WPI-CS-TR-98-24 November 1998

View Maintenance after View Synchronization

by

Anisoara Nica
Elke A. Rundensteiner

Computer Science Technical Report Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

View Maintenance after View Synchronization *

Anisoara Nica† and Elke A. Rundensteiner‡

(†) Department of EECS
University of Michigan, Ann Arbor
Ann Arbor, MI 48109-2122
anica@eecs.umich.edu

(‡)Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609-2280
rundenst@cs.wpi.edu

Abstract

Adaption of data warehouses is a critical task in dynamic environments such as the WWW where the underlying information sources (IS) change not only their contents but also their schemas. While current view technology assumes that the ISs do not change their schema, our Evolvable View Environment (*EVE*) project addresses this problem by evolving the view definitions affected by IS schema changes, which we coin *view synchronization*. In *EVE*, the view synchronizer rewrites the view definitions by replacing view components with suitable components from other ISs. However, after such a view redefinition process, the view extents, if materialized, must also be brought up-to-date. In this paper, we propose strategies to address this incremental adaptation of the view extent after view synchronization. One key idea of our approach is to regard the complex changes done to a view definition after synchronization as atomic units and to handle them in one optimized batch process, instead of treating the changes as a sequence of several primitive redefinitions. Second, we exploit knowledge of how the view definition was synchronized, especially the containment information between the old and new views, to achieve efficient view adaption. As we will demonstrate both these techniques lead to increased efficiency in view adaption. Third, we deal with the added difficulty that the IS schema evolution that triggered the view redefinition also may have removed the base information required for adaption from the IS. We illustrate that our techniques would successfully adapt views under the unavailability of base relations while currently known maintenance strategies from the literature would fail. Our experimental results conducted for comparing our solution approach to that of recomputation and redefinition techniques demonstrate that we achieve a performance gain of approximately 400% when the difference between the old and new extents is fairly small while being comparable with these alternate techniques in all other scenarios.

Keywords: Materialized view maintenance/adaptation, Warehouse data maintenance, Evolving information sources, View synchronization and preservation, Information descriptions.

1 Introduction

One important problem faced by applications using views is that current view technology only supports *static* view definitions meaning that views are assumed to be specified on top of information sources that do not change their schema. Once the underlying ISs change their schema (as is common for example for autonomous sources connected on the WWW), the views derived from them become undefined. We call this the *view synchronization* problem. In our previous work [RLN97], we introduced a taxonomy of view adaptation problems that shows

*This work was supported in part by several grants from NSF, namely, the NSF NYI grant #IRI 94–57609, the NSF CISE Instrumentation grant #IRIS 97–29878, and the NSF grant #IIS 97–32897. Dr. Rundensteiner would like to thank our industrial sponsors, in particular, IBM for the IBM partnership award and for the IBM corporate fellowship for one of her graduate students.

how the view synchronization problem is different from maintenance after data updates [GM95] or after explicit user-specified redefinition of the view [GMR95, MD96].

Our Evolvable View Environment *EVE* approach [RLN97, LNR97a, NLR98] focuses on the problem of data warehouse maintenance when base relations exhibit schema-level changes. In our previous work [LNR97b, NLR98, NR98a] we proposed different strategies for rewriting view definitions triggered by IS schema changes so that they are well-defined over the modified information space. Our proposed *view synchronization* algorithms also attempt to find view redefinitions that meet view preservation constraints specified by the view user, such as constraints imposed on the view extent or the view interface type. Unlike the strategies proposed for query rewriting using views in the database literature ([LRU96, SDJL96]), our proposed techniques address three new issues: (1) finding view rewritings that are not necessarily *equivalent* to the original view definition as long as they meet user requirements[1], (2) using semantic *containment information* for replacing the deleted information; and (3) preserving only indispensable attributes from the SELECT clause if preserving all is not possible.

In this paper, we now address a new problem that arises in the context of view synchronization under capability changes. Namely, if the views are materialized at the data warehouse site, then after the view synchronization process modifies the view definition the view extents must be brought up to date as well. This problem is similar to the explicit view redefinition problem that has recently been studied in the literature [MD96, GMR95]. That is, the view definition changes triggered by the IS changes could be mapped into a sequence of primitive changes assumed to be explicitly requested by a user for the view definition [MD96, GMR95]. Once this mapping is established, one can apply the maintenance-after-redefinition strategies proposed in [MD96, GMR95] for each such simple[2] change. However, as we will show here, treating synchronization as a sequence of primitive changes is very inefficient for keeping the view extent up-to-date. When the to-be-deleted attribute is still available during maintenance, we can treat our maintenance after view synchronization similar to the explicit view redefinition problem: the view definition changes done by the view synchronization process could be seen as a sequence of simple primitive changes of different types explicitly requested on the view definition. However, as we will demonstrate, this strategy is very inefficient as in the case of view synchronization the changes done to a view definition are fairly complex and intermediate results for each primitive change could potentially be huge. Hence, while the use of existing techniques is feasible for these cases, it is still not recommended to use them.

More importantly, the techniques proposed in [MD96, GMR95] would completely fail in the context of view synchronization when view maintenance is done *after* the capability change took place at the base relation site. The reason for this is that the deleted attribute or deleted relation that is to be queried during maintenance may no longer be available at the base relation site. The maintenance strategies for view redefinition proposed thus far in the literature all had made the simplifying assumption that the base relations *don't* change and are available during the process of maintenance. This is clearly an incorrect assumption in modern dynamic information

---

[1] We defined an extended view definition language (a derivative of SQL, which we call Evolvable-SQL or short E-SQL) that incorporates user preferences for change semantics of the view definition.

[2] A set of primitive changes are defined in [MD96, GMR95], such as drop an attribute from the SELECT clause, add a relation to the FROM clause.

environments. In our system, a new challenge arises: after a capability change of an autonomous IS, the *EVE* system has no longer access to the old data in the changed relation. For example, after an attribute A is dropped from a base relation R and the view definition V is being evolved accordingly by dropping a condition using the attribute $R.A$, one can no longer access the dropped attribute $R.A$, when attempting to maintain the affected view.

In this paper we propose view maintenance strategies after view synchronization that regard complex changes (the synchronization process in general changes components in one pass in all three clauses: SELECT , FROM and WHERE) of the view definition as atomic operations and handle associated maintenance tasks, such as querying ISs, as efficiently as possible. Moreover the maintenance strategies take advantage of the knowledge of how the view rewritings were obtained, especially the containment information between the old and new view extents, which is available in our system due to the view rewritings being performed by the system itself. The proposed techniques are general enough to be also applicable when complex (simultaneous) view redefinitions are *explicitly* done by the user and *directly applied* to the view definition.

In summary, the contributions of this current work are: (1) we introduce a new view adaptation problem, namely view maintenance after view synchronization and distinguish it from the view redefinition problem treated in [MD96, GMR95]; (2) we present strategies for the view maintenance after view synchronization for different sets of assumptions such as the availability of the to-be-deleted relation during the maintenance process, view evolution parameter values, and types of containment constraints; (3) we demonstrate that our techniques succeed in cases when those previous techniques would fail; and (4) we conduct experiments that compare our proposed strategies with rematerialization and redefinition techniques; the results illustrate the performance gains of our new algorithms of 400% over two other methods of view maintenance after redefinition.

The remainder of the paper is structured as follows. In Section 2 we discuss work related to the view adaptation problems we consider in *EVE*. In Section 3 we present the general *EVE* approach, in particular, a synchronization algorithm. Section 4 gives an extensive example of mapping the view maintenance after synchronization problem to the redefinition problem. In Section 5 we introduce strategies for view maintenance after synchronization. Section 6 summarizes the experimental results while Section 7 concludes the paper.

2 Related Work

To our knowledge, we are the first to study the problem of view synchronization caused by capability changes of participating information sources. In [RLN97], we establish a taxonomy of view adaptation problems that identifies alternate dimensions of the problem space, and hence serves as a framework for distinguishing our view synchronization problem from other (previously studied) view adaptation problems. In our previous work [LNR97a, NLR98, LNR97b, NR98a, NR98b], we introduce the overall *EVE* solution framework, and proposed diverse view synchronization strategies each based on the availability of different semantic constraints between ISs, such as join or containment constraints. In [LKNR98] we propose a cost model to rank the set of view rewritings found by the view synchronization process based on their expected maintenance costs and divergence from the extent. In this paper, we now present a first solution for incremental view maintenance after synchronization.

View maintenance after data updates was extensively studied in the literature [Wid95, GM95, BLT86, AAS97]. The main distinction between the classical view maintenance problem and our problem of maintenance after synchronization is the type of changes that trigger the maintenance process. In the case of maintenance after data updates, the changes are at the data level. That is, individual tuples of one of the base relations used in the view are changed. However, in the case of maintenance after synchronization there are three potential changes: (1) a schema-level change in one of the base relations (e.g., "*delete-attribute A*"); (2) this generally implies data changes as well in this base relation (e.g., all values for attribute A for all tuples get deleted), and (3) schema changes in the view definition itself (the view definition is synchronized to no longer be referring to the attribute A). This distinction makes the two problems complementary and, as we will propose in this paper, some subproblems arising in the process of view maintenance after synchronization can be reformulated so that techniques developed for view maintenance after data updates can now also be applied to solve them.

Gupta et al. [GMR95] and Mohania et al. [MD96] address the problem of how most efficiently to maintain a materialized view after a view redefinition *explicitly initiated by the user* takes place. They study under which conditions this view maintenance can efficiently take place with minimum access to base relations. In both approaches [GMR95, MD96], the base relations are assumed to remain unchanged (i.e., the data in the base relations is still available during the maintenance process). We compare our novel solutions to their techniques in our experiments, when appropriate.

Research on query reformulation using materialized views [LMS95, SDJL96] considers the problem of replacing an original query with a new expression containing materialized view definitions such that the new query is *equivalent* to the old one. To the best of our knowledge, there is no work done in this context of query reformulation using views with the goal of generating queries without *equivalence* but rather with some more relaxed view extent relationship (such as, the new reformulated query could be a subset of the original query).

3 *EVE*: The Evolvable View Environment Approach

In this section we review the principles of view synchronization in the context of *EVE* that are needed for the reminder of this paper.

3.1 MISD: The Model for Source Description

While individual ISs could be based on any data model, the schema exported by an IS when joining our integrated system is assumed to be described by a set of relations $IS.R_1$, $IS.R_2$, ..., $IS.R_n$. Besides these schema descriptions, our system has knowledge about the relationships among different ISs as we describe below. The descriptions of the ISs are stored in the meta knowledge base (MKB) and are used in the process of view evolution [LNR97a].

Example 1 *We will use the following example in the rest of the paper. Consider a large travel agency that has the headquarter in Detroit, USA, and many branches all over the world. It helps its customers to arrange flights,*

IS #	Descriptions
IS 1	**Customer(Name, State, Phone, Age)**
IS 2	**FlightRes(PName, Airline, FlightNo, Source, Dest, Date)**
IS 3	**Accident–Ins(Holder, Type, Amount, Age)**

Figure 1: Content Descriptions for Ex. 1

car rentals, hotel reservations, tours, and purchasing insurances. Some of the IS descriptions in the MISD format are listed in Fig. 1.

While the MISD model provides many types of semantic constraints [LNR97a, NLR98], below we only discuss those used in the remainder of this paper. A relation R is described by specifying its information source and its set of attributes as $IS.R(A_1, \ldots, A_n)$. Each attribute A_i is given a name and a data type to specify its domain of values. This information is specified by a *type integrity constraint* of the format $R(A_1, \ldots, A_n) \subseteq Type_1(A_1), \ldots, Type_n(A_n)$. It says that an attribute A_i is of type $Type_i$, for $i = 1, \ldots, n$. If two attributes are exported with the same name, they are assumed to have the same type.

Name	Syntax
Type Integrity Constraint	$\mathcal{TC}_{R.A_i} = (R(A_i) \subseteq Type_i(A_i))$
Partial/Complete Constraint	$\mathcal{PC}_{R_1,R_2} = (\pi_{A_{i_1},\ldots,A_{i_k}}(\sigma_{\mathcal{C}(A_{j_1},\ldots,A_{j_l})} R_1)\ \phi\ \pi_{A_{n_1},\ldots,A_{n_k}}(\sigma_{\mathcal{C}(A_{m_1},\ldots,A_{m_t})} R_2))$ $\phi \in \{\subset, \subseteq, \equiv, \supseteq, \supset\}$

Figure 2: Semantic Constraints for IS Descriptions.

The *partial/complete* containment constraints (denoted by $\mathcal{PC}$-constraint) describe that a fragment of a relation is part of or equal to a fragment of another relation for all extents of the two relations. The $\mathcal{PC}$-constraints, sometimes referred as *containment*, are used to decide if an evolved view is equivalent, subset of, or superset of the initial view. For two relations R_1 and R_2, the $\mathcal{PC}$ constraint is given as in Figure 2 where ϕ is $\{\subseteq, \supseteq, \equiv\}$ for the partial ($\subseteq$ and $\supseteq$) or complete ($\equiv$) information constraint, respectively; $A_{i_1}, \ldots, A_{i_k}$, $A_{j_1}, \ldots, A_{j_l}$ are attributes of R_1; and $A_{n_1}, \ldots, A_{n_k}$, $A_{m_1}, \ldots, A_{m_t}$ are attributes of R_2. The sets $A_{i_1}, \ldots, A_{i_k}$ of R_1 and $A_{n_1}, \ldots, A_{n_k}$ of R_2 are such that they pair-wise match, i.e., so that any attribute $R_1.A_{i_s}$, for $s = 1, \ldots, k$, has the same type as $R_2.A_{n_s}$.

Example 2 *The constraint in Eq. (1) states that the projection of relation **Accident–Ins** is a subset of the projection of relation **Customer** for the attributes **Holder** and **Age**, and **Name** and **Age**, respectively, with the some select conditions on both relations.*

$$\begin{aligned} &\mathcal{PC}_{Customer, Accident-Ins} = \\ &\pi_{Accident-Ins.Holder, Accident-Ins.Age}(\sigma_{(Accident-Ins.Amount > 1,000,000)\ AND\ (Accident-Ins.Age < 50)} \boldsymbol{Accident{-}Ins}) \\ &\quad \subseteq \pi_{Customer.Name, Customer.Age} \boldsymbol{Customer} \end{aligned} \tag{1}$$

3.2 E-SQL: The Evolvable SQL View Definition Language

In this section, we introduce Evolvable-SQL, representing one possible approach toward attaching evolution semantics to view definitions. E-SQL is an extension of the SQL view definition language (a detailed description of E-SQL can be found in [LNR97b]) that allows user evolution preferences to be specified for the view components. Evolution preferences, expressed as *evolution parameters*, allow the user to specify criteria based on which the view will be evolved by the system under capability changes at the ISs. As indicated in Fig. 3, each component

Evolution Parameter		Semantics
Attribute-	dispensable ($\mathcal{AD}$)	*true:* the attribute is dispensable *false:* the attribute is indispensable
	replaceable ($\mathcal{AR}$)	*true:* the attribute is replaceable *false:* the attribute is nonreplaceable
Condition-	dispensable ($\mathcal{CD}$)	*true:* the condition is dispensable *false:* the condition is indispensable
	replaceable ($\mathcal{CR}$)	*true:* the condition is replaceable *false:* the condition is nonreplaceable
Relation-	dispensable ($\mathcal{RD}$)	*true:* the relation is dispensable *false:* the relation is indispensable
	replaceable ($\mathcal{RR}$)	*true:* the relation is replaceable *false:* the relation is nonreplaceable
View-	extent ($\mathcal{VE}$)	$\equiv$: the new extent is equal to the old extent $\supseteq$: the new extent is a superset of the old extent $\subseteq$: the new extent is a subset of the old extent $\approx$: the new extent could be anything

Figure 3: View Evolution Parameters of E-SQL Language.

of the view definition (i.e., attribute, relation or condition) has attached two evolution parameters. One, the *dispensable parameter* (notation $\mathcal{XD}$, where $\mathcal{X}$ could be $\mathcal{A}$ (for attribute), $\mathcal{R}$ (for relation) or $\mathcal{C}$ (for condition)) specifies if the component could be dropped ($true$) or must be present in any evolved view definition ($false$). Two, the *replaceable parameter* (notation $\mathcal{XR}$) specifies if the component could be replaced in the process of view evolution ($true$) or must be left unchanged as defined in the initial view ($false$). In Fig. 3, each type of evolution parameter used by E-SQL is represented by a row in the table. Column one gives the parameter name and its abbreviation while column two lists the possible values each parameter can take (default values are underlined). Note that an E-SQL view definition having all evolution parameters set to the default values is semantically equivalent to the conventional SQL view definition, i.e., it cannot be evolved.

Example 3 *Let's assume a web-based travel agency TRAV has a promotion for its customers who travel to Asia by air. TRAV either sends promotion letters to these customers or calls them by phone. Therefore, it needs to find the customers' names, addresses, and phone numbers. The E-SQL definition of this Asia-Customer view is shown in Eq. (2).*

$$\begin{array}{ll}
\text{CREATE VIEW} & \textbf{Asia-Customer}\ (\mathcal{VE} = \supseteq)\ \text{AS} \\
\text{SELECT} & \textbf{C.Name}\ (\mathcal{AR} = true),\ \ \textbf{C.State}\ (\mathcal{AR} = true), \\
 & \textbf{C.Phone}\ (\mathcal{AD} = \textbf{true}) \\
\text{FROM} & \textbf{Customer C}\ (\mathcal{RR} = true),\ \ \textbf{FlightRes F} \\
\text{WHERE} & (\textbf{C.Name} = \textbf{F.PName})\ \text{AND}\ (\textbf{F.Dest} = \textbf{'Asia'})\ (\mathcal{CD} = true)
\end{array} \tag{2}$$

Assume the company is willing to put off the phone marketing strategy, if the customer's phone number cannot be obtained, e.g., the information provider of the ***Customer*** *relation decides to delete* ***Phone****. This preference is stated in the SELECT clause of Eq. (2) by the attribute-dispensable parameter $\mathcal{AD}$ = true for the attribute* ***Phone****. In addition, if the travel agent is willing to accept the customer information from other branches, we set the relation-replaceable parameter $\mathcal{RR}$ in the FROM clause to true for the relation* ***Customer****. Further, let's assume TRAV is willing to offer its promotion to all the customers who travel by air, if identifying who travels to*

Asia is impossible (i.e., the second WHERE *condition cannot be verified). This preference can be explicitly specified by associating the condition-dispensable parameter* $\mathcal{CD}$ = *true with that condition in the* WHERE *clause. The evolution parameters having default values are not shown.*

3.3 Formal Foundation for View Synchronization

For two relations R and R' with different attribute sets $Attr(R)$ and $Attr(R')$ such that $Attr(R) \cap Attr(R') \neq \emptyset$, we compare the extents of the two relations by comparing the projections on their common attributes. Fig. 4 summarizes all π-set operations defined on this common-subset-of-attributes notion, as well as π-equality among tuples.

Name	Set Operator	Semantics
π-equivalent	$R =_\pi R'$	$\forall\, t' \in R', \exists\, t \in R$ s.t. $t'[Attr(R) \cap Attr(R')] = t[Attr(R) \cap Attr(R')]$ and $\forall\, t' \in R', \exists\, t \in R$ s.t. $t'[Attr(R) \cap Attr(R')] = t[Attr(R) \cap Attr(R')]$
π-subset	$R' \subseteq_\pi R$	$\forall\, t' \in R', \exists\, t \in R$ s.t. $t'[Attr(R) \cap Attr(R')] = t[Attr(R) \cap Attr(R')]$
π-superset	$R' \supseteq_\pi R$	$\forall\, t \in R, \exists\, t' \in R'$ s.t. $t[Attr(R) \cap Attr(R')] = t'[Attr(R) \cap Attr(R')]$
π-equality	$t =_\pi t'$	$t \in R,\ t' \in R'$, *s.t.*$t[Attr(R) \cap Attr(R')] = t'[Attr(R) \cap Attr(R')]$;

Figure 4: Set Operators Augmented by the Notion of Common Subset of Attributes.

We use the notation $Attr(V(R))_{\text{SELECT}}(d, r)$ or $Attr(V(R))_{\text{WHERE}}(d, r)$ to denote all attributes of the relation R that are in the SELECT or WHERE clause of the view V with the evolution parameters set to d and r (d and r can be *false* (f) or *true* (t)), respectively. These and others notations are summarized in Fig. 5.

Notation	Definition
$Attr(V(R))_{\text{SELECT}}(d,r)$	$\{R.A \mid R.A$ in SELECT clause of V s.t. $\mathcal{AD}(R.A) = d$ AND $\mathcal{AR}(R.A) = r\}$
$Attr(V(R))_{\text{SELECT}}$	$\cup_{d,r \in \{false,true\}} Attr(V(R))_{\text{SELECT}}(d,r)$
$Attr(V(R))_{\text{WHERE}}(d,r)$	$\{R.A \mid R.A$ in a condition C in WHERE clause of V s.t. $\mathcal{CD}(C) = d$ AND $\mathcal{CR}(C) = r\}$
$Attr(V(R))_{\text{WHERE}}$	$\cup_{d,r \in \{false,true\}} Attr(V(R))_{\text{WHERE}}(d,r)$
$Attr(V(R))_{\text{WHERE}}^{\text{LOCAL}}$	$\{R.A \mid R.A$ in a *local* (select) condition C in WHERE clause of $V\}$
$Attr(V(R))_{\text{WHERE}}^{\text{JOIN}}$	$\{R.A \mid R.A$ in a *join* condition C in WHERE clause of $V\}$
$Attr(V(R))$	$Attr(V(R))_{\text{SELECT}} \cup Attr(V(R))_{\text{WHERE}}$

Figure 5: Notations for Attribute Sets.

The capability changes such as "*delete−attribute R.A*" and "*delete−relation R*" are said to *affect* views that refer to that particular attribute or relation in their definitions. For example, for the capability change "*delete−relation R*" with R in the FROM clause of the view V, the components of V that use R are said to be affected. These are: (1) the relation R in the FROM clause of V; (2) all attributes of relation R that appear in the SELECT clause of V; and (3) all conditions using attributes of relation R in the WHERE clause of V.

The semantics of the evolution parameters impose that all indispensable (i.e., the components with their *dispensable* evolution parameter set to *false*) components of a view must be preserved in any synchronized view definition (either exactly as they are in the original view definition or possibly replaced if they are replaceable). For a delete capability change, all *directly affected* components *must* be replaced or dropped in order for the view definition to satisfy the evolution parameters. Thus, a *legal rewriting* V' of an affected view V must satisfy the evolution parameters attached to each view component of V as well as the *view-extent* parameter of V (defined in Fig. 3). This latter condition requires that the extent of the view rewriting V' must be in the relationship specified by the view evolution parameter with the old view extent. For example, if the view-evolution parameter $\mathcal{VE}$ is "$\subseteq$", then $V' \subseteq_\pi V$ must be true after the view synchronization process.

3.4 The POC Strategy for View Synchronization

In the following, we introduce one of the view synchronization strategies used by our *EVE* system, namely the **PrOject Containment** (POC) algorithm [NR98a]. The POC algorithm uses *containment information* for replacing the deleted relation with another relation such that the redefined view satisfies the evolution parameters imposed by the E-SQL view specification. That is, all indispensable view components are preserved in the new view rewriting and the view-extent evolution parameter is satisfied.

We assume an E-SQL SPJ view V defined as in Eq. (3) with R a relation in the FROM clause. We emphasize in the view definition the attributes of the relation R used in the view because we will study the effect of the capability change "*delete−relation R*" on the view V. In order for the view V to be evolvable under this capability change, the view definition (Eq. (3)) must have the following properties: (1) there are no attributes of R in the SELECT clause with evolution parameters $(false, false)$, hence $Attr(V(R))_{\text{SELECT}}(f,f) = \emptyset$ (line 2 in Eq. (3)), (2) the relation R in the FROM clause is replaceable ($\mathcal{RR} = true$) (line 4 in Eq. (3)), and (3) attributes of R used in the WHERE clause are referred to in conditions with evolution parameters different than $(false, false)$, thus $Attr(V(R))_{\text{WHERE}}(f,f) = \emptyset$ (line 5 in Eq. (3)). The WHERE clause of the view V contains a conjunction of primitive clauses denoted by $\mathcal{CV}$ referring only to the attributes in $Attr(V(R))_{\text{WHERE}}(f,t)$, $Attr(V(R))_{\text{WHERE}}(t,t)$, $Attr(V(R))_{\text{WHERE}}(t,f)$ and $\bar{W}$ ($\bar{W}$ doesn't contain attributes from R).

$$
\begin{array}{ll}
\text{CREATE VIEW} & V \;\; (\mathcal{VE} = \delta) \text{ AS} \\
\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), Attr(V(R))_{\text{SELECT}}(t,t), Attr(V(R))_{\text{SELECT}}(t,f), \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
\text{FROM} & R(\mathcal{RR} = true), R_1, \ldots, R_n \\
\text{WHERE} & \mathcal{CV}(Attr(V(R))_{\text{WHERE}}(f,t), Attr(V(R))_{\text{WHERE}}(t,t), Attr(V(R))_{\text{WHERE}}(t,f), \bar{W})
\end{array}
\qquad (3)
$$

When a relation R is deleted, the POC algorithm must determine which alternate relation S is suitable for replacement of R in the view V (Eq. (3)) in order to construct a rewriting V'. Assume a relation S has been found to be a suitable replacement for R in V (Eq. (3)) based on the Conditions 1 trough 4 given below. Then POC will generate the view V' in Eq. (4) from V (in Eq. (3)) by replacing R with S and all attributes of R with their replacement in S (as stated in Condition 1). V' obtained in this way is shown in Eq. (4) with the new view components underlined.

$$
\begin{array}{ll}
\textsf{CREATE VIEW} & V' \ \ (\mathcal{VE} = \delta) \ \textsf{AS} \\
\textsf{SELECT} & \underline{S(Attr(V(R))_{\textsf{SELECT}}(f,t)), S(Attr(V(R))_{\textsf{SELECT}}(t,t))}, R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
\textsf{FROM} & \underline{S}, R_1, \ldots, R_n \\
\textsf{WHERE} & \underline{\mathcal{CV}(S(Attr(V(R))_{\textsf{WHERE}}(f,t), S(Attr(V(R))_{\textsf{WHERE}}(t,t)), \bar{W})}
\end{array}
\tag{4}
$$

In Eq. (4) $S(\ldots)$ denotes the attribute replacements from $S.\bar{B}$ corresponding to attributes of R that have replacements. $\mathcal{CV}(S(Attr(V(R))_{\textsf{WHERE}}(f,t)), S(Attr(V(R))_{\textsf{WHERE}}(t,t)), \bar{W})$ is the conjunction of primitive clauses in the WHERE clause of the view V (Eq. (3)) where attributes of relation R were replaced by the corresponding attributes in $S.\bar{B}$. Next we describe the four conditions of relations replaceability that are *sufficient* (but not *necessary*) for the view rewriting V' found by POC to be a legal rewriting. The proof can be found in [NR98a].

A relation $IS_2.S$ is said to be a replacement for $IS_1.R$ by POC if the following conditions are satisfied:

Condition 1: Project-Relation Containment. Assume that Eq. (5) is true for all states of relations R and S, i.e., the following $\mathcal{PC}$-constraint is defined in the MKB:

$$\mathcal{PC}_{S,R} = (\pi_{\bar{B}}(S) \ \phi \ \pi_{\bar{A}}(R)), \ \ \phi \ \in \{\subseteq, \equiv, \supseteq, \approx\}. \tag{5}$$

The attributes $S.\bar{B} \subseteq Attr(S)$[3] are called *replacements* for the attributes $R.\bar{A} \subseteq Attr(R)$. We use the notation $S(R.A_i)$ to indicate the replacement attribute of the attribute $R.A_i$ in S. That is, if the attribute $R.A_i \in R.\bar{A}$ has the same position as the attribute $S.B_i \in S.\bar{B}$ in the vectors $R.\bar{A}$ and $S.\bar{B}$, respectively, the replacement of the attribute $R.A_i$ is $S.B_i$ denoted by $S(R.A_i) = S.B_i$. If an attribute $R.C$ doesn't have a replacement in S (i.e., $R.C \notin R.\bar{A}$) then $S(R.C) = \emptyset$.

Condition 2: Indispensable Evolution Parameters Satisfaction. The relation S must contain replacements for *at least* all attributes of the relation R that are indispensable and replaceable in the view V, that is, all attributes of R in the SELECT clause with $\mathcal{AD} = false$ and $\mathcal{AR} = true$ and all attributes of the relation R that appear in the WHERE clause in a condition C with $\mathcal{CD} = false$ and $\mathcal{CR} = true$. This is formally stated in Eq. (6):

$$Attr(V(R))_{\textsf{SELECT}}(f,t) \ \cup \ Attr(V(R))_{\textsf{WHERE}}(f,t) \ \subseteq \ R.\bar{A}. \tag{6}$$

Condition 3: View-Extent Satisfaction. This condition expresses the properties that must hold for the $\mathcal{PC}$-constraint defined by Eq. (5) in order for the view-extent evolution parameter $\mathcal{VE}$ of the view V (Eq. (3)) to be satisfied if the attributes of relations R in V are to be replaced by their replacements in S.

<u>Case 1. $\mathcal{VE} = \subseteq$ or $\equiv$.</u> This case requires that all conditions in the WHERE clause of the view V be preserved regardless of their evolution parameters (Eqs. (7), (8)) in order to guarantee that the view-extent parameter is satisfied, i.e., $V' \equiv_\pi V$.

$$Attr(V(R))_{\textsf{WHERE}}(t,f) = \emptyset \tag{7}$$

$$Attr(V(R))_{\textsf{WHERE}}(f,t) \ \cup \ Attr(V(R))_{\textsf{WHERE}}(t,t) \ \subseteq \ R.\bar{A} \tag{8}$$

[3] We use the notation $X.\bar{B}$ to denote an ordered set of attributes of the relation X. And, we use $Attr(X)$ for the set of all attributes of relation X.

Case 2. $\mathcal{VE} = \supseteq$. In this case, the view extent evolution parameter $\mathcal{VE} =\supseteq$ would be satisfied even if the view is evolved by dropping conditions from the WHERE clause. However, we impose that at least the attributes of relation R used in the join conditions in the WHERE clause are replaced by attributes from $S.\bar{B}$. This condition is formalized in Eq. (9):

$$Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(t,f) = \emptyset \tag{9}$$
$$Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t) \; \cup \; Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(t,t) \; \subseteq \; R.\bar{A}$$

Case 3. $\mathcal{VE} = \approx$. As in Case 2, the view can be evolved by dropping conditions from the WHERE clause given that there is no restriction on the view extent parameter $\mathcal{VE} =\approx$. However, we impose that at least the attributes of relation R used in the join conditions in the WHERE clause are preserved by $S.\bar{B}$. This condition is expressed in Case 2, Eq. (9).

Condition 4: Relationship between $\mathcal{VE}$ and $\mathcal{PC}$-constraint. If the value of the view-extent parameter $\mathcal{VE}$ is different than "don't care", i.e., $\mathcal{VE} \neq\approx$, then the values of $\mathcal{VE}$ and the $\mathcal{PC}$ containment relationship ϕ (Eq. (5)) must satisfy the property given in Eq. (10).

$$\begin{array}{llll} \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \equiv, & \text{then } \phi \text{ must be } \equiv; \\ \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \subseteq, & \text{then } \phi \text{ must be } \subseteq \text{ or } \equiv; \text{ and} \\ \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \supseteq, & \text{then } \phi \text{ must be } \supseteq \text{ or } \equiv. \end{array} \tag{10}$$

Example 4 *We now illustrate the POC algorithm using the view* ***Customer-Passengers*** *defined to select all customers that fly in "Asia" and are younger than 20. The E-SQL view definition for* ***Customer-Passengers*** *is given in Eq. (11).*

$$\begin{array}{ll} \text{CREATE VIEW} & \textbf{Customer-Passengers } (\mathcal{VE} = \delta) \text{ AS} \\ \text{SELECT} & \textbf{C.Name } (\mathcal{AD} = false, \mathcal{AR} = true), \textbf{ C.Age } (\mathcal{AD} = true, \mathcal{AR} = true) \\ & \textbf{C.Phone } (\mathcal{AD} = true, \mathcal{AR} = true) \\ \text{FROM} & \textbf{Customer C } (\mathcal{RD} = true, \mathcal{RR} = true), \textbf{ FlightRes F } (\mathcal{RD} = true, \mathcal{RR} = true), \\ \text{WHERE} & (\textbf{C.Name = F.PName}) \; (\mathcal{CD} = false, \mathcal{CR} = true) \text{ AND } (\textbf{F.Dest = 'Asia'}) \text{ AND} \\ & (\textbf{C.Age} < 20)(\mathcal{CD} = false, \mathcal{CR} = true) \end{array} \tag{11}$$

We assume the capability change "delete-relation ***Customer****". In the MKB, a $\mathcal{PC}$-constraint is defined between the relations* ***Customer*** *and* ***Accidental–Ins*** *as shown in Eq. (12).*

$$\begin{array}{l} \mathcal{PC}_{Accident-Ins, Customer} \;=\; (\pi_{Accident-Ins.Holder, Accident-Ins.Age}(\textbf{Accident–Ins}) \\ \quad \subseteq \pi_{Customer.Name, Customer.Age}(\textbf{Customer})) \end{array} \tag{12}$$

The elements of the view ***Customer–Passengers*** *relative to relation* ***Customer*** *are given in Fig. 6.(The notations used are defined earlier in Fig. 5.)*

The relation ***Accident–Ins*** *has all the properties required by Conditions 1 to 4 falling into Case 1 in Condition 3: ϕ in our $\mathcal{PC}$-constraint from Eq. (12) is $\subseteq$, and all attributes from $Attr(V(R))_{\text{WHERE}}$ have replacements in* ***Accident–Ins*** *(Fig. 6). Then we can conclude that the only view-extent evolution parameter value for the view* ***Customer–Passengers*** *could be $\mathcal{VE} =\subseteq$ (thus δ in Eqs. (11) and (13) must be $\subseteq$). The POC algorithm obtains the legal view rewriting* ***Customer–Passengers'*** *defined in Eq. (13).*

Element of V	*Definition*
$Attr(V(R))_{SELECT}(f,t)$	***Customer.Name***
$S(Attr(V(R))_{SELECT}(f,t))$	***Accident–Ins.Holder***
$Attr(V(R))_{SELECT}(t,t)$	***Customer.Age, Customer.Phone***
$S(Attr(V(R))_{SELECT}(t,t))$	***Accident–Ins..Age***
$Attr(V(R))_{WHERE}(f,t)$	***Customer.Name, Customer.Age***
$S(Attr(V(R))_{WHERE}(f,t))$	***Accident–Ins.Holder, Accident–Ins.Age***

Figure 6: *Elements of the* ***Customer-Passengers*** *View Relative to* ***Customer*** *Relation.*

$$
\begin{array}{ll}
\text{CREATE VIEW} & \textbf{\textit{Customer-Passengers'}}\ (\mathcal{VE} = \subseteq)\ \text{AS} \\
\text{SELECT} & \underline{\textbf{\textit{A.Holder}}\ (\mathcal{AD} = false, \mathcal{AR} = true),\ \textbf{\textit{A.Age}}\ (\mathcal{AD} = true, \mathcal{AR} = true)} \\
\text{FROM} & \underline{\textbf{\textit{Accident–Ins A}}\ (\mathcal{RD} = true, \mathcal{RR} = true)},\ \textbf{\textit{FlightRes F}}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
\text{WHERE} & \underline{(\textbf{\textit{A.Holder = F.PName}})\ (\mathcal{CD} = false, \mathcal{CR} = true)}\ \text{AND}\ (\textbf{\textit{F.Dest = 'Asia'}}) \\
 & \underline{\text{AND}\ (\textbf{\textit{A.Age}} < 20)(\mathcal{CD} = false, \mathcal{CR} = true)}
\end{array}
\tag{13}
$$

4 Applying Techniques for Maintenance after View Redefinition to Maintenance after View Synchronization

Let's assume that we could apply the techniques for view maintenance after redefinition [GMR95, MD96] to bring the view extent up-to-date after view synchronization by treating the (possibly complex) view synchronization changes as a sequence of primitive view redefinition changes. [GMR95] defines the set of primitive changes such as drop/add an attribute from/to the SELECT clause, drop/add a condition from/to the WHERE clause, and drop/add a relation from/to the FROM clause.

Assume the capability change "*delete-relation R*" applied to a base relation R causes a complex change to a view V replacing the view components referring to this relation by the corresponding components from a replacement relation S. The complex view change initiated by the POC algorithm for finding legal view rewritings for an affected view V (Eq. (3)) is equivalent to the following sequence of primitive changes applied directly to the view definition V:

S1. Drop the select conditions referring to attributes of relation R from the WHERE clause, i.e., the ones using the attributes $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(f,t)$, $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(t,t)$ and $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(t,f)$[4].

S2. Drop the join conditions referring to attributes of relation R from the WHERE clause of the view V, i.e., the ones involving attributes from $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t)$, $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(t,t)$ and $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(t,f)$.

S3. Drop relation R from the FROM clause and the attributes $Attr(V(R))_{\text{SELECT}}(f,t)$, $Attr(V(R))_{\text{SELECT}}(t,t)$ and $Attr(V(R))_{\text{SELECT}}(t,f)$ from the SELECT clause of the view V.

[4] In [GMR95], the equijoin conditions are treated differently than the select conditions involving only one relation. Hence we need to separate the condition treatment into the two steps S1 and S2.

S4. Add relation S to the FROM clause; add the replacement attributes of S to the SELECT clause, i.e., $S(Attr(V(R))_{\text{SELECT}}(f,t))$ and $S(Attr(V(R))_{\text{SELECT}}(t,t))$ are added to the SELECT clause.

S5. Add the join conditions for which replacement attributes were identified from the relation S into the WHERE clause of the view V; i.e., the predicates $\mathcal{CV}(S(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t)),\ S(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(t,t)))$ are added to the WHERE clause of V.

S6. Add the local (select) conditions referring to attributes of S (replacing attributes of R) to the WHERE clause of the view V. Thus, the predicates $\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(f,t)), S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(t,t)))$ are added to the WHERE clause of the view V.

The above approach has two major drawbacks. One it can be used only if the relation R is still available even after the capability change "*delete−relation R*" has taken place. The reason for this is that, for example, step S1 is using relation R to retrieve more tuples from R. Two, intermediate steps such as S1 or S2 could considerably increase the size of the intermediate views, requiring unnecessarily overhead in term of source I/O time, messages exchanged, as well as overall computation time during querying the ISs for either R and/or S multiple times (see Example 5).

Example 5 *Let's now apply the steps S1 through S6 listed above for Example 4. Assume that the cardinalities of the relations are* |***Customer***| = 40000, |***FlightRes***| = 20000, |***Accident−Ins***| = 30000 *and* |***Customer-Passengers***| = 1000. *In Table 1, we show the intermediate results for the steps S1 through S6 listed above, where steps S4 and S5 were combined into one operation for simplicity. In this example, the biggest intermediate result of 10000 tuples is obtained by dropping relation* ***Customer*** *from the view, before using relation* ***Accident−Ins*** *for rewriting. Even if the first steps S1 through S3 were to be combined, the intermediate result of 10000 tuples would have to be kept before joining with relation* ***Accident-Ins***. *The final rewritten view* ***Customer-Passengers'*** *has only 400 tuples and in this particular example, we can prove that these tuples were in the view* ***Customer-Passengers*** *to begin with. This suggests that a strategy for directly finding the tuples that have to be deleted is preferable to dealing with these six small redefinitions one at a time.*

Step	*Primitive change*	*View Size after change*	*# Tuples retrieved from*		
			Customer	***FlightRes***	***Accident−Ins***
S1	*Drop* ***C.Age*** < 20	*8000*	*40000*	*10000*	*0*
S2	*Drop* ***C.Name = F.PName***	*10000*	*0*	*10000*	*0*
S3	*Drop* ***Customer, C.Name*** *Drop* ***C.Age, C.Phone***	*10000*	*0*	*0*	*0*
S4, S5	*Add* ***Accident−Ins*** *Add* ***A.Age, A.Holder*** *Add* ***A.Age*** < 20	*6000*	*0*	*0*	*6000*
S6	*Add* ***A.Holder = F.PName***	*400*	*0*	*10000*	*0*

Table 1: *Cardinalities of Intermediate Results for Steps S1 through S6.*

In the above example, we show that in the case of the view redefinition caused by view synchronization it is *inefficient* to consider the complex synchronization change as a sequence of local changes and to apply the maintenance techniques proposed in the literature for view redefinitions [GMR95] to handle each of these view changes one by one. In the example below, we now demonstrate applying the view redefinition techniques is not only very expensive but in fact *impossible* under capability changes due to the fact that the strategy proposed in [GMR95] assumes that nothing changes at the IS site.

Example 6 *Consider the step S1 in Example 5 that has as goal to drop the local condition* ($\textbf{C.Age} < 20$) *from the* WHERE *clause of the view* **Customer-Passengers**. *The extent of the redefined view will be bigger than the one of the original view, and by [GMR95] the new view after this one local change would be computed as* $\textbf{Customer-Passengers}^{S1} = \textbf{Customer-Passengers} \cup \textbf{Customer-Passengers}^{+}$, *where* $\textbf{Customer-Passengers}^{+}$ *is defined in Eq. (14):*

$$
\begin{array}{rl}
 & \textbf{Customer-Passengers}^{+} = \\
(\text{SELECT} & \textbf{C.Name, C.Age, C.Phone} \\
\text{FROM} & \textbf{Customer C , FlightRes F} \\
\text{WHERE} & \textbf{(C.Name = F.PName)} \text{ AND } \textbf{(F.Dest = 'Asia')} \text{ AND} \\
 & (NOT\ (\textbf{C.Age} < 20)))
\end{array}
\tag{14}
$$

Note that the expression $\textbf{Customer-Passengers}^{+}$ *is referring to the attribute* **Customer.Age** *which having been dropped is no longer available at the site of the relation* **Customer**. *Hence, this expression cannot be evaluated, and the view maintenance strategy fails.*

5 Materialized View Maintenance after View Synchronization

Using the knowledge of the containment between the to-be-deleted and replacement relations of the view rewriting V' generated by POC, we now propose techniques to directly find the tuples that need to be deleted from or added to the view V to obtain V'. We give a set of maintenance strategies for different combinations of system parameters such as the types of view definitions and values of the view-extent evolution parameter ($\{\approx, \subseteq, \supseteq, \equiv\}$). Moreover, we consider two possible scenarios defined by the availability of the deleted relation: the deleted relation is still available at the maintenance time (e.g., *before change* notification) and the deleted relation is *not* available at the maintenance time (e.g., *after change* notification). For all cases, we assume that we don't maintain duplicate counts. We assume that the old extent of the view V, the replacement relation S, and all other relations used in the FROM clause of the view V' are available during maintenance.

The cases we handle are described in Table 2.

We discuss in great detail Cases 1 through 4 of the strategy SYNCMAB and Cases 11 through 14 of the strategy SYNCMAA which are summarized in Figs. 7 and 8, respectively. The rest of the cases summarized in Figs. 9 and 10 are similar and hence are not further discussed.

Below there are four general strategies the optimizer can choose from for computing the view extent of the view rewriting V':

(I) *rematerialization* strategy: compute the view from scratch given its new definition (Eq. (4));

	Availability of deleted base relation	
(View Definition, $\mathcal{VE}$)	YES	NO
(SPJ ,$\subseteq$)	Case #1	Case #11
(SPJ ,$\equiv$)	Case #2	Case #12
(SPJ ,$\supseteq$)	Case #3	Case #13
(SPJ ,$\approx$)	Case #4	Case #14
(AGGR-SPJ ,$\subseteq$)	Case #5	Case #15
(AGGR-SPJ ,$\equiv$)	Case #6	Case #16
(AGGR-SPJ ,$\supseteq$)	Case #7	Case #17
(AGGR-SPJ ,$\approx$)	Case #8	Case #18

Table 2: Cases of Maintenance after View Synchronization.

(II) SYNCMAB strategy: use the containment information between the to-be-deleted relation R and its replacement S given by the $\mathcal{PC}$-constraint defined in Eq. (5) and apply the maintenance strategies we propose below for Cases 1 through 8. The name SYNCMAB stands for *SYNChronization-driven view MAintenance Before IS changes*;

(III) SYNCMAA strategy: apply specialized techniques for computing V' for the case when relation R is not available as applicable for Cases 11 to 18. The name SYNCMAA is an acronym for *SYNChronization-driven view MAintenance After IS changes*;

(IV) *redefinition* strategy: apply maintenance techniques for view redefinition for each primitive change in the sequence of changes necessary to obtain the new definition V' (see example in Section 4).

In all cases when the deleted relation is still available, i.e., SYNCMAB strategy for Cases 1 through 8, we can take advantage of the relationship given by the $\mathcal{PC}$-constraint defined by Condition 1, namely, Eq. (5): $\mathcal{PC}_{S,R} = (\pi_{\bar{B}}(S)\ \phi\ \pi_{\bar{A}}(R))$. Hence, the SYNCMAB strategy involves computing the difference-set between the to-be-deleted relation R and its replacement relation S (from the $\mathcal{PC}$-constraint). This makes it possible to use view maintenance techniques for data updates [GM95, BLT86, AAS97, Qua96] to incrementally maintain the view rewriting V'. In other words, we can recast our problem into a known data update propagation problem, and thus solve it in this new context. SYNCMAA strategy (Cases 11 through 18) is characterized by the unavailability of the relation R which implies that the difference between R and S cannot be found and hence we have to apply different algorithms for incrementally computing the new view extent.

In Section 5.1, SYNCMAB strategy (Cases 1 through 8) is presented, i.e., we define the techniques of using the containment information between R and S to compute the view extent of V'. In Section 5.2, the SYNCMAA strategy (Cases 11 through 18) is given, i.e., we outline algorithms for maintenance after synchronization problem when relation R is no longer available at the time of maintenance.

5.1 SYNCMAB Strategy: Maintenance after View Synchronization when the Deleted Relation is Available

Cases 1 through 8 are summarized in Figs. 7 and 8. The first column refers to the case and sub-case numbers as presented below. The second column lists the set of conditions defining each case, namely, the value of the view-extent parameter $\mathcal{VE}$, the relationship between the original view V and new view V' and the containment constraints between the deleted-relation R and its replacement S. The last column contains the steps of our maintenance algorithm as discussed in more detail below.

#	Parameters	Maintenance Strategy
1.1 2 3.1	$\mathcal{VE} = \equiv \mid \subseteq \mid \supseteq$ $V' \equiv_{\pi} V$ R *available* $\phi\ = \equiv$ $\pi_{S.\bar{B}}(S) = \pi_{R.\bar{A}}(R)$	INSERT INTO V' (SELECT $Attr(V(R))_{\text{SELECT}}(f,t), (Attr(V(R))_{\text{SELECT}}(t,t) \cap R.\bar{A}),$ $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ FROM V)
1.2	$\mathcal{VE}\ = \subseteq, V' \subseteq_{\pi} V$ R *available* $\phi\ = \subseteq$ $\pi_{S.\bar{B}}(S) \subseteq \pi_{R.\bar{A}}(R)$	$\boxed{0.}$ CREATE VIEW V^0 AS SELECT $Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t))$ $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ FROM $R, R_1, \ldots, R_n$ WHERE $\mathcal{CV}(R.\bar{A} \cap Attr(V(R))_{\text{WHERE}}(\_,t), \bar{W})$ INSERT INTO V^0 (SELECT $Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)),$ $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ FROM V) $\boxed{1.}$ $\Delta R^- =$ (SELECT $R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM R WHERE $\mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))$ EXCEPT (SELECT $S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM S WHERE $\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))$ $\boxed{2.}$ $\Delta V^0_- =$ (SELECT $Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)),$ $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ FROM $\Delta R^-, R_1, \ldots, R_n$ WHERE $\mathcal{CV}(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}, \bar{W}))$ $\boxed{3.}$ $V^0_- = V^0 \setminus \Delta V^0_-$ $\boxed{4.}$ INSERT INTO V' (SELECT * FROM V^0_-)
3.2	$\mathcal{VE}\ = \supseteq, V' \supseteq_{\pi} V$ R *available* $\phi\ = \supseteq$ $\pi_{S.\bar{B}}(S) \supseteq \pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{5.}$ $\Delta R^+ =$ (SELECT $S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM S WHERE $\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))$ EXCEPT (SELECT $R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM R WHERE $\mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})$) $\boxed{6.}$ $\Delta V^0_+ =$ (SELECT $Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)),$ $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ FROM $\Delta R^+, R_1, \ldots, R_n$ WHERE $\mathcal{CV}(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}, \bar{W}))$ $\boxed{7.}$ $V^0_+ = V^0 \cup \Delta V^0_+$ $\boxed{8.}$ INSERT INTO V' (SELECT * FROM V^0_+)
4	$\mathcal{VE}\ = \approx, V' \approx_{\pi} V$ R *available* $\phi\ = \approx$ $\pi_{S.\bar{B}}(S) \approx \pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{1.}$, $\boxed{2.}$, $\boxed{5.}$, $\boxed{6.}$, $\boxed{9.}$ $V^0_{-,+} = V^0 \setminus \Delta V^0_- \cup \Delta V^0_+$ $\boxed{10.}$ INSERT INTO V' (SELECT * FROM $V^0_{-,+}$)

Figure 7: SYNCMAB Strategy: SPJ View Maintenance after View Synchronization (Cases 1-4 as Defined in Table 2).

#	Parameters	Maintenance Strategy
5.1 6 7.1	$\mathcal{VE} = \equiv \mid \subseteq \mid \supseteq$ $V' \equiv_{\pi_{gr}} V$ $R\ available$ $\phi\ = \equiv$ $\pi_{S.\bar{B}}(S) = \pi_{R.\bar{A}}(R)$	INSERT INTO V' (SELECT $grAttr(V(R))(f,t), (grAttr(V(R))(t,t) \cap R.\bar{A}),$ $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ $\mathcal{F}'(aggAttr(V(R))(f,t), (aggAttr(V(R))(t,t) \cap R.\bar{A}), \bar{F})$ FROM V) $\forall f(X)(=F)\ in\ \mathcal{F} \exists f'(Y)\ in\ \mathcal{F}',\ s.t.\ (f(X), f'(Y)) \in$ $\{(MAX(X), MAX(F)), (MIN(X), MIN(F)), (SUM(X), SUM(F)),$ $(COUNT(*), SUM(F))(AVG(X), SUM(FxF')/SUM(F')) \mid F' = COUNT(*)\}$
5.2	$\mathcal{VE}\ = \subseteq, V' \subseteq_{\pi_{gr}} V$ $R\ available$ $\phi\ = \subseteq$ $\pi_{S.\bar{B}}(S) \subseteq \pi_{R.\bar{A}}(R)$	$\boxed{0.}$ CREATE VIEW V^0 AS SELECT $grAttr(V(R))(f,t), (grAttr(V(R))(t,t) \cap R.\bar{A}),$ $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ $\mathcal{F}(aggAttr(V(R))(f,t), (aggAttr(V(R))(t,t) \cap R.\bar{A}), \bar{F})$ FROM $R, R_1, \ldots, R_n$ WHERE $\mathcal{CV}(Attr(V(R))_{\text{WHERE}}(\_,t), \bar{W})$ GROUP BY $grAttr(V(R))(f,t), (grAttr(V(R))(t,t) \cap R.\bar{A}),$ $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ INSERT INTO V^0 (SELECT $grAttr(V(R))(f,t), (grAttr(V(R))(t,t) \cap R.\bar{A}),$ $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ $\mathcal{F}'(aggAttr(V(R))(f,t), aggAttr(V(R))(t,t) \cap R.\bar{A}), \bar{F})$ FROM V) $\boxed{1.}$ $\Delta R^- =$ (SELECT $R.\bar{A} \cap$ $(grAttr(V(R))(\_,t) \cup aggAttr(V(R))(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM R WHERE $\mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))$ EXCEPT (SELECT $S.\bar{B} \cap$ $S(grAttr(V(R))(\_,t) \cup aggAttr(V(R))(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM S WHERE $\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))$ $\boxed{2.}$ Update the AGGR-SPJ view V^0 after base relation data update "*delete-tuples* ΔR^-" [Qua96] $\boxed{4.}$ INSERT INTO V' (SELECT $*$ FROM V^0)
7.2	$\mathcal{VE}\ = \supseteq, V' \supseteq_{\pi_{gr}} V$ $R\ available$ $\phi\ = \supseteq$ $\pi_{S.\bar{B}}(S) \supseteq \pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{5.}$ $\Delta R^+ =$ (SELECT $S.\bar{B} \cap$ $S(grAttr(V(R))(\_,t) \cup aggAttr(V(R))(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM S WHERE $\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))$ EXCEPT (SELECT $R.\bar{A} \cap$ $(grAttr(V(R))(\_,t) \cup aggAttr(V(R))(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM R WHERE $\mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})$ $\boxed{6.}$ Update the AGGR-SPJ view V^0 after base relation data update "*add-tuples* ΔR^+" [Qua96] $\boxed{8.}$ INSERT INTO V' (SELECT $*$ FROM V^0)
8	$\mathcal{VE}\ = \approx, V' \approx_{\pi} V$ $R\ available$ $\phi\ = \approx$ $\pi_{S.\bar{B}}(S) \approx \pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{1.}$, $\boxed{5.}$, $\boxed{8.}$ Update the AGGR-SPJ view V^0 after base relation data update "*add-tuples* ΔR^+" AND "*delete-tuples* ΔR^-" [Qua96] $\boxed{10.}$ INSERT INTO V' (SELECT $*$ FROM V^0)

Figure 8: SYNCMAB Strategy: AGGR-SPJ View Maintenance after View Synchronization (Cases 5-8 as Defined in Table 2).

SYNCMAB - Case 1: SPJ views, $\mathcal{VE} = \subseteq$, relation R is available. This is the case when the containment constraint used for synchronization defined in Condition 4 from Section 3.4 is $\pi_{S.\bar{B}}(S)\phi\pi_{R.\bar{A}}(R)$ (Eq. (5). ϕ has to be either $\equiv$ or $\subseteq$ as given in Eq. (10). Thus we have two cases, $\phi = \equiv$ and $\phi = \subseteq$:

SYNCMAB - Case 1.1: $\phi = \equiv$. We can prove that in this case $V' =_\pi V$ with V and V' defined in Eqs. (3) and (4), respectively. The proof can be found in [Nic98] and is based on two facts: (1) all affected attributes from the WHERE clause of V have replacements in $S.\bar{B}$ (Condition 3) and (2) the $\mathcal{PC}$-constraint (Eq. (5)) states that the projection on attributes of R used in V is equal to the projection on attribute of S used in V'.

Algebraically, V' is a projection of V on attributes from the SELECT clause having replacements in S, i.e., all attributes from $Attr(V(R))_{\text{SELECT}}(f,t)$ and some of the attributes in $Attr(V(R))_{\text{SELECT}}(t,t))$ plus the attributes from all the other relations used in the view (i.e., $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$). This algebraic expression of V' is given in Eq. (15).

$$V' = \pi_{(Attr(V(R))_{\text{SELECT}}(f,t),(Attr(V(R))_{\text{SELECT}}(t,t) \cap R.\bar{A}),R_1.\bar{D}_1,\ldots,R_n.\bar{D}_n)}(V) \tag{15}$$

Given that V' is simply a projection of V, V' can be locally adapted by applying a simple projection query on the materialized view V as shown in Eq. (16).

$$\begin{array}{ll} \text{INSERT INTO } V' & \\ (\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (Attr(V(R))_{\text{SELECT}}(t,t) \cap R.\bar{A}), \\ & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\ \text{FROM} & V) \end{array} \tag{16}$$

SYNCMAB - Case 1.2: $\phi = \subseteq$. In this case, we know that the view rewriting V' is a subset of the view V, i.e., $V' \subseteq_\pi V$[5], based on the containment constraint $\pi_{\bar{B}}(S) \subseteq \pi_{\bar{A}}(R)$. Eq. (5) expresses that the replacement relation S is a subset of the current relation R. Let the view V^0 be obtained from the view V by projecting out the attributes of R that are *not* replaced by attributes of S in the view V' (i.e., the attributes from $Attr(V(R))_{\text{SELECT}}(t,f)$ that cannot be replaced in any legal rewriting as they are nonreplaceable) and dropping the conditions from the WHERE clause referring to attributes that have no replacements in $S.\bar{B}$. V^0 is defined in Eq. (17).

$$\begin{array}{ll} \text{CREATE VIEW} & V^0 \text{ AS} \\ \text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)) \\ & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\ \text{FROM} & R, R_1, \ldots, R_n \\ \text{WHERE} & \mathcal{CV}(R.\bar{A} \cap Attr(V(R))_{\text{WHERE}}(\_,t), \bar{W}) \end{array} \tag{17}$$

V^0 is defined in Eq. (17) and is initialized as defined in Eq. (18). Note that the extent of V^0 is not up-to-date (i.e., consistent to the base relations used in its definition) when its WHERE clause (Eq. (17)) is different than the

[5] The full proof is given in [Nic98].

WHERE clause of the view V. The reason for this is that some predicates were dropped in V^0 thus its extent could be bigger than the extent of V.

$$
\begin{array}{rl}
\text{INSERT INTO } V^0 & \\
(\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)), \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
\text{FROM} & V)
\end{array}
\tag{18}
$$

First we compute the set-difference between the relation R and S (i.e., the set of tuples in R that are not in S), denoted by ΔR^-, that satisfies the local conditions on R used in the WHERE clause of the view V. This set is defined by Eqs. (19) and (20) in algebraic and SQL notation, respectively:

$$
\begin{aligned}
\Delta R^- = {} & \pi_{R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})}(\sigma_{\mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})}(R)) \\
\setminus\ & \pi_{S(R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}))}(\sigma_{\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))}(S))
\end{aligned}
\tag{19}
$$

$$
\begin{array}{rl}
\Delta R^- = & \\
(\text{SELECT} & R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}) \\
\text{FROM} & R \\
\text{WHERE} & \mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})) \\
\text{EXCEPT} & \\
(\text{SELECT} & S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}) \\
\text{FROM} & S \\
\text{WHERE} & \mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))
\end{array}
\tag{20}
$$

The set-difference ΔR^- corresponds to a set of tuples from the relation R that generate a set of tuples in V which are *not* in V'. Thus we need to determine this set and use it to compute the extent of the view V'.

If we propagate these "deletions" to the rewritten view V^0 (by erasing the corresponding tuples from the view V^0) the view extent obtained – denoted by V^0_+ – is equal to the view extent of the view V'. That is, let $R^- = R \setminus_\pi \Delta R^-$, and let V^0_- be obtained from V^0 by replacing relation R with R^- (note that the attributes of relation R used in V^0 are all also attributes of R^-, from Conditions 2 and 3, Section 3.4). Then, V^0_- is defined as in Eq. (21).

$$
\begin{array}{ll}
\text{CREATE VIEW} & V^0_- \text{ AS} \\
\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)) \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
\text{FROM} & R^-, R_1, \ldots, R_n \\
\text{WHERE} & \mathcal{CV}(R.\bar{A} \cap Attr(V(R))_{\text{WHERE}}(\_,t), \bar{W})
\end{array}
\tag{21}
$$

The maintenance techniques to obtain the view V^0_- using the materialized view V^0 could be any of the view maintenance after data deletion strategies proposed in the literature [GM95, BLT86, AAS97]. For example, one of the techniques [BLT86] is to first compute relation ΔV^0_- (defined in Eq. (22)) to consist of tuples of V^0 generated by the "deleted" tuples of ΔR^-, and then to compute the rest of the tuples as $V^0_- = V^0 \setminus \Delta V^0_-$.

$$\begin{array}{rl} \multicolumn{2}{l}{\Delta V^0_- =} \\ (\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)), \\ & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\ \text{FROM} & \Delta R^-, R_1, \ldots, R_n \\ \text{WHERE} & \mathcal{CV}(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}, \bar{W})) \end{array} \tag{22}$$

Having computing the extent of the view V^0_-, we then can use it to (locally) update the view V'. The maintenance equation computing the final desired V' is given by Eq. (23).

$$\begin{array}{rl} \multicolumn{2}{l}{\text{INSERT INTO } V'} \\ (\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)), \\ & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\ \text{FROM} & V^0_-) \end{array} \tag{23}$$

We can prove that the above computation of the extent of view V' is correct based on the observation that in this particular case, all the conditions from the WHERE clause of the view V are preserved in the view definition V' as imposed by Condition 2, Section 3.4.

<u>SYNCMAB - Case 2: SPJ views, $\mathcal{VE} = \equiv$, relation R is available.</u> This case can be treated exactly as Case 1.1 above, with $\phi = \equiv$ as imposed by Condition 4. The strategy is outlined in Fig. 7.

<u>SYNCMAB - Case 3: SPJ views, $\mathcal{VE} = \supseteq$, relation R is available.</u> In general, in this case the view rewriting V' is bigger than the view V, $V' \supseteq_\pi V$[6]. ϕ used in the $\mathcal{PC}$-constraint (Eq. (5)) must be $\equiv$ or $\supseteq$ as imposed by Condition 4, Eq. (10). Thus we have to discuss the two cases of $\phi = \equiv$ and $\phi = \supseteq$:

<u>SYNCMAB - Case 3.1: $\phi = \equiv$.</u> In this case, we can prove that $V' =_\pi V$, i.e., this is identical to the Case 1.1 from above. The maintenance equation is given by Eq. (16) (see also Fig. 7).

<u>SYNCMAB - Case 3.2: $\phi = \supseteq$.</u> We proved in [Nic98] that the view rewriting V' is a superset of the view V, i.e., $V' \supseteq_\pi V$ based on the given $\mathcal{PC}$-constraint $(\pi_{\bar{B}}(S) \supseteq \pi_{\bar{A}}(R))$ in Eq. (5). Let the view V^0 be obtained as above (Eq. (17)) from the view V by projecting out the attributes of R that are *not* replaced by attributes of S in the view V', plus dropping the conditions from the WHERE clause whose attributes have no replacements in S.

As in Case 1.2, we can compute the set-difference between the relation R and S, denoted this time by ΔR^+. The set-difference ΔR^+ can be seen as modeling the set of new (inserted) tuples into relation R. This set of tuples in S that are not in R is defined by Eqs. (24) and (25), respectively in relational algebra and in SQL notation:

$$\Delta R^+ = \pi_{S(R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}))}(\sigma_{\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))}(S))$$

[6] The full proof is given in [Nic98].

$$\setminus \pi_{R.\bar{A}\cap(Attr(V(R))_{\text{SELECT}}(\_,t)\cup Attr(V(R))_{\text{WHERE}})}(\sigma_{\mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})}(R)) \tag{24}$$

$$\begin{array}{rl}
\Delta R^+ = & \\
(\text{SELECT} & S.\bar{B}\cap S(Attr(V(R))_{\text{SELECT}}(_,t)\cup Attr(V(R))_{\text{WHERE}}) \\
\text{FROM} & S \\
\text{WHERE} & \mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))) \\
\text{EXCEPT} & \\
(\text{SELECT} & R.\bar{A}\cap(Attr(V(R))_{\text{SELECT}}(_,t)\cup Attr(V(R))_{\text{WHERE}}) \\
\text{FROM} & R \\
\text{WHERE} & \mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))
\end{array} \tag{25}$$

Note that in Case 3.2, the set ΔR^- as defined in Eq. (20) is empty. And vice versa, in Case 1.2, the set ΔR^+ defined by Eq. (25) is empty. If we propagate these "insertions" to the view V^0 (by incrementally inserting new tuples in the view V^0), the view extent obtained, denoted by V^0_+, is equal to the extent of the view V'. That is, let $R^+ = R\cup_\pi \Delta R^+$, and V^0_+ (Eq. (26)) be obtained from V^0 by replacing relation R with R^+ and dropping the clauses from the WHERE clause that are dropped in the view rewriting V'. Condition 2 in Section 3.4 imposes that only local conditions could be dropped. Thus the conditions preserved in the view V' (and V^0_+) are the select conditions referring to attributes in $R.\bar{A}\cap(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(f,t)\cup Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(t,t))$ and all join conditions referring to attributes in $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}$.

$$\begin{array}{ll}
\text{CREATE VIEW} & V^0_+ \ \text{AS} \\
\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t),(R.\bar{A}\cap Attr(V(R))_{\text{SELECT}}(t,t)) \\
& R_1.\bar{D}_1,\ldots,R_n.\bar{D}_n \\
\text{FROM} & R^+,R_1,\ldots,R_n \\
\text{WHERE} & \mathcal{CV}(R.\bar{A}\cap Attr(V(R))_{\text{WHERE}}(_,t),\bar{W})
\end{array} \tag{26}$$

We have again reduced our problem of view maintenance after synchronization into the problem of maintenance under tuple insertions into one of the base relations. Hence, we can apply maintenance techniques designed to incrementally compute V^0_+ from the view V^0 [GM95, BLT86, AAS97]. To give an example, we can compute $V^0_+ = V^0 \cup \Delta V^0_+$ [BLT86], where ΔV^0_+ (Eq. (27)) is the set of tuples of the view V^0 generated by ΔR^+. Then we can compute ΔV^0_+ by applying view maintenance techniques for the view V^0 under insertion of the tuples ΔR^+ in the base relation R [BLT86].

$$\begin{array}{rl}
\Delta V^0_+ = & \\
(\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t),(R.\bar{A}\cap Attr(V(R))_{\text{SELECT}}(t,t)), \\
& R_1.\bar{D}_1,\ldots,R_n.\bar{D}_n \\
\text{FROM} & \Delta R^+,R_1,\ldots,R_n \\
\text{WHERE} & \mathcal{CV}(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}},\bar{W}))
\end{array} \tag{27}$$

After V^0_+ is computed, we then can use it to (locally) set the view V' using the maintenance strategy given by Eq. (28).

$$\begin{array}{rl} \text{INSERT INTO } V' & \\ (\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)), \\ & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\ \text{FROM} & V^0_+) \end{array} \tag{28}$$

The proof that this proposed technique computes the correct view extent of V' is based on the two facts: (1) only select conditions on R are dropped from the view rewriting V' and (2) the given $\mathcal{PC}$-constraint is $\mathcal{PC}_{S,R} = (\pi_{S.\bar{B}}(S) \supseteq \pi_{R.\bar{A}}(R))$.

SYNCMAB - Case 4: SPJ views, $\mathcal{VE} = \approx$, relation R is available. In this case the view rewriting V' could contain some of the tuples from V (but not necessarily all) and some new tuples. Thus, the technique we propose here is to compute first both ΔV^0_- and ΔV^0_+ as defined above in Eqs. (22) and (27), respectively. If we define $V^0_{-,+} = V^0 \setminus V^0_- \cup V^0_+$, then the view V' is computed as expressed in Eq. (29) (see Fig. 7):

$$\begin{array}{rl} \text{INSERT INTO } V' & \\ (\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)), \\ & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\ \text{FROM} & V^0_{-,+}) \end{array} \tag{29}$$

Example 7 *We give the following example to illustrate the most general case (Case 4) of the SYNCMAB strategy when the view extent evolution parameter is $\approx$. Let's assume the view **Customer-East** is defined in Eq. (30) and the $\mathcal{PC}$-constraint defined in Eq. (31). The capability change is "delete-relation **Customer**". A legal rewriting found by applying the POC algorithm is the view **Customer-East'** (Eq. (32)), where only the attributes **Customer.Name** and **Customer.Age** were replaced in the SELECT clause, and the local condition **C.State** = "MA″ was dropped from the WHERE clause (its dispensable parameter $\mathcal{CD}$ was set to true).*

$$\begin{array}{ll} \text{CREATE VIEW} & \textbf{Customer-East}\ (\mathcal{VE} = \approx)\ \text{AS} \\ \text{SELECT} & \textbf{C.Name}\ (\mathcal{AR} = true),\ \textbf{C.Age}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\ & \textbf{C.Phone}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\ \text{FROM} & \textbf{Customer C}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\ & \textbf{FlightRes F}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\ \text{WHERE} & (\textbf{C.Name} = \textbf{F.PName})\ (\mathcal{CR} = true)\ \text{AND}\ (\textbf{F.Dest} = \text{'}\textbf{Asia}\text{'})\ \text{AND} \\ & (\textbf{C.Age} < 20)(\mathcal{CR} = true)_{\text{AND}} \\ & (\textbf{C.State} = \text{“}MA''\text{)}(\mathcal{CD} = true, \mathcal{CR} = true) \end{array} \tag{30}$$

$$\begin{array}{l} \mathcal{PC}_{Customer, Accident-Ins} = (\pi_{Accident-Ins.Holder, Accident-Ins.Age}(\textbf{Accident-Ins}) \\ \quad \approx \pi_{Customer.Name, Customer.Age}(\textbf{Customer})) \end{array} \tag{31}$$

$$\begin{array}{ll} \textsf{CREATE VIEW} & \textbf{\textit{Customer-East'}}\ (\mathcal{VE} = \approx)\ \textsf{AS} \\ \textsf{SELECT} & \underline{\textbf{\textit{A.Holder}}\ (\mathcal{AR} = true),\ \textbf{\textit{A.Age}}\ (\mathcal{AD} = true, \mathcal{AR} = true)} \\ \textsf{FROM} & \underline{\textbf{\textit{Accident–Ins A}}\ (\mathcal{RD} = true, \mathcal{RR} = true)}, \\ & \underline{\textbf{\textit{FlightRes F}}\ (\mathcal{RD} = true, \mathcal{RR} = true),} \\ \textsf{WHERE} & \underline{(\textbf{\textit{A.Holder}} = \textbf{\textit{F.PName}})\ (\mathcal{CR} = true)}\textsf{AND}(\textbf{\textit{F.Dest}} = \textbf{\textit{'Asia'}}) \\ & \underline{\textsf{AND}(\textbf{\textit{A.Age}} < 20)(\mathcal{CR} = true)} \end{array} \tag{32}$$

Customer

Name	*Age*	*State*	*Phone*
John	*19*	*MA*	*111-1111*
Mary	*18*	*MA*	*222-2222*
Bob	*17*	*ON*	*999-9999*
Bill	*19*	*CA*	*888-8888*
....			

FlightRes

PName	*Dest*
John	*Asia*
Mary	*Asia*
Bob	*Asia*
Gill	*Asia*
....	

Accident-Ins

Holder	*Age*
Mary	*18*
Bob	*17*
Gill	*16*
....	

Table 3: *Data for the Relations Used in Example 7.*

The steps in the SYNCMAB strategy (see Fig. 7) presented above are for this example the following:

$\boxed{0.}$ *The view* ***Customer-East***^{0} *is defined as in Eq. (33) and its extent is initialized with the set* $\{(John, 19), (Mary, 18)\}$ *given the extents of the relations* ***Customer*** *and* ***FlightRes*** *from Table 3. Note that this extent is not up-to-date given the definition of the view* ***Customer-East***^{0}*.*

$\boxed{1.}$ *The expression* Δ***Customer***$^-$ *is* $\{(John, 19)\}$ *(Fig. 7).*

$\boxed{5.}$ *The expression* Δ***Customer***$^+$ *computes* $\{(Gill, 16), (Bob, 17)\}$ *(Fig. 7). Note that the relation* Δ***Customer***$^+$ *contains a new tuple* $(Gill, 16)$ *found only in the relation* ***Accident-Ins*** *plus a* π*-common tuple* $(Bob, 17)$ *of the relations* ***Customer*** *and* ***Accident-Ins****. The tuple* $(Bob, 17)$ *was not part of the initial view* ***Customer-East*** *as it fails the select condition (****C.State*** $='MA'$*) that now was dropped from the rewriting* ***Customer-East'****.*

$$\begin{array}{ll} \textsf{CREATE VIEW} & \textbf{\textit{Customer-East}}^0\ \textsf{AS} \\ \textsf{SELECT} & \textbf{\textit{C.Name, C.Age}} \\ \textsf{FROM} & \textbf{\textit{Customer C, FlightRes F}} \\ \textsf{WHERE} & (\textbf{\textit{C.Name}} = \textbf{\textit{F.PName}})\ \textsf{AND}\ (\textbf{\textit{F.Dest}} = \textbf{\textit{'Asia'}})\textsf{AND}\ (\textbf{\textit{C.Age}} < 20) \end{array} \tag{33}$$

Step $\boxed{9.}$ *is (artificially) considering two types of data-updates in the relation* ***Customer****:*

(i) insertion of the tuples from Δ***Customer***$^+$*: the extent of the view* ***Customer-East***^{0} *is now* $\{(John, 19), (Mary, 18), (Gill, 16), (Bob, 17)\}$*;*

(ii) deletion of the tuples from Δ***Customer***$^-$*: the extent of the view* ***Customer-East***^{0} *is now* $\{(Mary, 18), (Gill, 16), (Bob, 17)\}$*.*

In the step $\boxed{10.}$*, we set the extent of* ***Customer-East'*** *to* $\{(Mary, 18), (Gill, 16), (Bob, 17)\}$*. This corresponds exactly to its definition from Eq. (32).*

#	Parameters	Maintenance Strategy
11.1 12 13.1	$\mathcal{VE} = \equiv \mid \subseteq \mid \supseteq$ $V' \equiv_{\pi} V$ $R\ NOT\ available$ $\phi\ = \equiv$ $\pi_{S.\bar{B}}(S) = \pi_{R.\bar{A}}(R)$	INSERT INTO V' (SELECT $Attr(V(R))_{\text{SELECT}}(f,t)$, $(Attr(V(R))_{\text{SELECT}}(t,t) \cap R.\bar{A})$, $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ FROM V)
11.2	$\mathcal{VE}\ = \subseteq V' \subseteq_{\pi} V$ $R\ NOT\ available$ $\phi\ = \subseteq$ $\pi_{S.\bar{B}}(S) \subseteq \pi_{R.\bar{A}}(R)$	$\boxed{0.}$ CREATE VIEW V^0 AS SELECT $Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t))$ $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ FROM $R, R_1, \ldots, R_n$ WHERE $\mathcal{CV}((R.\bar{A} \cap Attr(V(R))_{\text{WHERE}}(\_,t), \bar{W})$ INSERT INTO V^0 (SELECT $Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t))$, $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ FROM V) $\boxed{1.}$ $\Delta R^- =$ (SELECT $R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM V) EXCEPT (SELECT $S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM S WHERE $\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))$ $\boxed{2.}$ $\Delta V^0_- =$ (SELECT $Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t))$, $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ FROM $\Delta R^-, R_1, \ldots, R_n$ WHERE $\mathcal{CV}(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}, \bar{W}))$ $\boxed{3.}$ $V^0_- = V^0 \setminus \Delta V^0_-$ $\boxed{4.}$ INSERT INTO V'(SELECT $*$ FROMV^0_-)
13.2	$\mathcal{VE}\ = \supseteq, V' \supseteq_{\pi} V$ $R\ NOT\ available$ $\phi\ = \supseteq$ $\pi_{S.\bar{B}}(S) \supseteq \pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{5.}$ $\Delta R^+ =$ (SELECT $S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM S WHERE $\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))$ EXCEPT (SELECT $R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM V) $\boxed{6.}$ $\Delta V^0_+ =$ (SELECT $Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t))$, $R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n$ FROM $\Delta R^+, R_1, \ldots, R_n$ WHERE $\mathcal{CV}(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}, \bar{W}))$ $\boxed{7.}$ $V^0_+ = V^0 \cup \Delta V^0_+$ $\boxed{8.}$ INSERT INTO V'(SELECT $*$ FROMV^0_+)
14	$\mathcal{VE}\ = \approx, V' \approx_{\pi} V$ $R\ NOT\ available$ $\phi\ = \approx$ $\pi_{S.\bar{B}}(S) \approx \pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{1.}$, $\boxed{2.}$, $\boxed{5.}$, $\boxed{6.}$, $\boxed{9.}$ $V^0_{-,+} = V^0 \setminus \Delta V^0_- \cup \Delta V^0_+$ $\boxed{10.}$ INSERT INTO V'(SELECT $*$ FROM$V^0_{-,+}$)

Figure 9: SYNCMAA Strategy: SPJ View Maintenance after View Synchronization (Cases 11-14 as Defined in Table 2).

#	Parameters	Maintenance Strategy
15.1 16 17.1	$\mathcal{VE} = \equiv \mid \subseteq \mid \supseteq$ $V' \equiv_{\pi_{gr}} V$ $R\ NOT\ available$ $\phi\ = \equiv$ $\pi_{S.\bar{B}}(S) = \pi_{R.\bar{A}}(R)$	INSERT INTO V' (SELECT $grAttr(V(R))(f,t), (grAttr(V(R))(t,t) \cap R.\bar{A}),$ $R_1.\bar{D_1}, \dots, R_n.\bar{D_n}$ $\mathcal{F}'(aggAttr(V(R))(f,t), (aggAttr(V(R))(t,t) \cap R.\bar{A}), \bar{F})$ FROM V) $\forall f(X)(= F)\ in\ \mathcal{F} \exists f'(Y)\ in\ \mathcal{F}',\ s.t.\ (f(X), f'(Y)) \in$ $\{(MAX(X), MAX(F)), (MIN(X), MIN(F)), (SUM(X), SUM(F)),$ $(COUNT(*), SUM(F))(AVG(X), SUM(FxF')/SUM(F')) \mid F' = COUNT(*)\}$
15.2	$\mathcal{VE}\ = \subseteq, V' \subseteq_{\pi_{gr}} V$ $R\ NOT\ available$ $\phi\ = \subseteq$ $\pi_{S.\bar{B}}(S) \subseteq \pi_{R.\bar{A}}(R)$	$\boxed{0.}$ CREATE VIEW V^0 AS SELECT $grAttr(V(R))(f,t), (grAttr(V(R))(t,t) \cap R.\bar{A}),$ $R_1.\bar{D_1}, \dots, R_n.\bar{D_n}$ $\mathcal{F}(aggAttr(V(R))(f,t), (aggAttr(V(R))(t,t) \cap R.\bar{A}), \bar{F})$ FROM $R, R_1, \dots, R_n$ WHERE $\mathcal{CV}(Attr(V(R))_{\text{WHERE}}(\_, t), \bar{W})$ GROUP BY $grAttr(V(R))(f,t), (grAttr(V(R))(t,t) \cap R.\bar{A}),$ $R_1.\bar{D_1}, \dots, R_n.\bar{D_n}$ INSERT INTO V^0 (SELECT $grAttr(V(R))(f,t), (grAttr(V(R))(t,t) \cap R.\bar{A}),$ $R_1.\bar{D_1}, \dots, R_n.\bar{D_n}$ $\mathcal{F}'(aggAttr(V(R))(f,t), aggAttr(V(R))(t,t) \cap R.\bar{A}), \bar{F})$ FROM V) $\boxed{1.}$ $\Delta R^- =$ (SELECT $R.\bar{A} \cap$ $(grAttr(V(R))(\_, t) \cup aggAttr(V(R))(\_, t) \cup Attr(V(R))_{\text{WHERE}})$ FROM V) EXCEPT (SELECT $S.\bar{B} \cap$ $S(grAttr(V(R))(\_, t) \cup aggAttr(V(R))(\_, t) \cup Attr(V(R))_{\text{WHERE}})$ FROM S WHERE $\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))$ $\boxed{2.}$ Update the AGGR-SPJ view V^0 after base relation data update "*delete-tuples* ΔR^-" [Qua96] (delete whole groups) $\boxed{4.}$ INSERT INTO V' (SELECT $*$ FROM V^0)
17.2	$\mathcal{VE}\ = \supseteq, V' \supseteq_{\pi_{gr}} V$ $R\ NOT\ available$ $\phi\ = \supseteq$ $\pi_{S.\bar{B}}(S) \supseteq \pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{5.}$ $\Delta R^+ =$ (SELECT $S.\bar{B} \cap$ $S(grAttr(V(R))(\_, t) \cup aggAttr(V(R))(\_, t) \cup Attr(V(R))_{\text{WHERE}})$ FROM S WHERE $\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))$ EXCEPT (SELECT $R.\bar{A} \cap$ $(grAttr(V(R))(\_, t) \cup aggAttr(V(R))(\_, t) \cup Attr(V(R))_{\text{WHERE}})$ FROM V) $\boxed{6.}$ Update the AGGR-SPJ view V^0 after base relation data update "*add-tuples* ΔR^+" [Qua96] (only new groups) $\boxed{8.}$ INSERT INTO V' (SELECT $*$ FROM V^0)
18	$\mathcal{VE}\ = \approx, V' \approx_{\pi} V$ $R\ NOT\ available$ $\phi\ = \approx$ $\pi_{S.\bar{B}}(S) \approx \pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{1.}$, $\boxed{5.}$, $\boxed{8.}$ Update the AGGR-SPJ view V^0 after base relation data update "*add-tuples* ΔR^+" AND "*delete-tuples* ΔR^-" [Qua96] (add/delete whole groups only) $\boxed{10.}$ INSERT INTO V' (SELECT $*$ FROM V^0)

Figure 10: SYNCMAA Strategy: AGGR-SPJ View Maintenance after View Synchronization (Cases 15-18 as Defined in Table 2).

5.2 SYNCMAA Strategy: Maintenance after View Synchronization when Deleted Relation is *NOT* Available

In the following we define the SYNCMAA strategy for handling the Cases 11-14 corresponding to SPJ synchronized views when the to-be-deleted relation R is no longer available for maintenance after synchronization. The SYNCMAA strategy uses the knowledge of the relationship between V' and V given by the view extent parameter $\mathcal{VE}$ which could be $\subseteq$, $\supseteq$, $\equiv$ or $\approx$. These relationships are defined in Section 3.3 using set semantics. We make the assumption that in any rewriting of the view V, all join attributes of R replaced in the WHERE clause of V' are among the attributes replaced in the SELECT clause, i.e., $S(Attr(V(R))_{\text{SELECT}}) \supseteq S(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}})$. We refer to this assumption in the rest of this section as the *inclusion assumption*. Note that when the view rewriting V' has this property, the view maintenance after synchronization could be done in most cases more efficiently by applying the SYNCMAA strategy for Cases 11 through 14 even if the relation R is still available during the maintenance.

The SYNCMAA strategy proposed below assumes that while the relation R is no longer available, the old view extent of V, the replacement relation S, and the rest of the relations in the FROM clause of V' $(R_1, \ldots R_n)$ are still available. On the other hand, in the SYNCMAB strategy, the possibly remote relation R was queried as well which in general will increase the costs of computation, for example, the amount of information sent/retrieved in the process of maintenance.

The SYNCMAA strategy applies the same principle for incrementally computing the view extent V' as the SYNCMAB for Cases 1 through 4, namely, reducing the problem of computing V' to a problem of view maintenance after data updates for the view V^0. Fig. 9 summarizes the cases we present below.

SYNCMAA - Case 11: SPJ views, $\mathcal{VE} = \subseteq$, relation R is NOT available. This is the case when Condition 4 imposes that ϕ in Equation (5) is either $\equiv$ or $\subseteq$ (Eq. (10)). Thus we have the two cases of $\phi = \equiv$ and $\phi = \subseteq$:

SYNCMAA - Case 11.1: $\phi = \equiv$. We treat this case exactly like for the SYNCMAB strategy, Case 1.1, as it doesn't require access to the relation R. The computation of the view rewriting V' is given in Fig. 9.

SYNCMAA - Case 11.2: $\phi = \subseteq$. The view rewriting V' is a subset of the view V, i.e., $V' \subseteq_\pi V$. Let the view V^0 be defined as in Section 5.1, Eq. (17) and be initialized as in Eq. (18). Because the *inclusion assumption* is imposed for any rewriting of the view V, V^0 has the property that all attributes from the WHERE clause of the relation R are in the SELECT clause as well. As in Case 1.2 of the SYNCMAB strategy, we want to compute the set-difference between the relation R and S. This time the relation R is no longer available. Hence we cannot use the definition given in Eq. (20) for ΔR^-. The new definition of the set-difference ΔR^- (given in Eq. (34)) computes the set of tuples from the view V that cannot be generated by any tuples in the relation S.

$$
\begin{array}{rl}
\Delta R^- = & \\
(\text{SELECT} & R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_, t) \cup Attr(V(R))_{\text{WHERE}}) \\
\text{FROM} & V) \\
\text{EXCEPT} & \\
(\text{SELECT} & S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_, t) \cup Attr(V(R))_{\text{WHERE}}) \\
\text{FROM} & S \\
\text{WHERE} & \mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))
\end{array} \tag{34}
$$

We treat again the set-difference ΔR^- as a set of tuples deleted from relation R and propagate these "deletions" to the view V^0. With the *inclusion-assumption* imposed on the definition of the view V^0, namely that all the WHERE attributes of R in V^0 are among the SELECT attributes, we can prove that the view extent obtained after propagation (denoted by V^0_-) is equal to the view extent of the view V'. The proof is immediate from the formulas of the relations ΔR^-, V^0_- and V'. The maintenance techniques used for propagation of the "deletions" to the view V^0 could be any view maintenance after deletion strategies proposed in the literature [GM95, BLT86, AAS97]. In Fig. 9 we give one of the techniques (same as in SYNCMAB strategy, Case 1.2).

SYNCMAA - Case 12: SPJ views, $\mathcal{VE} = \equiv$, relation R is NOT available. This case is treated exactly as Case 11.1 and is summarized in Fig. 9.

SYNCMAA - Case 13: SPJ views, $\mathcal{VE} = \supseteq$, relation R is NOT available. In general, in this case the view rewriting V' is a superset of the view V, $V' \supseteq_\pi V$[7] as imposed by Condition 4, Eq. (10). ϕ used in the $\mathcal{PC}$-constraint (Eq. (5)) must be $\equiv$ or $\supseteq$ which we treat separately in the two Cases 13.1 and 13.2 below.

SYNCMAA - Case 13.1: $\phi = \equiv$. In this case, we can prove that $V' =_\pi V$ ([Nic98]), thus the appropriate maintenance equation is given in Fig. 9. This is the same as for the Cases 11.1 and 12.

SYNCMAA - Case 13.2: $\phi = \supseteq$. We proved in [Nic98] that the view rewriting V' is a superset of the view V, i.e., $V' \supseteq_\pi V$. As in Case 1.2 for the SYNCMAB strategy, we want to compute the set-difference between the relation R and S, denoted by ΔR^+. The expression of ΔR^+ given in Eq. (35) computes the set of tuples in S that are potential candidates to generate new tuples in V'.

$$
\begin{array}{rl}
\Delta R^+ = & \\
(\text{SELECT} & S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_, t) \cup Attr(V(R))_{\text{WHERE}}) \\
\text{FROM} & S \\
\text{WHERE} & \mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))) \\
\text{EXCEPT} & \\
(\text{SELECT} & R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_, t) \cup Attr(V(R))_{\text{WHERE}}) \\
\text{FROM} & V
\end{array} \tag{35}
$$

If we propagate these "insertions" to the view V^0, the view extent obtained (denoted by V^0_+) is equal to the view extent of the view V'. We have again reduced our view maintenance after synchronization problem to

[7] See [Nic98], for the proof.

maintenance under tuple insertion in one of the base relation. Thus we can apply maintenance techniques to incrementally compute the view V^0_+ from the view V^0 [GM95, BLT86, AAS97]. After view V^0_+ is computed, we then can use it to (locally) set the view V'. The maintenance equation for the view V' is given in Fig. 9 under the Case 13.2.

SYNCMAA - Case 14: SPJ views, $\mathcal{VE} = \approx$, relation R is NOT available. We apply the same techniques as in Case 4 by first computing both ΔV^0_- and ΔV^0_+ as for Cases 11.2 and 13.2, respectively. If we define $V^0_{-,+} = V^0 \setminus V^0_- \cup V^0_+$, the view V' is computed as expressed in Fig. 9, Case 14.

Example 8 *Let's consider again Example 7 from Section 5.1 and apply the SYNCMAA strategy defined above when the relation **Customer** is not available. The view **Customer-East** is defined in Eq. (30), while the view rewriting under the capability change "delete-relation **Customer**"(found by the POC algorithm is the view **Customer-East'** (Eq. (32)). Note that **Customer-East'** has all the attributes of the relation **Accident-Ins** (replacing attributes of the relation **Customer**) from the WHERE clause among the attributes in the SELECT clause. Thus, we can apply the techniques defined above for Case 14 of the SYNCMAA strategy, when the view extent evolution parameter is $\approx$, and the $\mathcal{PC}$-constraint is given in Eq. (31). We now go over the steps defined in Table 9 for this particular example.*

*In step $\boxed{0.}$, the view **Customer-East**^{0} is defined as in Eq. (33) and its extent is initialized with the set $\{(John, 19), (Mary, 18)\}$. We assume the states of the relations **Customer**, **FlightRes**, and **Accident-Ins** given in Example 7. Without using the deleted relation **Customer**, the expression Δ**Customer**$^-$ (Eq. (36)) generates the tuple $\{(John, 19)\}$ (step $\boxed{1.}$), while the expression Δ**Customer**$^+$ (Eq. (37)) results in the tuples $\{(Gill, 16), (Bob, 17)\}$ (step $\boxed{5.}$).*

*Step $\boxed{9.}$ considers two types of data-updates to the relation **Customer**: (1) insertion of the tuples from Δ**Customer**$^+$; and (2) deletion of the tuples from Δ**Customer**$^-$. The view **Customer-East**^{0} after these updates is $\{(Mary, 18), (Gill, 16), (Bob, 17)\}$. In step $\boxed{10.}$ the extent of the view rewriting **Customer-East'** is set to the extent of the expression **Customer-East**^{0} and now corresponds to its final definition given in Eq. (32).*

$$
\begin{array}{rl}
 & \Delta \textit{\textbf{Customer-East}}^- = \\
(\text{SELECT} & \textit{\textbf{Name, Age}} \\
\text{FROM} & \textit{\textbf{Customer-East}}) \\
\text{EXCEPT} & \\
(\text{SELECT} & \textit{\textbf{Holder, Age}} \\
\text{FROM} & \textit{\textbf{Accident-Ins}} \\
\text{WHERE} & (\textit{\textbf{Age}} < 20))
\end{array}
\tag{36}
$$

$$
\begin{array}{rl}
 & \Delta \textit{\textbf{Customer-East}}^+ = \\
(\text{SELECT} & \textit{\textbf{Holder, Age}} \\
\text{FROM} & \textit{\textbf{Accident-Ins}} \\
\text{WHERE} & (\textit{\textbf{Age}} < 20)) \\
\text{EXCEPT} & \\
(\text{SELECT} & \textit{\textbf{Name, Age}} \\
\text{FROM} & \textit{\textbf{Customer-East}})
\end{array}
\tag{37}
$$

6 Experimental Results

We now present the experimental results obtained by comparing our proposed view maintenance techniques with two alternative solutions: view rematerialization and view redefinition techniques.

6.1 Description of the Experimental Process

We conduct the experiments using the following strategies. For a given E-SQL view definition V and a capability change *ch* "*delete-relation*", we first obtain a view redefinition V' by applying the POC algorithm (Section 3.4). For each pair V and V', we generate SQL code for computing the view extent of the new view V' in three ways:

(1) apply the techniques presented in Section 5, namely the SYNCMAB strategy[8];

(2) compute the view V' directly from its definition (*rematerialization* strategy);

(3) apply techniques for view redefinition (*redefinition* strategy).

The strategies for SYNCMAB , *redefinition* and *rematerialization* store the results into a new view leaving the old extent unchanged, i.e., we don't use in-place adaptation. Each SQL maintenance code obtained is submitted to the database interface of the commercial database system (we used Sybase) and executed there. We start the timer before sending the SQL code and stop the timer when the results are received back at the DB interface. Thus all the measurements charted in the following sections represent *elapsed time* for execution of the SQL queries. The database system used is SYBASE SQL Server running on a UNIX machine with 128MB RAM memory.

6.2 Description of the TPC-D Benchmark Schema and Database

The sample database used for these experiments is conforming to the TPC-D benchmark [TD95]. The TPC-D benchmark database schema which we use unchanged is described in Figure 6.2. The boxed attributes correspond to the primary keys. The formula such as SF*200k attached to the PART table represents the cardinality of the table. The size of the tables are factored by a SF (Scale Factor) to obtain a chosen database size. The names of the attributes when used in the rest of this section are prefixed by the string in the parenthesis following a table name (e.g., we use *p-name* to refer to the attribute *name* of the relation PART (p-)).

6.3 View Definitions Design for Experimental Evaluation

We have designed several experimental scenarios, each testing different cases of maintenance after synchronization methods for the SYNCMAB strategy as illustrated in Figs. 7 and 8. Figs. 12 and 13 depict a pair of view definitions. The labels above the arrows between the two view definitions indicate the case numbers discussed in the previous section in Table 2 that need to be applied for maintenance. For all four view definitions, the

[8]Note that we cannot compare the SYNCMAA strategy with *redefinition* strategy as the latter requires the to-be-deleted relation to be still available, and hence the *redefinition* strategy cannot be applied in any case when the SYNCMAA strategy can be. The SYNCMAA and SYNCMAB strategies differ only in the computation of the expressions ΔR^+ and ΔR^-. For this computation, the SYNCMAA strategy accesses only the replacement relation S and the old view extent V, while the SYNCMAB strategy accesses the replacement relation S and the to-be-deleted relation R. Hence, in general, we expect the SYNCMAA strategy to outperform the SYNCMAB strategy given that R is a remote relation (i.e., not at the view site).

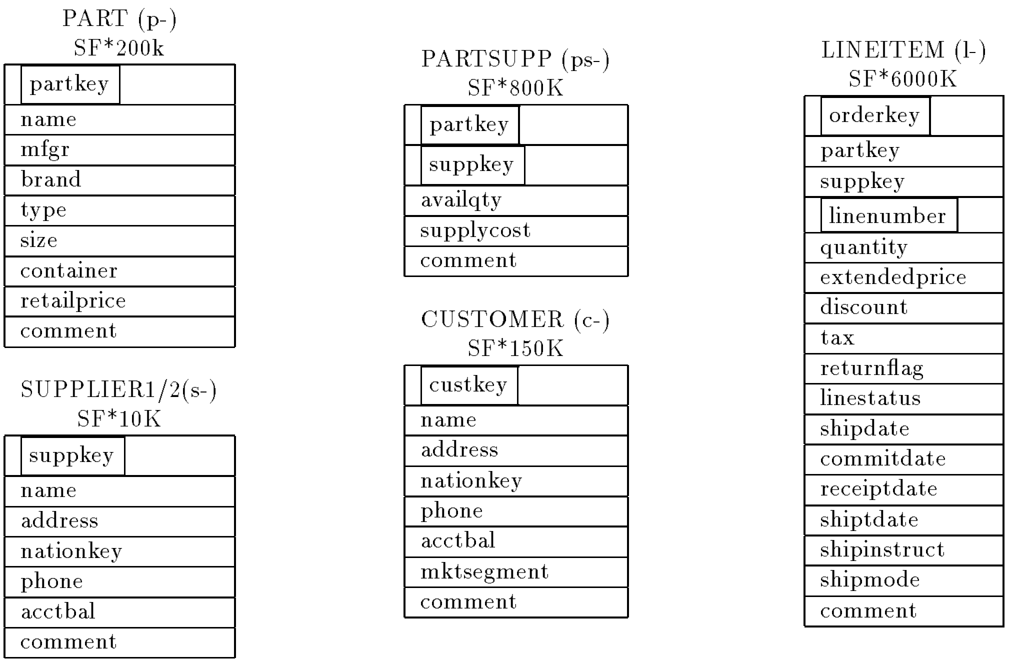


Figure 11: The TPC-D Benchmark Database Schema.

variable enclosed in curly braces (i.e., $\{baln\}$) is the parameter we use to vary the view sizes by randomly choosing its values from a range of possible TPC-D values. The values for the parameter $\{baln\}$ in the predicates (SUPPLIER1.s-acctbal $>$ $\{baln\}$) and (SUPPLIER2.s-acctbal $>$ $\{baln\}$) are chosen from $[0...10,000.00]$. The relations SUPPLIER1 and SUPPLIER2 are defined like the relation SUPPLIER in Fig. 6.2 in terms of their schema type. The extent of the relation SUPPLIER2 is a subset of the relation SUPPLIER1, ranging in our testbed in size from 0% to 100% of SUPPLIER1 size. The tables used for each test are populated at a scale factor SF=0.1 totaling 100M of data. All the base tables are physically ordered by their primary keys.

```
CREATE  VIEW V1 AS
SELECT  s_suppkey, s_name,
        s_address, s_nationkey,
        s_acctbal, s_comment,
        ps_availqty, s_quantity
FROM    SUPPLIER1 S1,
        PARTSUPP P,
        LINEITEM L
WHERE   S1.s_acctbal > {baln} and
        S1.s_suppkey = P.ps_suppkey and
        S1.s_suppkey = L.l_suppkey and
        P.ps_partkey = L.l_partkey
```

SYNCMA$\underset{\Longrightarrow}{\text{B}}$ - Case 1

SYNCMA$\underset{\Longleftarrow}{\text{B}}$ - Case 3

```
CREATE  VIEW V2 AS
SELECT  s_suppkey, s_name,
        s_address, s_nationkey,
        s_acctbal, s_comment,
        ps_availqty, s_quantity
FROM    SUPPLIER2 S2,
        PARTSUPP P,
        LINEITEM L
WHERE   S2.s_acctbal > {baln} and
        S2.s_suppkey = P.ps_suppkey and
        S2.s_suppkey = L.l_suppkey and
        P.ps_partkey = L.l_partkey
```

Figure 12: Cases 1 and 3 defined in Table 2 for SPJ Views: View Definitions for $V1$ and $V2$.

We use the test cases depicted in Fig. 12 to measure the performance of our maintenance techniques when the view to be synchronized is an SPJ view and the $\mathcal{PC}$-constraint is a subset (Case 1.2), equivalent (Cases 1.1 and 3.1) or superset (Case 3.2). In our scenarios, the $\mathcal{PC}$-constraint is defined as SUPPLIER1 $\supseteq$ SUPPLIER2. We describe the results obtained in these tests in Sections 6.4 and 6.5.

For the views $V3$ and $V4$ shown in Fig. 13 that contain aggregate attributes, we measure the performance

```
CREATE    VIEW V3 AS
SELECT    s_suppkey, s_name,
          s_address, s_nationkey,
          s_acctbal, s_comment,
          ps_availqty, SUM(s_quantity)
FROM      SUPPLIER1 S1,
          PARTSUPP P,
          LINEITEM L
WHERE     S1.s_acctbal > {baln}
          S1.s_suppkey = P.ps_suppkey
          S1.s_suppkey = L.l_suppkey
          P.ps_partkey = L.l_partkey
GROUP BY  s_suppkey, s_name,
          s_address, s_nationkey,
          s_acctbal, s_comment,
          ps_availqty
```

$\xRightarrow{\text{SYNCMAB}}$ - Case 5

$\xLeftarrow{\text{SYNCMAB}}$ - Case 7

```
CREATE    VIEW V4 AS
SELECT    s_suppkey, s_name,
          s_address, s_nationkey,
          s_acctbal, s_comment,
          ps_availqty, SUM(s_quantity)
FROM      SUPPLIER2 S2,
          PARTSUPP P,
          LINEITEM L
WHERE     S2.s_acctbal > {baln}
          S2.s_suppkey = P.ps_suppkey
          S2.s_suppkey = L.l_suppkey
          P.ps_partkey = L.l_partkey
GROUP BY  s_suppkey, s_name,
          s_address, s_nationkey,
          s_acctbal, s_comment,
          ps_availqty
```

Figure 13: Cases 5 and 7 defined in Table 2 for Aggregation Views: View Definitions for $V3$ and $V4$.

of our maintenance techniques again for different types of the $\mathcal{PC}$-constraint expressed as a subset (Case 5) or superset (Case 7). The experimental results are showed in Figs. 18 and 19 for Case 5, and 20 and 21 for Case 7.

6.4 Maintenance after Synchronization for Views with Deletion

The charts in Figs. 14 and 15 summarize our results for Cases 1.1 and 1.2 for different sizes of the original view $V1$ and the synchronized view $V2$ and for different sizes of the to-be-deleted relation SUPPLIER1 and its replacement SUPPLIER2. In these cases only deletion is performed by our SYNCMAB strategy (see Fig. 7). Given that the keys of the relations SUPPLIER1 and SUPPLIER2 are preserved in the views $V1$ and $V2$, respectively, we expect the deletion operation performed by SYNCMAB strategy to be very cheap and hence our technique is expected to outperform the other two methods.

The graphs in Fig. 14 are derived by varying the $\{baln\}$ parameter and thus controlling the view sizes to range from 0 to $10,000.00$ tuples. The x-axis values correspond to the fraction of tuples from SUPPLIER1 participating (i.e., generating tuples) in the view $V1$, while the y-axis indicates the elapsed time measured in seconds. Each chart corresponds to a different setting of the difference size between the to-be-deleted relation SUPPLIER1 and its replacement SUPPLIER2 ranging from SUPPLIER2 = 100% SUPPLIER1 (in the first chart) to SUPPLIER2 = 20% SUPPLIER1 (in the last chart). For each chart, the elapsed time increases proportionally with the percent of the relation SUPPLIER1 generating tuples in the view $V1$ corresponding to the x-axis values. In our setting, these percents directly influence the size of the original view $V1$ as well as the new synchronized view $V2$. This explains why the elapsed time for all three techniques increase to the right when the sizes of the two views $V1$ and $V2$ are increasing.

The SYNCMAB strategy, Case 1, in this scenario when the new view $V2$ is a subset of the original view $V1$, computes ΔR^- and only deletes tuples from $V1$ to find the extent of $V2$. The keys of the relations SUPPLIER1 and SUPPLIER2 are preserved in the views $V1$ and $V2$, respectively, and hence the deletion operation is very efficient once the set ΔR^- is computed. By contrast, the other two methods have to recompute all tuples of the view $V2$ which explains why their elapsed times are almost the same for all measurements. The results in all charts show a clear win of more than 400% of the SYNCMAB method compared to the other two methods.

In Fig. 15 we have the same experimental results represented in graphs from Fig. 14, depicted now with the x-axis representing the size of the difference between SUPPLIER1 and SUPPLIER2. This time, we plot together results corresponding to two cases: $V1 \equiv V2$ (Case 1.1 of the SYNCMAB strategy) and $V2 \subset V1$ (Case 1.2 of the SYNCMAB strategy). The upper chart depicts the results of the experiments falling in the Case 1.1, (i.e., $V1 \equiv V2$). We can see that in this case the SYNCMAB strategy will only compute the empty set ΔR^- and hence no deletion is performed. The SYNCMAB strategy accesses only relations SUPPLIER1 and SUPPLIER2 in order to compute the set ΔR^-. On the other hand, the other two methods recompute everything from scratch, being completely insensitive that the difference between the old and the new views is an empty set. These different techniques of computing $V2$ justify that the elapsed time for the SYNCMAB strategy is significantly below the times measured for both rematerialization and redefinition methods. While our method computes only the difference between the old and new extents and hence, reuse the old extent, the other two strategies recompute tuples that were available in the old extent.

The bottom chart corresponds to Case 1.2 of the SYNCMAB strategy when the view $V2$ is a proper subset of the original view $V1$. The SYNCMAB method needs to compute the non-empty set ΔR^- and use this set to delete appropriate tuples from $V1$. The deletion is very efficient when the keys of the relations SUPPLIER1 and SUPPLIER2 are preserved in the old and new views, respectively. The performance of the SYNCMAB strategy for this case is below the performance obtained for the SYNCMAB strategy for Case 1.1 (upper chart), but it still exceeds the other two methods by a wide margin of up to 400%.

6.5 Maintenance after Synchronization for Views with Insertion

The charts in Figs. 16 and 17 summarize our results for Cases 3.1 and 3.2 when the sizes of original and rewritten views are varied as well as the sizes of the to-be-deleted relation SUPPLIER2 and its replacement SUPPLIER1. In both Figs. 16 and 17, the x-axis and y-axis are defined as explained above in Section 6.4.

Our SYNCMAB strategy must perform insertion under capability change "*delete-relation* SUPPLIER2" (see Fig. 7). The first step of the SYNCMAB technique for Case 3 is to compute the expression ΔR^+ and then propagate these insertions into the new view $V1$. We expect the SYNCMAB strategy to perform very well when the size of ΔR^+ is relatively small which implies the number of tuples generated by ΔR^+ in $V1$ is smaller than the size of $V1$. But when the size of the expression ΔR^+ is close to the size of the replacement relation SUPPLIER1, the SYNCMAB strategy practically computes the view $V1$ from scratch when propagating these insertions into $V1$. In this latter case, we expect the performance of our strategy to be close to the performance of the rematerialization strategy.

The charts in Fig. 16 show that the SYNCMAB algorithm begins to perform not so well when the difference between the deleted relation SUPPLIER2 and the new replacement relation SUPPLIER1 is increasing and hence a bigger set of new tuples has to be computed for the new extent (i.e., ΔR^+ is increasing). The increased elapsed time of the SYNCMAB strategy toward the left side of the charts correspond to bigger sizes of the expression ΔR^+. All charts show a crossover point where the SYNCMAB strategy that first takes advantage of the expression ΔR^+ being empty set or relatively small, becomes inferior to the other two methods. The inferior performance on the left side of the charts is explained by the fact that the SYNCMAB strategy does the same amount of computation the rematerialization, for example, is doing *plus* computing first the relation ΔR^+ (see Fig. 7). Note that even when the differences between the relations SUPPLIER1 and SUPPLIER2 are big, our algorithm is still doing very well when ΔR^+ is small (the right side of the chats in Fig. 16).

In Fig. 17 we have the same experimental results represented in graphs from Fig. 16, depicted now with x-axis representing the size of the difference between SUPPLIER1 and SUPPLIER2. The upper chart depicts the results of the experiments falling in the Case 3.1, (i.e., $V1 \equiv V2$) corresponding to the expression ΔR^+ being empty set and hence the SYNCMAB strategy is not performing any insertion. The SYNCMAB strategy in this case accesses only relations SUPPLIER1 and SUPPLIER2 in order to compute the set ΔR^+ while the other two methods recompute everything from scratch. These computations justify that the elapsed time for the SYNCMAB strategy is significantly below the times measured for both rematerialization and redefinition methods.

The bottom chart corresponds to Case 3.2 of the SYNCMAB strategy when the rewritten view $V1$ is a proper superset of the original view $V2$. The SYNCMAB method needs to compute the non-empty set ΔR^+ and use this set to insert appropriate tuples into the view $V1$ that are not in $V2$. The insertion is done by computing these new tuples with the elapsed time being directly influenced by the size of the set ΔR^+. The performance of the SYNCMAB strategy for this case is then comparable to the other two methods when the size of the relation ΔR^+ is closed to the size of the replacement relation SUPPLIER1.

6.6 Discussion of the Experimental Results

The experiments presented in this section demonstrate a number of interesting points. First, they show that in all the cases the knowledge of how the rewritten view is obtained by the synchronization process can be exploited resulting in big performance gains by the SYNCMAB strategy. This information is the most valuable when the new rewriting view is equivalent to the original view and hence the SYNCMAB strategy performs very little computation to find the new view extent accessing only the to-be-deleted relation and its replacement. The gains in these cases are over 400% over the other two methods in our experimental setting. When deletion and insertion must be performed, the SYNCMAB strategy still outperforms the other two methods when the sizes of the ΔR^- and ΔR^+ are smaller relative to the size of the replacement relation. In these cases, the SYNCMAB strategy brings the new view extent up to date with minimum amount of work as the only expensive computation is directly influenced by the sizes of ΔR^- or ΔR^+. In the worst case scenario, when the size of ΔR^+ is close to the size of the replacement relation, the SYNCMAB strategy must perform the same amount of work as other methods for propagating insertions *plus* the extra computation for finding the expression ΔR^+ to begin with. In our experimental setting, the worst performance the SYNCMAB strategy displays is 115% of the performance of the rematerialization techniques. The 15% corresponds to extra time spent by the SYNCMAB strategy for computing the set ΔR^+. This penalty of 15% can be considered by an optimizer for assessing different maintenance strategies to be applied when the new view extent contains new tuples. In the case when the size of ΔR^+ is close to the size of the replacement relation, rematerialization should be preferred. When the deletion must be performed, the SYNCMAB strategy could be very efficient if keys are preserved in the view definitions.

7 Conclusions

While our previous work studied the problem of adapting view definitions themselves in order to keep views up and running in a dynamic environment [RLN97, LNR97a, NLR98, LKNR98, NR98c], in the present paper we address a new view extent adaptation problem.

The main contributions of this paper are:

- We describe a new view adaptation problem arising from view synchronization under capability changes, which we coin *view maintenance after view synchronization.*
- We distinguish this new problem from the explicit view redefinition problem, and demonstrate that the existing techniques for view redefinition from the literature [GMR95, MD96] would fail in several cases when applied to this new maintenance problem.
- We propose a series of maintenance strategies based on different system parameters such as the availability of base relations and the types of the view-extent evolution parameter. The proposed solution regards the complex changes done to the view definition by the view synchronizer as one atomic change and thus attempts to handle it in one optimized batch strategy. We exploit knowledge of how the view definition was synchronized, especially the containment information between the old and new views, to achieve efficient view adaption.
- We show via examples that our solutions work when other would fail and, in general, the view maintenance strategies proposed in this work are more efficient than previously known approaches for this problem of maintenance after view synchronization.
- We have performed a systematic experimental evaluation of our proposed algorithms. The experiment compares our techniques with rematerialization and redefinition strategies using the TPC-D benchmark [TD95]. Our experimental results demonstrate that we achieve a performance gain of approximately 400% when the difference between the old and new extents is fairly small while being comparable with these alternate techniques in all other scenarios.

In this work we have presented techniques for view maintenance after synchronization. The proposed strategies are however general enough to be used in other frameworks where complex (simultaneous) view redefinitions are *specified by the user* and *directly applied to the view definition.*

The work presented here has open a large spectrum of new problems to be solved, in general, for view adaptation under capability changes, and, in particular, for view synchronization and view maintenance after synchronization problems as defined in this paper. A most interesting problem is combining the process of view synchronization with data updates as it is very likely a data warehouse system to have to process capability changes and data changes in the same time. Extensive research has been done in the context of data warehouse maintenance under *simultaneous data updates* at distributed sources [ZGMHW95, ZWGM97]. We believe that more complex strategies must be designed to deal with the adaptation of the data warehouse under *simultaneous capability changes* and *data updates* of the underlying information sources [ZR98].

References

[AAS97] D. Agrawal, A. El Abbadi, and A. Singh. Efficient View Maintenance at Data Warehouses. In *Proceedings of SIGMOD*, pages 417–427, 1997.

[BLT86] J. A. Blakeley, P.-E. Larson, and F. W. Tompa. Efficiently Updating Materialized Views. *Proceedings of SIGMOD*, pages 61–71, 1986.

[GM95] A. Gupta and I.S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Warehousing*, 18(2):3–19, 1995.

[GMR95] A. Gupta, I.S. Mumick, and K.A. Ross. Adapting Materialized Views after Redefinition. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 211–222, 1995.

[LKNR98] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Data Warehouse Evolution: Trade-offs between Quality and Cost of Query Rewritings. Technical Report WPI-CS-TR-98-2, Worcester Polytechnic Institute, Dept. of Computer Science, 1998.

[LMS95] A.Y. Levy, A.O. Mendelzon, and Y. Sagiv. Answering Queries Using Views. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 95–104, May 1995.

[LNR97a] A. J. Lee, A. Nica, and E. A. Rundensteiner. Keeping Virtual Information Resources Up and Running. In *Proceedings of IBM Centre for Advanced Studies Conference CASCON97, Best Paper Award*, pages 1–14, November 1997.

[LNR97b] A. J. Lee, A. Nica, and E. A. Rundensteiner. The EVE Framework: View Synchronization in Evolving Environments. Technical Report WPI-CS-TR-97-4, Worcester Polytechnic Institute, Dept. of Computer Science, 1997.

[LRU96] A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external processors. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 227–237, Montreal, Canada, 3–5 June 1996.

[MD96] M. Mohania and G. Dong. Algorithms for Adapting Materialized Views in Data Warehouses. *International Symposium on Cooperative Database Systems for Advanced Applications*, December 1996.

[Nic98] A. Nica. *View Evolution Support for Information Integration Systems over Dynamic Distributed Information Spaces*. PhD thesis, University of Michigan in Ann Arbor, 1998.

[NLR98] A. Nica, A. J. Lee, and E. A. Rundensteiner. The CVS Algorithm for View Synchronization in Evolvable Large-Scale Information Systems. In *Proceedings of International Conference on Extending Database Technology (EDBT'98)*, pages 359–373, Valencia, Spain, March 1998.

[NR98a] A. Nica and E. A. Rundensteiner. The POC and SPOC Algorithms: View Rewriting using Containment Constraints in *EVE*. Technical Report WPI-CS-TR-98-3, Worcester Polytechnic Institute, Dept. of Computer Science, 1998.

[NR98b] A. Nica and E. A. Rundensteiner. Using Complex Substitution Strategies for View Synchronization. Technical Report WPI-CS-TR-98-4, Worcester Polytechnic Institute, Dept. of Computer Scien ce, 1998.

[NR98c] A. Nica and E. A. Rundensteiner. Using Containment Information for View Evolution in Dynamic Distributed Environments. In *Proceedings of International Workshop on Data Warehouse Design and OLAP Technology (DWDOT'98)*, Vienna, Austria, August 1998.

[Qua96] D. Quass. Maintenance Expressions for Views with Aggregation. In *Proceedings of the Workshop on Materialized Views: Techniques and Applications*, June 1996.

[RKL+98] E. A. Rundensteiner, A. Koeller, A. Lee, Y. Li, A. Nica, and X. Zhang. Evolvable View Environment (*EVE*) Project: Synchronizing Views over Dynamic Distributed Information Sources. In *Demo Session Proceedings of International Conference on Extending Database Technology (EDBT'98)*, pages 41–42, Valencia, Spain, March 1998.

[RLN97] E. A. Rundensteiner, A. J. Lee, and A. Nica. On Preserving Views in Evolving Environments. In *Proceedings of 4th Int. Workshop on Knowledge Representation Meets Databases (KRDB'97): Intelligent Access to Heterogeneous Information*, pages 13.1–13.11, Athens, Greece, August 1997.

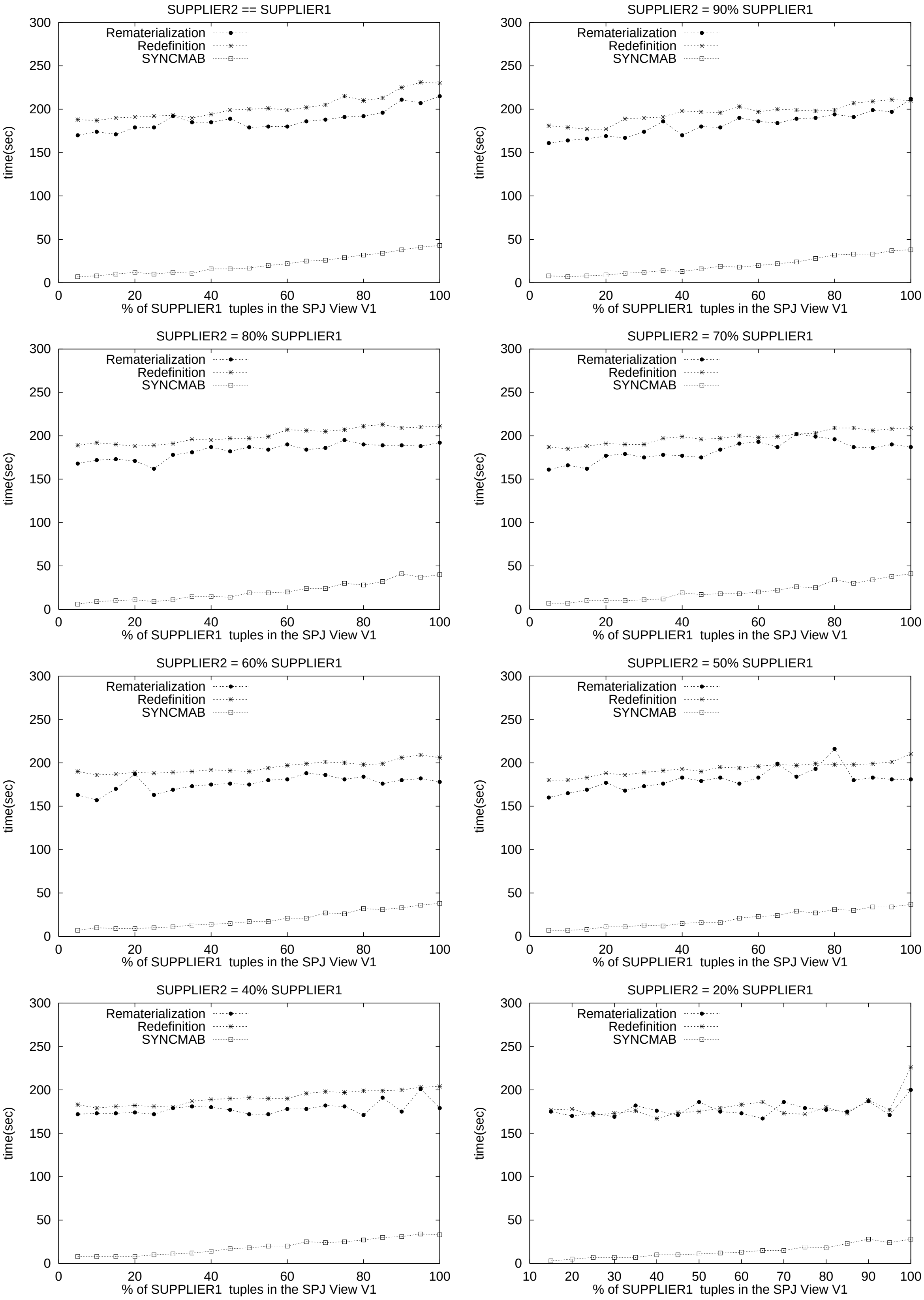
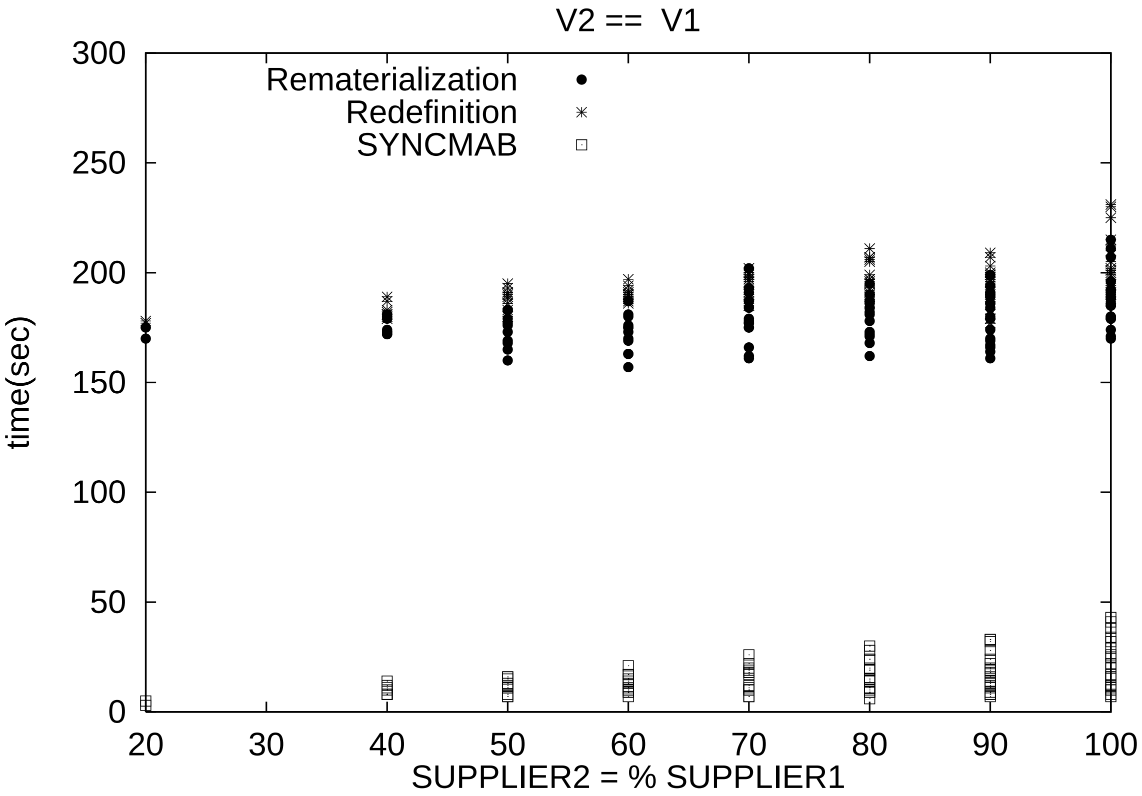
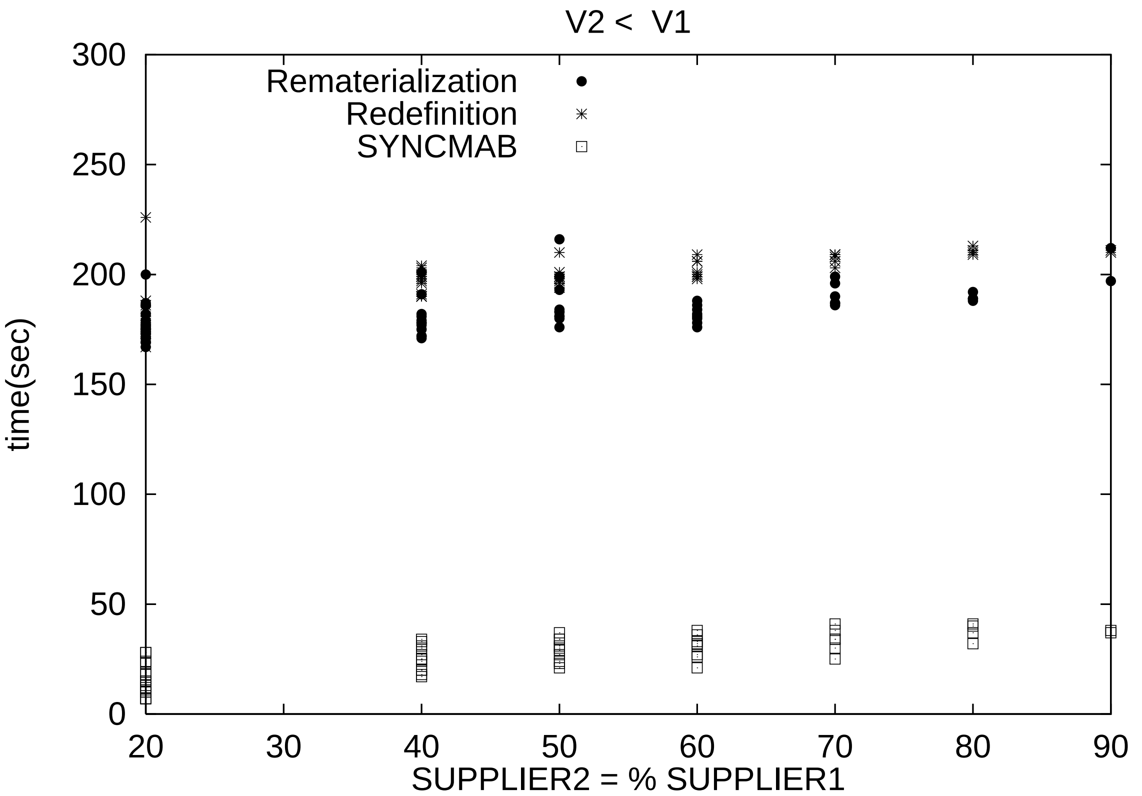


Figure 14: Case 1, for the SPJ view $V1$, the legal rewriting $V2$ when $V2 \subseteq V1$ using $\mathcal{PC}$-constraint SUPPLIER2 $\subseteq$ SUPPLIER1: the experimental results for maintenance after synchronization with *deletion* by varying the size of the SUPPLIER2 when SUPPLIER2 $\subseteq$ SUPPLIER1.



SYNCMAB strategy, Case 1.1: $V2 \equiv V1$



SYNCMAB strategy, Case 1.2: $V2 \subset V1$

Figure 15: Case 1, for the SPJ view $V1$, the legal rewriting $V2$ when $V2 \subseteq V1$ using $\mathcal{PC}$-constraint SUPPLIER2 $\subseteq$ SUPPLIER1: the experimental results for maintenance after synchronization with *deletion* by varying the difference between $V2$ and $V1$: Cases 1.1 (upper chart) and 1.2 (bottom chart).

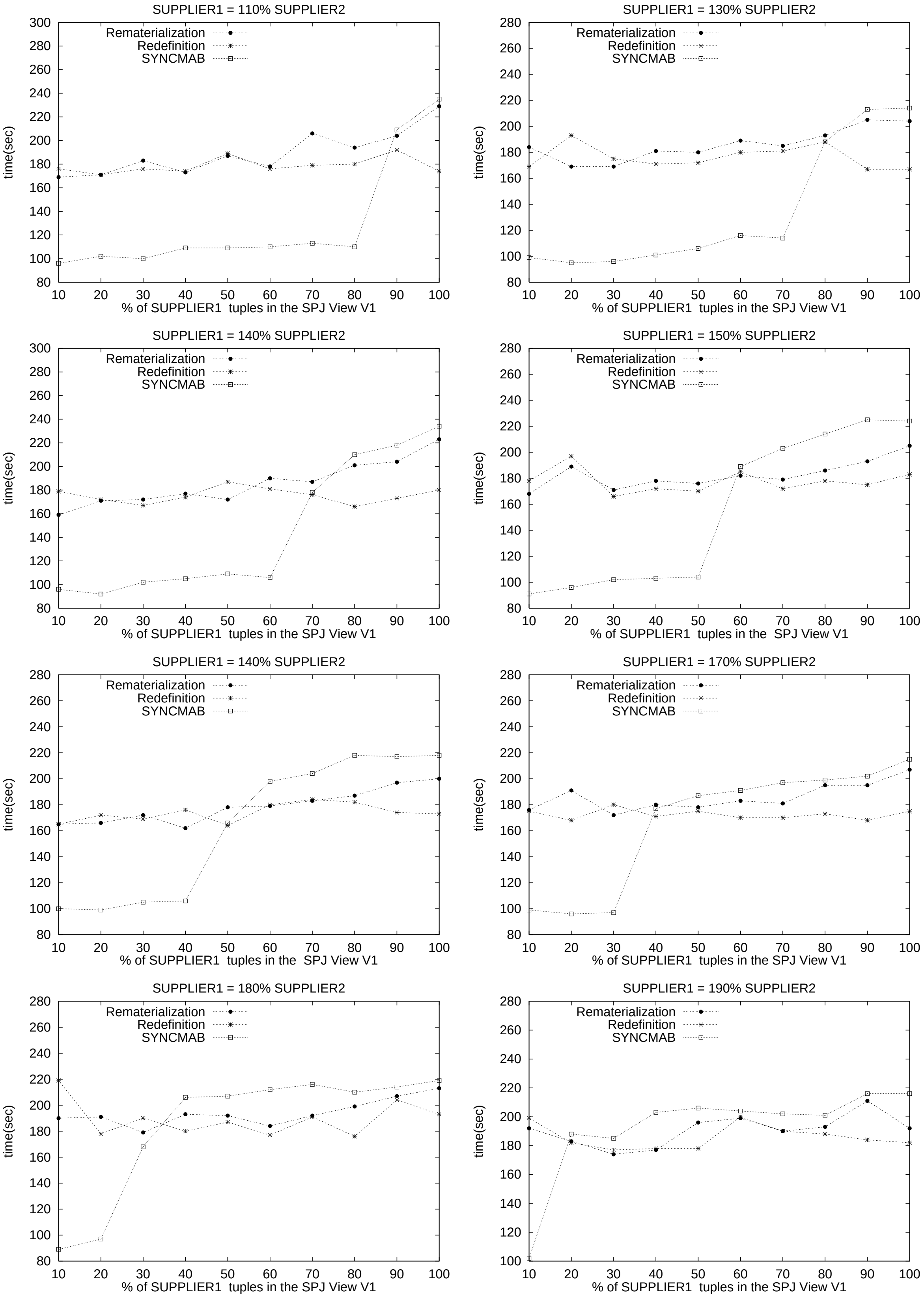
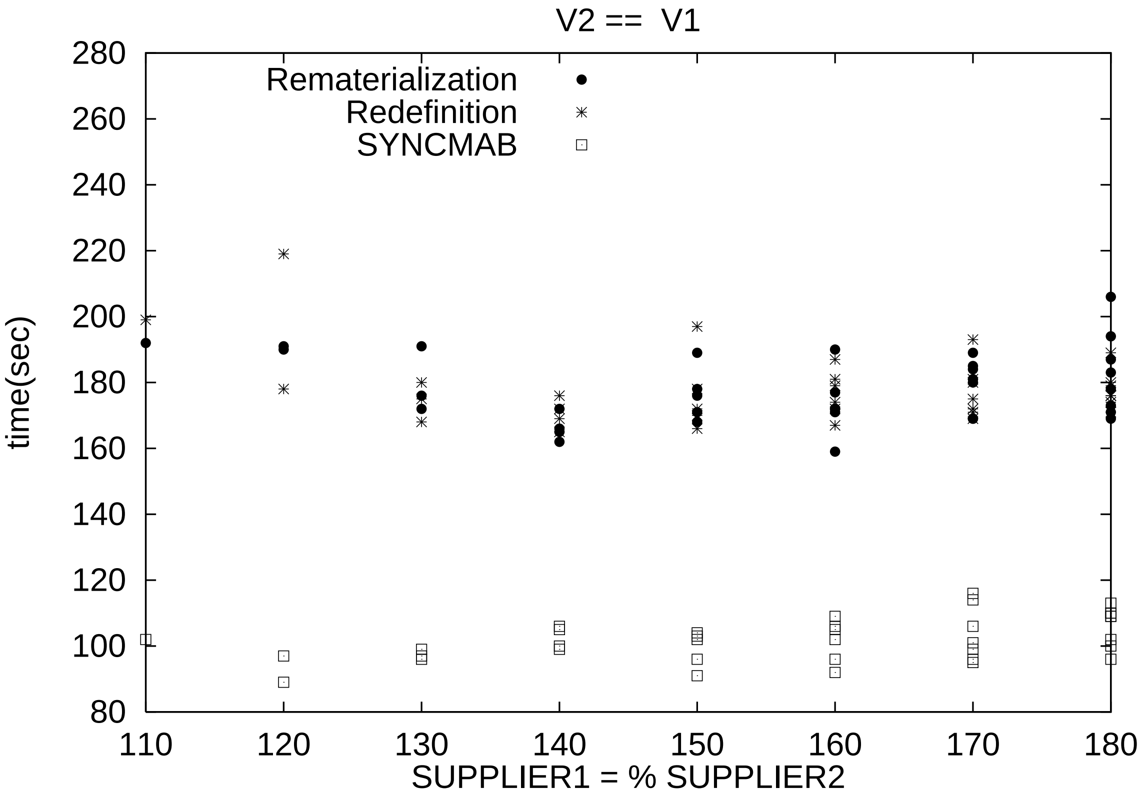
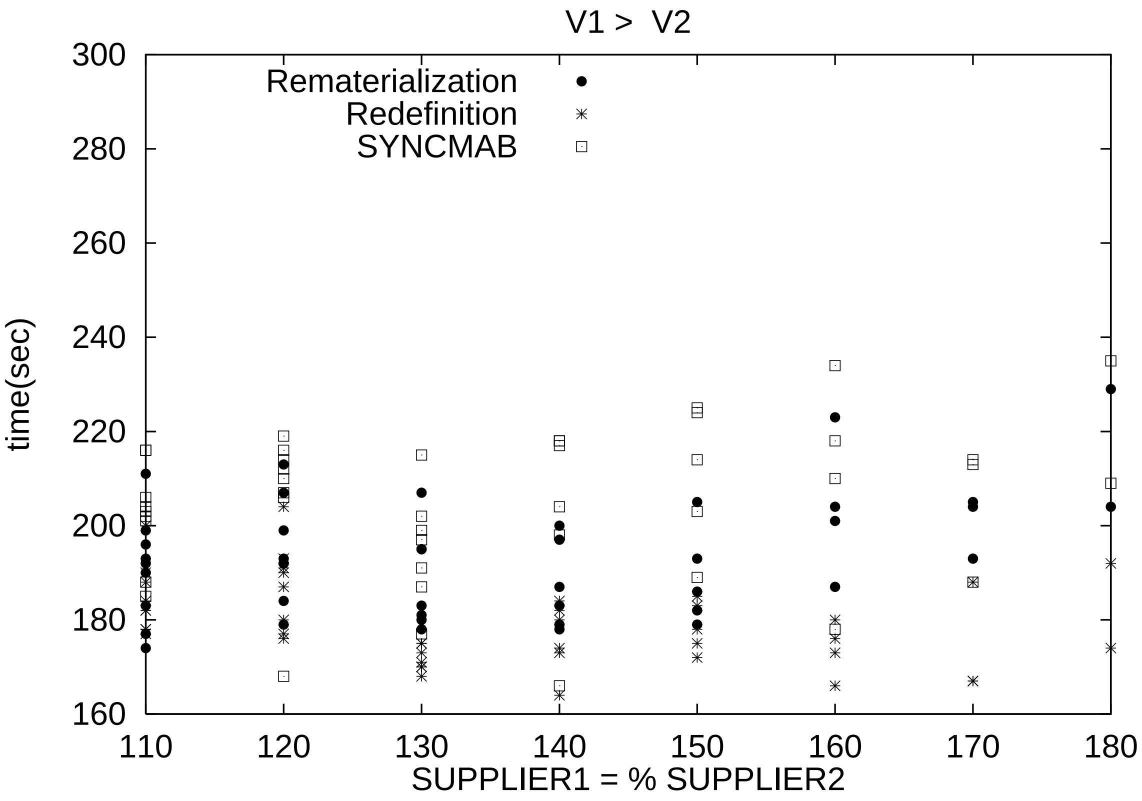


Figure 16: Case 3, for the SPJ view $V2$, the legal rewriting $V1$ when $V1 \supseteq V2$ using $\mathcal{PC}$-constraint SUPPLIER1 $\supseteq$ SUPPLIER2: the experimental results for maintenance after synchronization with *insertion* by varying the size of the SUPPLIER1 when SUPPLIER1 $\supseteq$ SUPPLIER2.



SYNCMAB strategy, Case 3.1: $V1 \equiv V2$



SYNCMAB strategy, Case 3.2: $V1 \supset V2$

Figure 17: Case 3, for the SPJ view $V2$, the legal rewriting $V1$ when $V1 \supseteq V2$ using $\mathcal{PC}$-constraint SUPPLIER1 $\supseteq$ SUPPLIER2: the experimental results for maintenance after synchronization with *insertion* by varying the difference between $V1$ and $V2$: Cases 3.1 (upper chart) and 3.2 (bottom chart)

.

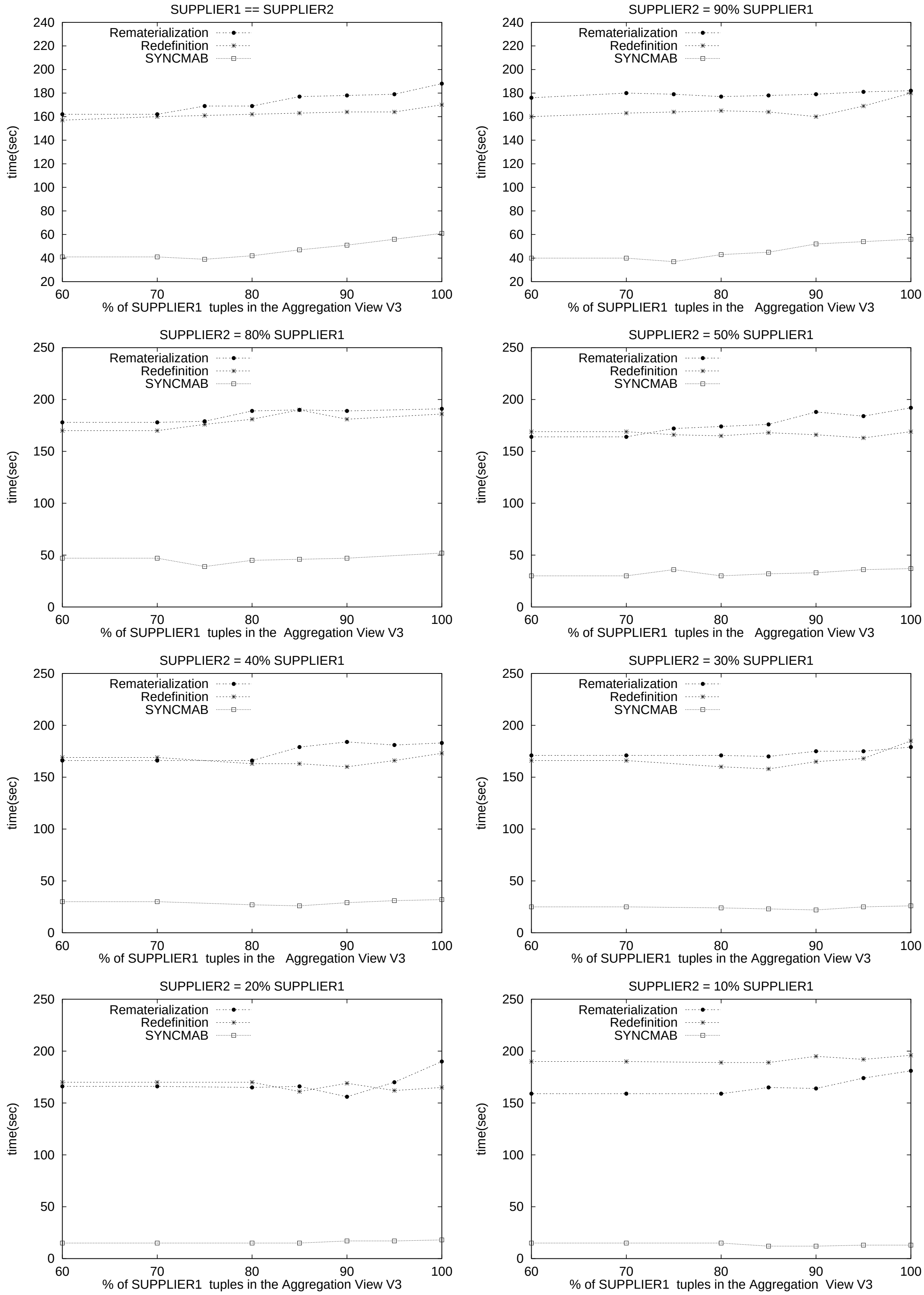
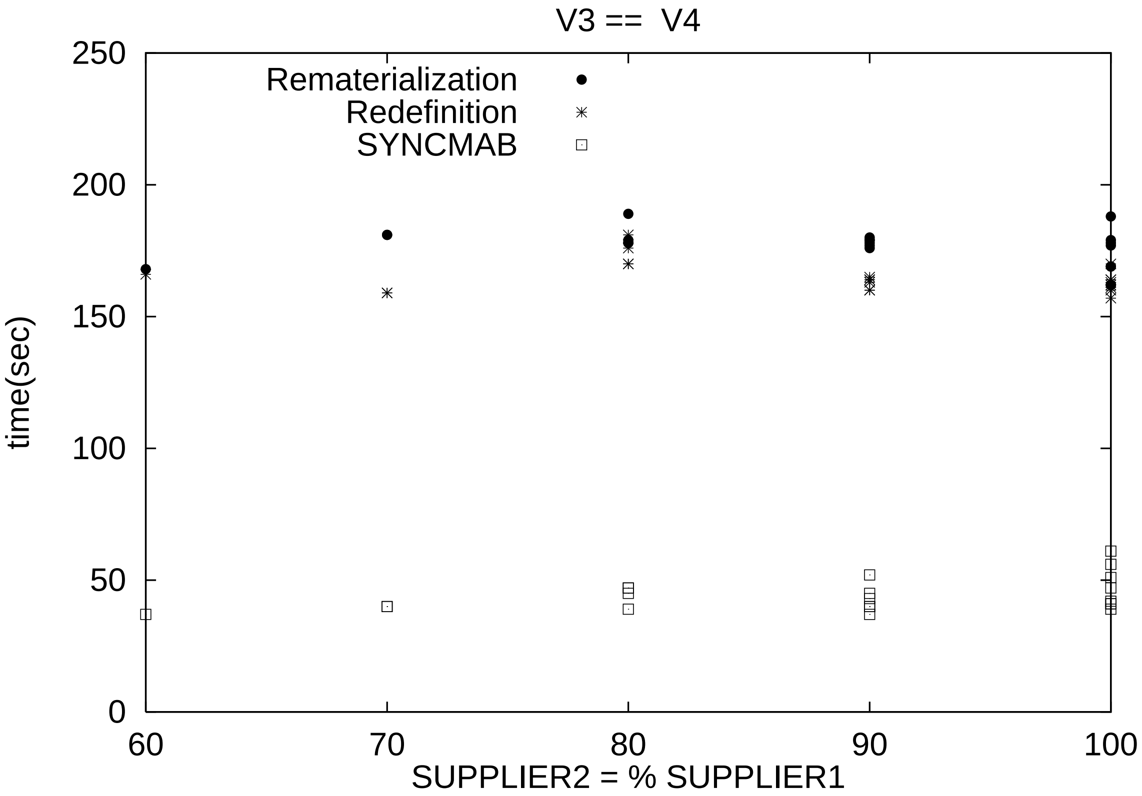
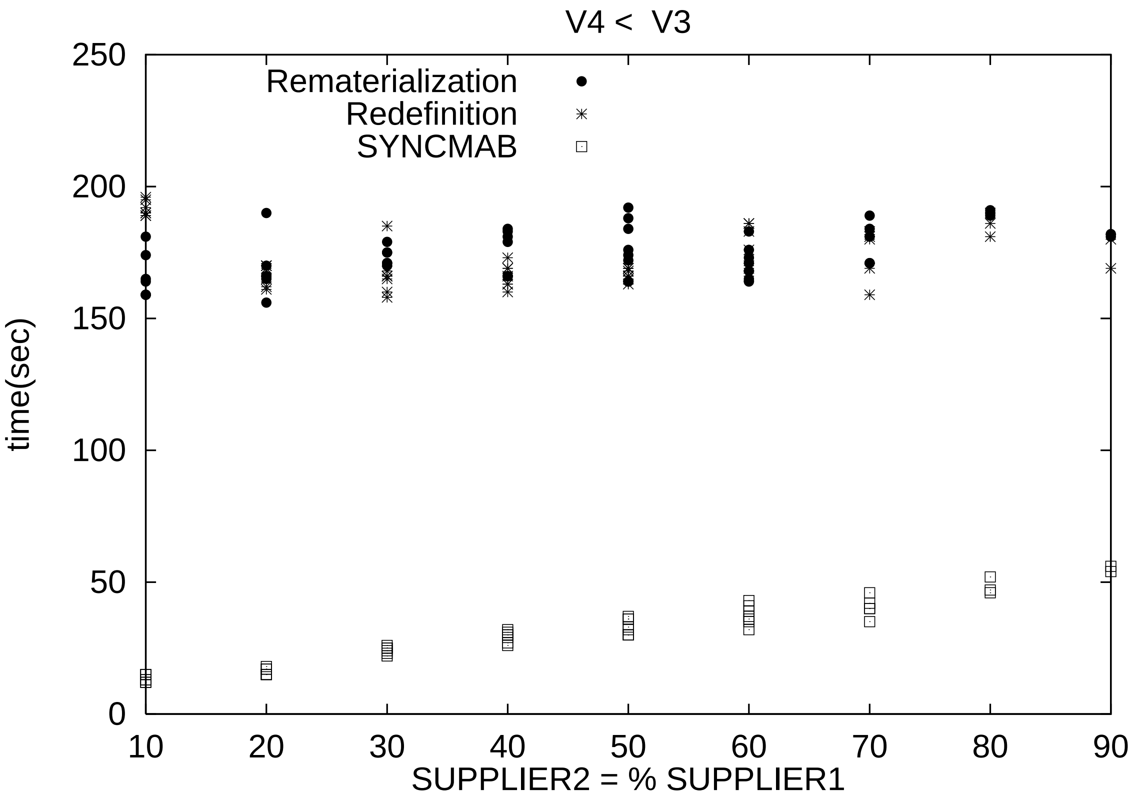


Figure 18: Case 5, for the AGGR-SPJ view $V3$, the legal rewriting $V4$ when $V4 \subseteq V3$ using $\mathcal{PC}$-constraint SUPPLIER2 $\subseteq$ SUPPLIER1: the experimental results for maintenance after synchronization with *deletion* by varying the size of the SUPPLIER2 when SUPPLIER2 $\subseteq$ SUPPLIER1.



SYNCMAB strategy, Case 5.1: $V3 \equiv V4$



SYNCMAB strategy, Case 5.2: $V3 \subset V4$

Figure 19: Case 5, for the AGGR-SPJ view $V3$, the legal rewriting $V4$ when $V4 \subseteq V3$ using $\mathcal{PC}$-constraint SUPPLIER2 $\subseteq$ SUPPLIER1: the experimental results for maintenance after synchronization with *deletion* by varying the difference between $V4$ and $V3$: Cases 5.1 (upper chart) and 5.2 (bottom chart).

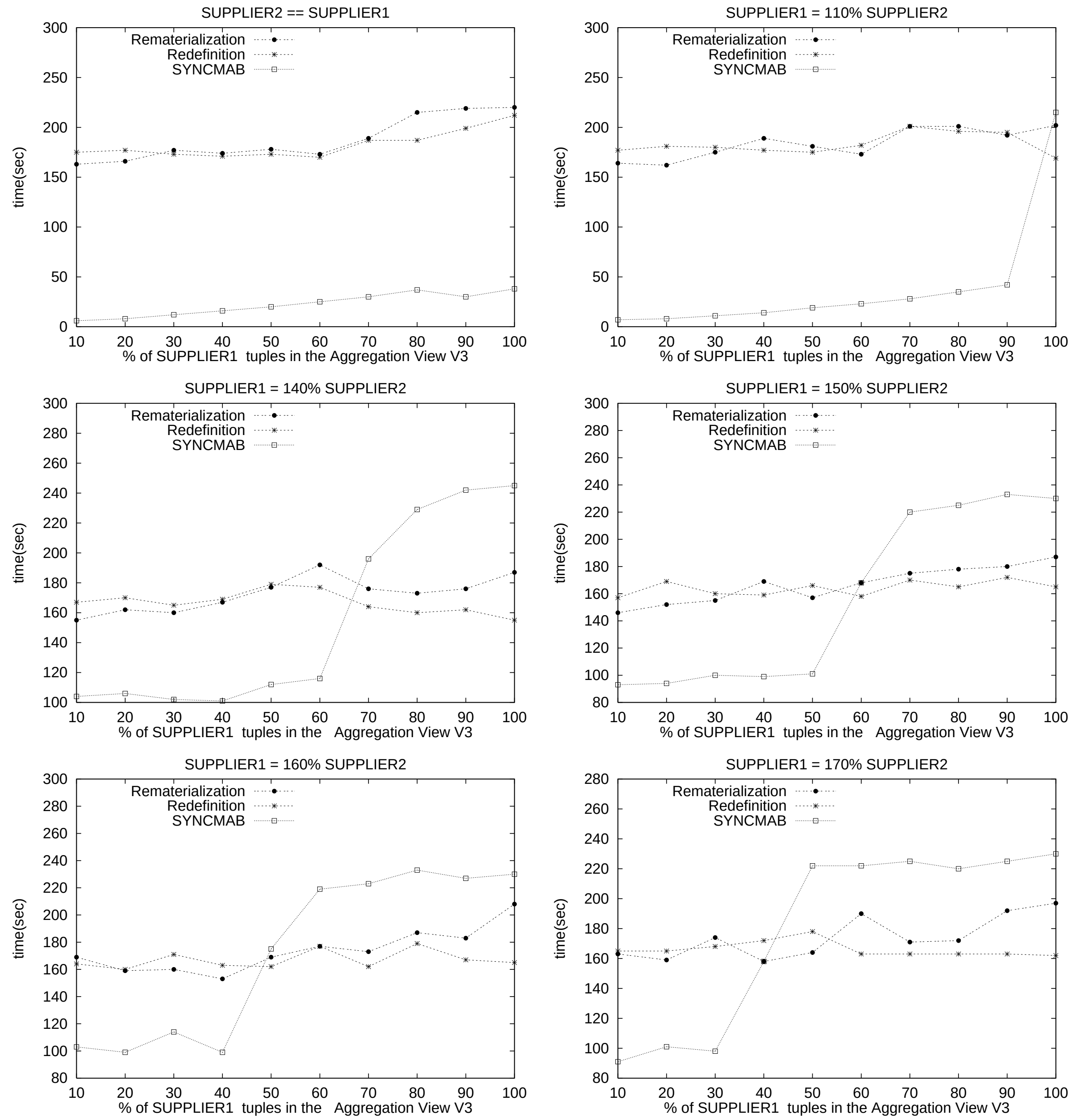
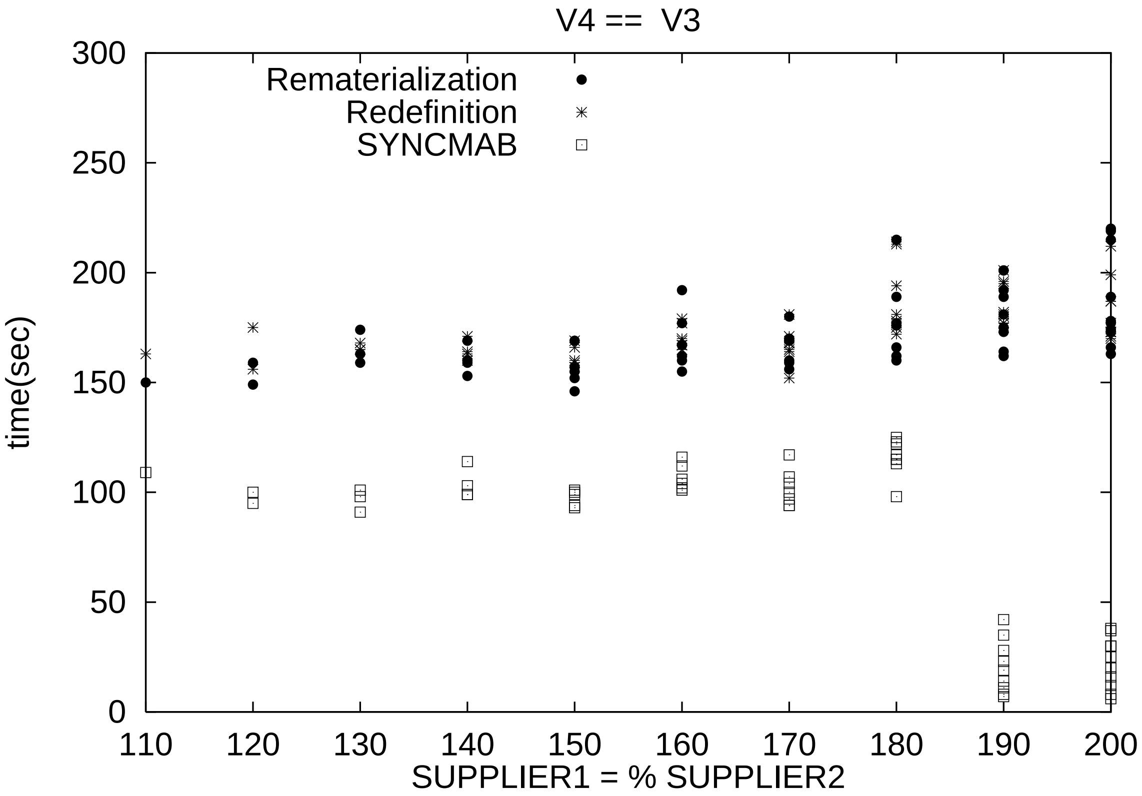
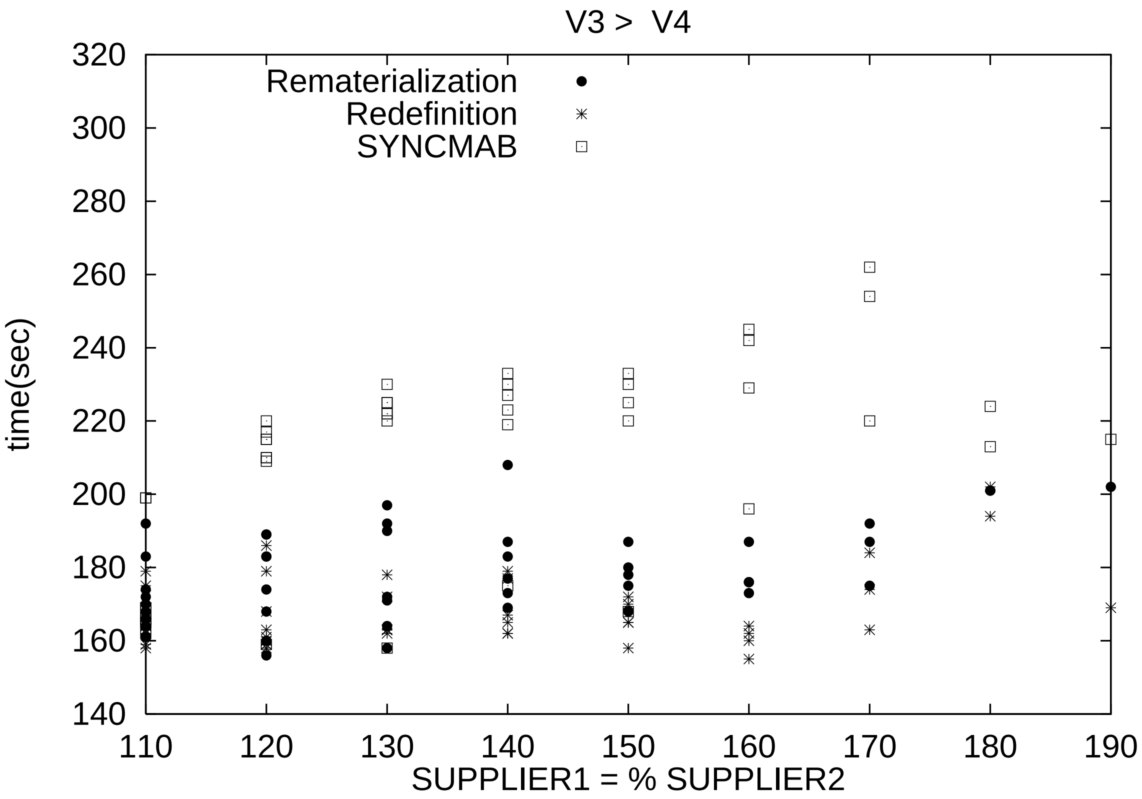


Figure 20: Case 7, for the AGGR-SPJ view $V4$, the legal rewriting $V3$ when $V3 \supseteq V4$ using $\mathcal{PC}$-constraint SUPPLIER1 $\supseteq$ SUPPLIER2: the experimental results for maintenance after synchronization with *insertion* by varying the size of the SUPPLIER1 when SUPPLIER1 $\supseteq$ SUPPLIER2.



SYNCMAB strategy, Case 7.1: $V3 \equiv V4$



SYNCMAB strategy, Case 7.2: $V3 \supset V4$

Figure 21: Case 7, for the AGGR-SPJ view $V4$, the legal rewriting $V3$ when $V4 \supseteq V3$ using $\mathcal{PC}$-constraint SUPPLIER1 $\supseteq$ SUPPLIER2: the experimental results for maintenance after synchronization with *insertion* by varying the difference between $V3$ and $V4$: Cases 7.1 (upper chart) and 7.2 (bottom chart).

[SDJL96] D. Srivastava, S. Dar, H.V. Jagadish, and A.Y. Levy. Answering Queries with Aggregation Using Views. In *International Conference on Very Large Data Bases*, pages 318–329, 1996.

[TD95] TPC-D. Benchmark standard specification. May 1995.

[Wid95] J. Widom. Research Problems in Data Warehousing. In *Proceedings of International Conference on Information and Knowledge Management*, pages 25–30, November 1995.

[ZGMHW95] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proceedings of SIGMOD*, pages 316–327, May 1995.

[ZR98] X. Zhang and E. A. Rundensteiner. Data Warehouse Maintenance Under Concurrent Schema and Data Updates. Technical report, Worcester Polytechnic Institute, Dept. of Computer Science, August 1998.

[ZWGM97] Y. Zhuge, J. L. Wiener, and H. Garcia-Molina. Multiple View Consistency for Data Warehousing. In *Proceedings of IEEE International Conference on Data Engineering*, pages 289–300, 1997.