Efficient View Self-Maintenance

Nam Huyn[*]
Stanford University
huyn@cs.stanford.edu

Abstract

We consider the problem of maintaining a materialized view without accessing the base relations. More specifically, we would like to find a maximal test that guarantees that a view is *self-maintainable* (abbrev SM) under a given update to the base relations, i.e., can be maintained using only the view definition, its contents and the update. We observe that SM evaluation can be separated into a view-definition-time portion where a maximal test is generated solely based on the view definition, and an update-time portion where the test can be efficiently applied to the view and the update. We call such a maximal test a *Complete Test for View Self-Maintainability* (abbrev CTSM).

This paper reports on some interesting new results for conjunctive-query views under insertion updates: 1) the CTSM's are extremely simple queries that look for certain tuples in the view to be maintained; 2) these CTSM's can be generated at view definition time using a very simple algorithm based on the concept of *Minimal Z-Partition*; 3) view self-maintenance can also be expressed as simple update queries over the view itself.

1 Introduction

In this paper, we consider the problem of determining self-maintainability (abbreviated SM) of views that are expressed as conjunctive queries over base relations. That is, given a view definition specified as a conjunctive query $\mathcal{Q}$, a materialized view V that is the result of applying $\mathcal{Q}$ to some database D, and an update μ to the base relations in D (as shown in Figure 1,) we would like to find a test:

- That only looks at the view definition $\mathcal{Q}$, view V and update μ,
- That determines whether or not $\mathcal{Q}(D^\mu)$ [1] depends only on V and μ, regardless of the actual database D, subject to the constraint that $V = \mathcal{Q}(D)$,
- And that is maximal in the sense that when the test answers negatively, there are database instances D_1 and D_2 that are both consistent with V but such that $\mathcal{Q}(D_1^\mu) \neq \mathcal{Q}(D_2^\mu)$.

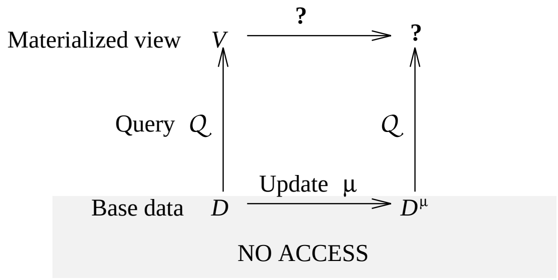

Figure 1: Elements in the view self-maintenance problem.

Example 1.1 Consider a large job brokerage house that uses a materialized view *match(P,J,S,L)* to keep track of good matches between people P with skill S and multi-sited jobs J at location L. View *match* derives from the following relations that reside in some remote personnel–project database:

apply(P,J,S) : person P applies for job J indicating that P has skill S to offer.

site(J,L) : L is one of job J's locations.

prefer(P,L) : P is willing to work at location L.

use(J,S) : S is one of job J's required skills.

View *match* is defined as follows:

$$match(P,J,S,L) \ :\!- \ apply(P,J,S), site(J,L), prefer(P,L), use(J,S). \quad (1)$$

In the first case, consider the insertion of tuple *site(java,montreal)*, and consider the test:

$$\mathcal{T}_1 : (\exists P,S) \ match(P, java, S, montreal)$$

[*]Work supported by ARO grant DAAH04–95–1–0192.

[1] D^μ denotes the result of applying μ to D.

If $\mathcal{T}_1$ is satisfied, we know that *site(java,montreal)* is already in the database, and no change is needed to bring the view up to date. Conversely, if $\mathcal{T}_1$ is not satisfied, there might be unseen candidates who applied for *java* with some unseen skill used in *java*, and who are willing to work in *montreal*. The reason these candidates didn't show up as a good match is that *java* was not located in *montreal* before the update. After the update, all these candidates will show up as good matches. So the new matches depend on which of these candidates are already in the database. Thus, $\mathcal{T}_1$ is a maximal test that guarantees view *match* be self-maintainable for the insertion of *site(java,montreal)*.

In the second case, consider the insertion of tuple *apply(philip,java,architect)*, and consider the test:

$$\mathcal{T}_2 : (\exists L)\ match(philip, java, architect, L)$$

Satisfaction of $\mathcal{T}_2$ is sufficient for *match* to be self-maintainable since no change is needed. However, $\mathcal{T}_2$ is not maximal. To see why, consider an instance of view *match* with the following tuples:

(betty,java,architect,quebec)
(philip,java,programmer,montreal)

From the first tuple, we know that *java* uses *architect*'s. From the second tuple, we can infer not only that *philip* applied for *java* as a *programmer*, but also that all locations of common interest to both *philip* and *java* should already be in the view. In the given view instance, there is only one such location, namely *montreal*. These locations do not depend on any particular skill. Thus, regardless of the instance of the underlying databases, we can safely conclude that the insertion causes exactly one new match to be added, namely the tuple *(philip,java,architect,montreal)*. Even though the view instance does not satisfy test $\mathcal{T}_2$, it remains self-maintainable. ∎

We call such maximal tests as $\mathcal{T}_1$ in Example 1.1 *Complete Tests for Self-Maintainability* (abbreviated CTSM). The problem of finding CTSM's has been studied in [TB88] and more recently in [GB95] for views that are conjunctive queries with arithmetic comparisons (aka select-project-join queries) but with single occurrence of predicates (i.e., no self-joins). Results from [TB88] and [GB95] gave necessary and sufficient conditions for *Conditionally autonomously Computable Updates* (abbreviated CAU). We would like to point out that the two notions SM and CAU are equivalent, even though [TB88] and [GB95] only explicitly mentioned that SM follows from CAU.

The approach used in [TB88] and [GB95] has two main disadvantages: efficiency of determining SM evaluation is not well understood, and construction of the test and its execution are intermingled, forcing most of SM evaluation to be done at update time. To date, efficient implementation of SM evaluation remains difficult.

In this paper, we explore the hypothesis that for conjunctive-query views at least, separation of SM evaluation into a view-definition-time portion and an update-time portion is possible. That is, a complete test can be constructed from the view definition alone. Figure 2 contrasts our approach with previous approach. Furthermore, the complete test can be efficient to execute, such as simple first order queries.

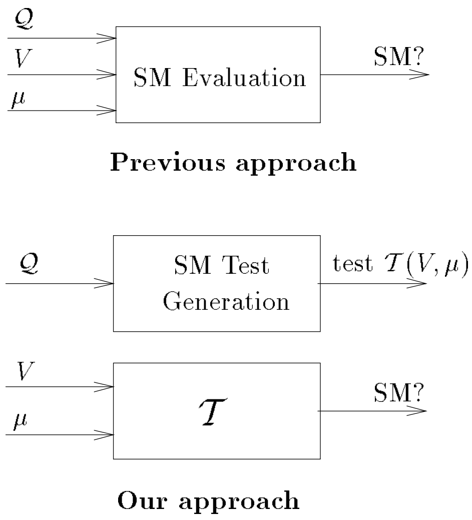

Figure 2: Separating SM test generation from SM test evaluation.

The main result of this paper essentially confirms our hypothesis:

- For CQ views with no self-joins and for insertion updates, view definitions have a very simple characterization using the concept of *Minimal Z-Partition* (Section 3.)
- Using this characterization, we derive CTSM's that turn out be simple queries on the view that essentially look for certain tuples. We found a class of CQ views where no SM evaluation is ever needed, simply because no view in this class is self-maintainable (Section 4.)
- View self-maintenance, expressible as simple queries, are also given for the general case of CQ (Section 4.)

- Finally, we show that our work dramatically improves over previous work on how efficiently CQ views can be self-maintained (Section 5.)

The obvious practical significance of our result is that these CTSM's not only can be efficiently precomputed but can also be optimized using traditional query optimizers at view definition time, thus minimizing work that needs to be done at update time.

2 Preliminaries

2.1 Notation, terminology and assumptions

Throughout the rest of this paper, the definition of a conjunctive-query view is represented as follows:

$$\mathcal{Q} : v(\bar{X}', \bar{U}', \bar{Z}') :- r(\bar{X}, \bar{U}), S(\bar{U}, \bar{Z}). \qquad (2)$$

where $\bar{U}$, $\bar{X}$ and $\bar{Z}$ denotes sets of variables, $\bar{X}'$, $\bar{U}'$ and $\bar{Z}'$ denote subsets of $\bar{X}$, $\bar{U}$ and $\bar{Z}$ respectively, r is the predicate for the updated relation, and S denotes a conjunction of subgoals.

$\bar{U}$ represents the *join* variables, i.e., the variables shared between the subqueries $r(\bar{X}, \bar{U})$ and $S(\bar{U}, \bar{Z})$. From the point of view of S, we also call $\bar{U}$ the *distinguished* variables (while we call $\bar{Z}$ the *nondistinguished* variables).

It is important not to confuse our definition of distinguished variables with that commonly used to designate those variables that are used in the head (i.e. those variables that are not "projected out"). We call the latter variables *exposed* and use ' to denote them. Thus, $\bar{X}'$ denotes a subset of the variables in $\bar{X}$ that are exposed. Variables that are not exposed are called *hidden*.

We sometimes call the variables in $\bar{X}$ the X-variables, $\bar{U}$ the U-variables, and $\bar{Z}$ the Z-variables.

Finally, we assume all predicates in the body have single occurrences, that is, no self-joins are allowed.

Example 2.1 The view definition

$$v(X, Z, T) :- r'(Y, X, 3, Z), s'(Z, T, Y, Z, 5).$$

where r' and s' are base relation predicates, is represented in our notation as

$$v(X, Z, T) :- r(X, Y, Z), s(Y, Z, T).$$

where the original subgoals are normalized (i.e., constant symbols are removed, multiple occurrences of variables consolidated, variables reordered) using predicates r and s. In (2)'s notation, $\bar{U}$ represents the join variables $\{Y, Z\}$, $\bar{U}' = \{Z\}$, $\bar{X} = \bar{X}' = \{X\}$, $\bar{Z} = \bar{Z}' = \{T\}$. ∎

We will use D, D_1 and D_2 to denote database instances, and D^μ, D_1^μ and D_2^μ the respective instances that result from applying update μ.

2.2 Definition of self-maintainability

Given a view definition $\mathcal{Q}$, a view V that is the result of applying $\mathcal{Q}$ to some database D and an update μ on D, we say that view V is self-maintainable (SM) under update μ if the new view that results from update μ is independent of the underlying database. The following more formally defines the SM notion.

Definition 2.1 (Self-Maintainability): View V is self-maintainable under μ if $\mathcal{Q}(D^\mu)$ is the same for every database instances D such that $\mathcal{Q}(D) = V$. ∎

Proposition 2.1 *SM and CAU are equivalent.* ∎

Proof: [TB88] showed that CAU implies SM. Now, we show that SM implies CAU as well. SM says that $\mathcal{Q}(D^\mu)$ is independent of D, provided that D is consistent with V. So to compute $\mathcal{Q}(D^\mu)$, we can choose the "canonical" database $D_c = \mathcal{Q}^{-1}(V)$ obtained as follows: each tuple in V binds the variables in the head; these bindings are extended to the hidden variables in the body by binding them to new constants. One can easily show that $\mathcal{Q}(D_c)$ is identical to V. Computing $\mathcal{Q}(D^\mu)$ is reduced to computing $\mathcal{Q}(D_c^\mu)$. We just defined a function that takes V as input and computes the new view that is consistent with the updated database. Thus the update is also conditionally autonomously computable. ∎

Equivalence between SM and CAU justifies our use of SM throughout the rest of this paper, which we find more natural for our purpose.

2.3 Approach to finding CTSM's

The approach we take can be summarized as follows:

- Find a syntactic characterization of $\mathcal{Q}$ for the purpose of deriving CTSM's.
- Based on a specific characterization of $\mathcal{Q}$, find a test condition that typically looks for the existence of certain tuples in V.
- Verify that the condition is sufficient for SM by showing that for any database D such that $\mathcal{Q}(D) = V$, $\mathcal{Q}(D^\mu)$ does not depend on D.
- Verify that the condition is necessary for SM by finding an appropriate counterexample consisting of database instances D_1 and D_2. Typically, D_1 is some canonical minimal database that is consistent with V. D_2 is typically obtained by introducing some perturbation (to be found) to D_1 that is sufficiently small to maintain consistency with V but sufficiently large to assure that $\mathcal{Q}(D_2^\mu)$ is different from $\mathcal{Q}(D_1^\mu)$.

2.4 Self-maintainability vs. self-maintenance

When a view is self-maintainable under a given update, how do we determine the actual updates to the view that will make it consistent with the updated base relations?

By definition of SM, $\mathcal{Q}(D^{\mu})$ does not depend on D as long as $\mathcal{Q}(D) = V$. In the worst case, we can pick D arbitrarily (e.g. the canonical database consistent with V), apply the update μ to D to obtain D^{μ} and run the query $\mathcal{Q}$ over D^{μ}. However, we can do much better: the required updates to view V can typically be derived from sufficiency proofs of the SM condition.

3 Minimal Z-Partition

The following example suggests that in general, for views defined by

$$v(\bar{X}', \bar{U}', \bar{Z}') :- r(\bar{X}, \bar{U}), S(\bar{U}, \bar{Z}).$$

complete tests for self-maintainability for insertion into r are not independent of the actual structure of S.

Example 3.1 Consider the following two different view definitions:

$$Q_1 : v(X, Y) :- r(X, Y), t(X, Y).$$
$$Q_2 : v(X, Y) :- r(X, Y), t_1(X), t_2(Y).$$

Consider the insertion of $r(a, b)$ and the problem of maintaining some view V defined by either Q_1 or Q_2. One can easily verify that while the condition that $V(a, b)$ holds is a CTSM in the first case, it is no longer a necessary condition for SM in the second case. In fact, it is not difficult to verify that a CTSM in the second case is given by a different condition, namely that $V(a, -) \wedge V(-, b)$ [2] holds. ∎

But how do we syntactically characterize $S(\bar{U}, \bar{Z})$ for the purpose of finding a CTSM? In the rest of this section, we develop the tool for characterizing S that will be used in later sections.

Definition 3.1 (Minimal Z-Partition): Let $S(\bar{U}, \bar{Z})$ be a conjunction of subgoals with distinct predicates where certain variables are designated as "distinguished" ($\bar{U}$ in our notation) and the remaining variables as "nondistinguished" ($\bar{Z}$ in our notation). A Z-partition for $S(\bar{U}, \bar{Z})$ is a partition of the subgoals into groups such that no two groups share the same Z-variable. A minimal Z-partition is a Z-partition such that further partitioning is not possible without introducing groups sharing the same nondistinguished variables. ∎

[2] $V(a, -)$ is a shorthand for $(\exists Y)V(a, Y)$. Similarly, $V(-, b)$ denotes $(\exists X)V(X, b)$.

Example 3.2 Consider the first case in Example 1.1 where relation *site* is updated. The following lists all subgoals whose relations are not subject to update:

$$apply(P, J, S), prefer(P, L), use(J, S)$$

where J and L are distinguished, P and S are nondistinguished. Any partitioning of these subgoals creates groups that share either P or S. Thus the minimal Z-partition consists of only one group that includes all the subgoals, as depicted in Figure 3 in connection hypergraph form where nondistinguished variables are labeled with a "*" and hyperedges are labeled with their predicate names. ∎

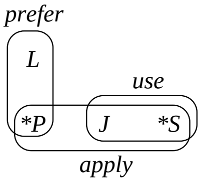

Figure 3: Only one group in the minimal Z-partition in Example 3.2.

Example 3.3 Consider the second case in Example 1.1 where relation *apply* is updated. The following lists all subgoals whose relations are not subject to update:

$$site(J, L), prefer(P, L), use(J, S).$$

Since L is the only nondistinguished variable, these subgoals can be partitioned into two groups: $\{use(J,S)\}$ and $\{site(J,L), prefer(P,L)\}$. This Z-partition is minimal and is illustrated in Figure 4 where each group in the partition is represented by hyperedges with the same shading. ∎

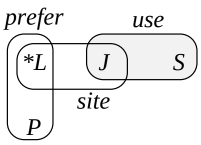

Figure 4: The two groups in the minimal Z-partition in Example 3.3.

Properties of minimal Z-partitions

- A minimal Z-partition always exists and is unique.
- Any group having no nondistinguished variables is a singleton that consists of some subgoal that uses no nondistinguished variables.
- Any group having some nondistinguished variables consists of subgoals that are all "interconnected" by nondistinguished variables. That is, suppose we cannot remove a subgoal from the group without also removing any other subgoal that can join with it via some nondistinguished variable. Then removing any subgoal would force us to remove all the subgoals from the group.

Algorithm for computing the minimal Z-partition

There is a simple one-pass algorithm that computes the minimal Z-partition. Scan the given list of subgoals and consider each subgoal in turn. If the subgoal has no nondistinguished variable, assign it to a new group. If the subgoal has some nondistinguished variable, look for an existing group that shares some nondistinguished variable with the subgoal. If none can be found, assign the subgoal to a new group. Otherwise, merge all such groups and assign the subgoal to the result.

4 Complete Tests for Self-Maintainability

This section presents solutions for finding CTSM for CQ views defined by (2). We only consider insertions of a single tuple into a base relation. We first present our result for the special case of (2) where $\bar{X}' = \bar{X} = \emptyset$, $\bar{U}' = \bar{U}$ and $\bar{Z}' = \bar{Z}$. This special case is important only because the technique used is applicable to the general case. We then present results for the general case. Due to space limitation, most proofs are omitted here. They can be found in [Hu96a].

4.1 Important special case

Consider the following view definition:

$$\mathcal{Q} : v(\bar{U}, \bar{Z}) \;\text{:-}\; r(\bar{U}), S(\bar{U}, \bar{Z}). \tag{3}$$

Let the minimal Z-partition of $S(\bar{U}, \bar{Z})$ consist of groups $g_1, \ldots, g_n$. Let $\bar{U}_i$ (resp. $\bar{Z}_i$) denote the set of distinguished (resp. nondistinguished) variables used in group g_i. A necessary and sufficient condition for self-maintainability is given in the following theorem.

Theorem 4.1 *For a view V defined by (3), a CTSM under the insertion of $r(\bar{a})$ is given by the following condition:*

$$\bigwedge_{i=1}^{n} (\exists \bar{U}, \bar{Z})[V(\bar{U}, \bar{Z}) \wedge \bar{U}_i = \bar{a}_i]^3 \tag{4}$$

To maintain view V (when the view is self-maintainable), insert tuples $(\bar{a}, \bar{z})$ for all $\bar{z}$ in the cross-product $\bar{z}_1 \times \ldots \times \bar{z}_n$ where $\bar{z}_i$ is obtained from the query

$$\{\bar{Z}_i \mid V(\bar{U}, \bar{Z}) \wedge \bar{U}_i = \bar{a}_i\} \tag{5}$$

∎

Proof: The full proof is given in the Appendix. ∎

Example 4.1 Consider the second case in Example 1.1 where tuple *(philip,java,architect)* is inserted into relation *apply(P,J,S)*. The minimal Z-partition was shown in Example 3.3 to consist of the groups {*use(J,S)*} and {*site(J,L),prefer(P,L)*}. The distinguished variables used in these groups are $\{J, S\}$ and $\{J, P\}$ respectively. A CTSM essentially looks for tuples *(P,J,S,L)* in *match* that agree with the inserted tuple over components $\{J, S\}$ or $\{J, P\}$, namely:

$$match(-, java, architect, -) \wedge$$
$$match(philip, java, -, -)$$

To bring view *match* up to date, simply add all tuples *(philip,java,architect,L)* such that $(philip, java, -, L)$ is already in *match*. ∎

Simplification

The complete test (4) for SM can often be simplified by eliminating any conjunct that is subsumed by another conjunct: when two groups g_i and g_j are such that $\bar{U}_i \subseteq \bar{U}_j$, the conjunct that corresponds to g_i can be eliminated without affecting the logical meaning of the test.

Example 4.2 Consider the view definition

$$v(U, V, W, Z, T) \;\text{:-}\; r(U, V, W), s_1(U, V), s_2(V, Z), s_3(W, Z), s_4(T).$$

The minimal Z-partition consists of three groups: group $\{s_1(U, V)\}$ using distinguished variables UV, group $\{s_2(V, Z), s_3(W, Z)\}$ using distinguished variables VW and group $\{s_4(T)\}$ using no distinguished

[3]Notation: $\bar{a}_i$ denotes the restriction of $\bar{a}$ over the $\bar{U}_i$ components.

variables. A complete test of self-maintainability under the insertion of $r(a,b,c)$ is given by

$$V(a,b,-,-,-) \wedge V(-,b,c,-,-) \wedge \\ V(-,-,-,-,-)$$

Since the last conjunct (meaning that V is nonempty) is subsumed by the other conjuncts, the condition can be simplified to:

$$V(a,b,-,-,-) \wedge V(-,b,c,-,-)$$

∎

4.2 General case where all distinguished variables are exposed

This case directly generalizes the special case (3), in which the X-variables are introduced to r and not all X-variables and Z-variables are exposed. That is, consider the view definition:

$$\mathcal{Q} : v(\bar{X}', \bar{U}, \bar{Z}') :- r(\bar{X}, \bar{U}), S(\bar{U}, \bar{Z}). \qquad (6)$$

Theorem 4.2 *For a view V defined by (6), when all distinguished variables are exposed, a CTSM under the insertion of $r(\bar{b}, \bar{a})$ is given by the following condition:*

$$\bigwedge_{i=1}^{n} (\exists \bar{X}', \bar{U}, \bar{Z}')[V(\bar{X}', \bar{U}, \bar{Z}') \wedge \bar{U}_i = \bar{a}_i] \qquad (7)$$

To maintain view V (when the view is self-maintainable), insert tuples $(\bar{b}', \bar{a}, \bar{z}')$ for all $\bar{z}'$ in the cross-product $\bar{z}'_1 \times \ldots \times \bar{z}'_n$ where $\bar{z}'_i$ is obtained from the query

$$\{\bar{Z}'_i \mid V(\bar{X}', \bar{U}, \bar{Z}') \wedge \bar{U}_i = \bar{a}_i\}^4 \qquad (8)$$

∎

Proof: The proof is not included due to space limitation. We have a proof very similar to the one for the special case of Section 4.1, where the database instance D_1 in the counterexample is constructed from V by padding the hidden variables with new constants, for each tuple from V. ∎

Example 4.3 Consider the view definition

$$v(U,V,W,Z) \;:-\; r(U,V,W,X), s_1(U,V), s_2(V,Z), \\ s_3(W,Z), s_4(T).$$

A CTSM under the insertion of $r(a,b,c,d)$ is given by

$$V(a,b,-,-) \wedge V(-,b,c,-)$$

To maintain V, add all tuples (a,b,c,z) such that $V(-,b,c,z)$ holds. ∎

[4] Notation: $\bar{Z}'_i$ denotes $\bar{Z}' \cap \bar{Z}_i$, that is, the exposed variables in $\bar{Z}_i$.

4.3 General case where some distinguished variables are hidden

When some of the distinguished variables (i.e. U-variables) are hidden, the view becomes "less self-maintainable" in some sense. Intuitively, any CTSM is expected to be stricter than when all distinguished variables are exposed. Consider the view definition:

$$\mathcal{Q} : v(\bar{X}', \bar{U}', \bar{Z}') :- r(\bar{X}, \bar{U}), S(\bar{U}, \bar{Z}). \qquad (9)$$

where $\bar{U}'$ is a proper subset of $\bar{U}$. There are two subcases we need to consider: the case where every group g_i either has no exposed $\bar{Z}_i$ or has no hidden $\bar{U}_i$, and the opposite case.

4.3.1 No group has both hidden distinguished and exposed nondistinguished variables

This is the case where for each group g_i, either $\bar{Z}'_i$ is empty or $\bar{U}_i \subseteq \bar{U}'$.

Theorem 4.3 *For a view V defined by (9), when no group has both hidden distinguished and exposed nondistinguished variables, a CTSM under the insertion of $r(\bar{b}, \bar{a})$ is given by the following condition:*

$$(\exists \bar{Z}') V(\bar{b}', \bar{a}', \bar{Z}') \qquad (10)$$

To maintain view V (when the view is self-maintainable), no tuples need to be inserted into V. ∎

Proof: The proof is not included due to space limitation. We have a proof that uses the same technique as for the other cases. The sufficiency proof involves showing that $\mathcal{Q}(D^{\mu}) = \mathcal{Q}(D) = V$. In the counterexample used, $\mathcal{Q}(D_1^{\mu}) = \mathcal{Q}(D_1)$. ∎

4.3.2 Some group has both hidden distinguished and exposed nondistinguished variables

The case where for some group g_i, $\bar{Z}'_i$ is nonempty and $\bar{U}_i \not\subseteq \bar{U}'$, is the worst case in the sense that the view is totally not self-maintainable, as stated in the following theorem.

Theorem 4.4 *When some group has both hidden distinguished and exposed nondistinguished variables, a view V defined by (9) is not self-maintainable under the insertion of $r(\bar{b}, \bar{a})$.* ∎

Example 4.4 Consider the view definition (1) in Example 1.1 and consider updating relation *apply*. If view *match* were defined to have only attributes *(P,J,L)*, it may still be self-maintainable. But when we further project out attribute P from the view, it is no longer self-maintainable. ∎

5 Complexity

Consider a materialized view V with n tuples and a conjunctive query Q with d subgoals.

To decide whether or not V is self-maintainable under a given insertion, the approach taken in [TB88, GB95] constructs a theory (that is, a set of first order sentences) about the base relations, and a set of tuples that could potentially be added to V. The view is self-maintainable if for every such tuple, we can either prove or disprove that the tuple will be added to V, based on the theory. Gupta et al. showed in [GB95] that such proof can be reduced to deciding containment of conjunctive queries for a number of query pairs proportional to n. Since there are $O(n)$ tuples to check, SM evaluation takes time

$$O(n^2 \times 2^d)$$

and uses auxiliary storage of size $O(n)$. Thus, for large views especially, implementation directly using this approach is not practical.

Referring to our results shown in the previous section, the CTSM's are essentially queries that are conjunctions of at most d subqueries of the form $(\exists \bar{T})\ V(\bar{T})$ where $\bar{T}$ agrees with the inserted tuple over some attributes. Since these subqueries are all closed, the CTSM's have no joins. Thus, using the CTSM's we derived, checking whether or not V is self-maintainable only takes time

$$O(n \times d)$$

(constant time if V is indexed on the $\bar{U}_i$ attributes), without using any auxiliary storage in the worst case.

Thus, compared with previous approach, our approach offers a significant improvement in both time and space. We have effectively precomputed many inferences that were carried out at runtime in the other approach.

6 Conclusion

Table 1 summarizes the main results of this paper. Interestingly, when all distinguished variables are exposed in the view, the CTSM does not depend on $\bar{b}$, the X-components of the inserted tuple. When we hide enough of the distinguished variables without also hiding related nondistinguished variables, the view ceases to be able to self-maintain.

The results also demonstrate that testing for self-maintainability not only can be practically implemented, but can also be efficiently implemented: the CTSM's that are generated at view-definition-time can be optimized and suggest ways to index the materialized view that can be exploited to speed up update-time testing and maintenance works.

We briefly mentioned that multivalued dependencies are an alternative technique to characterize view definitions, which lends itself easily to analysis involving dependencies on base relations.

Work is under way to find CTSM's for CQ views that allow self-joins in their definition.

Example 6.1 Consider for instance the following view definition

$$v(X,Y,Z) :- r(X,Y), t(X,Z), t(Y,Z).$$

A CTSM for inserting $r(a,b)$ is

$$\begin{array}{l} V(a,b,-) \vee V(b,a,-) \vee \\ {[V(a,a,-) \wedge V(b,b,-)]} \vee \\ {[V(a,a,-) \wedge (\forall Z)(V(a,a,Z) \Rightarrow P_b(Z))]} \vee \\ {[V(b,b,-) \wedge (\forall Z)(V(b,b,Z) \Rightarrow P_a(Z))]} \end{array}$$

where $P_y(Z)$ is defined to be

$$\begin{array}{l} V(-,y,Z) \vee V(y,-,Z) \vee \\ (\exists X)[(V(X,y,-) \vee V(y,X,-)) \wedge \\ \qquad (V(X,-,Z) \vee V(-,X,Z))] \end{array}$$

■

While the presence of self-joins introduces extra complexity in the CTSM, since components of the $S(\bar{U},\bar{Z})$ may now "commute" among themselves, it makes the view more self-maintainable. In fact, the more constraints we know hold among the base relations, the more self-maintainable the view becomes. We are currently investigating the use of generalized dependencies to capture this added constraint on S.

In future work, we plan to extend our techniques to analyzing views whose definition involves use of negation. Similar techniques have already been successfully used in our work on finding complete tests for constraint maintenance under limited data access ([Hu96b]), a different problem but related to the view maintenance problem.

7 Acknowledgments

We thank Prof. Jeff Ullman for many valuable discussions and comments regarding both technical contents and presentation of the material.

References

[GB95] Gupta A. and Blakeley J. A.: Using Partial Information to Update Materialized Views. In *Information Systems*, 20(8), pp. 641–662, 1995.

Complete characterization of view definition	*Description*
Form 1	All distinguished variables are exposed.
Form 2	No group has both hidden distinguished and exposed nondistinguished variables.
Form 3	Some group has both hidden distinguished and exposed nondistinguished variables.

	Complete SM test for inserting $r(b, \bar{a})$	*Maintenance expression*
Form 1	$\bigwedge_{i=1}^{n}(\exists \bar{X}', \bar{U}, \bar{Z}')[V(\bar{X}', \bar{U}, \bar{Z}') \wedge \bar{U}_i = \bar{a}_i]$	Insert $(\bar{b}', \bar{a}, \bar{z}')$ such that for all i, $\bar{z}_i' \in \{\bar{Z}_i' \mid V(\bar{X}', \bar{U}, \bar{Z}') \wedge \bar{U}_i = \bar{a}_i\}$
Form 2	$(\exists \bar{Z}')V(\bar{b}', \bar{a}', \bar{Z}')$	No update needed.
Form 3	*FALSE*	Not applicable.

Table 1: Summary of results for self-maintaining CQ views.

[Hu96a] Huyn N.: Efficient View Self-Maintenance, *Unpublished Technical Report*, available as URL `http://www-db.stanford.edu/pub/papers/cqvsm-tr.ps`, 1996.

[Hu96b] Huyn N.: Testing CQC¬ constraints under limited data access. *Unpublished Technical Report*, available as URL `http://www-db.stanford.edu/pub/papers/cqcnclt-tr.ps`, 1996.

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

A Appendix

View definition (3) can be rewritten as:

$$v(\bar{U}, \bar{Z}) \coloneq r(\bar{U}), S_1(\bar{U}_1, \bar{Z}_1), \ldots, S_n(\bar{U}_n, \bar{Z}_n).$$

where each S_i represents the conjunction of subgoals from group g_i as defined by the minimal Z-partition of $S(\bar{U}, \bar{Z})$. Another way to look at the abstract structure of $S(\bar{U}, \bar{Z})$ is to use multivalued dependencies, as depicted in Figure 5.

Sufficiency Proof for Theorem 4.1

To show that condition (4) is sufficient for SM, assume it is satisfied. Let D be a database instance consistent with V. We need to show that $Q(D^{\mu})$ does not depend on D.

$$\begin{aligned} & Q(D^{\mu}) \\ & = Q(D \cup \{r(\bar{a})\}) = Q(D) \cup \{(\bar{a}, \bar{z}) \mid S(\bar{a}, \bar{z}) \in D\} \\ & = V \cup \{(\bar{a}, \bar{z}) \mid S_1(\bar{a}_1, \bar{z}_1) \in D, \ldots, S_n(\bar{a}_n, \bar{z}_n) \in D\} \\ & = V \cup \{(\bar{a})\} \times \{(\bar{z}_1) \mid S_1(\bar{a}_1, \bar{z}_1) \in D\} \times \ldots \\ & \qquad \times \{(\bar{z}_n) \mid S_n(\bar{a}_n, \bar{z}_n) \in D\} \end{aligned}$$

For any i, there is a tuple $(\bar{u}, \bar{z})$ in V such that $\bar{u}_i = \bar{a}_i$ (i.e. $\bar{u}$ and $\bar{a}$ agree over $\bar{U}_i$) and D contains $r(\bar{u})$, $S_1(\bar{u}_1, \bar{z}_1)$, …,$S_n(\bar{u}_n, \bar{z}_n)$. Thus any $S_i(\bar{a}_i, \bar{z}_i')$ would join with these tuples to generate $v(\bar{u}, \bar{z}')$ where $\bar{z}'$ is obtained by replacing the $\bar{Z}_i$ components of $\bar{z}$ with $\bar{z}_i'$. Conversely, any tuple $(\bar{u}, \bar{z})$ in V such that $\bar{u}_i = \bar{a}_i$ implies the existence of some $S_i(\bar{a}_i, \bar{z}_i')$ where $\bar{z}_i' = \bar{z}_i$. Therefore $\{\bar{Z}_i \mid S_i(\bar{a}_i, \bar{Z}_i) \in D\} = \{\bar{Z}_i \mid V(\bar{U}, \bar{Z}) \wedge \bar{U}_i = \bar{a}_i\}$. Now we can rewrite $Q(D^{\mu})$ as:

$$\begin{aligned} & Q(D^{\mu}) \\ & = V \cup \{(\bar{a})\} \times \{(\bar{z}_1) \mid V(\bar{u}, \bar{z}) \wedge \bar{u}_1 = \bar{a}_1\} \times \ldots \\ & \qquad \times \{(\bar{z}_n) \mid V(\bar{u}, \bar{z}) \wedge \bar{u}_n = \bar{a}_n\} \end{aligned}$$

Therefore, not only we showed $Q(D^{\mu})$ is independent of D, but we also derived the view maintenance expression (5).

Necessity Proof for Theorem 4.1

To show that condition (4) is necessary for SM, assume it is not satisfied. We need to construct two database instances D_1 and D_2 that are both consistent with V but such that $Q(D_1^{\mu}) \neq Q(D_2^{\mu})$.

For D_1, we use the "canonical" database instance consistent with V, constructed the following way: each tuple in V binds the variables $\bar{U}$ and $\bar{Z}$ in the head of (3); substituting these bindings into the body

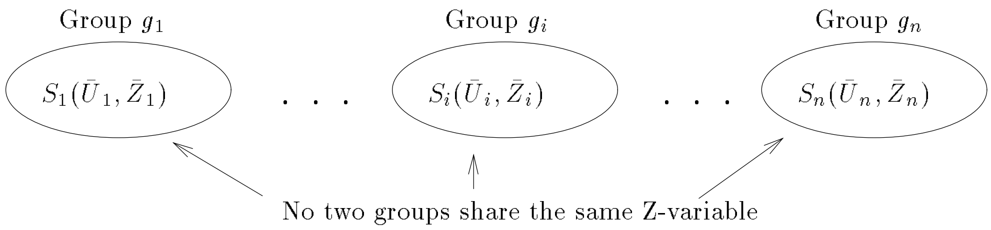

Figure 5: In the minimal Z-partition, $S(\bar{U}, \bar{Z})$ satisfies MVD $\bar{U} \twoheadrightarrow \bar{Z}_i$ for all i.

makes each subgoal into an atom, a ground atom in this case. The canonical instance consists of such ground atoms generated by all tuples in V.

To construct D_2, we add to D_1 a set Δ of new tuples (i.e., that is not already in D_1) as follows. Condition (4) can be written as $\bigwedge_{i=1}^{n} cond_i$. Since the condition is not satisfied, there is some $cond_i$ that is false. For each i such that $cond_i$ is false, new tuples are included into Δ according to which of the following categories group g_i belongs:

A If the group has no nondistinguished variable (i.e. $\bar{Z}_i = \emptyset$), it consists of a single subgoal, say $p(\bar{U}_i)$. It is not difficult to see that by construction of the canonical instance, D_1 could not possibly contain $p(\bar{a}_i)$. Therefore we include $p(\bar{a}_i)$ in Δ.

B If the group has some nondistinguished variable (i.e. $\bar{Z}_i \neq \emptyset$), we bind all nondistinguished variables in the group to new constants (say bind $\bar{Z}_i$ to $\bar{z}_i^{new}$). $S_i(\bar{a}_i, \bar{z}_i^{new})$ is a set of ground atoms each of which contains some new constant and thus cannot be in D_1. We therefore include $S_i(\bar{a}_i, \bar{z}_i^{new})$ in Δ.

This construction of Δ is illustrated in Table 2.

	g_i in Cat. A $cond_i$ false	g_i in Cat. B $cond_i$ false	g_i $cond_i$ true
D_1	$S_i(\bar{a}_i)$ absent		Some $S_i(\bar{a}_i, \bar{z}_i)$ present
Δ	Add $S_i(\bar{a}_i)$	Add $S_i(\bar{a}_i, \bar{z}_i^{new})$	No tuples added

Table 2: Construction of the counterexample.

Now that we have specified D_2, we need to verify that it is indeed consistent with V. Since $D_1 \subseteq D_2$ and Q is monotonic, we only need to make sure that Q cannot generate any new tuple when Δ is added to D_1. Any new tuple Q generates must use some tuple $t \in \Delta$ which falls into either Category A or Category B:

- For Category A, t includes $\bar{a}_i$ as components and since $cond_i$ is false, relation r in D_1 (or D_2) has no tuple that agrees with $\bar{a}$ over $\bar{U}_i$. Therefore, t cannot join with any tuple from r, and using t, Q cannot generate any new tuple.
- For Category B, using t from $S_i(\bar{a}_i, \bar{z}_i^{new})$ forces us to use all tuples from $S_i(\bar{a}_i, \bar{z}_i^{new})$. S_i generates exactly the tuple $(\bar{a}_i, \bar{z}_i^{new})$ which cannot join with any tuple from r since $cond_i$ is false. So again, Q cannot generate any new tuple is t is used.

Finally, to verify that $Q(D_1^{\mu}) \neq Q(D_2^{\mu})$, we need to find a tuple in $Q(D_2^{\mu})$ that is not in $Q(D_1^{\mu})$. Consider the tuple t' that joins the following facts from $Q(D_2^{\mu})$:

- $r(\bar{a})$,
- All the new facts from Δ (there is at least one such new fact),
- For each group g_i such that $cond_i$ is satisfied, we know that $(\exists \bar{Z}_i) S_i(\bar{a}_i, \bar{Z}_i)$ is satisfied in the canonical instance D_1. We arbitrarily choose some value $\bar{z}_i$ that satisfies $S_i(\bar{a}_i, \bar{z}_i)$. So we use all the facts in $S_i(\bar{a}_i, \bar{z}_i)$. These facts are old since they are all in D_1.

Tuple t' could not possibly be in $Q(D_1^{\mu})$ since it is derived from at least a new fact from Δ:

- If the new fact falls into Category A (say $p(\bar{a}_i)$), the only way t' can be in $Q(D_1^{\mu})$ is that $p(\bar{a}_i) \in D_1$, which we already know is not possible.
- If the new fact falls into Category B, one of its components must be a new constant. So t' must contain some new constant and thus cannot be in $Q(D_1^{\mu})$.