Using Partial Information to Update Materialized Views*

Ashish Gupta†
Department of Computer Science
Stanford University, CA 94305-2140
agupta@cs.stanford.edu

José A. Blakeley
Microsoft Corporation
Redmond, WA 98052-6399
joseb@microsoft.com

Abstract

This paper lays the theoretical foundations for identifying classes of views and data partitioning strategies that allow efficient incremental view maintenance using "partial information" about the underlying base relations. We present necessary and sufficient conditions for determining how a materialized **select-project-join** view can be updated using only the view definition, the current view materialization, and the update (i.e., no base relations); and also how to update views using only the view definition, the update, and the updated base relation (i.e., not the other base relations). We generalize the above results to use an arbitrary subset of the base relations and the current view materialization.

Our results are especially useful in distributed databases, in disconnected and mobile computing environments where the underlying database is not always accessible, and in data warehousing applications. Our techniques can be used to minimize remote data access and often to completely avoid remote access.

This paper gives the intuition and theory necessary for identifying and designing views that are efficiently maintainable using partial information.

1 Introduction

Today's computer communications networks give us access to many information sources. A common topology of these systems consists of a large network of workstations (or personal computers) connected to some central database that manages the access to a corporate database or information service. These workstations have substantial computing power as well as large amounts of secondary storage available. Caching commonly accessed queries at the workstations enhances the performance of distributed database systems by reducing communication costs and by relieving the central database from having to recompute a query repeatedly.

Materialized views are stored relations resulting from the evaluation of a query on some base relations or views. They are a form of data caching that can be used to enhance the performance and autonomy of various aspects of a database system by making data available at the point of need and reducing (or even eliminating) remote accesses. In mobile computing, a workstation (e.g., laptop) may be disconnected periodically from the network and the person carrying the workstation may need to continue to make decisions based on the data available in the workstation. Views are the backbone of the idea of *data warehouses* that are being

*The work of A. Gupta was supported by NSF grant IRI-91-16646.
†Current address: IBM Almaden Research Center, San Jose, CA 95120

proposed as a way of integrating and supporting queries on very large integrated heterogeneous systems [Z+94, RB].

However, the performance benefits provided by materialized views have a cost. As the base relations in the database are updated, the materialized views may become out of date. Therefore, efficient view maintenance mechanisms must be developed.

Consider a view definition $\mathcal{C}$ that involves relations $\{R_1, \ldots, R_n\}$ and defines view V. Assume that the update $\mathcal{U}$ affects some relation $R_u, 1 \leq u \leq n$. A view maintenance algorithm needs to compute the change to the view ΔV, and can use *intensional* information such as the view definition $\mathcal{C}$, the update $\mathcal{U}$, and the database schema; *extensional* information such as the contents of underlying base relations and the materialized view; or a *combination* of extensional and intensional information. Let us consider some previous view maintenance results along this spectrum:

1. **Intensional.** These strategies assume that only the view definition $\mathcal{C}$ and the update $\mathcal{U}$ are available to check if $\mathcal{U}$ affects the view [BC79, BCL89, Elk90, LS93]. The strategies are very efficient, but limited in applicability.
2. **Extensional.** These strategies assume that the underlying base relations $\{R_1, \ldots, R_n\}$ and view V are available to *incrementally* update the view [NY83, BLT86, Kuc91, QW91, CW94, DT92, UO92, GMS93]. Sometimes the base relations may be stored on multiple databases and could be expensive to access, or may not be accessible at all.
3. **Extensional and intensional.** Some of these strategies assume that the contents of the materialized view are available but no base relations are available to update the view [TB88]. We found the solution given in [TB88] is not correct. [BBC80] maintained additional stored aggregate information to efficiently update a subclass of SPJ views.

This paper considers the scenario where a view V is materialized at a user's workstation and the underlying base relations are stored at a central site or in other remote workstations. Accesses to data stored at the user's workstation are referred to as *local* accesses, and accesses to the base relations at a central site or other workstations as *remote* accesses. We derive techniques that permit the maintenance of V in the presence of updates to the underlying base relations using only local accesses. Ideally, if all underlying base relations were replicated at the local workstation, then one could simply recompute V (completely or incrementally) every time V became out of date. In practice, replicating the base relations may not be possible because of the amount of space consumed and the problems involved in maintaining all base relation replicas consistent under updates. However, keeping a small number of base relation replicas locally may be feasible. Hence, we explore techniques for view maintenance using "partial information,", *i.e.*, a subset of the set of underlying base relations and the materialized view. Example 1.1 illustrates the use of a base relation replica stored locally. Example 1.2 illustrates the use of the materialized view itself.

EXAMPLE 1.1 Consider the following schema for an employee-department database.

`emp`(E, D, S) % *employee number E in department D has salary S*
`dept`(M, D, MS) % *manager M in department D has salary MS*

Let view `ok_dept`(D, M) be defined to have all departments D that have a manager M and an employee whose salary is greater than the salary of manager M. `ok_dept` is defined as follows:

define view $\texttt{ok\_dept}(D, M)$ **as**
  select $\texttt{emp}.D$, $\texttt{dept}.M$ **from** emp, dept
  where $\texttt{emp}.D = \texttt{dept}.D$ **and** $\texttt{emp}.S > \texttt{dept}.MS$.

Let tuple $(e_2, d_1, 50)$ be inserted into relation emp. If only the view definition and the update are available then it is not possible to infer the effect of the insertion on the view, *i.e.*, the update cannot be inferred as *independent* (*irrelevant*) [BC79, BCL89, Elk90, LS93] with respect to the view.

We can do better if we assume that the updated base relation emp, containing tuple $(e_1, d_1, 65)$, is available locally. We can infer that view ok_dept will not change after the insertion (assuming that view ok_dept does not keep duplicates). This inference can be made by observing that if $\texttt{emp}(e_2, d_1, 50)$ would cause the insertion of tuple (d_1, m) in ok_dept, then the salary of manager m must be < 50. Thus, the salary of manager m would also be < 65, and (d_1, m) would be in ok_dept even before the insertion by virtue of tuple $\texttt{emp}(e_1, d_1, 65)$. This inference is made using the contents of the updated relation emp. Relation dept and view ok_dept are not used. □

EXAMPLE 1.2 Consider again the schema of Example 1.1 and let view $\texttt{good\_dept}(D, S, M)$ be defined to include in the projection list the salaries of employees that have a salary greater than their manager. Formally, good_dept is:

define view $\texttt{good\_dept}(D, S, M)$ **as**
  select $\texttt{emp}.D$, $\texttt{emp}.S$, $\texttt{dept}.M$ **from** emp, dept
  where $\texttt{emp}.D = \texttt{dept}.D$ **and** $\texttt{emp}.S > \texttt{dept}.MS$.

Suppose view good_dept in database DB contains tuple $(d_1, 45, m_1)$. That is, department d_1 has an employee with salary 45 and the salary of manager m_1 in department d_1 is less than 45. If tuple $(e_2, d_1, 50)$ is inserted into relation emp in DB, we can infer that view good_dept should have the additional tuple $(d_1, 50, m_1)$. This inference can be made, using the current view materialization and the inserted tuple, by observing that if manager m_1 has salary < 45 then the salary is also < 50. The two base relations are not used. □

Partial-information-based techniques are often less expensive, but may not always allow complete update of a view; sometimes the view can be partially updated. For instance, let the tuples $t_1 : \texttt{emp}(e_2, d_1, 50)$ and $t_2 : \texttt{emp}(e_3, d_1, 36)$ be inserted into emp in Example 1.2. The change to view good_dept caused by t_1 can be computed using only the old view, but the change caused by t_2 cannot be computed. Partial updates are useful in pipelined architectures, visually displaying answers, or starting other processes that operate with partial information. The updates not decided by the partial information techniques can be processed by alternative strategies in parallel. Such partial optimizations can be factored into the cost model to compare with other tuple-at-a-time methods, and with other global set-based update mechanism to update the entire view. Nevertheless, partial information is often all that is available, making our methods necessary.

The paper develops a formal basis for partial-information-based view-maintenance techniques enabling the inference of knowledge about the state of the underlying base relations using *local* information that includes the view definition, the update, the current view materialization, and varying amounts of base relation replicas. The results of this paper can be used to design views that can be maintained efficiently using partial information, and to decide how to partition and replicate data in a distributed system such that the partial information available to sites is useful for maintaining critical views efficiently. For instance, often it turns out that maintaining extra columns in the view allows efficient maintenance. The results of this paper are useful in making such decisions. Note, we consider **select-project-join** views in this paper.

We offer the following four main contributions. First, discuss how to update a view defined using view definition $\mathcal{C}$ in response to update $\mathcal{U}$ using only the current view materialization and the update, but not using any underlying base relation (Sections 2-4). This problem was posed by Tompa and Blakeley in [TB88] wherein they introduce the notion of *conditionally autonomously computable updates* as updates whose effect on the view can be computed using on the contents of the view. The Addendum presents an example where the solution proposed by Tompa and Blakeley [TB88] does not correctly compute the change to the view.

Second, we generalize the theory underlying the above solutions to use an arbitrary subset of the base relations $\{R_1, \ldots, R_n\}$ to determine the effect of $\mathcal{U}$ on the view (Section 5). Having a single solution framework comprehending all the possible combinations allows a system to choose the best option for every view.

Third, we consider how a specific subset of the set of base relations may be used to update a view in response to update $\mathcal{U}$. Namely, we use only the updated relation R_u (Section 6). This particular instance is particularly instructive because it shows how the results of this paper relate to a different, yet equivalent technique developed for checking integrity constraints. We give a reduction of the view maintenance problem to the problem of integrity constraint checking and then use results in [GW93, GSUW94].

Finally, we examine the computational complexity of evaluating the conditions generated by the theorems in this paper, and propose optimizations to reduce their complexity and increase their applicability (Section 7).

We stress however, that the methodology introduced in this paper is especially useful for determining and designing classes of efficiently maintainable views given partial information. Our results allow a designer to systematically analyse view definitions to determine if the view can be maintained using some partial information. This analysis can be done a database design and view definition time to accordingly partition/replicate database tables such that views are efficiently maintainable. We have successfully used the results of this paper to identify some such classes [GMS95, GMR95]. The intuition of how tuples interact with each other has also helped us develop the notion of *representative relations* [Gup94]. A representative relation is a subset of a base relation that captures the "relevant" part of a relation with respect to a particular view thereby allowing a subset of the original relation to be used in view maintenance or to be cached in case replication is used in some system.

Section 8 presents our conclusions.

Related Work

The problem of incremental view maintenance is a very well studied problem. Efficient solutions to the problem have been given assuming different amounts of information are available for maintenance. This paper lays the theoretical foundations for view maintenance when the available information may include some subset of the underlying relations and the view (in addition to the update and view definition). We do not consider using extra information like functional dependencies, other views, aggregated information about base tables.

[BC79, BCL89, Elk90, LS93] use only the update $\mathcal{U}$ and the view definition to derive the conditions under which the update does not affect the view. These conditions can be obtained using the framework in this paper, except for the non-SPJ views considered in [LS93]. Unlike the cited work, we do not focus on how to evaluate the resulting conditions. Instead we extend the framework to allow other information to be used and we use the cited results for evaluating the conditions.

[NY83, BLT86, Kuc91, QW91, CW94, DT92, UO92, GMS93] assume that all underlying relations involved in the view, and the view itself, are available. The results do not apply if some base relation is needed but is not available.

Another option is to use extra information for view maintenance. [BBC80] use stored aggregate information to efficiently maintain views (and the aggregate information) that are expressible by tuple calculus formulas involving two tuple variables. By contrast, we consider a larger class of views and do not maintain extra information. Our techniques nicely complement those of [BBC80].

[TB88] first considered view maintenance when no base relation, and only the materialized view, is available. Their solution was incorrect. We propose a solution to this problem and generalize the solution to consider the case when the available information includes an arbitrary subset of the base relations in addition to the view. This generalization has not been addressed before.

Materialized views have been used in other contexts, as discussed below, where the emphasis was on how best to use the views in the specified application and not so much on finding efficient view maintenance techniques. Materialized views generalize access paths in a DBMS. Thus, a B-tree index is also a materialized view. Join indexes [Va87] extend this view of materialized views as access path to joins of two relations, storing the join column and record ids of the joining tuples. "ViewCaches" [Rou91] are materialized views defined using extend join indexes even further to very simple select, project, join expressions. Access support relations [KM90] generalize the use of join indexes to two or more tables to represent path expressions in OODBs. Tsatalos et al. [TSI94] use "gmaps" to enhance independence of physical and logical storage representations in a DBMS. They explored ways in which a query optimizer could use gmap views in the generation of execution plans. While using a gmap, updating a logical base relation will translate to updating a series of different physical stores defined as views on the logical schema. All these approaches require a materialized view maintenance mechanism to keep the views current with respect to changes to the underlying database. Our results apply to all of the above uses of materialized views.

2 Updating Views Using Only the Materialized View

The main objective of Sections 2-4 is to develop techniques that enable a view maintenance system to infer knowledge about the state of the underlying base relations based on local information that includes the view definition $\mathcal{C}$, the update $\mathcal{U}$, and the current view materialization V without accessing base relations; and to use this knowledge to incrementally update the view. In Section 5 we illustrate how our techniques extend when a subset of the base relations is also available locally. The formulation of algorithms to carry out efficient computation of the rules derived in this paper is a subject of the authors' ongoing research. A discussion on the complexity of evaluating the rules derived in Sections 2-5 is given in Section 7.

We review (and sometimes redefine) basic concepts from [TB88] and introduce an example that will be used in the remainder of this paper. $\mathcal{C}$ is a select-project-join (SPJ) view definition for a materialized view V. The operation $\mathsf{INS}(R_u, T)$ denotes the insertion of a set of tuples T into relation R_u. Similarly, the operation $\mathsf{DEL}(R_u, \mathcal{C}_D)$ denotes the deletion of tuples from R_u satisfying the condition $\mathcal{C}_D$. A is the set of *distinguished attributes*, namely the attributes present in the view V. A^+ is the set of attributes that are *uniquely determined* by the set of attributes A (A^+ does not represent the closure of functional dependencies) [LY85]. A uniquely determined attribute, is an attribute not present in the view whose value can be unambiguously inferred for every tuple in the view using $\mathcal{C}$ and the current contents of V. $\alpha(\mathcal{C}_D)$ is the set of variables that are used in $\mathcal{C}_D$ (similarly for $\alpha(\mathcal{C})$). The rest of this section assumes that the update $\mathcal{U}$ is a deletion operation. Insertions are considered in Section 4.

EXAMPLE 2.1 Consider view `good_dept` as defined in Example 1.2:

define view `good_dept`(D, S, M) **as**
  select `emp.`D, `emp.`S, `dept.`M **from** `emp, dept`
  where `emp.`D=`dept.`D **and** `emp.`S > `dept.`MS.

and delete operation $\mathsf{DEL}(\texttt{dept}, MS \geq 65)$. Attribute E is not relevant to either the view or to the deletion condition. Therefore, E will be ignored from the discussion. Consider a particular materialization of the view `good_dept` with its corresponding "Z-extended" view `good_dept`^{z}:

`good_dept`				`good_dept`^{z}				
D	S	M		D	S	M	MS	
d_1	50	m_1	t_1	d_1	50	m_1	z_1	t_1^z
d_1	100	m_1	t_2	d_1	100	m_1	z_2	t_2^z
d_1	100	m_2	t_3	d_1	100	m_2	z_3	t_3^z

The "Z-extended" view is created by putting place holders (free variables) for the relevant attributes that have been projected away, *i.e.*, attributes that are used in the view/deletion condition, but not represented in the view. The Z-extended view along with the view condition $\mathcal{C}$ allow us to derive bounds for the values of the missing attributes (e.g., MS) in the current database that produced V. □

Definition 2.1 **(Z-extended view:** V^z**)**[1] Let V^z denote the Z-extension of view V where each tuple $t \in V$ is padded with a distinct variable z_k for each attribute in $Z = (\alpha(\mathcal{C}_D) \cup \alpha(\mathcal{C})) - A^+, 1 \leq k \leq |Z| * |V|$. Each place holder z_k is referred to as a Z-variable (Z-var). □

Given a view V in database DB, the Z-vars in the Z-extended view can be assigned values such that the instantiated tuples exist in the underlying database and satisfy the view definition. For instance, for Example 2.1 we get the following:

[1] This section builds the framework to handle deletions. Thus, many definitions in this section use the information about the delete condition $\mathcal{C}_D$. These definitions will be revisited in Section 4 when we discuss insertions.

EXAMPLE 2.2 The information that can be derived from the tuples t_1^z, t_2^z, t_3^z is:

$t_1^z:\ \exists X : [\texttt{emp}(\_, d_1, 50), \texttt{dept}(d_1, m_1, X), 50 > X].$
$t_2^z:\ \exists X : [\texttt{emp}(\_, d_1, 100), \texttt{dept}(d_1, m_1, X), 100 > X].$
$t_3^z:\ \exists X : [\texttt{emp}(\_, d_1, 100), \texttt{dept}(d_1, m_2, X), 100 > X].$

The scope of an existentially quantified variable is the formula within square brackets immediately following the existential quantification. We use "," to represent logical "and" inside formulas. □

The existence of appropriate tuples in the underlying relations can be captured by creating a template for every relation as follows:

Definition 2.2 (**Z-Projection of relation** R_i: R_i^z) Given a view V defined by $\mathcal{C}$, the Z-projection of a relation R_i used in $\mathcal{C}$ is defined as $R_i^z = \Pi_{G_i}(V^z)$, where $G_i = (\alpha(\mathcal{C}_D) \cup \alpha(\mathcal{C}) \cup A^+) \cap \alpha(R_i), 1 \leq i \leq n$. In other words, G_i represents the attributes from relation R_i that are present in the view or used in $\mathcal{C}_D$ or $\mathcal{C}$. The Z-projection of a base relation represents the smallest relation needed to produce the available view. □

EXAMPLE 2.3 The Z-projection of relation dept contains all the attributes of dept, *i.e.*, $\texttt{dept}^z$ is $\Pi_{D,M,MS}(\texttt{good\_dept}^z)$. The Z-projection of relation emp contains only attributes D and S, *i.e.*, $\texttt{emp}^z$ is $\Pi_{D,S}(\texttt{good\_dept}^z)$.

$\texttt{emp}^z$		$\texttt{dept}^z$		
D	S	M	D	MS
d_1	50	m_1	d_1	z_1
d_1	100	m_1	d_1	z_2
d_1	100	m_2	d_1	z_3

The tuple $\texttt{emp}(d_1, 100)$ appears twice in $\texttt{emp}^z$ and can be replaced by a single occurrence. Similarly, tuple $\texttt{dept}(m_1, d_1, z_1)$ and $\texttt{dept}(m_1, d_1, z_2)$ can be replaced by a single tuple because they differ only in the place holder, and both convey the same information. □

Note, a Z-projected relation does not say anything about tuples that are not represented in the view. The tuples that do not contribute to the view are not needed to determine if a deletion update is conditionally autonomously computable. We discuss tuples that are not represented in the view in Section 4, when we consider insertions.

In general, if two tuples in some Z-projected relation P^z can be unified, then only one of the tuples need be represented. For instance, in Example 2.3 tuples $\texttt{dept}(m_1, d_1, z_1)$ and $\texttt{dept}(m_1, d_1, z_2)$ can be unified. If the view is recomputed from the Z-projections of the base relations, the original view should be obtained. However, often we also obtain tuples that could be in the view if the place holders representing the missing attributes satisfied certain conditions. Consider the view definition of Example 2.1 applied to the relations $\texttt{emp}^z$ and $\texttt{dept}^z$ from Example 2.3. We refer to the resulting view as the project-join mapping $\mathsf{PJM}(\texttt{good\_dept}^z)$.

EXAMPLE 2.4 $\mathsf{PJM}(\texttt{good\_dept}^z)$ is obtained by taking the cross product of $\texttt{emp}^z$ and $\texttt{dept}^z$ and applying the view definition condition to each tuple in the cross product. The last column of the table specifies the view definition applied to the tuples. For some of the tuples in the cross product, say t, the constants are sufficient to infer that $\mathcal{C}(t)$ cannot hold ($\mathcal{C}(t)$ is the view definition condition applied to tuple t). For instance, a tuple in which $\texttt{emp}.D = d_1$ and $\texttt{dept}.D = d_2$. Such tuples are not represented in $\mathsf{PJM}(\texttt{good\_dept}^z)$. For other tuples, Z-variables can be instantiated to satisfy the view definition such that the distinguished variables constitute a tuple in the original view, for instance tuple t_1^z. Finally, for some tuples even though the Z-variables can be instantiated to satisfy $\mathcal{C}(t)$, the distinguished variables *do not* form a tuple in the original view, for instance tuple n_1 ($(d_1, 50, m_2)$ is not in good_dept). For such tuples we know that $\mathcal{C}(t)$ does not hold in the database DB (as indicated by the negated condition associated with n_1).

PJM(good_dept^z)						
D	S	M	D	MS		
d_1	50	m_1	d_1	z_1	t_1^z	$50 > z_1$
d_1	50	m_2	d_1	z_3	n_1	$\neg(50 > z_3)$
d_1	100	m_1	d_1	z_1	t_2^z	$100 > z_1$
d_1	100	m_2	d_1	z_3	t_3^z	$100 > z_3$

□

The above intuition is formalized as follows:

Definition 2.3 (Project-join mapping (PJM)) The project-join mapping $\mathsf{PJM}(V^z)$ of the Z-extended view V^z is defined as $\mathsf{PJM}(V^z) = R_1^z \times R_2^z \times \ldots \times R_m^z$. □

The PJM represents the largest relation that can be constructed using the Z-projection of the base relations. Every tuple μ in $\mathsf{PJM}(V^z)$ belongs to one of the following classes. $\mathcal{C}(\mu)$ is the view definition evaluated using tuple μ.

1. $\mathcal{C}(\mu)$ is satisfiable for some instantiation of the Z-variables in μ, and
   (a) the projection of the distinguished attributes of μ *is* in the materialized view V; or,
   (b) the projection of the distinguished attributes of μ *is not* in the materialized view V.
2. $\mathcal{C}(\mu)$ is *not* satisfiable for any instantiation of the Z-variables, *i.e.*, the constants in μ are sufficient to guarantee that μ cannot contribute to the view. Such tuples can be eliminated from the PJM. This elimination step is an optimization step and does not affect the correctness of our techniques.

Tuples of type (2) are eliminated because they do not contribute any useful information. Tuples of type ($1a$) constitute the set $\mathsf{pos}(V)$. This set is the same as V^z, the Z-extension of V. Tuples of type ($1b$) constitute the set $\mathsf{neg}(V)$ and are numbered n_i. Tuples of type ($1a$) assert the *existence* of certain values for the Z-variables that satisfy the view definition, and tuples of type ($1b$) assert the *absence* of certain values for the Z-variables. That is, the view definition condition applied to a tuple in $\mathsf{neg}(V)$ is not satisfiable in the current database instance.

EXAMPLE 2.5 Consider $\mathsf{PJM}(\texttt{good\_dept}^z)$ from our running example. The tuple n_1 gives us the following information:

$$n_1 : \ \forall X : [(\texttt{emp}(\_, d_1, 50), \texttt{dept}(m_2, d_1, X)) \Rightarrow \neg(50 > X)] \ .$$

□

The information provided by tuples of type ($1a$) is called **positive** information, and the information provided by tuples of type ($1b$) is called **negative** information.

Definition 2.4 (**Positive Information**) $\mathsf{pos}(V)$ provides positive information about a view. Each tuple t_i^z in $\mathsf{pos}(V)$, *i.e.*, in V^z, asserts that the Z-variables in t_i^z can be instantiated to give a tuple that can be obtained from the underlying database and that satisfies the view definition condition. Therefore, positive information consists of a conjunction of existentially quantified statements; one for each tuple in the Z-extended view. Example 2.2 illustrates this for our running example. $\square$

Definition 2.5 (**Negative Information**) $\mathsf{neg}(V)$ provides negative information about a view. Each tuple n_i in $\mathsf{neg}(V)$ asserts that the Z-variables in n_i cannot be instantiated to give a tuple that can be obtained from the underlying database and that satisfies the view definition condition. Therefore, negative information consists of a conjunction of universally quantified statements; one for each tuple in $\mathsf{neg}(V)$. Example 2.5 illustrates this for our running example. $\square$

Definition 2.6 (**Combined Information: $\mathcal{I}(V)$**) $\mathcal{I}(V)$ is a conjunction of the positive information obtained from the tuples in the Z-extended view V^z, and the negative information obtained from the tuples in $\mathsf{neg}(V)$. $\mathcal{I}(V)$ summarizes the information provided by the view V and the definition $\mathcal{C}$. $\square$

EXAMPLE 2.6 For view $\mathtt{good\_dept}$ in Example 2.1, $\mathcal{I}(\mathtt{good\_dept})$ is:

(1) $\exists X : [\mathtt{emp}(\_, d_1, 50), \mathtt{dept}(m_1, d_1, X), 50 > X] \wedge$
(2) $\exists X : [\mathtt{emp}(\_, d_1, 100), \mathtt{dept}(m_1, d_1, X), 100 > X] \wedge$
(3) $\exists X : [\mathtt{emp}(\_, d_1, 100), \mathtt{dept}(m_2, d_1, X), 100 > X] \wedge$
(4) $\forall X : [(\mathtt{emp}(\_, d_1, 50), \mathtt{dept}(m_2, d_1, X)) \Rightarrow \neg(50 > X)]$.

The components that have no variables, but are in the scope of a quantifier can be pulled out:

(1) $\mathtt{emp}(\_, d_1, 50) \wedge \mathtt{emp}(\_, d_1, 100) \wedge$
(2) $\exists X : [\mathtt{dept}(m_1, d_1, X), 50 > X] \wedge$
(3) $\exists X : [\mathtt{dept}(m_1, d_1, X), 100 > X] \wedge$
(4) $\exists X : [\mathtt{dept}(m_2, d_1, X), 100 > X] \wedge$
(5) $\forall X : [\mathtt{dept}(m_2, d_1, X) \Rightarrow 50 \leq X)]$.

Note, $50 > X \Rightarrow 100 > X$. Thus, line 3 does not add anything more than line 2 to $\mathcal{I}(\mathtt{good\_dept})$. $\square$

Any instantiation of the Z-variables in $\mathcal{I}(V)$ that satisfies $\mathcal{I}(V)$, guarantees that view V can be computed from the relations obtained from the instantiated Z-projected relations. Conversely, if the view V can be obtained from given base relations, then $\mathcal{I}(V)$ is satisfiable.

Lemma 2.1 *Given a database DB and view definition $\mathcal{C}$ that derives the view V from DB, the relations in DB satisfy $\mathcal{I}(V)$*[2]. $\square$

[2] We omit details of the proofs of lemmas and theorems in this paper due to the lack of space

Lemma 2.2 *If a database DB satisfies $\mathcal{I}(V)$, then DB derives a superset of view V using view definition $\mathcal{C}$.* $\square$

Therefore, $\mathcal{I}(V)$ characterizes the underlying relations that are represented in the view; either as positive information, or as negative information. However, tuples that are not represented in the view are disregarded. This last bit of information is not needed for deletions, but is needed for insertions as discussed in Section 4.

3 Deletions

In order to compute the effect of the deletion operation $\mathsf{DEL}(R_u, \mathcal{C}_D)$ on the view V, we consider two subproblems: (1) computing those tuples in V that are not affected by the deletion, and (2) computing those tuples in V that are definitely deleted from the view. If we are able to prove one or the other for every tuple $t \in V$, then the deletion can be performed locally (i.e., it is conditionally autonomously computable [TB88]). Often, it is possible to decide the status of a subset of the view after the deletion. The following lemma decides when a tuple $t \in V$ stays in the view.

Lemma 3.1 *Given a non-empty view V, its Z-extension V^z, and* $\mathsf{PJM}(V^z)$; *a tuple $t^z \in V^z$ stays in the view after applying deletion condition $\mathcal{C}_D$ to R_1 (for all underlying databases that result in view V given $\mathcal{C}$) if and only if* $\mathcal{I}(V) \Rightarrow \exists \bar{Z} : [R_1(\Pi_{R_1}(t^z)) \wedge \ldots \wedge R_n(\Pi_{R_n}(t^z)) \wedge \mathcal{C}(t^z) \wedge \neg\mathcal{C}_D(\Pi_{R_1}(t^z))]$, *where $\bar{Z}$ is the set of Z-variables in the tuple t^z.* $\square$

EXAMPLE 3.1 Consider the update $\mathsf{DEL}(\mathtt{dept}, MS \geq 65)$. We need to examine the effect of the deletion condition on the view from Example 2.1. First, consider tuple $\mathtt{good\_dept}(d_1, 50, m_1)$, *i.e.*, dept d_1 has a manager m_1 with salary < 50. The deletion condition would not affect manager m_1 because the salary is < 50 and does not meet the deletion condition. Therefore, $\mathtt{good\_dept}(m_1, d_1, 50)$ should not be affected by the deletion. $\mathtt{good\_dept}(m_1, d_1, 100)$ also stays unaffected due to the same dept tuple. However, tuple $\mathtt{good\_dept}(m_2, d_1, 100)$ is different because manager m_2 has salary < 100 and could be either < 65 or ≥ 65. Thus the status of $\mathtt{good\_dept}(m_2, d_1, 100)$ cannot be decided autonomously. We formally prove the above intuition. Recall that $\mathcal{I}(\mathtt{good\_dept})$ is:

(1) $\mathtt{emp}(\_, d_1, 50) \wedge \mathtt{emp}(\_, d_1, 100) \wedge$
(2) $\exists X : [\mathtt{dept}(m_1, d_1, X), 50 > X] \wedge$
(3) $\exists X : [\mathtt{dept}(m_2, d_1, X), 100 > X] \wedge$
(4) $\forall X : [\mathtt{dept}(m_2, d_1, X) \Rightarrow 50 \leq X)]$.

Consider the tuples of view good_dept in sequence. $\neg[\mathcal{C}_D(\mathtt{dept}(M, D, MS))]$ is $\neg(MS \geq 65)$ *i.e.*, $MS < 65$.

t_1 : Tuple t_1 stays in the view if we can prove that:

$$\mathcal{I}(\mathtt{good\_dept}) \Rightarrow \exists X : [\mathtt{emp}(\_, d_1, 50), \mathtt{dept}(m_1, d_1, X), 50 > X, 65 > X] \qquad \textit{i.e.}.$$
$$\mathcal{I}(\mathtt{good\_dept}) \Rightarrow \{\mathtt{emp}(\_, d_1, 50), \exists X : [\mathtt{dept}(m_1, d_1, X), 50 > X]\} .$$

We can prove the above statement by using lines 1 and 3 of $\mathcal{I}(\mathtt{good\_dept})$. Therefore, we can conclude that tuple t_1 stays in the view after the deletion.

t_2 : Can be inferred to stay in the view after the deletion.

t_3 : Can not be inferred to stay in the view. □

Now we consider the problem of determining when a tuple in the original view can be inferred as definitely deleted from the view.

Lemma 3.2 *Given view V, its Z-extension V^z, and* PJM*(V^z), a tuple $t \in V$ is deleted from the view after applying deletion condition $\mathcal{C}_D$ to R_1 (for all underlying databases that result in view V given $\mathcal{C}$) if and only if $\mathcal{I}(V) \Rightarrow \forall \bar{Z} : \neg[R_1(\Pi_{R_1}(t^z)) \wedge \ldots \wedge R_n(\Pi_{R_n}(t^z)) \wedge \mathcal{C}(t^z) \wedge \neg \mathcal{C}_D(\Pi_{R_1}(t^z))]$, where $\bar{Z}$ is the set of Z-variables in the tuple t^z.* □

EXAMPLE 3.2 Consider the view and delete operation from Example 2.1. Recall $\mathcal{I}(\texttt{good\_dept})$ from Example 3.1 and let the delete operation be $\mathsf{DEL}(\texttt{dept}, MS \geq 50)$. That is, $\mathcal{C}_D(\texttt{dept}(M, D, MS)$ is $MS \geq 50$. Intuitively, tuple t_3 should be deleted. The reason is as follows: tuple t_3 is derivable because there is some manager m_2 in department d_1 with salary < 100. If m_2 had salary < 50, then tuple $\texttt{good\_dept}(m_2, d_1, 50)$ also would have been in the view; but is not. Therefore, we can infer that $50 \leq m_2.MS < 100$ and therefore manager m_2 will be deleted, making t_3 underivable in the new state. First, we consider tuple t_3.

t_3 : Tuple t_3 is deleted from the view if we can prove that:

$$\mathcal{I}(\texttt{good\_dept}) \Rightarrow \forall X : \neg[\texttt{emp}(\_, d_2, 100), \texttt{dept}(m_2, d_1, X), X < 100, X < 50].$$

Observe that the negation of the RHS, $\exists X : [\texttt{emp}(\_, d_2, 100), \texttt{dept}(m_2, d_1, X), X < 50]$, contradicts line 4 of $\mathcal{I}(\texttt{good\_dept})$: $\forall X[\texttt{dept}(m_2, d_1, X) \Rightarrow 50 \leq X)]$. Therefore, we can infer that the above implication holds and that tuple t_3 can be inferred to be deleted.

t_1 : Cannot be inferred to be deleted.

t_2 : Cannot be inferred to be deleted. □

Theorem 3.1 *If a delete operation is conditionally autonomously computable, then every tuple $t \in V$ satisfies either the condition of Lemma 3.1 or the condition of Lemma 3.2.* □

4 Insertions

Recall that the insert operation was $\mathsf{INS}(R_u, T)$, where T is a set of tuples to be inserted into relation R_u. Multiple tuples may need to be inserted into view V to keep the materialized view consistent with an insertion operation. Therefore, given the insertion, we need to determine if: (a) some new tuple is definitely inserted into the view; or (b) no tuple can be inserted into the view independent of the underlying database. In order to answer the first question, we need to build the possible candidates for insertion into the view. In Section 4.1 we consider the first question. Subsequently, in Section 4.2 we tackle the second question.

4.1 Insertion into Base Relation Inserts Tuple Into View

In this section we discuss the problem of computing tuples that are definitely inserted into the view because of $\mathsf{INS}(R_u, T)$. The notions of Z-extended view V^z, and of the Z-projection of a relation $\texttt{emp}^z$ need to be changed from the way they were used for deletions. The set of relevant attributes is now defined by $Z = \alpha(\mathcal{C}) - A^+$ (rather than by $Z = (\alpha(\mathcal{C}_D) \cup \alpha(\mathcal{C})) - A^+$). The change occurs because insertions are completely instantiated unlike the deletion condition that could use uninstantiated attributes that were not in the view.

Definition 4.1 (**Z-extended view:** V^z) V^z is the Z-extension of view V where each tuple $t \in V$ is padded with a Z-variable z_k for each attribute in $Z = \alpha(\mathcal{C}) - A^+, 1 \leq k \leq |Z| * |V|$. □

Definition 4.2 (**Z-Projection of relation** R_i: R_i^z) Given view V defined by $\mathcal{C}$, the Z-projection of relation R_i used in $\mathcal{C}$ is defined as $R_i^z = \Pi_{G_i}(V^z)$, where $G_i = (\alpha(\mathcal{C}) \cup A^+) \cap \alpha(r_i), 1 \leq i \leq n$. □

Definition 4.3 (**Candidate tuples**) The set of *candidate tuples* I^z to be inserted into the view V as a result of $\mathsf{INS}(R_u, T)$, is given by $I^z = R_1^z \times R_2^z \times \ldots \times R_{u-1}^z \times T \times R_{u+1}^z \times \ldots R_m^z$. This is the usual incremental computation rule [UO92]. □

The set I^z represents all new tuples that can be built using the information provided by the newly inserted tuples T and the current view materialization V. As in the case of the PJM, some tuples can be eliminated from I^z. $\mu \in I^z$ can be removed from consideration if:

1. $\mathcal{C}(\mu)$, the view definition condition applied to tuple μ, *is not* satisfiable for any instantiation of the Z-variables in $\mathcal{C}(\mu)$; or, if
2. $\Pi_A(\mu) \in V$, *i.e.*, the view already contains the tuple that μ can contribute.

Both these elimination steps are optimizations and not needed for correctness of our techniques.

EXAMPLE 4.1 Suppose tuples $\texttt{emp}(\_, d_1, 60)$ and $\texttt{emp}(\_, d_2, 90)$ are inserted into relation $\texttt{emp}$. Using $\texttt{dept}^z$ as in Example 2.3, we obtain I^z:

I^z						
D	S	M	D	MS		
d_1	60	m_1	d_1	z_1	ν_1	$60 > z_1$
d_1	60	m_2	d_1	z_3	ν_2	$60 > z_3$

Tuples $< d_2, 90, m_1, d_1, z_1 >$ and $< d_2, 90, m_2, d_1, z_3 >$ do not appear in I^z because $d_2 \neq d_1$ independent of the constant assigned to the Z-variables z_1 and z_3. □

Lemma 4.1 *Given a non-empty view V, Z-extension V^z, project-join mapping $\mathsf{PJM}(V^z)$, set of tuples T to be inserted into relation $R_u, 1 \leq u \leq n$, and the set of candidate tuples I^z, a tuple $t^z \in I^z$ contributes the tuple $\Pi_A(t^z)$ to the view V for all underlying databases yielding V, if and only if* $\mathcal{I}(V) \Rightarrow \exists \bar{Z} : [R_1(\Pi_{R_1}(t^z)) \wedge \ldots \wedge R_n(\Pi_{R_n}(t^z)) \wedge \mathcal{C}(t^z)]$. □

EXAMPLE 4.2 Consider each candidate tuple from Example 4.1. Recall that $\mathcal{I}(\texttt{good\_dept})$ is:

(1) $\texttt{emp}(\_, d_1, 50) \wedge \texttt{emp}(\_, d_1, 100) \wedge$
(2) $\exists X : [\texttt{dept}(m_1, d_1, X), 50 > X] \wedge$
(3) $\exists X : [\texttt{dept}(m_2, d_1, X), 100 > X] \wedge$
(4) $\forall X : [\texttt{dept}(m_2, d_1, X) \Rightarrow 50 \leq X)]$.

ν_1 : For ν_1 to be definitely inserted into the view, we need to prove that:

$$\mathcal{I}(\texttt{good\_dept}) \Rightarrow \exists X : [\texttt{emp}(\_, d_1, 60), \texttt{dept}(m_1, d_1, X), 60 > X] \quad \textit{i.e.}.$$
$$\mathcal{I}(\texttt{good\_dept}) \Rightarrow \exists X : [\texttt{dept}(m_1, d_1, X), 60 > X].$$

The RHS can be proved from line 2 of $\mathcal{I}(\texttt{good\_dept})$ thereby inferring that tuple ν_1 contributes the tuple $(d_1, 60, m_1)$ to view good_dept.

ν_2 : It is not possible to infer that ν_2 is definitely inserted into the view. □

To determine if a tuple was definitely inserted into the view it is sufficient to consider only those tuples that are represented in the view (because the view definition is monotonic). Informally, this is the case because in order for a tuple t to definitely be inserted for all databases that result in V, t should be inserted in the smallest such database; namely one that consists only of the tuples represented in the view and no more.

4.2 Insertion into Base Relation *Does Not* Insert any Tuple Into View

However, in order to prove that an inserted tuple does not contribute to *any* database that yields V, we need to consider all possible databases that could result in the view V and not just the smallest such database. Therefore, we need to consider the tuples that are not reflected in view V, but *may* be in the underlying base relations. The PJM is extended to include information about tuples that are not represented in the view by introducing an extra tuple, called the Z-tuple, in the Z-projection of each underlying relation.

Definition 4.4 (Z-Projection of relation R_i: R_i^z) Given view V defined by $\mathcal{C}$, the Z-projection of a relation R_i used in $\mathcal{C}$ is defined as $R_i^z = \Pi_{G_i}(V^z) \cup \{z'_1, \ldots, z'_{k_i}\}$, where $G_i = (\alpha(\mathcal{C}) \cup A^+) \cap \alpha(r_i), 1 \leq i \leq n$ and k_i is the number of attributes in G_i. The variables $\{z'_1, \ldots, z'_{k_i}\}$ are unique and do not occur anywhere else. We refer to this extra tuple as the Z-tuple for relation R_i. □

The Z-tuple of relation emp represents those tuples of emp that do not contribute to the view V. We associate a condition with the Z-tuple to capture the fact that the Z-tuple does not unify with any of the other tuples in the Z-projection of the relation (Conditions $C'(\texttt{e})$ and $C'(\texttt{d})$ in Example 4.3). PJM(V^z) is defined as before, *i.e.*, as the cross product of the Z-projections of the relations used in $\mathcal{C}$. For every tuple in the PJM that uses the Z-tuple, the condition associated with the Z-tuple is combined with the view definition condition (illustrated in Example 4.3).

Z-tuples are not needed to determine the effect of deletions or to determine if a tuple is definitely inserted, for the following reason. For deletions, the only tuples whose deletion affects the view are tuples represented in the view. Therefore, tuples not represented in the view do not need to be considered. Similarly, when determining the tuples that are inserted into the view for all underlying databases giving the particular view, we need to consider the database that has the bare minimum tuples to generate the given view, and no more. Therefore, the tuples not represented in the view are not needed.

EXAMPLE 4.3 Consider view `good_dept` from Example 2.1, and the Z-projections of relations `emp` and `dept` computed using Definition 4.4

$\texttt{emp}^z$		$\texttt{dept}^z$		
D	S	M	D	MS
d_1	50	m_1	d_1	z_1
d_1	100	m_2	d_1	z_3
z'_1	z'_2	z'_3	z'_4	z'_5

The additional condition associated with tuple $\texttt{emp}(\_, z'_1, z'_2)$ is:

$\mathcal{C}'(\texttt{e})$: $\neg(z'_1 = d_1 \wedge z'_2 = 50) \wedge \neg(z'_1 = d_1 \wedge z'_2 = 100)$.

The additional condition associated with tuple $\texttt{dept}(z'_4, z'_5, z'_3)$ is:

$\mathcal{C}'(\texttt{d})$: $\neg(z'_3 = m_1 \wedge z'_4 = d_1) \wedge \neg(z'_3 = m_2 \wedge z'_4 = d_1)$.

The corresponding project-join mapping is given in Figure 1 on page 23. Note that the tuples that use a Z-tuple in them, are all numbered n_i to reflect that none of them contributes to the view. In addition, each such tuple that uses the Z-tuple from relation `emp` has $\mathcal{C}'(\texttt{e})$ associated with it and tuples from relation `dept` have $\mathcal{C}'(\texttt{d})$ associated with them. Therefore, $\mathcal{I}(\texttt{good\_dept})$ is:

(1) $\texttt{emp}(\_, d_1, 50) \wedge \texttt{emp}(\_, d_1, 100) \wedge$
(2) $\exists X : [\texttt{dept}(m_1, d_1, X), 50 > X] \wedge$
(3) $\exists X : [\texttt{dept}(m_2, d_1, X), 100 > X] \wedge$
(4) $\forall X : [\texttt{dept}(m_2, d_1, X) \Rightarrow 50 \leq X)] \wedge$
(5) $\forall X, Y, Z : [(\texttt{emp}(\_, d_1, 50), \texttt{dept}(X, Y, Z), \mathcal{C}'(\texttt{d})) \Rightarrow \neg(Y = d_1, 50 > Z)] \wedge$
(6) $\forall X, Y, Z : [(\texttt{emp}(\_, d_1, 100), \texttt{dept}(X, Y, Z), \mathcal{C}'(\texttt{d})) \Rightarrow \neg(Y = d_1, 100 > Z)] \wedge$
(7) $\forall V, W, X, Y, Z : [(\texttt{emp}(\_, V, W), \texttt{dept}(X, Y, Z), \mathcal{C}'(\texttt{e}), \mathcal{C}'(\texttt{d})) \Rightarrow \neg(V = Y, W > Z)] \wedge$
(8) $\forall V, W, Z : [(\texttt{emp}(\_, V, W), \texttt{dept}(m_1, d_1, Z), \mathcal{C}'(\texttt{e})) \Rightarrow \neg(V = d_1, W > Z)] \wedge$
(9) $\forall V, W, Z : [(\texttt{emp}(\_, V, W), \texttt{dept}(m_1, d_2, Z), \mathcal{C}'(\texttt{e})) \Rightarrow \neg(V = d_1, W > Z)]$.

□

The modified construction for $\mathcal{I}(V)$ allows us to prove a stronger version of Lemma 2.2.

Lemma 4.2 *If a database DB satisfies $\mathcal{I}(V)$, then DB derives view V using view definition $\mathcal{C}$.* □

The set of candidate tuples also changes because the Z-tuples now need to be considered (for all relations other than the relation to which insertions are made). The candidate tuples are computed using Definition 4.3 on page 12 with the changed definition of the Z-projection of a relation. We now state the lemma that gives the way to infer when insertions definitely contribute to the view. Because of lack of space, we omit illustrating the use of the lemma.

Lemma 4.3 *Consider a non-empty view V, Z-extension V^z, project-join mapping $\mathsf{PJM}(V^z)$, and set of tuples T to be inserted into relation R_u. Let $I^+ \subseteq I^z$ be the set of tuples that are definitely inserted into the view (as determined by Lemma 4.1). Tuple $t^z \in (I^z - I^+)$ does not contribute tuple $t = \Pi_A(\mu)$ to the view V for any underlying database, if and only if $\mathcal{I}(V) \Rightarrow \forall \bar{Z} : \neg[R_1(\Pi_{R_1}(t^z)) \wedge \ldots \wedge R_n(\Pi_{R_n}(t^z)) \wedge \mathcal{C}(t^z))]$.* □

In general, using partial information to prove that an insert operation definitely inserts tuples into a view could be difficult. Additional information about other relations can help make the inferences possible. We demonstrate the use of the above techniques in special cases such as in integrity constraint checking, and when some additional base relations are also available (see Section 5).

Theorem 4.1 *Consider a non-empty view V, Z-extension V^z, project-join mapping* $\mathsf{PJM}(V^z)$, *and set of tuples T to be inserted into relation R_u. Let I^z be the set of candidate tuples. The insertion operation is conditionally autonomously computable if and only if every tuple in I^z is either definitely inserted into the view, as per Lemma 4.1 or definitely not inserted, as per Lemma 4.3.* □

5 Base Relations Also Help!

In this section we illustrate the fact that it is possible to update a view using some base relations in addition to the materialized view when the view may not be updateable using either only the materialized view or only the updated relation. Extensions of the results of Sections 3 and 4 handle these cases. The details are discussed in [Gup94].

EXAMPLE 5.1 Consider a variant of view `ok_dept` that contains only the names of departments that have at least one employee who makes more than some manager in that department.

`ok_dept`(D) :- `emp`(E, S, D) & `dept`(M, D, MS) & $S > MS$.

We assume that the contents of the view `ok_dept` and relation `emp` are available. Let `ok_dept` contain two tuples (d_1) and (d_2) and `emp` contain the four tuples: $(a, d_1, 10)$, $(b, d_1, 20)$, $(c, d_2, 30)$, $(d, d_3, 45)$. Suppose tuple `emp`$(b, d_1, 20)$ is deleted. Using the contents of `ok_dept` and `emp` it is possible to infer that the view stays unchanged. However, if either `emp` or `ok_dept` were unavailable, the inference could not have been made. □

Intuitively, base relations are used by unifying each available relation with its Z-projection that is created as before. That is, the available base relations are used to instantiate the Z-variables in the Z-projection. In addition, the extra Z-tuple introduced in Definition 4.4 for each available base relation will be instantiated by all those tuples that do not unify with other tuples in the Z-projection. The net effect of this unification process is that some Z-variables in $\mathcal{I}(V)$ are get replaced by constants. Fewer variables in $\mathcal{I}(V)$ result in more restricted implication conditions for the lemmas of Sections 3 and 4. A condition that has fewer variables is more often provable (and less expensive to prove) than a condition that has more variables. Positive and negative information are treated differently for unification. Addendum 2 contains a detailed example that illustrates how to build $\mathcal{I}(V)$. Lemmas 3.1, 3.2, 4.1, and 4.3 hold exactly as before.

5.1 Building Representative Relations

Not all the tuples in a base relation provide new information that can be used to update a view or enforce a constraint. Some tuples provide *less* information than other tuples with respect to a view.

EXAMPLE 5.2 Consider again view `ok_dept` from Example 1.1. Let tuples $t_1 = (e_1, d_1, 65)$ and $t_2 = (e_2, d_1, 50)$ be in relation `emp`. As argued in Example 1.1, tuple t_2 does not contribute anything more than t_1 to view `ok_dept`. In other words, tuple t_1 *subsumes* t_2 with respect to view `ok_dept`. Therefore, tuple t_1 represents itself *and* tuple t_2 in the view maintenance process (and any other tuples subsumed by t_1). Hence, if t_1 is present in `emp`, tuple t_2 need not be accessed to update view `ok_dept`. □

Often, the information contained in a base relation can be summarized by smaller *representative* relations. This notion is useful from the point of view of optimizing constraint checking, view maintenance, and intelligent caching. Representative relations can be computed using ideas described in this paper and in [GW93, GSUW94] by defining the conditions under which one tuple *subsumes* another tuple with respect to a given view definition. Subsumption with respect to a view definition defines a partial order on the tuples in any relation. The tuples not subsumed by any other tuple determine the contents of the view. These tuples constitute the *representative relation*. The representative relation can be cached at multiple sites in order to expedite view maintenance. In addition, even if the entire relation is available, only the representative portion need be accessed for view maintenance or integrity constraint checking.

6 Updating Views using Only the Updated Relation

In this section we consider the case when only the view definition $\mathcal{C}$, the update $\mathcal{U}$, and the updated relation R_u are used to update the view. We first consider the problem of determining when the update *does not* contribute to the view. Subsequently we discuss when the update *definitely* contributes to the view.

6.1 Update Does Not Contribute to View

Example 1.1 illustrates this case. This problem was discussed in the context of integrity constraints in [GW93, GSUW94]. In this paper we extend the results to view maintenance. We give a reduction from the view maintenance problem to the integrity constraint checking problem. Note, how closely the reduction mimics a special case of the reasoning process developed thus far in this paper. The results in [GW93, GSUW94] were developed independently exploiting the fact that a constraint is essentially a 0-ary view.

EXAMPLE 6.1 Consider view `ok_dept` defined in Example 1.1, this time expressed as a Datalog rule.

$$\texttt{ok\_dept}(D, M) \text{ :- } \texttt{emp}(E, D, S) \;\&\; \texttt{dept}(D, M, MS) \;\&\; S > MS.$$

If view `ok_dept` is consistently materialized then the following constraint assertion holds:

$$\mathcal{A}_{ok}: \quad \forall\, E,D,S,M,MS : [(\texttt{emp}(E, D, S) \wedge \texttt{dept}(M, D, MS) \wedge S > MS) \Rightarrow \texttt{ok\_dept}(D, M)]\ .$$

Let view `ok_dept` be consistent before an update to relation `emp`, *i.e.*, assertion $\mathcal{A}_{ok}$ holds. If the update to `emp` can be inferred not to affect the truth value of assertion $\mathcal{A}_{ok}$, then we can conclude that the contents of view `ok_dept` do not need to be updated. The results in [GW93, GSUW94] can be used to check the truth value of assertion $\mathcal{A}_{ok}$ using only the contents of updated relation `emp`. □

In general, consider view definition $\mathcal{C}$ that defines relation head:

$\mathcal{C}$: head :– body.

body is the body of a Datalog rule defining head. For a given database DB, if the view materialization is consistent then the following integrity constraint assertion holds:

$\mathcal{A}$: body $\Rightarrow$ head.

Let some relation R_u in body be updated by update $\mathcal{U}$. The view relation head remains unchanged in the new database state if and only if the assertion $\mathcal{A}$ holds in the new database. [GW93, GSUW94] describe how to check $\mathcal{A}$ only using relation R_u. The papers give algorithms to derive a query $\mathcal{Q}_u$ on relation R_u such that if $\mathcal{Q}_u$ is satisfiable, then update $\mathcal{U}$ does not violate the constraint. If the query is not satisfiable, then the constraint cannot be checked using only R_u. The results can be used to update views using the transformation from $\mathcal{C}$ to $\mathcal{A}$ described above.

EXAMPLE 6.2 Continuing with Example 6.1, assume that only relation emp is available for view maintenance. Let the update $\mathcal{U}$ consist of inserting tuple (e, d, s) into emp. The update does not contribute to the view ok_dept if and only if the update does not violate assertion $\mathcal{A}_{ok}$. The update does not violate the constraint if the following query on emp returns a non-empty answer:

select * **from** emp
where emp.$D = d$ **and** emp.$S \geq s$.

Note, tuple emp$(e_1, d_1, 65)$ satisfied the above query when tuple $(e_2, d_1, 50)$ was inserted into relation emp (validating the intuition of Example 1.1). □

Queries of the above form can be derived algorithmically for a superset of the following classes of views when the update is either the insertion, deletion, or modification of (sets of) tuples in one of the base relations used in the view:

1. Views defined by select-project-join expressions that use the arithmetic comparisons $=, <, \leq, >, \geq$ and restricted use of $+, -$.
2. SQL views that use arithmetic comparisons $=, <, \leq, >, \geq$ and restricted use of $+, -$ with an arbitrary number of positive and negative (**not exists**, **not in**) subqueries where the positive subqueries could be nested queries of arbitrary depths.

The technical details and SQL-grammar for the above classes are discussed at length in [Gup94].

6.2 Update *Definitely* Contributes to View

We prove that it is not possible to infer that an update *definitely* contributes to a view if only a subset of participating base relations is available and the materialized view is not available.

Theorem 6.1 *Let a view V defined by $\mathcal{C}$ use relations $\{L_1, \ldots, L_m, R_1, \ldots, R_n\}, m \geq 1, n \geq 1$. Let relation L_u be updated by update $\mathcal{U}$. It is not possible to infer that $\mathcal{U}$ definitely inserts a tuple into view V by any partial information based technique that uses only $\mathcal{C}$, $\{L_1, \ldots, L_m\}$, and $\mathcal{U}$.* □

Theorem 6.1 is proved by contradiction by building instances of relations $\{R_1, \ldots, R_n\}$ such that for all relations $\{L_1, \ldots, L_m\}$, the resulting view V is empty (proof in [Gup94]).

7 Evaluability and Efficiency Issues

This section discusses the complexity of evaluating the conditions of Lemmas 3.1, 3.2, 4.1, and 4.3. We also discuss optimizations for the evaluation process.

Before we discuss the complexity issues, we motivate how the results might be used. As will be clear from the following discussion that using the techniques outlined in this paper as *the* algorithm to maintain views using partial information is not feasible practically. However, the results provide the necessary theory to identify subclasses of SPJ views that can be efficiently maintained. The optimizations described below have the same purpose. We have used the results of this paper to identify syntactically classes of views that can be always maintained without using any base relations [GMS95]. Such views are especially useful in data warehousing and data visualization applications because then the view can be maintained efficiently in an instance independent fashion.

7.1 Evaluation

Recall that the positive information in $\mathcal{I}(V)$ consists of existentially quantified statements of the form:

$$S : \exists \bar{X} : [r_1(\bar{X}_1) \wedge \ldots \wedge r_n(\bar{X}_n) \wedge \mathcal{C}(\bar{X})].$$

Where r_is are relations used in the view definition, $\bar{X}_i$s are subsets of the variables in $\bar{X}$ and $\mathcal{C}$ is a conjunction of selection conditions. The negative information in $\mathcal{I}(V)$ can be expressed as a conjunction of statements of the form $\neg S$. $\mathcal{I}(V)$ is therefore of the form:

$$S_1 \wedge S_2 \wedge \ldots \wedge S_m \wedge \neg S_{m+1} \wedge \ldots \wedge \neg S_n.$$

The condition checked by Lemma 3.1 is of the form:

$$S_1 \wedge S_2 \wedge \ldots \wedge S_m \wedge \neg S_{m+1} \wedge \ldots \wedge \neg S_n \Rightarrow S_k.$$

The above condition can be rewritten as:

$$S_1 \wedge S_2 \wedge \ldots \wedge S_m \Rightarrow S_{m+1} \vee \ldots \vee S_n \vee S_k.$$

Note, each S_i in the LHS is existentially closed and the LHS is satisfiable. Therefore, the above statement is equivalent to the disjunction of m statements:

$$\bigvee_{i \in \{1,\ldots,m\}} (S_i \Rightarrow S_{m+1} \vee \ldots \vee S_n \vee S_k).$$

Similarly, each S_i in the RHS is existentially closed. Thus, the above statement is equivalent to:

$$\bigvee_{i \in \{1,\ldots,m\}} \bigvee_{j \in \{k,m+1,\ldots,n\}} (S_i \Rightarrow S_j).$$

Lemmas 3.2, 4.1, and 4.3 also result in similar conditions. We refer to the implications of the form $S_i \Rightarrow S_j$ as $\mathcal{B}$. Determining the validity of $\mathcal{B}$ is the same as the problem of conjunctive query containment with built-in predicates [Ull89, ZO93, GSUW94]. Using results from [ZO93, Gup94], $\mathcal{B}$ can be reduced to an implication of the form:

$$\mathcal{A} : (C_1 \Rightarrow C_2 \vee \ldots \vee C_q).$$

Where each C_j uses the selection conditions in the original view definition. The sentence $\mathcal{A}$ can be evaluated by the theory for real numbers with inequalities, if the selection conditions use only inequalities and reals. Depending on the class of selection condition used by the view, the appropriate theory is used for inference.

Consider the results of [SKN89] for the case when the selection conditions are arithmetic comparisons over the reals. The problem can be represented as determining the validity of the implication $\sigma_Q \Rightarrow \sigma_T$, where σ_Q and σ_T are conjunctions and disjunctions of arithmetic order constraints. In the case when σ_Q uses the inequality sign or when σ_T uses disjunction, the problem of determining if $\sigma_Q \Rightarrow \sigma_T$ is NP-complete (by reducing 3-SAT). In our case, $\mathcal{A}$ could have disjuncts on the RHS making the tests NP-complete. However, the size of $\mathcal{A}$ is bounded by the total number of attributes of the relations that participate in the view.

In addition, the number of sentences of form $\mathcal{A}$ are proportional to the number of tuples in the materialized view. Therefore, for each of the lemmas, we need to check a disjunction of $|V|^2$ conditions of the form $\mathcal{A}$ where $|V|$ is the size of the view. Checking each disjunct is exponential in "a" the total number of attributes involved in selections conditions in the view definition. However, as soon as a true disjunct is found, the process halts successfully. The worst case complexity of checking the conditions of the lemmas is $O(|V|^2 * 2^a)$.

7.2 Efficient Evaluation of $\mathcal{B}$

Representative relations can also be used to efficiently check the conditions of Lemmas 3.1, 3.2, 4.3, and 4.1. $\mathcal{I}(V)$ need be generated only for the representative portion of the view, making the worst case complexity of the test proportional to the size of $\mathsf{representative}(V)$. The complexity of computing the representative relation depends on the class of selection conditions used in the view definition. For instance, the representative relation for a relation R that is used in a view definition that uses the arithmetic comparison operators over the reals, can be computed in time $O(|R|^2 * 2^a)$. Note, a is typically very small and this cost is incurred at "compile" time, *i.e.*, whenever the system has extra computing cycles the representative can be computed and stored. The savings are visible at run-time when the view maintenance actually happens and the representative relation is used. The representative may be much smaller than the original relation. For instance, in Example 1.1 the representative relation for emp contains only the highest paid employee in each department. Thus, the representative relation has as many tuples as the number of departments. Additionally, it is possible to incrementally maintain a representative relation in time proportional to the size of the representative relation.

Subsumption allows the inferences made for one tuple to be used for another tuple. For instance, let tuples t_1 and t_2 be inserted into a relation R that participates in view definition $\mathcal{C}$. Let t_1 subsume t_2 with respect to $\mathcal{C}$. Then we need not determine the effect of tuple t_2 once the effect of t_1 has been computed. Subsumption can also be used to cascade deletions from a subsuming tuple to all subsumed tuples, or propagate insertions from a subsumed tuple to subsuming tuples without recomputing the view definition for each tuple.

The efficiency of evaluating $\mathcal{B}$ is related to efficiency of evaluating the arithmetic implications $\mathcal{A}$. One such method is discussed in [ZO93]. It is also possible to replace $\mathcal{A}$ by a sufficient condition that implies $\mathcal{A}$ and is linear in the number of attributes of the view [GSUW94]. Finally, restricted classes of arithmetic comparisons can be evaluated using specialized techniques from computational geometry and operations research. Unification of all these techniques under a common framework is a subject of further research.

7.3 Discussion

Some classes of views are more efficiently undateable given partial information. In addition, some types of updates are easier to handle. For instance, when all the attributes involved in the selection and join conditions are in the view, the effect of a deletion can be determined directly. All tuples in the view that satisfy the deletion condition are eliminated. However, insertions may not be as straightforward to handle.

In general, it is easier to autonomously maintain views that retain a large number of the attributes involved in the selection and join conditions, than views that retain few of these relevant attributes. The number of "missing" attributes determines the amount of unknown information (and number of variables) in $\mathcal{I}(V)$ and therefore determine the complexity of autonomously updating a view. Another dimension is the relationship between the missing attributes. It is easier to handle cases where the missing attributes are compared to non-missing attributes. For instance, consider a view that uses conditions $A > B$ & $C > D$ and only two of the four attributes are part of the view. More updates can be handled autonomously when one attribute from each of the pairs A, B and C, D is missing, rather than if either both A, B or both C, D are missing.

8 Conclusions

This paper gives a formal basis for partial-information-based view-maintenance techniques enabling the inference of knowledge about the state of the underlying base relations using *local* information that includes the view definition, the update, the current view materialization, and varying amounts of base relation replicas, without remote accesses. We describe how such knowledge can be used to incrementally maintain views. In addition to assisting incremental view maintenance, the results of this paper can be used to *design* views that can be maintained efficiently using partial information, to *define* data partition and replication policies in a distributed system such that the partial information available to sites is useful for maintaining critical views efficiently, and to efficiently enforce integrity constraints.

We describe the complexity of the methods and suggest ways of optimizing the methods. Our results formally explain previous specialized view maintenance techniques (e.g. [CW91] modulo their use of functional dependencies).

Our techniques are especially useful when the status of some particular view tuples need to be determined quickly. The method works tuple by tuple, *i.e.*, determines the status of each individual tuple in the existing materialized view after the change. Whether the method can be used to determine if the view stays unchanged (on an insertion or deletion), depends on the view itself and the update. Often, the underlying base relations may not even all be available, for instance, in a distributed system that has suffered from partial network failure, or in mobile computing environments where the entire database is not always accessible. In such cases, view materialization techniques based on partial information, like the ones presented in this paper, are valuable.

Scenarios where only partial information is available are increasing. Thus it is beneficial to use the results of this paper to identify and design classes of views that can be efficiently maintained using partial information. We are currently researching subclasses of the class of SPJ views, that can be efficiently maintained using partial information [GMR95, GMS95]. Often the best way to facilitate maintenance is to retain extra columns in the view being computed. We are also investigating strategies for intelligently partitioning data in a distributed framework in order to facilitate efficient view maintenance using partial information (refer Section 7.3).

Acknowledgments

We thank Prof. Ullman for valuable feedback on the ideas described in this paper. We also thank the referees for their extremely useful feedback on corrections and improvements, to both technical and presentation aspects.

References

[BLT86] J. A. Blakeley, P. Larson, F. W. Tompa. Efficiently Updating Materialized Views. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 61–71, 1986.

[BBC80] P. A. Bernstein, B. T. Blaustein, and E. M. Clarke. Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data. In *Proceedings of the Sixth conference on Very Large Data Bases*, pages 126–136, 1980.

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

[BC79] P. O. Buneman, E. K. Clemons. Efficiently Monitoring Relational Databases. *ACM Transactions on Database Systems*, Vol. 4, No. 3, pages 368–382 (September 1979).

[CW91] S. Ceri and J. Widom. Deriving Production Rules for Incremental View Maintenance. In *Proceedings of the 17th VLDB Conference, Barcelona, Spain*, 1991.

[CW94] S. Ceri and J. Widom. Deriving Incremental Production Rules for Deductive Data. In *Information Systems*, 19(6), pages 467–490, 1994.

[DT92] G. Dong and R. Topor. *Incremental Evaluation of Datalog Queries.* In *Proceedings of the International Conference on Database Theory (ICDT)*, 1992.

[Elk90] C. Elkan. Independence of Logic Database Queries and Updates. In *Proceedings of the Ninth Symposium on Principles of Database Systems (PODS)*, pages 154–160, Nashville, TN, 1990.

[Gup94] Ashish Gupta. *Partial Information Based Integrity Constraint Checking.* PhD thesis, Stanford University, Department of Computer Science, STAN-CS-TR-95-1534, 1994.

[GMS95] A. Gupta, H. V. Jagadish, and I. S. Mumick. Data Warehousing Using Self-Maintainable Views. Unpublished document.

[GMR95] A. Gupta, I. S. Mumick, and K. Ross. Adapting Materialized Views after Redefinitions In *SIGMOD* 1995.

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

[GSUW94] A. Gupta, S. Sagiv, J. D. Ullman, and J. Widom. Checking Integrity Constraints Using Partial Data. In *Proceedings of the Thirteenth Symposium on Principles of Database Systems (PODS)*, pages 45-55, 1994.

[GW93] A. Gupta and J. Widom. Local Checking of Global Integrity Constraints . In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 49–59, 1993.

[HD92] J. V. Harrison and S. Dietrich. Maintenance of Materialized Views in a Deductive Database: An Update Propagation Approach. In *Workshop on Deductive Databases, JICSLP*, 56–65, 1992.

[KM90] Alfons Kemper and Guido Moerkotte. Access support in object bases. In *Proceedings of ACM SIGMOD 1990 International Conference on Management of Data*, pages 364–374, May 1990.

[Kuc91] V. Kuchenhoff. On the Efficient Computation of the Difference Between Consecutive Database States. In *Second International Conference, Deductive and Object-Oriented Databases*, 1991.

[LY85] P. A. Larson and H. Z. Yang. Computing Queries from Derived Relations. In *Proceedings of the Eleventh International Conference on Very Large Databases*, pages 259–269, 1985.

[LS93] A.Y. Levy and Y. Sagiv. Queries Independent of Updates. In *Proceedings of the Nineteenth International Conference on Very Large Data Bases*, pages 171–181, Dublin, Ireland, 1993.

[NY83] J. M.Nicolas and Yazdanian. An Outline of BDGEN: A Deductive DBMS. In *Information Processing*, pages 705–717, 1983.

[QW91] X. Qian and G. Wiederhold. *Incremental Recomputation of Active Relational Expressions.* In *TKDE*, 1991.

[RB] Red Brick Systems. *RISQL Self-Study Guide.* Los Gatos, CA: Red Brick Systems, 1993.

[Rou91] N. Roussopoulos. The Incremental Access Method of View Cache: Concept, Algorithms, and Cost Analysis. In *ACM-TODS*, 16(3):535-563, 1991.

[SKN89] X. Sun, N. Kamel, and L. M. Ni. Solving Implication Problems in Database Applications. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp 185–192, 1989.

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

[TSI94] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for physical data independence. In *Proceedings of the 20^{th} International Conference on Very Large Databases*, pages 367–378, 1994.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, Volumes 1 and 2. Computer Science Press, New York, 1989.

[UO92] T. Urpi and A. Olive. A Method for Change Computation in Deductive Databases. In *Proceedings of the Eighteenth VLDB*, pages 225–237, Vancouver, British Columbia, 1992.

[Va87] P. Valduriez. Join Indices. In *ACM Trans. Database Systems*, 12(2):218-246, 1987.

[VGT91] A. V. Gelder and R. W. Topor. Safety and Translation of Relational Calculus Queries. *ACM Transactions on Database Systems*, 16(3):235–278, (June 19911991).

PJM(good_dept^z)						
D	S	M	D	MS		
d_1	50	m_1	d_1	z_1	t_1^z	$50 > z_1$
d_1	50	m_2	d_1	z_3	n_1	$\neg(50 > z_3)$
d_1	100	m_1	d_1	z_1	t_2^z	$100 > z_1$
d_1	100	m_2	d_1	z_3	t_3^z	$100 > z_3$
d_1	50	z'_3	z'_4	z'_5	n_2	$\neg(z'_4 = d_1 \wedge 50 > z'_5)$
d_1	100	z'_3	z'_4	z'_5	n_3	$\neg(z'_4 = d_1 \wedge 100 > z'_5)$
z'_1	z'_2	z'_3	z'_4	z'_5	n_4	$\neg(z'_1 = z'_4 \wedge z'_2 > z'_5)$
z'_1	z'_2	m_1	d_1	z_1	n_5	$\neg(z'_1 = d_1 \wedge z'_2 > z_1)$
z'_1	z'_2	m_2	d_1	z_1	n_6	$\neg(z'_1 = d_1 \wedge z'_2 > z_1)$

Figure 1: Project-join-mapping for Example 4.3

[ZO93] X. Zhang and M. Z. Ozsogoglu. On Efficient Reasoning with Implication Constraints. In *Conference on Deductive and Object Oriented Databases*, pages 236–252, 1993.

[Z+94] Yue Zhuge, Hector Garcia-Molina, Joachim Hammer, and Jennifer Widom. View Maintenance in a Warehousing Environment To appear in SIGMOD 1995.

Recommended by Yannis Ioannidis, Area Editor

Addendum 1

The view maintenance problem discussed in this paper was motivated by [TB88]. In this section we present an example where the algorithm in [TB88] does not apply. We assume the reader is familiar with that paper.

EXAMPLE 8.1 Consider Example 4.1 from [TB88]. Replace relation R_1 in [TB88] by the relation R'_1, and keep relation R_2 the same. Using the view definition given below, and the relations R'_1, and R_2, we get view V:

$v(B_1, A_2, B_2, C_2)$:- $r_1(A_1, B_1)$ & $r_2(A_2, B_2, C_2)$ & $A_1 > A_2$ & $A_1 < C_2$.

R'_1		R_2			V				
A_1	B_1	A_2	B_2	C_2	B_1	A_2	B_2	C_2	
16	1	17	18	23	2	17	18	23	t_1
18	2	19	14	25	2	19	14	25	t_2
23	2	24	18	33	3	19	14	25	t_3
24	3								

The deletion condition is the same as in [TB88]: $\mathsf{DEL}(R'_1, (A_1 < 20) \vee (A_1 > 22))$, *i.e.*, delete all those R'_1 tuples whose A_1 attribute has value either < 20 or > 22. Every tuple in relation R'_1 (as given in this example) satisfies this deletion condition *i.e.*, all tuples in R'_1 get deleted resulting in a modified view that is empty.

However, if the reasoning of Examples 4.2, 4.3 and 4.4 from [TB88] is applied we do not get this answer (some tuples remain in the view). Given that the view is the same as in the paper, Examples 4.2, 4.3 and 4.4 remain the same as described in [TB88] and result in the conclusion that tuples t_1 and t_2 should not be deleted from the view. This conclusion is incorrect. □

In [TB88] the tuples of the PJM of type ($1a$) (refer Section 2) were treated as being universally quantified, and were considered multiple times, leading to assumptions that were stronger than should be made. For instance, if tuples t_i^z in the PJM contributed a tuple to the view, [TB88] assumed that the condition $\mathcal{C}(t_i^z)$ was universally quantified leading to incorrect inferences.

Addendum 2

How to use Available Base Relations

EXAMPLE 8.2 Consider the following view definition:

`ok_dept`(D) :- `emp`(E, S, D) & `dept`(M, D, MS) & $S > MS$.

Consider a materialization of the view `ok_dept`, its Z-extended view `ok_dept`^{z}, and a particular instance of relation `emp` that is used to compute the view. (Henceforth we ignore the first attributes of relations `emp` and `dept` because those attributes are not used in any computation).

ok_dept	ok_dept^z			emp	
D	S	D	MS	S	D
d_1	z_1	d_1	z_2	10	d_1
				20	d_1
d_2	z_3	d_2	z_4	30	d_2
				45	d_3

The Z-projections of relations emp and dept, and PJM(ok_dept^z) obtained using the materialized view are:

emp^z		dept^z		PJM(ok_dept^z)					
S	D	D	MS	S	D	D	MS		
z_1	d_1	d_1	z_2	z_1	d_1	d_1	z_2	t_1^z	$z_1 > z_2$
z_3	d_2	d_2	z_4	z_3	d_2	d_2	z_4	t_2^z	$z_3 > z_4$
z'_1	z'_2	z'_3	z'_4	z'_1	z'_2	z'_3	z'_4	n_1	$\neg(z'_2 = z'_3, z'_1 > z'_4)$
				z'_1	z'_2	d_1	z_2	n_2	$\neg(z'_2 = d_1, z'_1 > z_2)$
				z'_1	z'_2	d_2	z_4	n_3	$\neg(z'_2 = d_2, z'_1 > z_4)$
				z_1	d_1	z'_3	z'_4	n_4	$\neg(d_1 = z'_3, z_1 > z'_4)$
				z_1	d_2	z'_3	z'_4	n_5	$\neg(d_2 = z'_3, z_1 > z'_4)$

The additional condition associated with tuple emp$(_, z'_2, z'_3)$ is:

$C'(\texttt{emp}(\_, z'_2, z'_3))$: $\neg(z'_2 = d_1 \vee z'_2 = d_2)$ *i.e.* $(z'_2 \neq d_1 \wedge z'_2 \neq d_2)$.

The additional condition associated with tuple dept$(_, z'_2, z'_3)$ is:

$C'(\texttt{dept}(\_, z'_2, z'_3))$: $\neg(z'_2 = d_1 \vee z'_2 = d_2)$ *i.e.* $(z'_2 \neq d_1 \wedge z'_2 \neq d_2)$.

$\mathcal{I}$(ok_dept) is:

$$
\begin{array}{ll}
(1) & \exists W, Z : [\texttt{emp}(W, d_1), \texttt{dept}(d_1, Z), W > Z] \wedge \\
(2) & \exists W, Z : [\texttt{emp}(W, d_2), \texttt{dept}(d_2, Z), W > Z] \wedge \\
(3) & \forall W, X, Y, Z : [(\texttt{emp}(W, X), \texttt{dept}(Y, Z), X \neq d_1, X \neq d_2, Y \neq d_1, Y \neq d_2) \Rightarrow \\
 & \hfill \neg(W > Z, X = Y)] \wedge \\
(4) & \forall W, X, Z : [(\texttt{emp}(W, X), \texttt{dept}(d_1, Z), \underline{X \neq d_1}, X \neq d_2) \Rightarrow (\underline{X \neq d_1} \vee W \leq Z)] \wedge \\
(5) & \forall W, X, Z : [(\texttt{emp}(W, X), \texttt{dept}(d_2, Z), X \neq d_1, \underline{X \neq d_2}) \Rightarrow (\underline{X \neq d_2} \vee W \leq Z)] \wedge \\
(6) & \forall W, Y, Z : [(\texttt{emp}(W, d_1), \texttt{dept}(Y, Z), \underline{Y \neq d_1}, Y \neq d_2) \Rightarrow (\underline{Y \neq d_1} \vee W \leq Z)] \wedge \\
(7) & \forall W, Y, Z : [(\texttt{emp}(W, d_2), \texttt{dept}(Y, Z), Y \neq d_1, \underline{Y \neq d_2}) \Rightarrow (\underline{Y \neq d_2} \vee W \leq Z)].
\end{array}
$$

Note that lines 4, 5, 6, and 7 are tautologies and can be eliminated from the conjunction. Therefore, tuples n_2, n_3, n_4, and n_5 do not contribute anything to $\mathcal{I}$(ok_dept). The resulting $\mathcal{I}$(ok_dept) is:

$$
\begin{array}{ll}
(1) & \exists W, Z : [\texttt{emp}(W, d_1), \texttt{dept}(d_1, Z), W > Z] \wedge \\
(2) & \exists W, Z : [\texttt{emp}(W, d_2), \texttt{dept}(d_2, Z), W > Z] \wedge \\
(3) & \forall W, X, Y, Z : [(\texttt{emp}(W, X), \texttt{dept}(Y, Z), X \neq d_1, X \neq d_2, Y \neq d_1, Y \neq d_2) \Rightarrow \\
 & \hfill \neg(W > Z, X = Y)].
\end{array}
$$

Consider the deletion operation $\mathsf{DEL}(\texttt{dept}, MS > 25)$. If we did not have access to the relation `emp`, but only to the materialized view, it would not be possible to conclude anything about the effect of this delete operation on the view. However, given the particular `emp` relation we can infer that tuple t_1^z can be derived only by either tuple $\texttt{emp}(10, d_1)$ or tuple $\texttt{emp}(20, d_1)$ because no other `emp` tuple has the department attribute value d_1. Thus, there has to be a tuple (d_1, MS) in relation `dept` where MS is less than either 10 or 20 and therefore ≤ 25. In either case, this tuple will not be deleted by $\mathsf{DEL}(\texttt{dept}, MS > 25)$ and thus will help derive tuple $\texttt{ok\_dept}(d_1)$ even after the deletion. Hence the tuple $\texttt{ok\_dept}(d_1)$ definitely stays in the view.

Nothing can be deduced about tuple $\texttt{ok\_dept}(d_2)$. □

How do we automatically make the above inference?

We describe the process assuming that only one relation is available and later generalize the process to use an arbitrary number of available relations. Assume that relation L is the available relation and $\mathcal{I}(V)$ was obtained for some view definition using view V. We redefine $\mathcal{I}(V)$ to reflect the availability of relation L. For every tuple $t \in L$ do the following:

- For every existentially quantified component E, if t unifies with the occurrence of relation l in that component, instantiate l by t to obtain $E(t)$. Replace E by $E \vee E(t)$.
- For every universally quantified component F, if t unifies with the occurrence of relation l in that component, instantiate l by t to obtain $F(t)$. Replace F by $F \wedge F(t)$.

Finally eliminate all those existentially and universally quantified terms of $\mathcal{I}(V)$ that have uninstantiated occurences of relation l. The two steps carried out above, substitute the available relation into the PJM.

EXAMPLE 8.3 Recall that using the view definition, view `ok_dept`, and contents of relation `emp` from Example 8.2, $\mathcal{I}(\texttt{ok\_dept})$ is:

$$\begin{array}{ll}
(1) & \exists W, Z : [\texttt{emp}(W, d_1), \texttt{dept}(d_1, Z), W > Z] \wedge \\
(2) & \exists W, Z : [\texttt{emp}(W, d_2), \texttt{dept}(d_2, Z), W > Z] \wedge \\
(3) & \forall W, X, Y, Z : [(\texttt{emp}(W, X), \texttt{dept}(Y, Z), X \neq d_1, X \neq d_2, Y \neq d_1, Y \neq d_2) \Rightarrow \\
& \hfill \neg(W > Z, X = Y)].
\end{array}$$

Now we commence the process of unifying tuples in the available relation `emp` with occurrences of `emp` as terms in $\mathcal{I}(\texttt{ok\_dept})$. Consider $\texttt{emp}(10, d_1)$, that unifies with $\texttt{emp}(W, d_1)$ in line 1 and with $\texttt{emp}(W, X)$ in line 3. Following the steps outlined above, we obtain:

$$\begin{array}{ll}
(1) & \exists W, Z : [(\texttt{emp}(10, d_1), \texttt{dept}(d_1, Z), 10 > Z) \vee (\texttt{emp}(W, d_1), \texttt{dept}(d_1, Z), W > Z)] \wedge \\
(2) & \exists W, Z : [\texttt{emp}(W, d_2), \texttt{dept}(d_2, Z), W > Z] \wedge \\
(3a) & \forall Y, Z : [(\texttt{emp}(10, d_1), \texttt{dept}(Y, Z), \underline{d_1 \neq d_1}, d_1 \neq d_2, Y \neq d_1, Y \neq d_2) \Rightarrow \\
& \hfill \neg(10 > Z, Y = d_1)] \wedge \\
(3) & \forall W, X, Y, Z : [(\texttt{emp}(W, X), \texttt{dept}(Y, Z), X \neq d_1, X \neq d_2, Y \neq d_1, Y \neq d_2) \Rightarrow \\
& \hfill \neg(W > Z, X = Y)].
\end{array}$$

Note, line $3a$ is always true because $d_1 \neq d_1$ is always false. Therefore, line $3a$ can be eliminated. Substituting $\texttt{emp}(20, d_1)$ and using similar reasoning results in:

$$
\begin{array}{ll}
(1) & \exists W, Z : [(\texttt{emp}(10, d_1), \texttt{dept}(d_1, Z), 10 > Z) \vee (\texttt{emp}(20, d_1), \texttt{dept}(d_1, Z), 20 > Z) \vee \\
 & \qquad (\texttt{emp}(W, d_1), \texttt{dept}(d_1, Z), W > Z)] \wedge \\
(2) & \exists W, Z : [\texttt{emp}(W, d_2), \texttt{dept}(d_2, Z), W > Z] \wedge \\
(3) & \forall W, X, Y, Z : [(\texttt{emp}(W, X), \texttt{dept}(Y, Z), X \neq d_1, X \neq d_2, Y \neq d_1, Y \neq d_2) \Rightarrow \\
 & \qquad \neg(W > Z, X = Y)].
\end{array}
$$

Consider $\texttt{emp}(30, d_2)$ that unifies with $\texttt{emp}(W, d_2)$ line 2 and with $\texttt{emp}(W, X)$ in line 3. The resulting expression, after eliminating the term produced by unifying with line 3, is:

$$
\begin{array}{ll}
(1) & \exists W, Z : [(\texttt{emp}(10, d_1), \texttt{dept}(d_1, Z), 10 > Z) \vee (\texttt{emp}(20, d_1), \texttt{dept}(d_1, Z), 20 > Z) \vee \\
 & \qquad (\texttt{emp}(W, d_1), \texttt{dept}(d_1, Z), W > Z)] \wedge \\
(2) & \exists W, Z : [(\texttt{emp}(30, d_2), \texttt{dept}(d_2, Z), 30 > Z) \vee (\texttt{emp}(W, d_2), \texttt{dept}(d_2, Z), W > Z)] \wedge \\
(3) & \forall W, X, Y, Z : [(\texttt{emp}(W, X), \texttt{dept}(Y, Z), X \neq d_1, X \neq d_2, Y \neq d_1, Y \neq d_2) \Rightarrow \\
 & \qquad \neg(W > Z, X = Y)].
\end{array}
$$

$\texttt{emp}(45, d_3)$ does not unify with either line 1 or 2. However, line 3 introduces an additional line:

$$
\begin{array}{ll}
(1) & \exists W, Z : [(\texttt{emp}(10, d_1), \texttt{dept}(d_1, Z), 10 > Z) \vee (\texttt{emp}(20, d_1), \texttt{dept}(d_1, Z), 20 > Z) \vee \\
 & \qquad (\texttt{emp}(W, d_1), \texttt{dept}(d_1, Z), W > Z)] \wedge \\
(2) & \exists W, Z : [(\texttt{emp}(30, d_2), \texttt{dept}(d_2, Z), 30 > Z) \vee (\texttt{emp}(W, d_2), \texttt{dept}(d_2, Z), W > Z)] \wedge \\
(3) & \forall W, X, Y, Z : [(\texttt{emp}(W, X), \texttt{dept}(Y, Z), X \neq d_1, X \neq d_2, Y \neq d_1, Y \neq d_2) \Rightarrow \\
 & \qquad \neg(W > Z, X = Y)] \wedge \\
(4) & \forall Y, Z : [(\texttt{emp}(45, d_3), \texttt{dept}(Y, Z), d_3 \neq d_1, d_3 \neq d_2, Y \neq d_1, Y \neq d_2) \Rightarrow \\
 & \qquad \neg(45 > Z, d_3 = Y)] .
\end{array}
$$

Once all the tuples of relation E are considered, the existential clauses that have uninstantiated e terms can be eliminated; namely the last disjuncts in lines 1 and 2. In addition, the tuples of E are known to be true and can be pulled out of all the conjunctions they occur in. Terms like $d_3 \neq d_1$ that are true, can also be eliminated. These two simplifications yield:

$$
\begin{array}{ll}
(1) & \texttt{emp}(10, d_1), \texttt{emp}(20, d_1), \texttt{emp}(30, d_2), \texttt{emp}(45, d_3) \wedge \\
(2) & \exists Z : [(\texttt{dept}(d_1, Z), 10 > Z) \vee (\texttt{dept}(d_1, Z), 20 > Z)] \wedge \quad \%\ \textit{by removing}\ \texttt{emp}\ \textit{constants} \\
(3) & \exists Z : [\texttt{dept}(d_2, Z), 30 > Z] \wedge \qquad \%\ \textit{and disjuncts involving uninstantiated}\ \texttt{emp}\ \textit{terms} \\
(4) & \forall W, X, Y, Z : [(\textit{emp}(W, X), \textit{mydept}(Y, Z), X \neq d_1, X \neq d_2, Y \neq d_1, Y \neq d_2) \Rightarrow \\
 & \qquad \neg(W > Z, X = Y)] \wedge \\
(5) & \forall Y, Z : [(\texttt{dept}(Y, Z), Y \neq d_1, Y \neq d_2) \Rightarrow \neg(45 > Z, d_3 = Y)] .
\end{array}
$$

Further, '$\texttt{dept}(d_1, Z), 10 > Z$' can be eliminated from line 2 because $Z < 10 \Rightarrow Z < 20$. The value of $\mathcal{I}(V)$ thus obtained, can be used just as before. □

We have described how to build $\mathcal{I}(V)$ given the instantiation of one of the base relations. In case more than one base relation is available, the same process can be used effectively.