the set of changes to the referenced relations (without access to the full referenced relations). We obtain complete conditions and syntactic restrictions under which SPJ views (which are views defined using the select-from-where clause of SQL) are self-M with respect to insertions, deletions, and updates and present algorithms for their self-maintenance.
- We model updates directly for the purpose of view maintenance. We show that view maintenance is easier when updates are modeled directly rather than as a set of deletions and insertions; in particular updates can often be self-maintained even when insertions cannot be self-maintained.

2 Background and Notation

We consider SPJ views, *i.e.*, `SELECT-PROJECT-JOIN` views written using `SELECT-FROM-WHERE` clauses. A SP (`SELECT-PROJECT`) view is a SPJ view with only one relation occurrence in the `FROM` clause.

Definition 2.1 (Distinguished Attribute:) An attribute A of a relation R is said to be distinguished in a view V if attribute A appears in the `SELECT` clause defining view V. $\square$

We use keys to define updates and to derive self-maintenance algorithms. A relation may have several keys, and any one of these could be used for any of the results we derive ($\mathsf{key}(R)$ refers to the key attributes of R).

Definition 2.2 (Updates:) For a relation R that has a key, a tuple in R is said to be updated if one or more attributes of the tuple are assigned a value different from its original value. An update to tuple r that results in tuple r' is represented as $\mu(r, r')$. $\square$

For each of the three types of modifications (inserts, deletes, and updates) we identify classes of views that are self-M with respect to the modified relation R. We omit mentioning "with respect to modified relation R" when the context makes it clear that relation R is modified. Also, henceforth the phrase "with respect to R" implicitly assumes that R is used in the view under consideration.

Definition 2.3 (Self Maintainability with respect to a Modification:) A view V is said to be self-M with respect to a modification type (insertion, deletion, or update) if for all database states, the view can be self-maintained in response to a modification of that type to the base relations. $\square$

Definition 2.4 (Local and Join Predicates:) Consider a predicate $p(\bar{X})$ in the `WHERE` clause of a view. $p(\bar{X})$ is said to be a local predicate if all variables in $\bar{X}$ appear in a single relation in the `FROM` clause. Otherwise, $p(\bar{X})$ is said to be a join predicate . $\square$

Definition 2.5 (Derivation tree [RSUV89]:)(Informal) A derivation tree for a tuple t is a tree representation of the manner in which the tuple is derived from base relations. For a tuple t in a base relation, the derivation tree consists of a single node labeled with tuple t. For a tuple t in a view defined as a join between relations $R_1, \ldots, R_k$, a derivation tree consists of a root node labeled with the tuple t, and one child node for each of the tuples ($r_1 \in R_1, r_2 \in R_2, \ldots, r_k \in R_k$) that join to derive tuple t. $\square$

Definition 2.6 (Tuple Derivation:) Consider view V defined using relation R, and possibly other relations. A tuple r in R derives a tuple v in V if r appears in some derivation tree for tuple v. $\square$

In the paper we consider SP and SPJ views that are satisfiable [Shm87]. That is, there exists a database for which the view is non-empty. Unsatisfiable views are always empty and hence trivially self-M. Also, we assume that the views do not include valid predicates (like ($X > 10$ `OR` $X < 20$)).

3 Self-Maintenance for SPJ Views

Whether a view is self-M depends on both the definition of the view and on the type of modification, and also on other finer distinctions such as which attribute of a relation is updated, or the actual value of the modified attribute, or the presence of functional dependencies and other integrity constraints. In this paper we restrict ourselves to classifying self-maintainability with respect to the relation being modified, the type of modification, and key information. We do not consider the other finer granularity distinctions in this paper.

Insertions Views are self-M for insertions only under very limited circumstances. We prove that it is not possible to self-maintain an SPJ (select-project-join) view joining at least two distinct relations upon an insertion into a component relation. Even a view involving a self-join of a relation with itself may not be self-M.

Theorem 3.1 *The self-maintainability of an SPJ view in response to insertions is decidable, and can be tested in time exponential in the number of self joins in the SPJ view.* □

Proposition 3.1 *An SPJ view that takes the join of two or more distinct relations is not self-M with respect to insertions.* □

Proposition 3.2 *All SP views are self-M with respect to insertions. An SPJ view defined using self-joins over a single relation R is self-M if every join is based on* $\mathsf{key}(R)$. □

Algorithm Outline: For SP views self-maintenance can be done by a selection and a projection on the newly inserted tuples. For an SPJ view with self-joins on a key each newly inserted tuple joins only with itself, so the view definition is evaluated on only the set of inserted tuples.

Deletions

Theorem 3.2 *An SPJ view V that takes the join of one or more relations $R_1, \ldots, R_n$ is self-M with respect to deletions to R_1 if and only if, for every database instance, and for every occurrence of relation R_1 in the view definition, the following holds: Given a tuple t in view V, let a derivation tree for the tuple t use the tuple r in R_1 for the stated occurrence of R_1 in the view definition. Then, it is possible to identify the key of such a tuple r of R_1 from the tuple t and the view definition, without referring to the contents of any of the relations R_i.* □

A sufficient condition for the conditions of Thm 3.2 to be true is when, for some key of relation R_1, each key attribute is either retained in the view, or is equated to a constant in the view definition. Then, given a tuple t in the view, we can identify the tuple r of R_1 from the key attributes.

Algorithm outline for views satisfying Theorem 3.2: For deleted tuple r in relation R_1, check $\mathsf{key}(r)$ satisfies any predicates that equate key attributes to constants in the view definition. Then, look for tuples in the view that have the same values on the remaining key attributes as the deleted tuple. Delete all such tuples from the view.

Updates Updates have been modeled as deletions followed by insertions in previous view maintenance work. This model may lose information that could be useful for incremental view maintenance. Also, such a representation of an update means that a view is self-M with respect to updates only if the view is self-M with respect to both inserts and deletes. In fact, the following lemma may suggest that indeed the above conclusion holds for self-maintenance in response to updates.

Lemma 3.3 *Let V be an SPJ view, and R one of the relations in the* `FROM` *clause, such that there is at least one predicate on R. Then, the view maintenance problem for V in response to any insertion or a deletion into R can be reduced to the view maintenance problem for V in response to an update to R.* □

However, a view may be self-M with respect to updates even it it is not self-M with respect to both insertions and deletions because by modeling an update directly, not as a delete plus an insert, we retain the link between the deleted and inserted tuples. Thus, often most of the attributes of the new tuple that needs to be inserted into the view can be obtained from the deleted tuple, enabling self-maintenance.

The information about the "deleted" view tuple that is retained by directly modeling updates facilitates self-maintenance in two respects: the conditions for self-maintenance becomes less onerous than in the case of simple insertions, so that many views that are not self-M with respect to insertions can be self-M with respect to updates; and the computational effort required to perform the insertion is decreased, whether or not the view is self-M.

Whether a view is self-M depends upon the attributes being updated. The following definition captures the property that enables self-maintenance.

Definition 3.1 (Exposed Variable:) Given a view definition V, a variable, or equivalently, an attribute, A, of a relation used in the view definition is said to be exposed if it is involved in some predicate. A variable that is not exposed is called a non-exposed variable. □

A non-exposed attribute does not affect the combinations of tuples that contribute to the view.

Theorem 3.3 *A SPJ view V that joins two or more distinct relations is self-M with respect to updates to relation R_1 if and only if either:*

- *The updated attributes are non-exposed and are not distinguished with respect to view V, or*
- *The updated attributes are non-exposed with respect to view V and V is self-M with respect to deletions to the relation R_1.*

□

Algorithm Outline: The algorithm involves joining the modified tupels with the view on the key attributes to obtain the new value for the modified view tuple.

4 Related Work

Autonomously Computable Views of [BCL89] Blakeley *et al.* [BCL89] defined a view to be *Autonomously computable* with respect to a given update if the view can be maintained using only the materialized view for all database instances for only the given instance of the update. Autonomously Computability thus differs from self-M in that it considers a specific instance of the update, while self-M is defined over all possible updates. The update language of [BCL89] is more powerful – it permits deletions and updates to be specified using arbitrary conditions. In contrast, they do not consider views with self joins and do not use key information for deriving self-M conditions.

View Maintenance: View maintenance [BLT86, CW91, GMS93, HD92, Kuc91, UO92] bas been generally studied as a problem to compute the changes to the view given the changes to the underlying relations, while having access to the view and the underlying relations. [TB88, GB94] deduce instance specific conditions for maintaining views without accessing base relations. Also, the above work models updates as deletions + insertions and not directly as updates that may be more easily maintainable.

Self-M is also indirectly related to work in data integration and updatable views.

5 Conclusions and Future Work

We propose materialized views as a way to provide fast access to integrated data in a data warehouses We define self-maintainability as a desirable property for such views that allows the integrated data to be updated in response to modifications to the base relations without accessing the base relations and thereby avoiding remote accesses and concurrency control. We discuss syntactic restrictions that ensure self-maintainability of SPJ views. We find that SPJ views are self-M with respect to deletions and updates to non-exposed attributes if the keys of the joined relations are included in the view. Also, SPJ views are usually not self-M with respect to insertions. Modeling updates directly, and not as deletions + insertions, facilitates view maintenance and can make some views self-M with respect to updates.

An important application of the results is that they provide guidance to a designer on how to define views in the integrated database so that they may be maintained efficiently. Many times simply keeping an extra attribute can make a view self-M, and greatly reduce the maintenance work. We recommend that integrated views be defined so as to be self-M for as large a class of modifications as possible.

Future Work This paper opens up a new area of work on view maintenance issues for data warehousing. How can functional dependencies, more general than keys, be used for for self-maintenance? How to self-maintain a *set* of views? How to self-maintain aggregate views?

References

[BCL89] J. A. Blakeley, N. Coburn, and P. Larson. Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates. *ACM Transactions on Database Systems*, 14(3):369–400, 1989.

[BLT86] J. A. Blakeley, P. Larson, and F. W. Tompa. *Efficiently Updating Materialized Views. SIGMOD 1986.*

[CW91] S. Ceri and J. Widom. Deriving Production Rules for Incremental View Maintenance. In *VLDB 1991.*

[GB94] A. Gupta and J. A. Blakeley. Using Partial Information to Update Materialized Views In *Information Systems*, 1995.

[GJM94] A. Gupta, H. V. Jagadish, and I. S. Mumick. Data integration using self-maintainable views. Technical memorandum, AT&T Bell Laboratories, November 1994.

[GMS93] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining Views Incrementally. In *Proceedings of ACM SIGMOD 1993 International Conference on Management of Data*, pages 157–167.

[HD92] J. V. Harrison and S. Dietrich. Maintenance of Materialized Views in a Deductive Database: An Update Propagation Approach. In *Workshop on Deductive Databases, JICSLP*, 1992.

[LMSS95] J. Lu, G. Moerkotte, J. Schu, and V. Subrahmanian. Efficient maintenance of materialized mediated views. In *SIGMOD*, 1995.

[Kuc91] V. Kuchenhoff. On the Efficient Computation of the Difference Between Consecutive Database States. In *DOOD, LNCS 566*, 1991.

[RSUV89] R. Ramakrishnan, Y. Sagiv, J. D. Ullman, and M. Vardi. Proof-tree Transformation Theorems and Their Applications. In *PODS*, 1989.

[Shm87] O. Shmueli. Decidability and expressiveness aspects of logic queries. In *PODS*, pages 237–249, 1987.

[TB88] F. W. Tompa and J. A. Blakeley. Maintaining Materialized Views without Accessing Base Data. *Information Systems*, 13(4):393–406, 1988.

[UO92] T. Urpi and A. Olive. A Method for Change Computation in Deductive Databases. In *VLDB*, 1992.

[Z+94] Y. Zhuge, H.G arcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *SIGMOD 1995.*