Efficient Self-Maintenance of Materialized Views

A TECHNICAL REPORT

Nam Huyn
Stanford University
huyn@cs.stanford.edu

Abstract

We address the problem of incrementally maintaining a materialized view using the view instance and the update but with only limited access to the base data. We give necessary and sufficient conditions for self-maintainability (handling updates without looking at all base data) for conjunctive-query (CQ) views. These conditions, which we call Complete Tests for Self-Maintainability or CTSM in short, are generated at view-definition time and are expressed as safe, nonrecursive Datalog queries. We show both a direct approach that yields surprisingly simple CTSM's for a class of CQ views with self-joins, when no knowledge of the base data is available, and a query-containment-based approach that can be systematically used to find CTSM's for more complex CQ views with or without additional knowledge about the base data.

1 Introduction

The growing popularity of data warehouses [RED, IK93, ZGHW95] has renewed interests in work on materialized view maintenance. To date, much work has been done on view maintenance [TB88, GJM94, GB95, GM95, QGMW95], yet efficiently maintaining views still remains a difficult problem, especially when not using all the base data. But why can't all base data be used?

Consider a collection of materialized views in a data warehouse that draws their contents from a variety of external sources, as depicted in Figure 1. Some of the base data may reside in remote sources or may even be archived, other may be produced by only virtual data sources (e.g. mediators [Wie92]) as a result of expensive transformations, making their access expensive. At the extreme, when some data sources are disconnected from the warehouse, accessing them becomes impossible.

View *self-maintenance* (see [GM95] for a more refined taxonomy), that is, maintenance not using all the base data, can be used as an approach to optimize view maintenance: maintenance plans that make use of cheaply accessible data are considered first, those using more expensive data are only considered as a fall-back position.

Not using all base data to maintain a materialized view raises many issues:

- Self-maintainability: before even addressing the issue of how to maintain the view, how do we know if there is enough information to maintain the view? Are there algorithms that decide self-maintainability (abbreviated SM)?
- Self-maintenance: if the view is self-maintainable, how do we bring it up to date without using only a subset of the base data?

In order to use view self-maintenance as a practical approach to maintenance optimization, the algorithms themselves must be efficient: if accessing the base data is cheaper than running the self-maintenance algorithms, the purpose of SM would be defeated. Thus, efficiency considerations raise new challenges:

- There are two principal approaches to self-maintenance: compile-time and run-time ([Huyn96a]). In the compile-time approach, we would like to determine, at view definition time, whether the view is SM under some class of update. In the run-time approach, the specific update and the actual view contents are available to help decide SM. Both approaches have shortcomings: the former is likely to answer the question negatively for most views, while the latter is likely to incur much work at run-time. The challenge here is to find a middle ground where for example, instance-specific update-time tests are generated at view-definition time.
- A SM test always guarantees a view is self-maintainable if it evaluates to true. However, a test that is not the most general might fail to detect situations where the view can actually self-maintain. The most general SM test is thus desirable. But can we always find most general SM tests that are efficient?
- There are many forms SM tests can take. However, having them in the special form of queries (for instance in safe, nonrecursive Datalog or SQL) is beneficial: not only we can execute them on a traditional query evaluation engine, but we can also optimize these tests using traditional query optimization techniques, especially at compile time. But do SM tests always admit a query form?

Example 1.1 To remain competitive in the global airline business, a large international air carrier constantly has to reallocate resources and adjust its promotional packages, based on its operational data. For this purpose, a data warehouse has been setup to keep track of certain business patterns. A materialized view `P` is used to keep track of the popular routes and can be defined in SQL as

```
CREATE VIEW P(X,Y)
  AS SELECT rtrip.X, rtrip.Y
     FROM   rtrip, nstop NS1, nstop NS2
     WHERE  rtrip.X = NS1.X AND
            rtrip.Y = NS1.Y AND
            NS1.X = NS2.Y AND
            NS1.Y = NS2.X
```

or equivalently in Datalog as

```
p(X,Y) :- rtrip(X,Y,U),nstop(X,Y),nstop(Y,X).
```

Relation `rtrip(X,Y,U)` indicates that individual `U` purchased a roundtrip ticket going from `X` to `Y` and back to `X`. Relation `nstop(X,Y)` indicates there is currently a minimum of two daily nonstop flights from `X` to `Y`. Since the base relations all reside in operation databases, it is desirable to minimize their disturbance during view maintenance. Suppose the warehouse is notified of a new roundtrip ticket purchase, say when tuple `(chicago,london,john)` is inserted to relation `rtrip`.

Without looking at any base relation, how can we determine if there is enough information to update view `P`? Suppose for a moment that `P` contains the tuple `(chicago,london)`. Using this information, we can easily verify that `P` needs no update since `(chicago,london)` is already a popular route. But if `(chicago,london)` is not in `P`, do we have to give up maintaining `P` on the ground that further information on the base relations is required to correctly maintain `P`? The answer is negative and to see why, suppose view `P` consists of the tuple `(london,chicago)`. No matter what extension the base relations might have, we know that there must be at least two daily nonstop flights between `london` and `chicago` in both directions. Thus, there is only one way to update `P`, namely insert `(chicago,london)`. So, while test

$$\tau_1 : \texttt{(chicago,london)} \in \texttt{P}$$

guarantees that P can be maintained without looking at the base relations, test

$$\tau_2 : \texttt{(chicago,london)} \in \texttt{P} \vee \texttt{(london,chicago)} \in \texttt{P}$$

is strictly more general than τ_1. It turns out τ_2 is the most general self-maintainability test under the insertion of `rtrip(chicago,london,john)`.

How does the situation change if we now assume `rtrip` can be accessed during the maintenance of P? With this additional information, τ_2 still guarantees maintainability but is no longer the most general test. In fact, if `(london,chicago,-)` $\in$ `rtrip`, whether or not to insert `(chicago,london)` into P only depends on whether `(london,chicago)` is already in P. Thus, the most general test when `rtrip` is available is

$$\tau_3 : \texttt{(chicago,london,-)} \in \texttt{rtrip} \vee \texttt{(london,chicago,-)} \in \texttt{rtrip}.$$

Note that all the tests in this example can be written in SQL. For instance τ_3 can be implemented as the following fragment

```
WHERE EXISTS (SELECT * FROM rtrip
              WHERE  (rtrip.X = ''chicago'' AND rtrip.Y = ''london'') OR
                     (rtrip.X = ''london'' AND rtrip.Y = ''chicago''))
```

∎

In the remainder of this paper, we only consider self-maintainability tests that are most general. We call them *Complete Tests for Self-Maintainability* or CTSM in short. We will use the hybrid approach where CTSM's are generated at view-definition time and we will look for such generation algorithms. Lastly but not the least, we are only interested in CTSM's in efficient query form, that is, tests expressible as safe nonrecursive Datalog queries over the given view and any additional known base relations.

Related Work

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). Although [TB88] and [GB95] gave necessary and sufficient conditions for *Conditionally Autonomously Computable Updates* (a notion very similar to self-maintainability), these conditions refer to the symbols used in the extension of the view to maintain and are not expressible as efficient queries. Efficient implementation of SM evaluation remains difficult with their approach. [QGMW95] solves a different but related problem, namely that of making a view self-maintainable by introducing auxiliary views to materialize and by exploiting dependencies in the base data. In [GJM94] the compile-time SM problem is addressed, rather than the run-time SM problem we are interested in here. Finally, [Huyn96a] solved the basic SM problem (without using any information about the base relations) using CTSM's in efficient query form. However the solution there is limited to conjunctive-query views with no self-joins.

Results

- We extend the work in [Huyn96a] using the direct approach, to conjunctive-query (CQ) views that allow *exposed* self-joins.

- We show how an alternative approach based on query containment can be applied to solve the basic SM problem for a class of CQ views not handled by the direct approach: arbitrary self-joins are allowed when all body variables are exposed.
- We show how to use the containment-based approach to solve the SM problem with the presence of additional information about the base data, namely when some base relations and/or additional views are given.

2 Preliminaries

Notation, terminology and assumptions

Throughout the rest of this paper, the definition of a CQ view is represented as follows:

$$\mathcal{Q} : v(\bar{X}', \bar{U}', \bar{Z}') \coloneq r(\bar{X}, \bar{U}), S(\bar{U}, \bar{Z}). \tag{1}$$

where $\bar{U}$, $\bar{X}$ and $\bar{Z}$ denote 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. We assume predicate r does not occur in S. Unless noted otherwise, repeated predicates are allowed within S.

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 $v(\bar{X}, \bar{U}', \bar{Z}) \coloneq r(\bar{X}, \bar{U}), S(\bar{U}, \bar{Z})$, where v, r, and S denote `v'(X,Z,T)`, `r'(Y,X,3,Z)`, and `s'(Z,T,Y,Z,5)` respectively, $\bar{U}$ represents the join variables {Y,Z}, $\bar{U}' =$ {Z}, $\bar{X} =$ {X}, $\bar{Z} =$ {T}. ∎

Note that this notation ignores constants and multiple occurrence of variables in the subgoals. The order in which variables occur in the subgoals is also ignored. While this "normalized" notation works well when there is no repeated predicate among the subgoals, it can be confusing when two subgoals use the same predicate, since the fact that the two normalized subgoals actually use identical predicates is not captured in the notation. The reader will be explicitly warned should such a potential for confusion arise.

Self-maintainability

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. Given a query $\mathcal{Q}$, a view V that is the result of applying $\mathcal{Q}$ to some database D, and an update μ to 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. More precisely, V is self-maintainable under μ if $\mathcal{Q}(D^\mu)$ is the same for every "valid" database instance D. A database instance D is said to be *valid* if it satisfies $\mathcal{Q}(D) = V$ and any other additional constraints. For instance, if some of the base relations are given, any valid D must include these relations exactly as they are given.

Efficient Tests in Query form

We are only interested in efficient tests in query form, that is, tests whose conditions can be expressed in nonrecursive, safe Datalog [Ull88]. We emphasize query safety here since test evaluation must be finite. These requirements ensure that the tests can be implemented in a traditional language such as SQL and run on traditional query engines. In the remainder of this paper, all the CTSM's will be given in the form of safe queries.

3 Basic SM Problem Solved With Direct Approach

In the basic SM problem, only a view definition, a view, and an update are given, and no knowledge of the base relations is assumed. A direct approach was first used in [Huyn96a] to solve the basic SM problem, in which a closed form solution is given and its correctness directly proven, without using additional tools to derive the solution. In the following, we first introduce the notion of *Minimal Z-Partition* that is used in the direct approach. We then summarize results from [Huyn96a], followed by new results that extend the previous results to CQ views that allow some form of self-joins.

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* of $S(\bar{U}, \bar{Z})$ is a partition of the subgoals into groups such that no two groups share the same Z-variable. A Z-partition is *minimal* if further partitioning is not possible without introducing groups sharing the same nondistinguished variables. The groups in a Z-partition of $S(\bar{U}, \bar{Z})$ will be denoted by $g_1, \ldots, g_n$, the distinguished variables used in group g_i by $\bar{U}_i$, and the nondistinguished variables by $\bar{Z}_i$. ∎

Consider for example the conjunction of subgoals `s(J,L)`, `t(P,L,P)`, `u(3,J,S)`, where only `L` is nondistinguished. The minimal Z-partition consists of two groups: $g_1 = \{$`s(J,L),t(P,L,P)`$\}$ and $g_2 = \{$`u(3,J,S)`$\}$. This partitioning is also depicted in Figure 2 in hypergraph form, where groups are defined by hyperedges that share some nondistinguished variables.

3.1 CQ views with no self-joins

The basic SM problem for CQ views with no self-joins was completely solved in [Huyn96a]. Only a brief summary of the results is given here. For proof details, we refer the reader to [Huyn96a-TR].

The case where all distinguished variables are exposed is treated by the following theorem.

Theorem 3.1 *([Huyn96a]) Consider a view V defined by*

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

where S has no repeated predicates. Then a CTSM under the insertion of $r(\bar{b}, \bar{a})$ is given by the following condition:

$$\bigwedge_{i=1}^{n} (\exists \bar{X}', \bar{U}, \bar{Z}')[V(\bar{X}', \bar{U}, \bar{Z}') \wedge \bar{U}_i = \bar{a}_i] \tag{2}$$

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

$$\{\bar{Z}'_i \mid V(\bar{X}', \bar{U}, \bar{Z}') \wedge \bar{U}_i = \bar{a}_i\} \tag{3}$$

∎

The remaining case where some distinguished variables are hidden is treated by the following theorem.

Theorem 3.2 *([Huyn96a]) Consider a view V defined by*

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

where $\bar{U}' \subset \bar{U}$ and S has no repeated predicates. Then, if there is a group g_i such that $\bar{Z}'_i \neq \emptyset$ and $\bar{U}_i \not\subseteq \bar{U}'$, the view is not self-maintainable. Otherwise, a CTSM under the insertion of $r(\bar{b}, \bar{a})$ is given by the following condition:

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

and V is already up to date. ∎

Example 3.1 Consider view `V` defined by

```
CREATE VIEW V(P,J,L,S)
  AS SELECT r.P, r.J, s.L, r.S
     FROM   r, s, t, u
     WHERE  r.J = s.J AND r.P = t.P AND r.S = u.S AND
            s.J = u.J AND s.L = t.L AND
            t.P = t.Q AND u.I = 3
```

over relations `r(J,P,S)`, `s(J,L)`, `t(P,L,Q)`, `u(I,J,S)`, or equivalently in Datalog by

```
v(P,J,L,S) :- r(J,P,S),s(J,L),t(P,L,P),u(3,J,S).
```

Let us insert `r(a,b,c)`. The minimal Z-partition has two groups using the sets {`P`,`J`} and {`J`,`S`} of distinguished variables respectively. Applying Theorem 3.1, `v` is self-maintainable under the insertion if and only if it satisfies the condition `v(b,a,-,-)` $\wedge$ `v(-,a,-,c)` which can be implemented by the following SQL fragment

```
WHERE EXISTS (SELECT * FROM V WHERE V.P = ``b'' AND V.J = ``a'') AND
      EXISTS (SELECT * FROM V WHERE V.J = ``a'' AND V.S = ``c'')
```

The following is an SQL query that maintains `V`

```
INSERT INTO V
SELECT ``b'', ``a'', v1.L, ``c''
FROM   V v1
WHERE  v1.P = ``b'' AND v1.J = ``a''
```

Consider another view `V'` similarly defined as `V` except that `S` is projected out. Even though the second group uses distinguished variable `S` that is now hidden, it does not use any nondistinguished variable. According to Theorem 3.2, view `V'` is SM if and only if `v'(b,a,-,c)` is true. Finally consider view `V''` similarly defined as `V` except that now both `S` and `P` are projected out. Theorem 3.2 tells us that `V''` is not self-maintainable. ∎

3.2 CQ views with Exposed self-joins

The previous results are a good promising first step toward solving the basic SM problem for general CQ views. However, their restriction to CQ views without self-joins limits their usefulness in many data warehousing applications where self-joins are often found. We now extend the solution to CQ views that allow a limited form of self-joins called *exposed* self-joins. All proofs for the results in this section can be found in the Appendix.

Definition 3.2 (Exposed Self-Joins): Let $v(\bar{X}', \bar{U}', \bar{Z}')$:- $r(\bar{X}, \bar{U}), M(\bar{U}, \bar{Z})$ be a conjunctive query, where r is the updated predicate and M is a conjunction of subgoals. Predicate r is not used in M, but repeated predicates may be used in M. The self-joins in the query are said to be exposed when all subgoals in M with repeated predicates use only variables from $\bar{U}'$. ∎

Example 3.2 Consider query

$$v(X, Y, T) \text{ :- } r(X, Y, T), s(X, Y, X), s(Y, 1, Y), t(X, Z), u(T, Z)$$

and assume r is the updated predicate. Since the subgoals with predicate s use variables X and Y that occur in both the head and the r subgoal, the query has exposed self-joins. If Y is dropped from the head, the query no longer has the property. ∎

First, the case when the join variables are totally exposed is addressed by the following theorem.

Theorem 3.3 *Consider a view V defined by*

$$\mathcal{Q} : v(\bar{X}', \bar{U}, \bar{Z}') \text{ :- } r(\bar{X}, \bar{U}), M(\bar{V}), S(\bar{W}, \bar{Z}). \tag{4}$$

where $\bar{V} \cup \bar{W} = \bar{U}$, M is a conjunction of subgoals with repeated predicates, and S a conjunction of subgoals with no repeated predicates. Consider the minimal Z-partition of $S(\bar{W}, \bar{Z})$ in which group g_i uses distinguished variables W_i and nondistinguished variables Z_i. Then the CTSM under the insertion of $r(\bar{b}, \bar{a})$ is given by the following condition:

$$\tau_{\bar{a}} \wedge \bigwedge_{i=1}^{n} (\exists \bar{X}', \bar{U}, \bar{Z}')[V(\bar{X}', \bar{U}, \bar{Z}') \wedge \bar{W}_i = \bar{a}_i] \tag{5}$$

where $\tau_{\bar{a}}$ is defined as the query

$$\tau_{\bar{a}} \text{ :-} M(\bar{V}), \bar{U} = \bar{a}.$$
$$M(\bar{V}) \text{ :-} v(\bar{X}', \bar{U}, \bar{Z}').$$

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

$$\{\bar{Z}'_i \mid V(\bar{X}', \bar{U}, \bar{Z}') \wedge \bar{W}_i = \bar{a}_i\} \tag{6}$$

∎

Example 3.3 Consider the problem defined in Example 1.1, where `p` is the predicate for view `P`. `P` is self-maintainable under the insertion of `rtrip(chicago,london,john)` if and only if `P` satisfies τ defined by the following program

```
       τ  :-  nstop(chicago,london),nstop(london.chicago).
nstop(X,Y)  :-  p(X,Y).
nstop(Y,X)  :-  p(X,Y).
```

After expanding the `nstop` subgoals, τ can be simplified to `p(chicago,london)`∨`p(london,chicago)`.
∎

Example 3.4 Consider a more complicated view V defined by

```
v(X,Y,T) :- r(X,Y,T),s(X,Y,X),s(Y,1,Y),t(X,Z),u(T,Z)
```

and let us insert `r(1,2,3)`. View V is self-maintainable if and only if the condition `v(1,-,3)` $\wedge\ \tau_{1,2,3}$ is satisfied. $\tau_{1,2,3}$ is defined by the program

```
    τ1,2,3  :-  s(1,2,1),s(2,1,2).
  s(X,Y,X)  :-  v(X,Y,T).
  s(Y,1,Y)  :-  v(X,Y,T).
```

and rewritten as `v(1,2,-)` $\wedge$ [`v(2,1,-)` $\vee$ `v(-,2,-)`], which further simplifies to `v(1,2,-)`. In other words, the CTSM is `v(1,-,3)` $\wedge$ `v(1,2,-)` or in SQL

```
WHERE EXISTS (SELECT * FROM V
              WHERE  (V.X = 1 AND V.T = 3) OR
                     (V.X = 1 AND V.Y = 2))
```

∎

The remaining case where some distinguished variables are hidden is addressed by the following theorem.

Theorem 3.4 *Consider a view V defined by*

$$\mathcal{Q} : v(\bar{X}', \bar{U}', \bar{Z}') \text{ :- } r(\bar{X}, \bar{U}), M(\bar{V}), S(\bar{W}, \bar{Z}). \tag{7}$$

where $\bar{U}' \subset \bar{U}$, $\bar{V} \cup \bar{W} = \bar{U}$ and $\bar{V} \subseteq \bar{U}'$, M is a conjunction of subgoals with repeated predicates and S a conjunction of subgoals with no repeated predicates. Consider the minimal Z-partition of $S(\bar{W}, \bar{Z})$ in which group g_i uses distinguished variables W_i and nondistinguished variables Z_i. Then, if a group g_i has some hidden distinguished variables ($\bar{W}_i \not\subseteq \bar{U}'$) and some exposed nondistinguished variable ($\bar{Z}'_i \neq \emptyset$), V is not self-maintainable. Otherwise, a CTSM under the insertion of $r(\bar{b}, \bar{a})$ is given by the following condition:

$$\tau_{\bar{a}} \wedge (\exists \bar{Z}')\ V(\bar{b}', \bar{a}', \bar{Z}') \tag{8}$$

where $\tau_{\bar{a}}$ is defined as the query

$$\tau_{\bar{a}} \text{ :-} M(\bar{V}), \bar{U} = \bar{a}.$$
$$M(\bar{V}) \text{ :-} v(\bar{X}', \bar{U}', \bar{Z}').$$

and V is already up to date. ∎

4 Containment-Based Approach

While the problem of finding CTSM in query form was previously solved using a direct approach, the solutions are limited to CQ views that use a restricted form of self-join. The solutions obtained by a directed approach are elegant and simple, yet extending them a larger class of views turns out to be difficult.

Alternatively, an approach based on query containment (QC) can be applied to find query CTSM for broader class of views and seems to lend itself more easily to extensions of the basic SM problem where we have additional knowledge of the base relations. For instance, some of the base relations may be known, additional views may be given, or certain base data dependencies may be known.

In the following, we consider CQ views that allow arbitrary self-joins but with all body variables exposed in the head. This class of views includes certain views that were not previously handled in the direct approach (for example, views where self-joins use nondistinguished variables). We first define *minimal valid database*, a notion central to reducing SM to QC. We then show how a containment-based approach can be applied to solve the basic SM problem. Finally, we show how the approach can be adapted to solve the SM problem where some of the base relations are known and additional views are given. All proofs in this section can be found in the Appendix.

Definition 4.1 (Minimal valid database $\mathcal{Q}^{-1}(V)$): Given a conjunctive query $\mathcal{Q}$ and a view $V = \mathcal{Q}(D)$, consider building a database instance as follows: for every tuple in V that matches $\mathcal{Q}$'s head, the resulting bindings are substituted into $\mathcal{Q}$'s body; a new constant is substituted for any variable in $\mathcal{Q}$'s body that does not occur in $\mathcal{Q}$'s head; the database instance consists of all the resulting ground atoms from the $\mathcal{Q}$'s body. In general a database instance constructed this way is not unique, but all such instances are isomorphic to each other. When all body variables in $\mathcal{Q}$ are exposed, the database instance obtained above is unique and denoted by $\mathcal{Q}^{-1}(V)$. Recall that a database instance D is said to be *valid* if $\mathcal{Q}(D) = V$. It is easy to show that $\mathcal{Q}^{-1}(V)$ is not only valid but also minimal in the sense that it is contained in any valid database, provided that one exists (see Appendix). ∎

For example, consider a view `V` defined by

$$\mathcal{Q} : \texttt{v(X,Y,Z)} :- \texttt{r(X,Y)}, \texttt{s(X,Z)}, \texttt{s(Y,Z)}.$$

Suppose $\texttt{V} = \{\texttt{(1,2,a)}, \texttt{(2,3,a)}\}$. The first tuple generates the atoms `r(1,2)`, `s(1,a)` and `s(2,a)`. The second tuple generates `r(2,3)`, `s(2,a)` and `s(3,a)`. Thus $\mathcal{Q}^{-1}(\texttt{V}) = \{\texttt{r(1,2)}, \texttt{r(2,3)}, \texttt{s(1,a)}, \texttt{s(2,a)}, \texttt{s(3,a)}\}$ is the minimal valid database. The relations in $\mathcal{Q}^{-1}(\texttt{V})$ can simply be defined by reversing the implication in the view definition.

4.1 CQ views with general self-joins

In the basic SM problem, we are given a view $V = \mathcal{Q}(D)$ and an insertion μ. We would like to find the most general test, in a safe query form, that guarantees that V is self-maintainable w.r.t. μ, i.e.

$$(\forall D_1, D_2)\ [\mathcal{Q}(D_1) = \mathcal{Q}(D_2) = V \Rightarrow \mathcal{Q}(D_1 \cup \mu) = \mathcal{Q}(D_2 \cup \mu)] \tag{9}$$

Reducing SM to QC

The problem is that (9) uses two databases instead of a single one usually found in a query containment formulation. However, when $\mathcal{Q}$ is a CQ with all body variables exposed, we can use the canonical database $D_0 = \mathcal{Q}^{-1}(V)$ as one of the two valid databases in (9) and an arbitrary superset of D_0 as the other valid database. Therefore, (9) is logically equivalent to the condition

$$(\forall D)\ \mathcal{Q}(D_0 \cup D) = V \Rightarrow \mathcal{Q}(D_0 \cup D \cup \mu) = \mathcal{Q}(D_0 \cup \mu)$$

or, since $\mathcal{Q}$ is monotonic and D_0 is a valid database,

$$(\forall D)\ \mathcal{Q}(D_0 \cup D) \subseteq V \Rightarrow \mathcal{Q}(D_0 \cup D \cup \mu) \subseteq \mathcal{Q}(D_0 \cup \mu)$$

or equivalently

$$(\forall D)\ \mathcal{Q}(D_0 \cup D \cup \mu) \not\subseteq \mathcal{Q}(D_0 \cup \mu) \Rightarrow \mathcal{Q}(D_0 \cup D) \not\subseteq V \tag{10}$$

To solve this (boolean) query containment problem, let us first elaborate on each of the queries in term of the view definition. To be more precise, consider the following view definition, where M denotes a conjunction of atoms whose predicates may be repeated:

$$\mathcal{Q} : v(\bar{X}, \bar{Y}, \bar{Z}) \coloneq r(\bar{X}, \bar{Y}), M(\bar{Y}, \bar{Z}).$$

The canonical database D_0 consists of the predicates r_0 and M_0 (strictly speaking, M_0 actually denotes several predicates, since M represents many subgoals) that can be defined by the rules (again, note that the second rule actually denotes many rules since the head is a conjunction of many atoms):

$$\begin{array}{rcl} r_0(\bar{X}, \bar{Y}) & \coloneq & v(\bar{X}, \bar{Y}, \bar{Z}). \\ M_0(\bar{Y}, \bar{Z}) & \coloneq & v(\bar{X}, \bar{Y}, \bar{Z}). \end{array}$$

Define an auxiliary predicate $v_{\bar{b}}$ that represents the Z-component of all tuples gained by the view after $r_0(\bar{a}, \bar{b})$ is inserted into D_0:

$$v_{\bar{b}}(\bar{Z}) \quad \coloneq \quad M_0(\bar{b}, \bar{Z}).$$

Assume the subgoals in M are indexed 1 through n. For any subset $I \subseteq \{1, \ldots, n\}$, let M_I denote the result of replacing the predicate for each subgoal indexed by I in M with the corresponding predicate from D_0.

The query on the left of (10) consists of rules (one for each $I \subseteq \{1, \ldots, n\}$):

$$panic \quad \coloneq \quad M_I(\bar{b}, \bar{Z}), \neg v_{\bar{b}}(\bar{Z}). \tag{11}$$

The query on the right of (10) consists of rules (one for each $J \subseteq \{1, \ldots, n\}$):

$$panic \quad \coloneq \quad r_0(\bar{X}, \bar{Y}), M_J(\bar{Y}, \bar{Z}), \neg v(\bar{X}, \bar{Y}, \bar{Z}). \tag{12}$$

Note that in (11), the rule for $I = \emptyset$ will never succeed and can thus be dropped from further consideration, since it characterizes when D_0 is not valid. Furthermore, note that in (12), we did not include those rules with predicate r (instead of r_0), since there is obviously no containment mappings from these rules to the rules in (11) which do not use r at all.

Example 4.1 Consider a view `V` defined by `v(X,Y,Z) :- r(X,Y),s(X,Z),s(Y,Z)` and the insertion of `r(1,2)`. The canonical database consists of relations $\mathtt{r}_0$ and $\mathtt{s}_0$ defined by the following rules:

$$\begin{array}{lcl} \mathtt{r_0(X,Y)} & \mathtt{:-} & \mathtt{v(X,Y,Z).} \\ \mathtt{s_0(X,Z)} & \mathtt{:-} & \mathtt{v(X,Y,Z).} \\ \mathtt{s_0(Y,Z)} & \mathtt{:-} & \mathtt{v(X,Y,Z).} \end{array}$$

The query on the left of the containment equation (10) is defined by the following rules:

$$\begin{array}{rlcl} & \mathtt{v_{12}(Z)} & \mathtt{:-} & \mathtt{s_0(1,Z), s_0(2,Z).} \\ r_1: & \mathtt{panic} & \mathtt{:-} & \mathtt{s_0(1,Z), s(2,Z), \neg v_{12}(Z).} \\ r_2: & \mathtt{panic} & \mathtt{:-} & \mathtt{s(1,Z), s_0(2,Z), \neg v_{12}(Z).} \\ r_3: & \mathtt{panic} & \mathtt{:-} & \mathtt{s(1,Z), s(2,Z), \neg v_{12}(Z).} \end{array}$$

The query on the right of the containment equation is defined by the following rules:

$$\begin{array}{rlcl} r_4: & \mathtt{panic} & \mathtt{:-} & \mathtt{r_0(X,Y), s_0(X,Z), s(Y,Z), \neg v(X,Y,Z).} \\ r_5: & \mathtt{panic} & \mathtt{:-} & \mathtt{r_0(X,Y), s(X,Z), s_0(Y,Z), \neg v(X,Y,Z).} \\ r_6: & \mathtt{panic} & \mathtt{:-} & \mathtt{r_0(X,Y), s(X,Z), s(Y,Z), \neg v(X,Y,Z).} \end{array}$$

∎

Solving the QC problem

The queries we want to compare both consist of rules with the same head (*panic*) and whose body contains subgoals with both "constant" predicates and "variable" predicates, and one negated subgoal with a constant predicate. A *constant* predicate refers to a predicate whose relation is totally known (either because it is given or derived from other known relations). If the relation for a predicate is totally unknown, the predicate is said to be *variable*. Such rules have the form

$$panic :\!-\ A(\bar{X}, \bar{Y}), B(\bar{Y}, \bar{Z}), \neg r(\bar{X}, \bar{Y}, \bar{Z}). \tag{13}$$

where A represents the subgoals with constant predicates, B the subgoals with variable predicates , and r a constant predicate.

The constant predicates can be eliminated by replacing any subgoal with such a predicate by a disjunction of equality comparisons involving the symbols in the relation for the constant predicate. Thus, if p is a constant predicate with relation P, the subgoal $p(\bar{X})$ can be expanded to $\bigvee_{\bar{t} \in P} \bar{X} = \bar{t}$, and $\neg p(\bar{X})$ to $\bigwedge_{\bar{t} \in P} \bar{X} \neq \bar{t}$. By eliminating all constant predicates and by factoring out all the disjunctions, a CQ of the form (13) can be rewritten as a union of CQ's with only variable predicates but also with inequality comparisons. Recall the following result from [GSUW94] paraphrased here:

Theorem 4.1 *([GSUW94]) Consider the conjunctive queries $Q_i : H_i :\!-\ B_i, A_i$ where B_i represents a conjunction of ordinary subgoals and A_i a conjunction of arithmetic comparisons. B_i is assumed not to use the same variable twice or any constant. Let M be the set of containment mappings from (H_2,B_2) to (H_1,B_1). Then $Q_1 \subseteq Q_2$ if and only if $A_1 \Rightarrow \bigvee_{h \in M} h(A_2)$.* ∎

Note that if a conjunction of ordinary subgoals uses the same variable twice or uses constants, we can always *rectify* it (that is, eliminate all constants and duplicate variables) by introducing enough new variables and equate these new variables with the constants and duplicate variables. In the case of Theorem 4.1, A_i would include all equality comparisons that result from the rectification of B_i.

Even though Theorem 4.1 can be used to decide the containment of our queries, the problem is that using this straightforward approach leads to containment test expressions involving symbols from the relation for the constant predicates, but these symbols are not known at view definition time. We would rather like to have test expressions that refer to the given relations as a whole rather than their contents, as is usually the case with traditional database queries.

The solution is to use the relation contents of the constant predicates only as a conceptual tool and to develop containment tests that actually use these constant predicates in their *unexpanded* form. To this end, we have extended the containment results from [GSUW94] to deal symbolically with constant predicates.

Lemma 4.1 *Consider the queries $Q_i : H_i :\!-\ B_i, A_i$ where B_i represents a conjunction of ordinary subgoals and A_i a boolean combination of arithmetic comparisons. B_i is assumed not to use the same variable twice or any constant. Let M be the set of containment mappings from (H_2,B_2) to (H_1,B_1). Then $Q_1 \subseteq Q_2$ if and only if $A_1 \Rightarrow \bigvee_{h \in M} h(A_2)$.* ∎

Theorem 4.2 *Consider the queries $Q_i : H_i(\bar{X}'_i, \bar{Y}'_i, \bar{Z}'_i) :\!-\ A_i(\bar{X}_i, \bar{Y}_i), B_i(\bar{Y}_i, \bar{Z}_i), \neg c_i(\bar{X}_i, \bar{Y}_i, \bar{Z}_i)$ where A_i represents a conjunction of subgoals with constant predicates, B_i a conjunction of subgoals with variable predicates and c_i an atom with a constant predicate. Let M be the set of containment mappings from (H_2,rectified B_2) to (H_1,B_1). Let $G_h(\bar{Y}_1, \bar{Z}_1)$ be the result of applying some $h \in M$ to the equality comparisons obtained from the rectification of B_2. Then $Q_1 \subseteq Q_2$ if and only if:*

$$\begin{aligned}(\forall \bar{X}_1, \bar{Y}_1, \bar{Z}_1)\ \ A_1(\bar{X}_1, \bar{Y}_1) \wedge \bigwedge_{h \in M} [\neg G_h(\bar{Y}_1, \bar{Z}_1) \vee [(\forall \bar{X}_2)\ A_2(\bar{X}_2, h(\bar{Y}_2)) \Rightarrow c_2(\bar{X}_2, h(\bar{Y}_2), h(\bar{Z}_2))]] \\ \Rightarrow c_1(\bar{X}_1, \bar{Y}_1, \bar{Z}_1)\end{aligned} \tag{14}$$

∎

The proof can be found in the Appendix. This result can be extended to containment between unions of queries of the form (13) in a straightforward manner. Thus, the original SM problem reduces to a QC problem that can be solved using a test in the form (14) of a first order query over known relations.

Example 4.2 Consider solving the containment in Example 4.1. This containment is the conjunction of subcontainments between each *panic* rule in the left query on the one hand, and the right query on the other hand. Consider for instance the subcontainment between query $\{r_1\}$ and query $\{r_4, r_5, r_6\}$. It holds if and only if:

$$\begin{aligned}(\forall \mathtt{Z})\ & \mathtt{s}_0\mathtt{(1,Z)} \wedge [(\forall \mathtt{X})\ \mathtt{r}_0\mathtt{(X,2)} \wedge \mathtt{s}_0\mathtt{(X,Z)} \Rightarrow \mathtt{v(X,2,Z)}] \wedge [(\forall \mathtt{Y})\ \mathtt{r}_0\mathtt{(2,Y)} \wedge \mathtt{s}_0\mathtt{(Y,Z)} \Rightarrow \mathtt{v(2,Y,Z)}] \\ & \wedge [\mathtt{r}_0\mathtt{(2,2)} \Rightarrow \mathtt{v(2,2,Z)}] \\ & \Rightarrow \mathtt{v}_{12}\mathtt{(Z)} \end{aligned} \tag{15}$$

∎

Making the query test safe

The first-order query (14) that tests for SM is not obviously safe in general, since $\bar{Z}_1$ is not range-restricted. To analyze it, let us rewrite it in the following slightly more general form

$$(\forall \bar{Z})\ a_1(\bar{Z}_1) \wedge a_2(\bar{Z}_2) \wedge \ldots \wedge a_n(\bar{Z}_n) \Rightarrow r(\bar{Z}) \tag{16}$$

where r represents a safe query (a query is said to be safe when all variables used in the query are range-restricted), and $a_i(\bar{Z}_i)$ denotes a (generally unsafe) query having one of the following forms:

- $\alpha_i(\bar{Z}_i)$: $(\forall \bar{X}_i)\ p_i(\bar{X}_i) \Rightarrow q_i(\bar{X}_i, \bar{Z}_i)$, where p_i and q_i represent safe queries.
- $\beta_i(\bar{Z}_i)$: $(\forall \bar{X}_i)\ p_i(\bar{X}_i, \bar{Z}'_i) \Rightarrow q_i(\bar{X}_i, \bar{Z}_i)$ where $\bar{Z}'_i \subseteq \bar{Z}_i$ and $\bar{Z}'_i \neq \emptyset$, and p_i and q_i represent safe queries.
- $\gamma_i(\bar{Z}_i)$: $\neg[(\mu_1 = \nu_1) \wedge \ldots \wedge (\mu_m = \nu_m)]$, where the μ_j's and ν_j's are either constants or variables from $\bar{Z}_i$. We assume that no comparison involves the same variable.

While (16) is not safe in general, the question is whether and how they can be made safe. The answer is stated in the following theorem.

Theorem 4.3 *Formula* $(\forall \bar{Z})\ a_1(\bar{Z}_1) \wedge \ldots \wedge a_n(\bar{Z}_n) \Rightarrow r(\bar{Z})$*, where* $a_i(\bar{Z}_i)$ *is either of the form* $\alpha_i(\bar{Z}_i)$*,* $\beta_i(\bar{Z}_i)$ *or* $\gamma_i(\bar{Z}_i)$*, can always be rewritten as a safe query.* ∎

The proof can be found in the Appendix. We use a constructive proof that shows how to rewrite (16) to a logically equivalent formula that is safe. The details of the construction in the proof can be used as an algorithm that computes an equivalent but safe query.

Example 4.3 The unsafe query (15) can be rewritten to the following safe query:

$$\begin{aligned}& \mathtt{r}_0\mathtt{(2,2)} \wedge (\forall \mathtt{Z})\ [\mathtt{s}_0\mathtt{(1,Z)} \wedge \beta_1\mathtt{(Z)} \wedge \beta_2\mathtt{(Z)} \wedge \mathtt{v(2,2,Z)} \Rightarrow \mathtt{v}_{12}\mathtt{(Z)}] \vee \\ & \neg\mathtt{r}_0\mathtt{(2,2)} \wedge (\forall \mathtt{Z})\ [\mathtt{s}_0\mathtt{(1,Z)} \wedge \beta_1\mathtt{(Z)} \wedge \beta_2\mathtt{(Z)} \Rightarrow \mathtt{v}_{12}\mathtt{(Z)}]\end{aligned} \tag{17}$$

where β_1(Z) denotes $(\forall$X$)$ [$\texttt{r}_0$(X,2) $\wedge$ $\texttt{s}_0$(X,Z) $\Rightarrow$ v(X,2,Z)] and β_2(Z) denotes $(\forall$Y$)$ [$\texttt{r}_0$(2,Y) $\wedge$ $\texttt{s}_0$(Y,Z) $\Rightarrow$ v(2,Y,Z)].

Test (17) can either be directly implemented or further optimized. Query optimization is beyond the scope of this work, but as an example, (17) can be simplified to:

$$\texttt{r}_0\texttt{(2,2)} \vee (\forall \texttt{Z})\ [\texttt{s}_0\texttt{(1,Z)} \wedge \neg(\exists \texttt{X})(\texttt{r}_0\texttt{(X,2)} \wedge \texttt{s}_0\texttt{(X,Z)}) \wedge \neg(\exists \texttt{Y})(\texttt{r}_0\texttt{(2,Y)} \wedge \texttt{s}_0\texttt{(Y,Z)}) \Rightarrow \texttt{v}_{12}\texttt{(Z)}]$$

This condition can be implemented as the following SQL fragment

```
WHERE EXISTS (SELECT * FROM r0 r1 WHERE r1.X = 2 AND r1.Y = 2)
OR    NOT EXISTS (SELECT * FROM s0 s1
                  WHERE s1.X = 1
                  AND   NOT EXISTS (SELECT * FROM r0 r2, s0 s2
                                    WHERE  r2.X = s2.X
                                    AND    r2.Y = 2
                                    AND    s2.Z = s1.Z)
                  AND   NOT EXISTS (SELECT * FROM r0 r3, s0 s3
                                    WHERE  r3.X = 2
                                    AND    r3.Y = s3.X
                                    AND    s3.Z = s1.Z)
                  AND   NOT EXISTS (SELECT * FROM v12
                                    WHERE  v12.Z = s1.Z)
```

where the views r0(X,Y), s0(X,Z) and v12(Z) are appropriately defined over v. ∎

4.2 Solving the SM problem when additional knowledge about the base relations is available

In the basic SM problem, we assume no knowledge about the base relations. In practice however, we may be given additional knowledge about the base data. For instance, to find the cheapest plan for maintaining a materialized view, we may need to consider arbitrary subsets of the base relations to be available. Also, in a data warehousing environment, it is not uncommon to have additional views that are materialized in the warehouse. These additional views can be exploited to maintain the given view.

Using known base relations

The underlying database now consists of a portion D_{known} of constant predicates (of known relations) and a portion $D_{unknown}$ of variable predicates (of unknown relations). Now given a view $V = \mathcal{Q}(D_{known} \cup D_{unknown})$, consider the canonical database $D_0 = D_{known} \cup \mathcal{Q}^{-1}(V)$. It is easy to show that D_0 is a valid database and any valid database consists of D_0 augmented with some set D of tuples for the variable predicates (see Appendix). Thus self-maintainability of V with respect to a given insertion μ can be captured by the condition:

$$(\forall D)\ \mathcal{Q}(D_0 \cup D) = V \Rightarrow \mathcal{Q}(D_0 \cup D \cup \mu) = \mathcal{Q}(D_0 \cup \mu)$$

or, since $\mathcal{Q}$ is monotonic and D_0 is a valid database,

$$(\forall D)\ \mathcal{Q}(D_0 \cup D) \subseteq V \Rightarrow \mathcal{Q}(D_0 \cup D \cup \mu) \subseteq \mathcal{Q}(D_0 \cup \mu)$$

or equivalently

$$(\forall D)\ \mathcal{Q}(D_0 \cup D \cup \mu) \not\subseteq \mathcal{Q}(D_0 \cup \mu) \Rightarrow \mathcal{Q}(D_0 \cup D) \not\subseteq V \tag{18}$$

where D consists of only those variable predicates in $\mathcal{Q}$. This condition can be reduced to the familiar containment test of queries similar to (11) and (12), where I and J are now restricted to subgoals with variable predicates. These queries consist of rules of the form (13) and we already know how to test their containment.

Example 4.4 Consider the view P in Example 1.1 whose definition is rewritten here as:

```
p(X,Y) :- rtrip'(X,Y),nstop(X,Y),nstop(Y,X).
```

where `rtrip'(X,Y)` is obtained from `rtrip(X,Y,U)` by projecting `U` out. Assume `rtrip` is given (and thus `rtrip'` is known). The canonical database consists of the known `rtrip'`, plus relation $\texttt{nstop}_0$ defined by:

$$\begin{array}{lll} \texttt{nstop}_0\texttt{(X,Y)} & \texttt{:-} & \texttt{p(X,Y).} \\ \texttt{nstop}_0\texttt{(Y,X)} & \texttt{:-} & \texttt{p(X,Y).} \end{array}$$

Under the insertion of `rtrip'(chicago,london)` (which follows from the insertion of `rtrip(chicago,london,john)`), the query on the left of containment (18) has the following rules:

$$\begin{array}{rll} \texttt{p}_{c,l} & \texttt{:-} & \texttt{nstop}_0\texttt{(chicago,london)},\texttt{nstop}_0\texttt{(london,chicago)}. \\ \texttt{panic} & \texttt{:-} & \texttt{nstop}_0\texttt{(chicago,london)},\texttt{nstop(london,chicago)},\neg\texttt{p}_{c,l}. \\ \texttt{panic} & \texttt{:-} & \texttt{nstop(chicago,london)},\texttt{nstop}_0\texttt{(london,chicago)},\neg\texttt{p}_{c,l}. \\ \texttt{panic} & \texttt{:-} & \texttt{nstop(chicago,london)},\texttt{nstop(london,chicago)},\neg\texttt{p}_{c,l}. \end{array}$$

and the query on the right has the following rules:

$$\begin{array}{lll} \texttt{panic} & \texttt{:-} & \texttt{rtrip'(X,Y)},\texttt{nstop}_0\texttt{(X,Y)},\texttt{nstop(Y,X)},\neg\texttt{p(X,Y)}. \\ \texttt{panic} & \texttt{:-} & \texttt{rtrip'(X,Y)},\texttt{nstop(X,Y)},\texttt{nstop}_0\texttt{(Y,X)},\neg\texttt{p(X,Y)}. \\ \texttt{panic} & \texttt{:-} & \texttt{rtrip'(X,Y)},\texttt{nstop(X,Y)},\texttt{nstop(Y,X)},\neg\texttt{p(X,Y)}. \end{array}$$

With the understanding that these queries use `nstop` as the only variable predicate, their containment can be solved using previous techniques and can further be simplified to

$$\begin{array}{ll} & [\texttt{rtrip'(chicago,london)} \Rightarrow \texttt{p(chicago,london)}] \\ \wedge & [\texttt{rtrip'(london,chicago)} \Rightarrow \texttt{p(london,chicago)}] \\ \Rightarrow & \texttt{p}_{c,l} \end{array}$$

and ultimately to `rtrip(chicago,london,-)` $\vee$ `rtrip(london,chicago,-)`. This test can implemented as the following SQL fragment

```
WHERE EXISTS (SELECT * FROM rtrip
              WHERE  (rtrip.X = ``chicago'' AND rtrip.Y = ``london'') OR
                     (rtrip.X = ``london'' AND rtrip.Y = ``chicago''))
```

∎

Using Additional views

Besides being given a view $V = \mathcal{Q}(D)$ to maintain, we are also given additional views $V_i = \mathcal{Q}_i(D)$ for $i = 1, \ldots, n$ to be used to maintain view V. The canonical database in this case is $D_0 = \mathcal{Q}^{-1}(V) \cup \bigcup_i \mathcal{Q}_i^{-1}(V_i)$. Again, it is easy to show that D_0 is valid (here, a database D is said to be valid if $\mathcal{Q}(D) = V$ and $\mathcal{Q}_i(D) = V_i$ for all i), and any valid database consists of D_0 augmented with some D (see Appendix). Then V is self-maintainable with respect to a given insertion μ if and only if

$$(\forall D)\ \mathcal{Q}(D_0 \cup D) = V \wedge \bigwedge_i \mathcal{Q}_i(D_0 \cup D) = V_i \Rightarrow \mathcal{Q}(D_0 \cup D \cup \mu) = \mathcal{Q}(D_0 \cup \mu)$$

or, since $\mathcal{Q}$ and the $\mathcal{Q}_i$'s are monotonic, and D_0 is a valid database,

$$(\forall D)\ \mathcal{Q}(D_0 \cup D) \subseteq V \wedge \bigwedge_i \mathcal{Q}_i(D_0 \cup D) \subseteq V_i \Rightarrow \mathcal{Q}(D_0 \cup D \cup \mu) \subseteq \mathcal{Q}(D_0 \cup \mu)$$

or equivalently

$$(\forall D)\ \mathcal{Q}(D_0 \cup D \cup \mu) \not\subseteq \mathcal{Q}(D_0 \cup \mu) \Rightarrow \mathcal{Q}(D_0 \cup D) \not\subseteq V \vee \bigvee_i \mathcal{Q}_i(D_0 \cup D) \not\subseteq V_i \tag{19}$$

While queries similar to (11) (resp. (12)) can be derived from the left (resp. right) hand side of the containment equation (19), the following additional rules are obtained from the right hand side (assuming $\mathcal{Q}_i : v_i(\bar{X}_i) :- M^i(\bar{X}_i)$):

$$panic \quad :- \quad M^i_I(\bar{X}_i), \neg v_i(\bar{X}_i).$$

for every i and for every subset I of subgoals in M_i.

Again, these queries consist of rules of the form (13) and we already know how to test their containment.

References

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

[GJM94] Gupta A., Jagadish H. V. and Mumick I. S.: Data Integration Using Self-Maintainable Views. In *Technical Memorandum 113880-941101-32,* AT&T Bell Labs, November 1995.

[GM95] Gupta A. and Mumick I. S.: Maintenance of Materialized Views: Problems, Techniques, and Applications. In *IEEE Data Engineering Bulletin, Special Issue on Materialized VIews & Data Warehousing,* 18(2), June 1995.

[GSUW94] Gupta A., Sagiv Y., Ullman J. D. and Widom J.: Constraint Checking with Partial Information. *Proc. 13th ACM Symp. on Principles of Database Systems,* 1994, pp. 45–55.

[Huyn96a] Huyn N.: Efficient View Self-Maintenance. In *Proc. Workshop on Materialized Views: Techniques and Applications,* pp. 17–25, 1996.

[Huyn96a-TR] Huyn N.: Testing Efficient View Self-Maintenance. *Unpublished Technical Report* available as `URL http://www-db.stanford.edu/pub/huyn/1996/cqvsm-tr.ps,` 1996.

[IK93] Inmon W. H. and Kelley C.: Rdb/VMS: Developing the Data Warehouse. *QED Publishing Group,* Boston, Massachussetts, 1993.

[QGMW95] Quass D., Gupta A., Mumick I. and Widom J.: Making Views Self-Maintainable for Data Warehousing. In *Technical report, Stanford University,* 1995.

[RED] Red Brick Systems. *Red Brick Warehouse,* 1995

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

[Ull88] Ullman J. D.: *Principles of Database and Knowledge-Base Systems,* Volumes 1 and 2. Computer Science Press, 1988.

[Wie92] Wiederhold G.: Mediators in the architecture of future information systems. In *IEEE Computer,* 1992.

[ZGHW95] Zhuge Y., Garcia-Molina H., Hammer J. and Widom J: View Maintenance in a Warehousing Environment. In *Proc. ACM SIGMOD,* San Jose, California, 1995.

A Self-Maintainability of Views with Exposed Self-Joins

The proofs in this section parallel those in the case where the view has no self-joins. In the following, we adapt the proofs from [Huyn96a-TR] in order to handle the presence of exposed self-joins.

A.1 Join variables totally exposed

In this section, we give the proof for Theorem 3.3.

Condition (5) is sufficient for SM

Let D be a database instance consistent with V. Assume (5) holds. We need to show that $Q(D^{\mu})$ does not depend on D. Let $\bar{a}_V$ and $\bar{a}_W$ denote the projections of $\bar{a}$ onto the components in $\bar{V}$ and $\bar{W}$ respectively. For every group g_i in the minimal Z-partition of of S, let $\bar{a}_i$ denote the projection of $\bar{a}$ onto the components in $\bar{U}_i$ (the distinguished variables from g_i).

$$\begin{aligned}
&Q(D^{\mu})\\
&= Q(D \cup \{r(\bar{b}, \bar{a})\}) = Q(D) \cup \{(\bar{b}', \bar{a}, \bar{z}') \mid M(\bar{a}_V) \in D, S(\bar{a}_W, \bar{z}) \in D\}\\
&= V \cup \{(\bar{b}', \bar{a})\} \times \{(\bar{z}'_1) \mid S_1(\bar{a}_1, \bar{z}_1) \in D\} \times \ldots \times \{(\bar{z}'_n) \mid S_n(\bar{a}_n, \bar{z}_n) \in D\}\\
&= V \cup \{(\bar{b}', \bar{a})\} \times \{(\bar{z}'_1) \mid V(\bar{x}', \bar{u}, \bar{z}') \wedge \bar{u}_1 = \bar{a}_1\} \times \ldots \times \{(\bar{z}'_n) \mid V(\bar{x}', \bar{u}, \bar{z}') \wedge \bar{u}_n = \bar{a}_n\}
\end{aligned}$$

Condition (5) is necessary for SM

Assume (5) is false. The proof, very similar to the case where the view definition has no self-joins (see [Huyn96a-TR]), consists of finding two databases D_1 and $D_2 = D_1 + \Delta$ that are both consistent with V prior to update but that derive different views after update.

For D_1, we again use the "canonical" database (constructed in the usual way). While it is easy to show consistency (i.e. $Q(D_1) = V$) when Q has no self-join, consistency does not always hold in the general case with self-joins. However, when the variables used in subgoals with repeated predicates are all exposed in the head, consistency can be shown.

Also, it is easy to show that since (5) is false, $M(\bar{a}_V)$ and $S(\bar{a}_W, -)$ cannot both hold in D_1. Thus $Q(D_1^{\mu}) = Q(D_1)$.

For D_2, Δ is constructed in a fashion similar to the case with no self-join, with the following addition: M is treated as a group and when $\tau_{\bar{a}}$ is false, we add to Δ the atoms in $M(\bar{a}_V)$ that are not already in D_1. Note that since M may have subgoals with repeated predicates, it may generate new tuples other than just $(\bar{a}_V)$. However, there cannot be any tuple in r that can join with these new tuples, since by construction of the canonical database D_1, since if $r(-, \bar{c})$ is in D_1, it must be the case that $M(\bar{c}_V)$ is also in D_1. Thus, the addition of Δ to D_1 does not cause Q to generate new tuples, and $Q(D_2) = Q(D_1)(= V)$.

Now, by construction of Δ, $Q(D_2^{\mu})$ contains a tuple $(\bar{b}', \bar{a}, \bar{z}')$ for some $\bar{z}'$. This tuple cannot be in V since otherwise, it would contradict the hypothesis that (5) is false. Thus $Q(D_2^{\mu}) \neq Q(D_1^{\mu})$.

A.2 Join variables partially exposed

In this section, we give the proof for Theorem 3.4.

Condition (8) is sufficient for SM

Let D be a database instance consistent with V. Assume (8) holds. We need to show that $Q(D^{\mu})$ does not depend on D.

$$Q(D^{\mu}) = Q(D \cup \{r(\bar{b}, \bar{a})\}) = Q(D) \cup \{(\bar{b}', \bar{a}', \bar{z}') \mid M(\bar{a}_V) \in D, S(\bar{a}_W, \bar{z}) \in D\}$$

Since $\tau_{\bar{a}}$ is true by hypothesis, $M(\bar{a}_V) \in D$. $Q(D^{\mu})$ is reduced to

$$V \cup \{(\bar{b}', \bar{a}', \bar{z}') \mid S(\bar{a}_W, \bar{z}) \in D\}$$

which can be shown to reduce to simply V, using a proof similar to the case of no self-joins (see [Huyn96a-TR]).

Condition (8) is necessary for SM

Assume (8) is false. Again, the proof is very similar to the case where the view definition has no self-joins (see [Huyn96a-TR]), where we construct two databases D_1 and $D_2 = D_1 + \Delta$ that are both consistent with V prior to update but that derive different views after update.

For D_1, we again use the "canonical" database (constructed in the usual way). Again it is easy to show that $Q(D_1) = V$, especially since the variables used in M are all exposed in the head. Furthermore, $Q(D_1^{\mu}) = Q(D_1)$ can be shown using a proof similar to the case with no self-join.

For D_2, Δ is constructed in a fashion similar to the case with no self-join, with the following addition: M is treated as a group and when $\tau_{\bar{a}}$ is false, we add to Δ the atoms in $M(\bar{a}_V)$ that are not already in D_1.

The rest of the proof, namely that $Q(D_2) = Q(D_1)$ and then $Q(D_2^{\mu}) \neq Q(D_1^{\mu})$, parallels the corresponding proof in [Huyn96a-TR] and is not repeated here.

B Canonical Databases

In this section, we show that under some circumstances, canonical databases are minimal valid databases. Since the notions of canonical database and valid database are relative to what information is available, we will consider each case in turn. In the following, we assume that views are defined as conjunctive queries (that may have self-joins), and that that in any view definition, all body variables are exposed in the head.

B.1 Single View

Given a single view $V = \mathcal{Q}(D)$, the canonical database D_0 in this case is simply $\mathcal{Q}^{-1}(V)$. On the one hand, at least V can be regenerated from D_0, that is, $\mathcal{Q}(D_0) \supseteq V$. On the other hand, since every atom in D_0 must be true in order to explain every tuple in V, any valid database D necessarily contains D_0. Since $\mathcal{Q}$ is monotonic ,$\mathcal{Q}(D_0) \subseteq \mathcal{Q}(D)$. Assuming that a valid database exists, we infer that $\mathcal{Q}(D_0) \subseteq V$. Therefore, $\mathcal{Q}(D_0) = V$. Since any valid database contains D_0, D_0 is the minimal valid database.

B.2 Multiple Views

When views $V_i = \mathcal{Q}_i(D)$ for $i = 1, \ldots, n$ are given, the canonical database D_0 is $\bigcup_i \mathcal{Q}_i^{-1}(V_i)$. We want to show D_0 is valid, that is, $\mathcal{Q}_i(D_0) = V_i$ for all i.

First, using the same argument as in the case of single view, D_0 has all the tuples needed to justify every tuple in each V_i. So $\mathcal{Q}_i(D_0) \supseteq V_i$ for all i.

On the other hand, since all atoms in D_0 are necessary to justify each V_i, any valid database D must contain D_0. Since each $\mathcal{Q}_i$ is monotonic, $\mathcal{Q}_i(D_0) \subseteq \mathcal{Q}_i(D)$. In other words, provided that a valid database exists, $\mathcal{Q}_i(D_0) \subseteq V_i$.

B.3 Known Base Relations

The fact that some of the base relations are known only places additional constraints on the valid databases, i.e., that any valid database includes these known relations exactly as they are given. The remaining (variable) portions of these valid databases are subject to the same arguments as above.

C Containment of Queries with Constant Predicates

C.1 Proof for Lemma 4.1

Since a boolean combination of arithmetic comparisons can always be expressed as a disjunction of conjunctions of arithmetic comparisons, it is sufficient to prove the following lemma.

Lemma C.1 *Consider the queries $\mathcal{P} : G$:- $R, \bigvee_i A_i$ and $\mathcal{Q} : H$:- $S, \bigvee_j B_j$, where R and S represent conjunctions of ordinary subgoals that do not use the same variable twice or any constant, A_i and B_j represent conjunctions of arithmetic comparisons. Let M be the set of containment mappings from (H,S) to (G,R). Then $\mathcal{P} \subseteq \mathcal{Q}$ if and only if $(\bigvee_i A_i) \Rightarrow \bigvee_{h \in M} h(\bigvee_j B_j)$.* ∎

Proof: The queries we want to compare are just unions of conjunctive queries with arithmetic comparisons. Applying Theorem 4.1 (paraphrased from [GSUW94]), the containment holds if and only for every i, the following holds

$$A_i \Rightarrow \bigvee_j \bigvee_{h \in M} h(B_j)$$

The disjunction over j can be pushed all the way into the B_j's. Furthermore, since the right hand side of each of these implications is independent of i, the A_i's can be regrouped into a disjunction. ∎

C.2 Proof for Theorem 4.2

The proof essentially consists of putting the given queries in an appropriate form so that Lemma 4.1 can be applied. In fact, an extension of the lemma will be used instead, where: (1) only the second query needs rectification; (2) the second query is a union of subqueries, in which case the implicant in the containment condition needs to be augmented with a disjunction over each subquery.

Query $\mathcal{Q}_1$ already has the required form, since: $A_1(\bar{X}_1, \bar{Y}_1)$ can be viewed as a disjunction of conjunctions of equality comparisons involving variables in $\bar{X}_1$ and $\bar{Y}_1$ on the one hand, and constant symbols from known relations on the other hand; similarly, $\neg c_1(\bar{X}_1, \bar{Y}_1, \bar{Z}_1)$ can be viewed as a conjunction of disjunctions of inequality comparisons involving variables in $\bar{X}_1$, $\bar{Y}_1$ and $\bar{Z}_1$ on the one hand, and constant symbols from known relations on the other hand.

Query $\mathcal{Q}_2$ can be viewed as a union, over each tuple $(\bar{x}_2, \bar{y}_2) \in A_2$, of the following subquery:

$$H_2(\bar{X}'_2, \bar{Y}'_2, \bar{Z}'_2) \text{ :- } B_2(\bar{Y}_2, \bar{Z}_2), (\bar{Y}_2 = \bar{y}_2), \neg c_2(\bar{x}_2, \bar{y}_2, \bar{Z}_2)$$

which, after rectification, takes the form:

$$H_2(\bar{X}'_2, \bar{Y}'_2, \bar{Z}'_2) \text{ :- } D_2(\bar{Y}_2, \bar{Z}_2, \bar{N}), E(\bar{N}, \bar{Y}_2, \bar{Z}_2), (\bar{Y}_2 = \bar{y}_2), \neg c_2(\bar{x}_2, \bar{y}_2, \bar{Z}_2)$$

where $\bar{N}$ represents new variables introduced, D_2 has no duplicate variables or constants, and E represents the additional equality comparisons introduced by the rectification.

Applying the extension of Lemma 4.1 as mentioned above to $\mathcal{Q}_1$ and the modified $\mathcal{Q}_2$ (a union of subqueries), the containment holds if and only if:

$$\begin{aligned}(\forall \bar{X}_1, \bar{Y}_1, \bar{Z}_1) \quad & A_1(\bar{X}_1, \bar{Y}_1) \wedge \neg c_1(\bar{X}_1, \bar{Y}_1, \bar{Z}_1) \\ & \Rightarrow \bigvee_{(\bar{x}_2, \bar{y}_2) \in A_2} \bigvee_{h \in M} h[E(\bar{N}, \bar{Y}_2, \bar{Z}_2) \wedge (\bar{Y}_2 = \bar{y}_2) \wedge \neg c_2(\bar{x}_2, \bar{y}_2, \bar{Z}_2)]\end{aligned}$$

where M is the set of mappings from (H_2, D_2) to (H_1, B_1) that map variables in $\bar{Y}_2$, $\bar{Z}_2$, $\bar{N}$ to variables in $\bar{Y}_1$, $\bar{Z}_1$ and constants. Note that neither M nor E depends on $\bar{x}_2$ and $\bar{y}_2$. We use this fact to push the outer disjunction inside and turn the disjunction into an existential statement, to obtain the following for the implicant:

$$\bigvee_{h \in M} h[E(\bar{N}, \bar{Y}_2, \bar{Z}_2) \wedge (\exists\ \bar{x}_2, \bar{y}_2)(A_2(\bar{x}_2, \bar{y}_2) \wedge (\bar{Y}_2 = \bar{y}_2) \wedge \neg c_2(\bar{x}_2, \bar{y}_2, \bar{Z}_2))]$$

By defining $G_h(\bar{Y}_1, \bar{Z}_1)$ to be $h[E(\bar{N}, \bar{Y}_2, \bar{Z}_2)$, and by replacing $\bar{y}_2$ with $h(\bar{Y}_2)$, we obtain:

$$\bigvee_{h \in M} G_h(\bar{Y}_1, \bar{Z}_1) \wedge (\exists\ \bar{x}_2)(A_2(\bar{x}_2, h(\bar{Y}_2)) \wedge \neg c_2(\bar{x}_2, h(\bar{Y}_2), h(\bar{Z}_2)))$$

D Making Certain Implication Formulas Safe

In some cases, the problem of finding complete tests for view self-maintainability can be solved using query containment techniques. For at least conjunctive-query views, the question of self-maintainability can be reduced to a question of containment of union of conjunctive queries with inequality comparisons. Solving such containment problems often results in first order test formulas that are generally not safe. Since it is desirable to implement these tests as queries that can be evaluated on conventional query engines, we are lead to wonder whether the test formulas can always be made safe (e.g., rewritten in safe, nonrecursive Datalog with negation).

The following notation convention will be used throughout the rest of this paper.

- Predicates with name starting with p, q, r, s, t will be used to represent safe queries and finite relations
- Queries that are not necessarily safe are represented with predicates with name starting with a, b, c, d, e.
- $\bar{Z}$ denotes an ordered set of variables. $\bar{Z}_i$, $\bar{Z}'$ denote subsets of $\bar{Z}$, $\bar{Z}'_i$ denotes a subset of $\bar{Z}_i$.

The formulas in question are of the form

$$(\forall \bar{Z})\ a_1(\bar{Z}_1) \wedge a_2(\bar{Z}_2) \wedge \ldots \wedge a_n(\bar{Z}_n) \Rightarrow r(\bar{Z}) \tag{20}$$

where r represents a safe query and $a_i(\bar{Z}_i)$ denotes a (generally unsafe) query having one of the following forms:

- $\alpha_i(\bar{Z}_i)$: $(\forall \bar{X}_i)\ p_i(\bar{X}_i) \Rightarrow q_i(\bar{X}_i, \bar{Z}_i)$, where p_i and q_i represent safe queries.
- $\beta_i(\bar{Z}_i)$: $(\forall \bar{X}_i)\ p_i(\bar{X}_i, \bar{Z}'_i) \Rightarrow q_i(\bar{X}_i, \bar{Z}_i)$ where $\bar{Z}'_i \subseteq \bar{Z}_i$ and $\bar{Z}'_i \neq \emptyset$, and p_i and q_i represent safe queries.

- $\gamma_i(\bar{Z}_i)$: $\neg[(\mu_1 = \nu_1) \wedge \ldots \wedge (\mu_m = \nu_m)]$, where the μ_j's and ν_j's are either constants or variables from $\bar{Z}_i$. We assume that no comparison involves the same variable.

Formula (20) is generally not safe and the question of whether or not it can always be made safe is not immediately obvious. In the following, we will show that formula (20) can always be rewritten as a safe (boolean) query: we will first give a lemma that deal with the α_i subqueries, and another lemma that deal with the β_i and γ_i subqueries. The two lemmas are then combined to show the main theorem.

Lemma D.1 *Let I be some set of indices and let $b(\bar{Z}')$ denote some arbitrary (not necessarily safe) query. Formula $(\forall \bar{Z})\ (\bigwedge_{i\in I} \alpha_i(\bar{Z}_i)) \wedge b(\bar{Z}') \Rightarrow r(\bar{Z})$ can be rewritten as the disjunction over all subsets $H \subseteq I$ of*

$$(\bigwedge_{i\in H} s_i) \wedge (\bigwedge_{i'\in I-H} \neg s_{i'}) \wedge (\forall \bar{Z})\ [(\bigwedge_{i\in H} t_i(\bar{Z}_i)) \wedge b(\bar{Z}') \Rightarrow r(\bar{Z})]$$

where s_i and $t_i(\bar{Z}_i)$ are safe queries. ∎

Proof: Let s_i be the safe (boolean) query $(\exists \bar{X}_i)\ p_i(\bar{X}_i)$, and let $t_i(\bar{Z}_i)$ be the safe query $(\exists \bar{X}_i)\ q_i(\bar{X}_i, \bar{Z}_i) \wedge \alpha_i(\bar{Z}_i)$. Since $\alpha_i(\bar{Z}_i)$ is logically equivalent to $(\neg s_i \vee t_i(\bar{Z}_i))$, the premise of the implication can be expanded and rewritten as the disjunction over all subsets $H \subseteq I$ of

$$(\bigwedge_{i\in H} s_i) \wedge (\bigwedge_{i'\in I-H} \neg s_{i'}) \wedge (\bigwedge_{i\in H} t_i(\bar{Z}_i)) \wedge b(\bar{Z}')$$

The original formula now has the form $(\forall \bar{Z})\ [\bigvee_H (s_H \wedge t_H(\bar{Z}_H) \wedge b(\bar{Z}'))] \Rightarrow r(\bar{Z})$, which is equivalent to

$$\bigwedge_H [s_H \Rightarrow (\forall \bar{Z})\ (t_H(\bar{Z}_H) \wedge b(\bar{Z}') \Rightarrow r(\bar{Z}))]$$

Since the s_H's are mutually exclusive, we can rewrite the latter formula as

$$\bigvee_H [s_H \wedge (\forall \bar{Z})\ (t_H(\bar{Z}_H) \wedge b(\bar{Z}') \Rightarrow r(\bar{Z}))]$$

∎

Lemma D.2 *Let J and K be two disjoint sets of indices and let $p(\bar{Z}')$ denote a safe query. For any $H \subseteq J$, let $\bar{Z}'_H$ denotes $\bar{Z}' \cup \bigcup_{j\in H} \bar{Z}_j$. When $\bar{Z}' \subset \bar{Z}$, formula $(\forall \bar{Z})\ p(\bar{Z}') \wedge (\bigwedge_{j\in J} \beta_j(\bar{Z}_j)) \wedge (\bigwedge_{k\in K} \gamma_k(\bar{Z}_k)) \Rightarrow r(\bar{Z})$ can be rewritten as the conjunction over all $H \subseteq J$ of either*

$$(\forall \bar{Z})\ [p(\bar{Z}') \wedge (\bigwedge_{j\in H} t_j(\bar{Z}_j)) \wedge (\bigwedge_{j'\in J-H} \neg s_{j'}(\bar{Z}'_{j'})) \wedge (\bigwedge_{k\in K} \gamma_k(\bar{Z}_k)) \Rightarrow r(\bar{Z})]$$

for those H such that $\bar{Z}'_H = \bar{Z}$, or

$$\neg(\exists \bar{Z}'_H)\ [p(\bar{Z}') \wedge (\bigwedge_{j\in H} t_j(\bar{Z}_j)) \wedge \bigwedge_{j'\in J-H, \bar{Z}'_{j'} \subseteq \bar{Z}'_H} \neg s_{j'}(\bar{Z}'_{j'}) \wedge \bigwedge_{k\in K, \bar{Z}_k \subseteq \bar{Z}'_H} \gamma_k(\bar{Z}_k)]$$

for those H such that $\bar{Z}'_H \subset \bar{Z}$, where the $s_j(\bar{Z}'_j)$'s and $t_j(\bar{Z}_j)$'s are safe queries. ∎

Proof: Let $s_j(\bar{Z}'_j)$ be $(\exists \bar{X}_j)\ p_j(\bar{X}_j, \bar{Z}'_j)$ and $t_j(\bar{Z}_j)$ be $[(\exists \bar{X}_j)\ q_j(\bar{X}_j, \bar{Z}_j)] \wedge \beta_j(\bar{Z}_j)$. Rewrite $\beta_j(\bar{Z}_j)$ as $(t_j(\bar{Z}_j) \vee \neg s_j(\bar{Z}'_j))$, that is, the union of a finite set and a cofinite set. The premise of the original formula expands to a disjunction, and the formula itself expands to a conjunction (over H) of implications of the form

$$(\forall \bar{Z})\ [p(\bar{Z}') \wedge (\bigwedge_{j \in H} t_j(\bar{Z}_j)) \wedge (\bigwedge_{j' \in J-H} \neg s_{j'}(\bar{Z}'_{j'})) \wedge (\bigwedge_{k \in K} \gamma_k(\bar{Z}_k)) \Rightarrow r(\bar{Z})] \tag{21}$$

If $Z'_H = Z$, (21) is safe and we are done. Otherwise, if $Z'_H \subset Z$, consider the condition

$$(\exists \bar{Z}'_H)\ [p(\bar{Z}') \wedge (\bigwedge_{j \in H} t_j(\bar{Z}_j)) \wedge \bigwedge_{j' \in J-H, \bar{Z}'_{j'} \subseteq \bar{Z}'_H} \neg s_{j'}(\bar{Z}'_{j'}) \wedge \bigwedge_{k \in K, \bar{Z}_k \subseteq \bar{Z}'_H} \gamma_k(\bar{Z}_k)] \tag{22}$$

If there is a value $\bar{z}'_H$ satisfying (22), there would be infinitely many ways to extend $\bar{z}'_H$ to the remaining variables in $\bar{Z}$ to satisfy the premise in (21), since each of the remaining conjuncts in (21)'s premise uses a variable not in $\bar{Z}'_H$. But since r is finite, (21) cannot be true. Conversely, if (22) is false, the premise in (21) is not satisfiable and (21) is vacuously true. ∎

Theorem D.1 *Formula* $(\forall \bar{Z})\ a_1(\bar{Z}_1) \wedge \ldots \wedge a_n(\bar{Z}_n) \Rightarrow r(\bar{Z})$*, where* $a_i(\bar{Z}_i)$ *is either of the form* $\alpha_i(\bar{Z}_i)$*,* $\beta_i(\bar{Z}_i)$ *or* $\gamma_i(\bar{Z}_i)$*, can always be rewritten as a safe query.* ∎

Proof: For some I, J and K, the original formula can be rewritten as

$$(\forall \bar{Z})\ (\bigwedge_{i \in I} \alpha_i(\bar{Z}_i)) \wedge (\bigwedge_{j \in J} \beta_j(\bar{Z}_j)) \wedge (\bigwedge_{k \in K} \gamma_k(\bar{Z}_k)) \Rightarrow r(\bar{Z}) \tag{23}$$

Applying Lemma D.1, (23) can be rewritten as the disjunction over all subsets $H \subseteq I$ of $(\bigwedge_{i \in H} s_i) \wedge (\bigwedge_{i' \in I-H} \neg s_{i'}) \wedge \varphi_H$, where φ_H is defined as

$$(\forall \bar{Z})\ (\bigwedge_{i \in H} t_i(\bar{Z}_i)) \wedge (\bigwedge_{j \in J} \beta_j(\bar{Z}_j)) \wedge (\bigwedge_{k \in K} \gamma_k(\bar{Z}_k)) \Rightarrow r(\bar{Z})] \tag{24}$$

Let $\bar{Z}_H = \bigcup_{i \in H} \bar{Z}_i$. If $\bar{Z}_H = \bar{Z}$, since the subquery $(\bigwedge_{i \in H} t_i(\bar{Z}_i))$ in (24) is safe, (24) is safe and we are done. Otherwise, if $\bar{Z}_H \subset \bar{Z}$, we can apply Lemma D.2 to rewrite (24) as a safe query. ∎

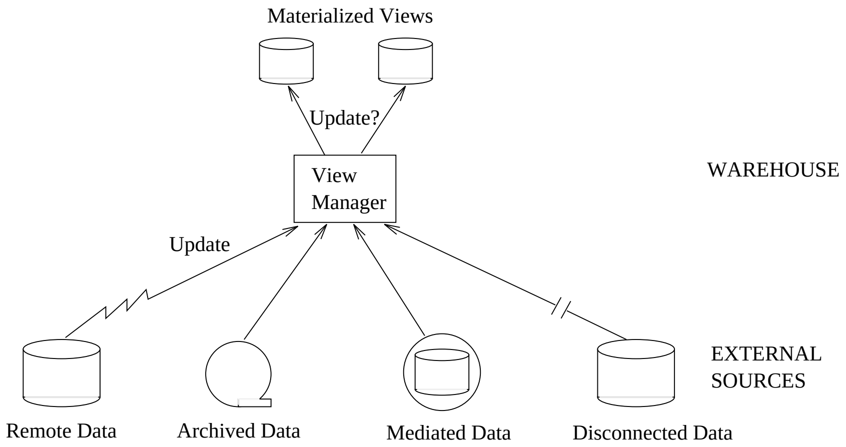

Figure 1: A typical warehouse

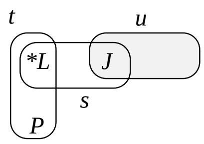

Figure 2: Z-partition in hypergraph form.