VIEW EVOLUTION SUPPORT FOR INFORMATION INTEGRATION SYSTEMS OVER DYNAMIC DISTRIBUTED INFORMATION SPACES

by

Anisoara Nica

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1999

Doctoral Committee:

Associate Professor Elke A. Rundensteiner, Chair
Associate Professor Edmund H. Durfee
Associate Professor Atul Prakash
Professor Daniel Teichroew
Professor Toby J. Teorey

To my family for their loving support

ACKNOWLEDGEMENTS

I would like to thank my advisor, Elke A. Rundensteiner, for the valuable lessons she taught me in the last five years and for the guidelines in all the research work presented in this dissertation. My thanks also go to members of my dissertation committee: Professor Durfee, Professor Prakash, Professor Teichroew, and Professor Teorey for their support in the finalization on the dissertation. Thanks to many sponsors I had during my graduate studies: Digital Library Initiative project sponsored by NSF/ARPA/NASA under CERA IRI-9411287; NSF grant NYI #IRI-9457609 awarded to Professor Rundensteiner; and the Sloan foundation for the Sloan fellowship I had in the summer of 1997. I would like to thank the Software Systems Research Lab (SSRL) members at the University of Michigan for being so supportive to my work and studies during the last five years. Thanks to my student colleagues: Amy Lee from University of Michigan, Andreas Koeller and Xin Zhang from Worcester Polytechnic Institute for collaborating with me on the EVE project, which is the topic of this dissertation. I also thank to the members of the database group at the University of Waterloo, particularly Professor Tompa, Professor Salem and Glenn Paulley, for their warm hospitality during the year 1997-1998, which I spent visiting this university.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

Appendix

CHAPTER I

Introduction

Most approaches to information integration systems to date are based on static apriori defined views (or schemas) that gather information from heterogeneous information sources and provide the user with a uniform view of the information space for browsing and querying [3, 19, 12, 68, 35, 34, 63, 58, 20]. In this dissertation, we address issues related to the evolution and maintenance of such information integration systems, when the underlying information sources change their capabilities (e.g., schema level changes).

One domain of applicability of the information integration systems is provided by commercial applications, where there is a need for enterprises to gather all of their relevant information into a single place for in-depth analysis, and there is desire to decouple such analysis from on-line transaction processing systems. Analytical processing that involves very complex queries (often with aggregates) is one of the primary uses of such systems. Generally, information integration systems are based on customized integration of data across dynamic, highly heterogeneous large information spaces. Such applications could be found in very diverse domains ranging from business (e.g., trade-market analysis, etc), and libraries (e.g., museums, multimedia on-line resources) to science (e.g., decision support applications, integration of scientific results, etc.).

In this introductory chapter, a background to information integration systems is given first. Next, a discussion of the research problems in the general context of information integration and in the particular context of data warehousing is presented. The discussion identifies new largely unexplored issues. It is followed by an outline of the approach taken in this dissertation for solving some of these problems. The chapter concludes with a summary of the rest of the dissertation.

1.1 Background to Information Integration Systems

In recent years, more and more information has become available in on-line digital formats in Digital Libraries and the World Wide Web which, while offering unique opportunities to everyone for making use of this wealth of information, also introduces many new challenges to the database community. To name a few, these include:

- a large number of heterogeneous data sources and other service resources available in the environment,
- a continuously changing configuration of sources joining or leaving the system or changing their interfaces and content, and
- a wide diversity in data structures as well as query interfaces ranging from fixed APIs to declarative languages.

Typical users of these new systems are not database experts. Hence they are neither familiar with, nor willing to utilize the rigid interfaces and query languages offered by today's DBMS systems. Key challenges that thus need to be addressed are to allow such users to query the information space without having to:

- be aware of which information sources the data is located in;

- know the exact query language used by each source (and consequently need to learn multiple such languages);
- utilize structured query languages (such as SQL) requiring knowledge about the particular schema of the source accessed;
- deal with differences between structured and unstructured information sources;
- know how to rewrite a query into subqueries, each targeting only one information source; and
- know how to combine data from more than one source into one composite response.

The approaches to information search and retrieval range from one-information-source-at-a-time keyword search such as on-line search engines – e.g., Lycos, Yahoo! – to highly integrated federated multidatabase systems [63]. Lately, new approaches that combine these two extremes, such as Information Manifold [24], SIMS [3], GARLIC [19] and TSIMMIS [12, 20] are based on apriori defined global schemas or unified world views that allow heterogeneous information sources to join such systems by uniformly describing their capabilities against these schemas. These systems – denoted by the term *integration systems* – have the main goal of providing integrated access to *information* - data stored in standardized SQL databases, object repositories, knowledge bases, file systems, document retrieval systems, etc.

The solutions given by database research to the data integration problem could be classified into two categories: *on-demand* (or lazy) approaches and *data warehouse* (or eager) approaches. The two categories differ mostly in the strategies that integrated systems adopt to answer to queries posed to such systems.

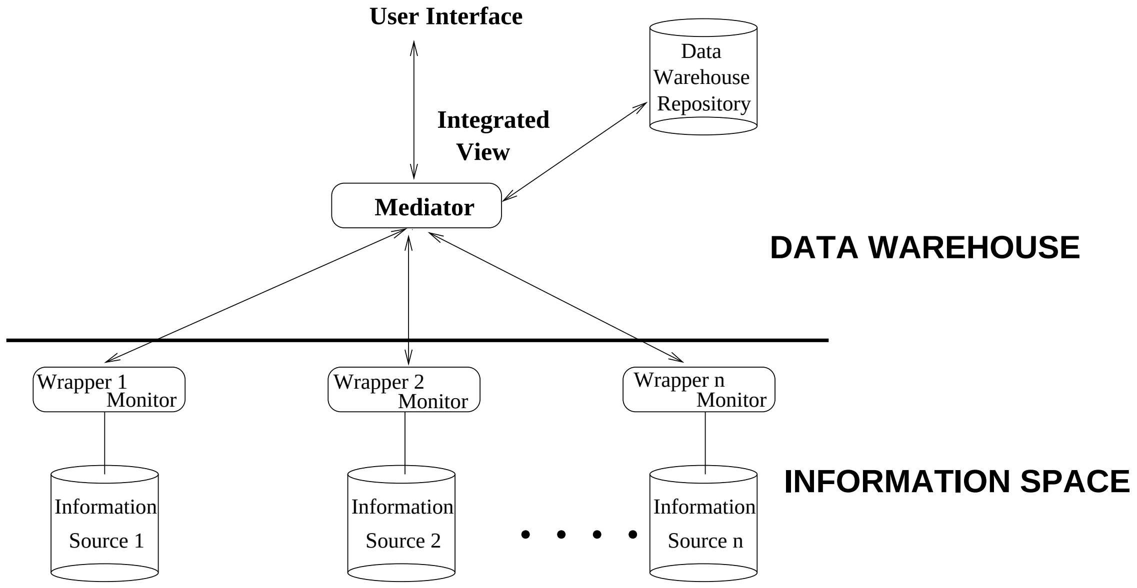


Figure 1.1: General Framework of Data Warehouse Systems.

The on-demand approach is generally based on the following two-step process:

Step 1. The system accepts a query, determines the appropriate set of information sources to answer the query, and generates the subqueries to be sent to each relevant information source.

Step 2. The system collects subquery results from the information sources, and presents the answer to the user (most probably after filtering and merging the partial results from different sources).

The on-demand solution to integration is thus suitable for querying information spaces that change rapidly, have a large amount of data per information source, or a large number of participating information sources. Also, if the diversity of client (users and applications) requests is large, the lazy approach is appropriate as each client is likely to be interested in a different portion of the data. However, query processing may be inefficient, especially when the same requests are issued multiple times, when information sources are slow to answer, expensive, or periodically unavailable, and significant processing is required for the second step in translation, filtering, merging, and shipping the results back to the clients.

An alternative to this on-demand approach is the data-warehouse (or eager) approach to data integration where the system "prepares" in advance for answering the user queries. The steps for this process are as follows:

Step 1. Information from each source that may be of interest is prepared (extracted, translated and filtered) in advance, organized in an integrated view with other relevant information, and stored (at the data warehouse site) in a repository.

Step 2. A user query is evaluated directly against the repository, without accessing the original information sources.

The main advantage of the warehousing approach is that the integrated information is available for immediate querying and analysis by clients. Thus, the warehousing approach is appropriate for applications requiring specific and predictable portions of the data warehouse repository, and have high query performance requirements (e.g., response time). On the other hand, the data presented to the user is not necessarily in the most recent state, i.e., the changes done at the information source sites may not always have been already propagated to the data warehouse repository.

There are many applications for which the data warehouse approach provides a suitable solution to information integration. These include enterprises that need to gather information into a single site used only for in-depth analysis or analytical processing that involves usually aggregation queries. Examples of such uses of data warehouses are data mining, on-line analytical processing (OLAP), and multi-dimensional analysis, all part of what is coined in the database literature as *decision support* applications.

1.1.1 General Architecture of Data Warehouses

In a typical integration architecture (Figure 1.1), an Information Source (IS) joins an information integration system via a wrapper that translates common-language queries coming from the system users into source-specific queries that are locally supported at the data warehouse site. The set of ISs may include non-traditional data sources such as flat files, HTML and SGML documents, knowledge bases, legacy systems, and traditional relational or object-oriented systems. The monitor module shown in Figure 1.1 is customized for each information system (IS) and is responsible for detecting and advertising to the data warehouse changes at the IS site both at the data level and at the schema level. When an information source joins or leaves the warehouse system or when some changes are reported to the data warehouse, the mediator is responsible for updating the information in the warehouse, which may include filtering and merging of retrieved data, or even querying other ISs during this process. Also, the mediator provides user access to the data warehouse repository by answering the queries posed from client applications.

The data warehouse architecture could be more complex, involving distribution or replication of the data warehouse repository for the purpose of increasing performance in both user response time and in the maintenance of the data warehouse itself.

Also for performance and cost reasons, hybrid systems that combine data warehouse and lazy approaches are an alternative solution to the information integration problem, which inherit some of the advantages but also disadvantages of both approaches (as mentioned above).

1.2 Research Challenges

Given the general architecture depicted in Figure 1.1, we outline in this section the main research issues raised by the warehousing approach [69] while in the next section (Section 1.2.1) we discuss in more detail one of them. Namely, we further characterize the *view synchronization* problem which is the topic of this dissertation.

1. Wrapper Related Issues. The translation between the data warehouse and IS specific query language, and detection of changes for a participating IS are issues that appear in any information integration system approach. They have been studied intensively in projects dealing with information integration [3, 19, 68, 58, 20]. The challenge is coming from the fact that we need different wrapper and monitor modules for each type of IS, as both components are dependent on data structure and capabilities for each participating IS. The research done in this direction is developing techniques and tools for automating the process of implementing such modules [68, 58]. In this dissertation we don't address issues related to wrapping and change detection but rather assume that change notification and wrapper modules are available to our system.

2. Mediator Related Issues. The mediator tasks include, in general, constant maintenance of the data warehouse repository under changes of the underlying information space (e.g., changes that are advertised by the monitors and wrappers of the ISs). A data warehouse is in a broad sense a set of materialized views over the base relations residing at decoupled IS sites. However, because the information space is assumed to be dynamic (e.g., ISs leaving and joining the system, changing their capabilities) and highly heterogeneous, conventional view maintenance techniques suitable to centralized database systems cannot be used in a data warehouse approach. The reasons,

to name a few, include the fact that ISs operate independently from the warehouse where the views themselves are kept, thus they are not willing and often not capable to participate in the maintenance process. Hence, maintenance strategies have to deal, for example, with the unavailability of locking capabilities or statistical information (for query optimization) that are used in conventional view maintenance techniques.

The most studied problem in this context is the maintenance problem after data updates (data changes in the underlying information sources) [22, 53, 38, 21, 54, 15, 71]. The research challenges here include the optimization of the data maintenance process itself (e.g., *self-maintainability* problem, *query optimization* problem) as well as optimization of the amount of data stored in the data warehouse [38, 21].

One largely unexplored problem related to the maintenance is warehouse evolution under capability changes. A mediator must be able to handle schema changes of the information sources, as well as the addition of new information sources and the removal of old ones. The focus of this dissertation is on addressing the issues related to the evolution of a data warehouse under capability changes. Given that this problem as of now is unexplored, we first set out to propose a taxonomy of view adaptation problems, which provides a context for our problem space. We then characterize the issues this dissertation will address.

1.2.1 A Taxonomy of View Adaptation Problems in Information Integration Systems

We discuss in more detail here one of the research issues identified above: view adaptation after information source changes.

Once a unified customized view of the information space is defined in an information integration system, the view has to be adapted – i.e., maintained up-to-date

– when the underlying information sources change their data or their capabilities. Often, such views are materialized – i.e., the integrated data is kept at the view site [69]. We refer to any process that changes the view definition or the view extent (in the case of materialized view data) as a *view adaptation* process. In the following we give a taxonomy of view adaptation problems in evolving environments, based upon types of changes that either a view or an information source could undergo. This taxonomy represents a suitable classification, based on which we can single out several view adaption problems. Some of these problems have already been studied in the research literature – e.g., materialized view maintenance [16, 22, 53, 71], and view redefinition [17, 37].

On the other hand, this taxonomy helps us identify a new view adaptation problem, which we call *view synchronization*. This problem corresponds to the process of view definition adaptation triggered by capability changes of ISs, and is the main topic of the present dissertation.

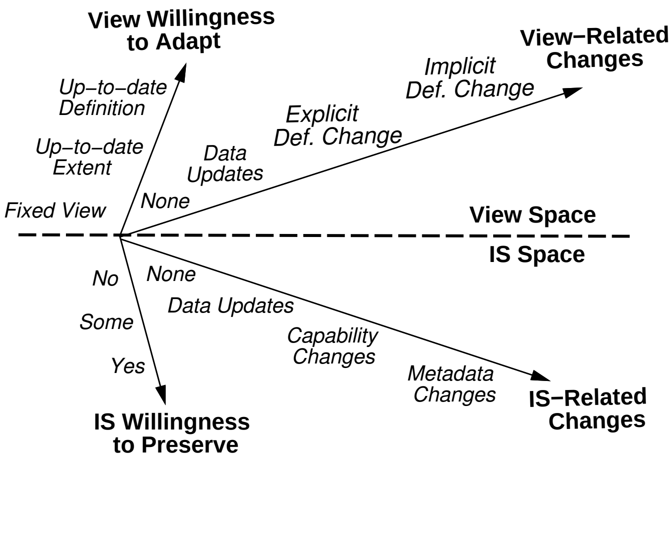


Figure 1.2: A Taxonomy of View Adaptation Problems.

1.2.2 Dimensions of the View Adaptation Space

We identify four dimensions that define the coordinates for different view adaptation problems (Figure 1.2):

IS-Related Changes dimension: There are three types of changes that can be made at the IS side, namely (1) changing the *metadata*, such as adding the fact that a relation R in IS_1 is contained in another relation S in IS_2, (2) changing the *capability* of an IS, such as deleting/adding an attribute or a relation, and (3) updating the (data) content of an IS, such as adding a new tuple to a relation.

IS Willingness to Preserve dimension: This dimension refers to the willingness of an IS to conform to the information needed by views defined on top of it. Possible values along this dimension correspond to the IS being willing and able to preserve *all, some*, or *not any* of its capabilities (either query interface, schema or even data content) referenced in a view V — denoted by *Yes*, *Some*, and *No* values on this dimension, respectively. If an IS for example opts to continue to preserve its data as it is in use by at least one of its derived views V, then V's data and/or definition would not need to be adapted after an IS change.

View-Related Changes dimension: Most previous work in the materialized view literature has focussed on processing changes at the (view) content level, either triggered by updates at the information site or by view updates specified directly through the view interface (*Data Updates* value on this dimension). However, similar to schema changes at an IS, capability changes also could take place for a view. Such definition changes could be either *explicitly requested* by the view maintainer, or they could be *implicitly triggered* due to other changes in the underlying environment.

View Willingness to Adapt dimension: This dimension considers the degree of desired flexibility in view adaptation in the context of changes, i.e., if the *view data* or *view definition* needs to be adapted after an IS-related or view-related change. This dimension refers to the criticality of different components of the view (e.g., view extent, view definition (VD), etc.) during the lifetime of the view, in order for the view to still be useful to the view user. One option (denoted by *Fixed View* value on this dimension) corresponds to not being able to handle any changes affecting the view - be it data or schema. The second option (denoted by *Up-to-date Extent* value on this dimension) corresponds to the view user's desire to maintain the *view extent* after any view-related or IS-related changes so that it always corresponds to the most up-to-date value set. The third option, which is the key focus of our work, corresponds to the view user's desire to adapt the *view definition* under ISs capability changes. This is denoted by *Up-to-date Definition* value on this dimension in Figure 1.2. The later represents a large range of options such as: (1) preserving the view interface and the original view extent while allowing the view definition to be changed, e.g., taking data from other sources, (2) preserving the view interface but accepting any view extent returned by the new view definition, (3) allowing some components to be dropped if no replacements can be found, versus (4) placing no constraints on the view adaptation process – which can survive any kind of capability changes.

1.2.3 Classification of View Adaptation Problems

We give below a **Classification of View Adaptation Problems** and introduce two new problems that are the focus of this dissertation, namely *view synchronization* and *materialized view maintenance after view synchronization.*

In Figure 1.2, we depict the four dimensions that we use to define several view adaptation problems characterized by specific combinations of the coordinates. A set of four coordinates, one from each dimension, corresponds to a set of **apriori** conditions for adaptation process (on either IS or data warehouse space) that leads to a specific view adaptation process.

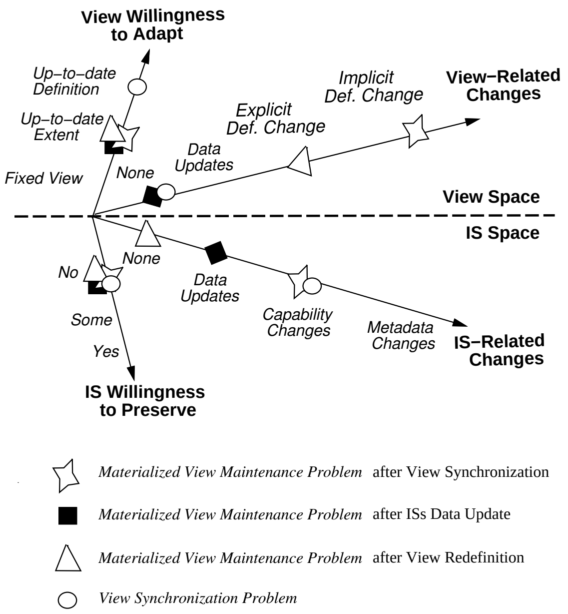


Figure 1.3: Problem Spaces in our View Adaptation Taxonomy.

Problem 1. *View Synchronization* is defined as a dynamic process that adapts *the view definition* triggered by IS-related capability changes. The coordinates that define this problem are marked by circles in Figure 1.3. One key distinction of this problem as compared to the others we have identified is that it is characterized by the *Up-to-date Definition* value on the **View Willingness to Adapt** dimension. This characterizes the fact that the view user desires for the *view definition* to adapt with changing IS capabilities instead of the view becoming undefined. The *None* value on the **View-Related Changes** dimension means that apriori no other change occurs on the view. The changes of the underlying ISs we consider are capability changes as indicated by the *Capability Changes* value on the **IS-Related Changes** dimension. No assumptions about the modified ISs being willing to preserve their old capabilities are being made, as indicated by the *No* value on the **IS Willingness to Preserve** dimension[1]

The rest of the adaptation problems depicted in Figure 1.3 are defined by a common view adaptability preference: maintaining the extent of the materialized view up-to-date as indicated on the **Willingness to Adapt** dimension.

<u>Problem 2.</u> The problem of *materialized view maintenance after base relation updates* is well studied in the database literature (see [16, 69] for an overview of problems and solutions). Its coordinates are marked by black squares in Figure 1.3. The problem is characterized by (1) the view has no explicit apriori changes, (2) the IS changes its data extent, (3) the view user wants the view extent up-to-date, and (4) the IS doesn't preserve its old data.

<u>Problem 3.</u> The problem of *materialized view maintenance after view redefinition* has been recently studied by Gupta et al. [17] and by Mohania et al. [37]. Its coordinates are marked by triangles in Figure 1.3. The main difference to <u>Problem 1.</u> is that the process is (1) triggered by the view's explicit change of its definition, and (2) *no* changes at the IS side occurred.

<u>Problem 4.</u> The fourth problem, named by us *materialized view maintenance after view synchronization*, is one of the problem this dissertation tackled and it has to our knowledge never been explored before (marked by stars in Figure 1.3). When the view definition is adapted to the IS capability changes, the view extent also needs to be updated in response to view synchronization. The apriori conditions defining this new view maintenance problem are: (1) the view definition was synchronized, i.e., its definition was *im-*

[1] For this to be possible, the evolution preferences for the desired view adaptability properties for each view element under capability changes must be incorporated into the view definition.

plicitly changed by the view synchronization process, (2) the IS changes its capability (this information is needed by the process of view maintenance in order to know what exactly was the change that triggers it), (3) the view user wants the view extent up-to-date, and (4) the IS doesn't preserve its old data or schema. The distinguishing characteristic of this maintenance problem over the other two (Problem 1 and Problem 2) is that potentially both ISs capabilities and data are changed (e.g., after the attribute A of relation R is deleted from the schema), relation R's data is changed as well, that is the column corresponding to attribute A is deleted from R, and hence both are not accessible for maintaining the view derived from it).

1.3 Dissertation Focus

This dissertation addresses one of the unexplored issues that information integration systems raise, namely support for the evolution and data maintenance of data warehouses when the underlying information sources change their *capabilities* (e.g., schema level changes). More precisely, the focus on the dissertation is on the Problem 1 and Problem 4 presented above in the Section 1.2.3 as subproblems of view adaptation in data warehouse approach to information integration.

The first contribution of this dissertation is the design of the overall information integration framework that lays the basis for tackling the above mentioned two problems. Secondly, as part of the overall approach, a data model for information source description and an extension of SQL view definition language are presented. Next, providing a solution for Problem 1, we define semantics for view synchronization and design view synchronization strategies for different types of available constraints and capability changes. Another contribution is the solution we propose for Problem 4, maintenance of materialized views after view synchronization, by developing algorithms that incrementally update the view data after the synchronization process took place. Furthermore, a working system incorporating the view synchronization algorithms presented in this dissertation has been developed. To evaluate the proposed view maintenance techniques for Problem 4, we use a testbed based on the TPC-D benchmark [64] where we compare our strategies with conventional data maintenance approaches, namely, recomputation and redefinition. We outline each of these contributions in more depth in the following subsections.

1.3.1 Evolvable View Evolution EVE Approach

We describe here the proposed overall approach, which we call *Evolvable View Environment* (*EVE*) (Figure 1.4), followed by the detailed descriptions of the problems tackled in this dissertation. Our environment can be divided into two spaces, i.e., the user space and the information space (which matches the general architecture of an information integration system depicted in Figure 1.1). The information space is populated by a large number of heterogeneous, distributed, and autonomous ISs. The information space is very dynamic, e.g., the ISs can join, leave and change their data and capabilities at any time without regard to views defined upon them. Similar to other large-scale systems (Figure 1.1), an IS is integrated in the global framework via a wrapper called Information Source Interface (ISI) in Figure 1.4. The main functionality of an ISI is to translate common-language queries into source-specific queries supported by each IS. The ISI is assumed to be intelligent so that it can extract not only raw data, but also meta information that describes each IS, such as schema level information or relationships with other ISs. Any IS that supports a query interface can participate in this environment. When an IS joins *EVE*, it advertises its capabilities and content to the system, and this IS description knowledge is then kept in a meta knowledge base (MKB)[2].

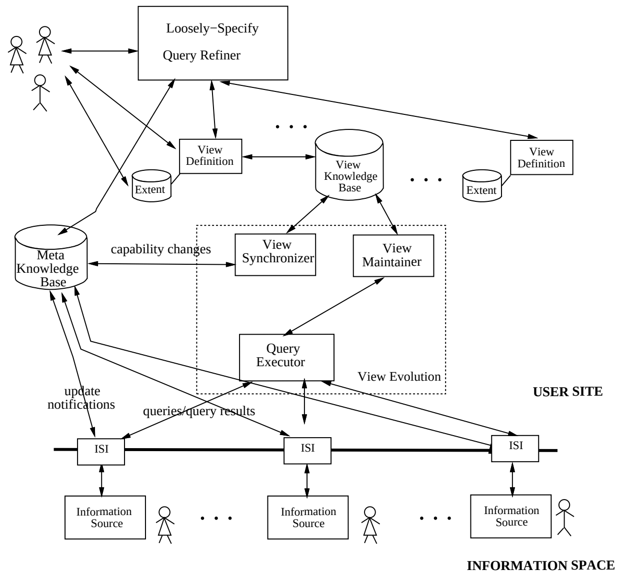


Figure 1.4: The *EVE* Framework: View Evolution in an Evolving Environment.

Generally, a system user starts by posing loosely-specified queries against the MKB. The *loosely-specified query refiner* disambiguates these queries into well-defined queries. The user could choose to keep for further reference these well defined queries as view definitions (VDs) in the View Knowledge Base (VKB). When the system is notified about a capability change of one of the ISs, *EVE* first identifies which VDs are affected. The *view evolution* module consists of three tools: the *view synchronizer* that evolves those affected VDs after a capability change of an IS; the *view maintainer* that is responsible for bringing the view extents up-to-date after view

[2]The information providers have strong economic incentives to provide the meta knowledge of their individual ISs as well as of the relationships with other ISs, since populating the MKB not only advertises their resources to potential view users, but it also increases the utilization of their data set in cases where users of other competitive ISs are in need of alternate sources (especially, if they offer the same information at a better price).

synchronization in the case when the views are materialized; and the *query executor* (corresponding to a mediator from the general architecture depicted in Figure 1.1) that translates query requests from the view maintainer into several smaller query fragments to be sent to individual ISs and then assembles the results submitted from the ISs into one coherent response.

1.3.2 Research Issues Related to View Synchronization Support

In this dissertation, the general framework for view adaptation in evolving environments called *EVE* (Section 1.3.1) supports preservation of view components instead of completely disabling views with each IS change. One key component of the *EVE* framework is the language for capturing descriptions of the content, capabilities as well as interrelationships of all ISs in the system. Descriptions of ISs expressed in this language are maintained in a meta-knowledge base (MKB) available during the view evolution process. We assume that information sources are described in the language proposed in this dissertation (Section 2.3), that is applicable to a wide range of information sources ranging from more structured DBMSs to more unstructured web resources.

A second key component of the *EVE* framework is the *view definition language* (E-SQL) (Section 2.4) which is essentially SQL extended with view evolution preferences. E-SQL allows the view definer to control the view evolution process triggered by environment changes by indicating the criticality and dispensability of the different components of the view definition. For example, a view definer could indicate that the attribute **Name** is indispensable to the view, whereas the attribute **Address** is desirable yet can be omitted from the original view definition, if keeping it becomes impossible, without jeopardizing the utility of the view.

Given a view defined in E-SQL and a MKB, we have defined the formal foundation for what are *legal rewritings* of a view affected by capability changes. This includes properties characterizing MKB constraints and the level of preservation achieved by the view redefinition process given evolution preferences.

Based on this formal foundation, this dissertation includes strategies for solving this view adaptation problem, i.e., algorithms for handling the standard set of capability changes that can affect a view definition, such as delete relation, delete attribute, etc. The *view synchronization algorithms* finds valid replacements for affected (deleted) components of the existing view definitions based on the semantic constraints captured in the MKB. First, we design view synchronization algorithms that require only the availability of *containment information* among information sources in order to evolve views. Unlike the strategies proposed for query rewriting using views in the database literature ([33, 61]), our techniques address three new issues: (1) finding view rewritings that are not necessarily *equivalent* to the original view definition, (2) preserving only indispensable attributes from the SELECT clause if preserving all is not possible; and (3) using semantic *containment information* for replacing the deleted view components. The first proposed technique, *PrOject Containment* view synchronization (POC) algorithm finds a replacement for an entire relation or only one of the attributes used in a view based on containment constraints between the projections of the affected relation and replacement relation. In the second algorithm, we consider the select-project containment information between the to-be-replaced relation and its replacement, thus we call it *Select-PrOject Containment* view synchronization (SPOC) algorithm. We present the algorithms and give formal proofs that the rewritings found are consistent with our semantics for legal view rewritings.

Second, we propose a strategy for the view synchronization problem by exploiting other semantic constraints captured in the MKB such as *join and function-of constraints*, which we call *Complex View Synchronization* (*CVS*). Rather than just providing so-called 'one-step-away' view rewriting as we use for POC and SPOC algorithms, CVS finds replacements that correspond to possibly complex pieces of information from several ISs, succeeding in determining view rewritings through multiple join constraints given in the MKB.

Another objective is to define a cost model that allows to measure the quality of alternate solutions found by the synchronization process (as we expect to find more than one redefinition for an affected view). To solve this problem, we present the *QC-Model* which is capable of comparing different view rewritings (generated as replacements for an original view that has become invalid). It is designed to measure the *efficiency* of a view rewriting in the two dimensions of *quality* and *cost* for any candidate view rewritings found by a synchronization process. Based on this cost model we discuss heuristics that can be used to prune the search space for the best legal rewriting for a given view, information space, and capability change.

The synchronization process changes the affected view definitions and hence makes the view extent (i.e., the view data) out-of-date. In this dissertation we developed *maintenance algorithms* for materialized views after synchronization. We introduce a new view adaptation problem, namely *view maintenance after view synchronization*, we distinguish it from the view redefinition problem treated in [37, 17] demonstrating that our techniques succeed in cases when those previous techniques would fail. We present view maintenance 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. We perform a systematic experimental evaluation of our proposed algorithms and present the experimental results done using TPC-D benchmark for comparing our techniques with recomputation and redefinition strategies. For certain types of view synchronization, our view maintenance techniques perform four times better for both SPJ and aggregation views.

To summarize, the dissertation makes the following contributions:

- *The Design of a Model for Information Description Language* that captures content and capability properties of heterogeneous information sources.
- *The Development of a View Definition Language* that allows view users to express preferences about the view evolution process.
- *The Introduction of the Concept of synch-equivalence* between E-SQL specifications to express the real evolution flexibility of an E-SQL view definition.
- *The Definition of the View Evolution Semantics* that lay the foundation of what is a legal rewriting the synchronization process could produce, for both SPJ and aggregation views.
- *The Design of Algorithms for View Synchronization* that achieve the proposed view evolution semantics in the presence of different types of constraints specified for underlying information sources, for SPJ and aggregation E-SQL views as well.
- *The Proposal of a Cost Model* based on which view redefinitions found by synchronization process could be analytically ranked based on the user/application requirements.

- *The Development of Materialized View Maintenance after View Synchronization* techniques that incrementally adapt view extents after redefinition.
- *The Implementation of a TPC-D Benchmark Testbed* for performance evaluation of the view maintenance algorithms and comparison with alternative techniques for view maintenance.
- *The Design and Implementation of a Working System* for information integration with support for view evolution under capability changes in distributed environments.

1.4 Thesis Outline

The reminder of this dissertation is organized as follows. In Chapter II we outline the general framework of the *EVE* and introduce the key components of the *EVE* solution. Chapters III and IV present our view synchronization algorithms tailored for different constraints among information sources, namely containment and join constraints, respectively. Chapter VI introduces an analytical cost model for assessment of quality and cost of view rewritings obtained by our view synchronization strategies. Chapter V investigates strategies for view maintenance after synchronization problem and presents the testbed and experimental results for the proposed approach. Chapter VII discusses related work relevant to our *EVE* approach. In Chapter VIII we conclude this dissertation by presenting the conclusions and future research.

CHAPTER II

Evolvable View Evolution Framework

In this introductory chapter, we focus on defining the general *EVE* framework that provides our solution to the view synchronization problem for view adaptation under capability changes. We put forth that it is important for the person in charge of defining the virtual information resource (i.e., view) to be able to express preferences about the view evolution process (instead of our system making automatic and generic choices) – as these view definers are the ones that know the criticality and dispensability of the different components of a view for applications and end users of the view. As these view evolution preferences refer to specific components of view definition, in our system the view definer can directly embed their preferences about view evolution into the view definition itself. We design 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 (see Section 2.4). Such view preference specification would allow us to avoid human interaction each and every time a change occurs in the environment.

To facilitate the replacement finding task, we exploit a model for information source description (MISD) for capturing the capabilities of each IS as well as the interrelationships between ISs. Similar to other mediation systems such as the University of Michigan Digital Library system [5] and the Garlic project [19], each IS registers its description expressed in an integration model in a Meta Knowledge Base (MKB) when joining the system. This Meta Knowledge Base (MKB) thus represents a resource that can be exploited when searching for an appropriate substitution for the affected components of a view in the global environment. MISD is introduced in Section 2.3.

Based on this solution framework of E-SQL and the MISD, we introduce strategies for evolving views transparently. Our proposed view rewriting process, which we call *view synchronization*, finds a view redefinition that meets all view preservation constraints specified by the E-SQL view definition. That is, it identifies and extracts appropriate information from other ISs as replacement of the affected components of the view definition and produces an alternative view definition. Hence, given a view defined in E-SQL and a MKB, we define in Section 2.5 a formal foundation for what are *legal rewritings* of the view affected by capability changes. This includes both evolution semantics associated with E-SQL evolution parameters as well as properties that new components used to replace the obsolete ones must have.

The E-SQL view definition language allows any combination of the evolution parameters to be set for the view components in order to simplify the specifications task for the users. Because of the relationships between components, e.g., an attribute in the `SELECT` clause is coming from a relation in the `FROM` clause, the synchronization process cannot take full advantage of the apparent flexibility of all combinations of evolution parameters. We introduce in Section 2.6 the synch-equivalence concept to express the real evolution flexibility of an E-SQL view definition. We also present a strategy for finding the strongest synch-equivalent E-SQL definition for a given view specification, and prove that our transformation rules are correct. Finding the synch-equivalent definitions is essential for helping the view definer to understand the semantics associated with a view definition and makes the implicit E-SQL evolution semantics explicit.

The remainder of the chapter is structured as follows. In Section 2.1, we present the *EVE* framework, and in Section 2.2 we introduce a web-based travel agency example used as running example throughout the dissertation. The extended view definition language, E-SQL, designed to add flexibility to current view technology is presented in Section 2.4. In Section 2.3, we present the information source description model (MISD), while the formal semantics for synchronization is given in Section 2.5. The Section 2.6 gives a characterization of the concept of the strongest synch-equivalent view definition and provides algorithms of how to compute it. We conclude this chapter with Section 2.7.

The work presented in this chapter has been published in several papers in [45, 44, 60, 30, 40]. An earlier version of the MISD model is published by the author and her adviser in [45, 44]. The extension of the SQL (i.e., E-SQL), MISD for EVE and the overall EVE framework is the result of the collaboration with members of the EVE project. The author also introduced the concept of the strongest E-SQL view definition in [47].

2.1 Evolvable View Environment (*EVE*) Framework

The *view synchronization process* attempts to evolve views, when they are affected by capability changes triggered by the participating ISs. In this section we present the Evolvable View Environment (*EVE*) framework that we propose for tackling the view synchronization problems in dynamic environments (Figure 2.1). We give an architectural overview of the *EVE* framework and outline our key design decisions.

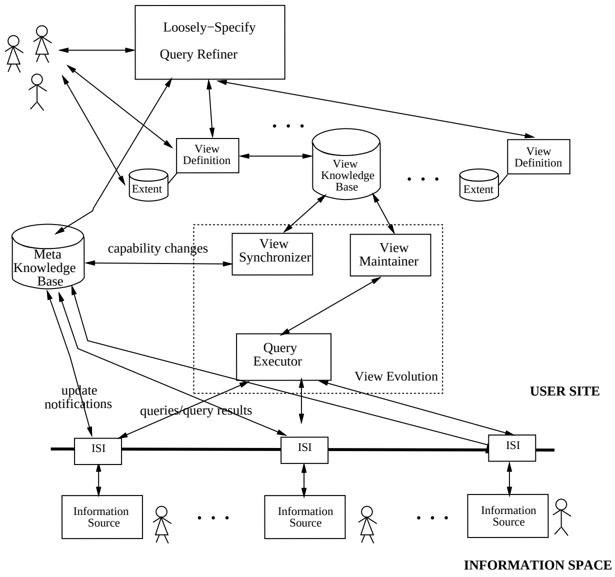


Figure 2.1: The Framework of Evolvable View Environment (EVE).

IS Registration. Our environment can be divided into two spaces, i.e., the view space and information space. The information space is populated by a large number of external ISs. External ISs are heterogeneous and distributed. Most importantly, they are dynamic and can autonomously change their capabilities, when desired. They could even join or leave the system at any time. An IS is "integrated" in the global framework via a wrapper that serves as a bridge between the information space and the view space. The main functionality of a wrapper is to translate the messages specified in the underlying data definition/manipulation languages into a common language used in the view site, and vice versa. The wrapper is assumed to be intelligent so that it can extract not only raw data, but also meta information about the IS, such as changes at the schema level of the IS, performance data, or relationships with other ISs. Any IS that supports a query interface can participate in our environment.

Meta Knowledge Base (MKB). When an IS joins *EVE*, it advertises to the MKB its capabilities, data model (e.g., the semantic mappings from its concepts to the concepts already in the MKB), and data content. The information providers have strong economic incentives to provide the meta knowledge of their individual ISs as well as the relationships with other ISs, since populating the MKB makes their data known by the view users, and thus increases the data utilization of their data set (especially, if they offer the same information at a better price).

We have designed a model for information source descriptions (MISD) [30, 60] that is capable of describing the content and capabilities of heterogeneous ISs. MISD captures meta knowledge such as an attribute must have a certain type (type integrity constraint), one relation can be meaningful joined with another relation if certain join constraints are satisfied (join constraint), a fragment of a relation is partially or completely contained in another fragment of some other relation (partial/complete information constraint), and so on (see Section 2.3). The IS descriptions collected in the MKB form an information pool that is critical in finding appropriate replacements for view components when view definitions become undefined (See Section 5) and for translating loosely-specified user requests into precise query plans [45].

MKB Evolution. When an underlying IS makes a change to its capabilities (e.g., adds a new relation), the MKB no longer reveals the IS correctly in the sense that the meta knowledge describing the IS and the actual capabilities of the IS are distinct. For this, we have designed the MKB Evolution process to react to capability changes in the information space. In our framework, each IS will via the wrapper interface notify the MKB of any such capability changes so that they can be properly registered in the MKB. The MKB Evolver module will then take appropriate actions to update the MKB [39]. For example, deleting an attribute A from a relation S may cause the MKB evolver to modify a subset constraint between two relations S and R, e.g., "$S \subset R$", into the constraint "$S \subset$ (project all attributes of R besides A from R)". In other cases, some constraints may have to be completely removed from the MKB if they contain references to the deleted attribute.

View Maintenance. The *view maintainer* tool (Figure 2.1) in general is in charge of propagating data updates executed on an IS site to all affected views. In our system, this tool will also be in charge of bringing the view content up-to-date after the view definition already has been changed by the view synchronizer in response to a capability change.

View Synchronization. The *view synchronizer* tool (Figure 2.1) evolves affected views transparently according to users' preferences expressed by our extended view definition language E-SQL. View synchronization is the focus of this paper, and we will present replacement strategies and view synchronization algorithms in later sections.

Global Consistency Checking Across Sources. There are two types of inconsistencies (related to meta knowledge) in *EVE*. The first one is that constraints expressed in the MKB do not correspond to the information actually provided by ISs; and the second one is that different assertions in the MKB contradict each other. The first type of inconsistency occurs when (1) either an IS provider makes an error when entering a MISD description, (2) an update occurred at one IS that causes a constraint that used to hold to become invalid, or (3) the usage and hence content of an IS changes over time without proper notification to the MKB. For example, the information provider for IS_1 inserts the fact that the relation R is equivalent to a relation S in another site IS_2 into the MKB. Now, the provider of IS_2, that is not aware of this assertion made about S in IS_2, inserts a new tuple t that makes the assertion become false.

There are alternative approaches for resolving this inconsistency. For example, (1) insert the tuple t into the relation R as well, (2) reject the insertion into S, (3) modify the invalid assertion in the MKB so to make it valid (i.e., in this case change "$\mathrm{IS}_1.R \equiv \mathrm{IS}_2.S$" into "$\mathrm{IS}_1.R \subset \mathrm{IS}_2.S$"), or (4) remove the invalid assertion from the MKB. Since checking and enforcing constraints across distributed autonomous ISs is an extremely difficult problem all on its own, in this work we assume that providers of individual ISs are in charge of assuring that their data is consistent with the meta knowledge collected in the MKB. We do not at this time incorporate a tool into our *EVE* framework that resolves possible inconsistencies. However, once being notified about the entry or removal of some data item by an IS, *EVE* will notify the creators of all constraints in the MKB that may possibly be violated by this data modification. For example, on inserting a new tuple t into the relation S in the above example, both the providers of S and R are notified that the update occurred and that the constraint "$\mathrm{IS}_1.R \equiv \mathrm{IS}_2.S$" may now be inconsistent. It is up to the providers of IS_1 and IS_2 to determine how to handle this situation, once given the notification.

MKB Consistency. The second type of MKB consistency concerns conflicts between the constraints entered in the MKB, and thus can be detected by our MKB Consistency Checker module without help from the IS providers. One example of this type of conflict is that one information provider declares that a relation R of IS_1 is a strict subset of a relation S in another site IS_2, and at the same time the provider of S claims that the extent of S is a strict subset of R. This is clearly an inconsistency. Our MKB consistency checker discovers such controversial meta knowledge using various types of inference techniques. Once detected, inconsistent assertions are reported to responsible information providers to have the differences resolved.

2.2 Running Example: The Global Travel Agency Service

We use a travel consolidator service provider as running example throughout this dissertation. Below we describe the relevant information sources (expressed using relations in our system) and two example views, while additional relations and views are added later, as needed.

Example 1 *Consider a large travel consolidator which has a headquarter in Detroit, USA, and many branches all over the world. It helps its customers to arrange flights, car rentals, hotel reservations, tours, and purchasing insurances. Therefore, the travel consolidator needs to access many disparate information sources, including domestic as well as international sites. Since the connections to external information sites, such as the overseas branches, are very expensive and have low availability, the travel consolidator materializes the query results (views) at its headquarter or other US branches (at the view site). Some of the relevant ISs are listed in the table in Table 2.1.*

IS 1: *Customer Information* ***Customer(Name, Address, Phone, Age)***
IS 2: *Tour Information* ***Tour(TourID, TourName, Type, Duration)***
IS 3: *Tour Participant Information* ***Participate(Participant, TourID, StartingDate)***
IS 4: *Flight Reservation Information* ***FlightRes(PName, Airline, FlightNo, Source, Dest, Date)***
IS 5: *Insurance Information* ***Accident–Ins(Holder, Amount, Birthday)***
IS 6: *Car Rental Information* ***CarRental(Name, Address, Phone, City, State, Country)***
IS 7: *Hotel Information* ***Hotel(Name, Address, Phone, City, State, Country)***

Table 2.1: Descriptions of Relevant Information Sources.

Assume the headquarter maintains complete sets of information of the customers, tours, and tour participants in the following formats:

Customer(Name, Address, Phone, Age),

Tour(TourID, TourName, Type, Duration)

- *where* ***Type*** *attribute has values in the set {luxurious, economy, super−valued}, and* ***Participate(Participant, TourID, StartingDate)*** *that states which customer joins which tour starting on what day. We further assume the local branches keep partial sets of information of its local customers, the tours offered locally, and the participation information of its local customers. The flight reservation information* ***FlightRes(PName, Airline, FlightNo, Source, Dest, Date)*** *is managed by each individual airline company. Insurance information*

Accident–Ins(Holder, Amount, Birthday) *is kept by each individual insurance company. The car rental company and lodging information,*

CarRental(Name, Address, Phone, City, State, Country) *and*

Hotel(Name, Address, Phone, City, State, Country), *are managed by each individual company, respectively.*

Let's assume that the travel agency has a promotion for the customers who travel to Asia. Therefore, the travel agency needs to find the customers' names, addresses, and phone numbers in order to send promotion letters to these customers or call them by phone. The SPJ *view definition can be specified as follows:*

```
CREATE VIEW  Asia-Customer AS
SELECT       Name, Address, Phone
FROM         Customer C, FlightRes F                          (2.1)
WHERE        (C.Name = F.PName) AND (F.Dest = 'Asia')
```

In addition, the travel consolidator wants to study the correlation between the type *of tour a customer (older than 18) participates in and the type of accidental insurance she buys. Therefore, this second aggregation view definition can be specified as:*

$$\begin{array}{ll}
\text{CREATE VIEW} & \textbf{\textit{Tour-Insurance}} \text{ AS} \\
\text{SELECT} & \textbf{\textit{C.Age, T.Type, T.Duration, MAX(I.Amount)}} \\
\text{FROM} & \textbf{\textit{Customer C, Participate P, Tour T, Accidental-Insurance I}} \\
\text{WHERE} & \textbf{\textit{(C.Name = P.Participant)}} \text{ AND } \textbf{\textit{(P.TourID = T.TourID)}} \text{ AND} \\
 & \textbf{\textit{(C.Name = I.Holder)}} \text{ AND } \textbf{\textit{(C.Age}} > 18) \\
\text{GROUP BY} & \textbf{\textit{C.Age, T.Type, T.Duration}}
\end{array} \quad (2.2)$$

Note that the views 2.1 and 2.2 are static apriori-specified queries. We use this travel consolidator service example to demonstrate the usage of and interactions among proposed evolution parameters in later sections.

2.3 MISD: Model for Information Source Description

While individual ISs could be based on any data model, we assume that the information source interface (ISI) agent of an IS describes the schema exported by the IS as a set of relations $IS.R_1$, $IS.R_2, \ldots, IS.R_n$ that reside at the IS. A relation description contains three types of information specifying its data structure and content, its query capabilities as well as its relationships with exported relations from other ISs that semantically express the operations allowed between ISs. The descriptions of the ISs are stored in the meta knowledge base (MKB) (see Figure 2.1) and are used in the process of view evolution, when alternative rewritings must be found for the affected views after a capability change at an IS[1]. Below we introduce the MISD model with all MISD constraints summarized in Table 2.2 [45, 44, 30, 31].

[1] We assume in this work that information sources support at least SPJ and aggregation SQL queries with conjunctions of primitive clauses in the WHERE clause, and thus we are not discussing how query capabilities are described in our system [44].

Name	Syntax
Type Integrity Constraint	$\mathcal{TC}_{R.A_i} \;=\; (R(A_i) \subseteq Type_i(A_i))$
Order Integrity Constraint	$\mathcal{OC}_R \;=\; (R(A_1,\ldots,A_n) \subseteq \mathcal{C}(A_{i_1},\ldots,A_{i_k}))$
Join Constraint	$\mathcal{JC}_{R_1,R_2} = (C_1 \text{ AND } \cdots \text{ AND } C_l)$
Partial/Complete Constraint	$\mathcal{PC}_{R_1,R_2} = (\pi_{\bar{A}_1}(\sigma_{\mathcal{C}(\bar{B}_1)}R_1)\ \theta\ \pi_{\bar{A}_2}(\sigma_{\mathcal{C}(\bar{B}_2)}R_2))$ $\theta \in \{\subset,\subseteq,\equiv,\supseteq,\supset\}$

Table 2.2: Semantic Constraints for IS Descriptions.

2.3.1 Data Content Description

The model used to describe the basic units of information available in each of the ISs is the relational model. A base relation is an n-ary relation, with $n \geq 2$. A relation name is not required to be unique in the MKB, but the pair (IS name, relation name) is. A relation R is described by specifying its information source and its set of attributes as follows:

$$IS.R(A_1,\ldots,A_n). \tag{2.3}$$

2.3.2 Type Integrity Constraints

Each attribute A_i is given a name and a data type to specify its domain of values. This information is specified by using a type integrity constraint with the format $Type_i(A_i)$. A *type constraint* for a relation $R(A_1,\ldots,A_n)$ is specified as:

$$R(A_1,\ldots,A_n) \subseteq Type_1(A_1),\ldots,Type_n(A_n) \tag{2.4}$$

which says that an attribute A_i is of type $Type_i$, for $i = 1,\ldots,n$. For simplicity, we assume that the attribute types are primitive. If two attributes are exported with the same name, they are assumed to have the same type (which must be reflected by the type constraints for their relations).

2.3.3 Order Integrity Constraints

The *order constraints* specify data constraints that are satisfied by any tuple of a relation at any time. For a relation $R(A_1, \ldots, A_n)$, an order constraint is defined by:

$$R(A_1, \ldots, A_n) \subseteq \mathcal{C}(A_{i_1}, \ldots, A_{i_k}) \tag{2.5}$$

where $(A_{i_1}, \ldots, A_{i_k})$ is a subset of the attributes of R and $\mathcal{C}(A_{i_1}, \ldots, A_{i_k})$ is a conjunction of primitive clauses defined over the attributes. A primitive clause has one of the following forms: $< attribute - name > \theta < attribute - name >$ or $< attribute - name > \theta < value >$ with $\theta \in \{<, \leq, =, \geq, >\}$. Expression (2.5) specifies that for any state of the database R, and for any tuple $t \in R$, $\mathcal{C}(t[A_{i_1}], \ldots, t[A_{i_k}])$[2] is satisfied.

Example 2 *An insurance relation* ***Expensive-Insurance****, containing all expensive accidental insurances that cover more than* $1,000,000, *can be expressed by the following order constraint:*

$$\textbf{\textit{Expensive-Insurance(Holder, Type, Amount, Birthday)}} \subseteq (\ \textbf{\textit{Amount}} > 1,000,000).$$

2.3.4 Join Constraints

In order to evolve views defined over a set of relations exported from different ISs, the IS description also contains a specification to characterize the IS content in terms of its relationship with other sources in the environment. A join constraint is used to specify a meaningful way to combine information from two ISs, i.e., a join condition

[2] The expression $t[A]$ refers to the value of the attribute A in the tuple t.

between two relations is used to capture our knowledge of their interrelationship. The join condition is a conjunction of primitive clauses (not necessarily equijoin clauses). Formally, a join constraint is of the form:

$$\mathcal{JC}_{R_1,R_2} = (C_1 \text{ AND } \cdots \text{ AND } C_l) \tag{2.6}$$

where $C_1, \ldots, C_l$ are primitive clauses over the the attributes of R_1 and R_2. Expression (2.6) gives a default, legal join condition that could be used to join R_1 and R_2, specifying that the join relation $J = R_1 \bowtie_{(C_1 \cdots\ AND\ \cdots C_l)} R_2$ is a meaningful way of combining the two relations.

Example 3 *For our running Example 2.2, some of the join constraints are given in the table of Table 2.3 (the underlined names are the relations for which the join constraints are defined).*

For example, for the relations ***Customer(Name, Address, PhoneNo, Age)*** *and* ***Person(SSN, Name, PermanentAddress)****, the join constraint:*

$$\begin{aligned} \mathcal{JC}_{\textbf{\textit{Person, Customer}}} &= \\ (\ \textbf{\textit{Person.Name}} = \textbf{\textit{Customer.Name}})\ & \textit{AND} \\ (\ \textbf{\textit{Person.PermanentAddress}} = \ \textbf{\textit{Customer.Address}}) & \end{aligned} \tag{2.7}$$

states that the two relations ***Customer*** *and* ***Person*** *can be meaningfully joined on the attributes* ***Name*** *and* ***Address****.*

2.3.5 Partial/Complete Information Constraints

The *partial/complete* ($\mathcal{PC}$) information constraints make it possible to 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 are used to decide if an evolved

$\mathcal{JC}$	**Join Constraint**
JC1	**Customer.Name = FlightRes.PName**
JC2	**Customer.Name = Accident−Ins.Holder** AND **Customer.Age** > 1
JC3	**Customer.Name = Participant.Participant**
JC4	**Participant.TourID = Tour.TourID**
JC5	**Hotels.Address = RentACar.Location**
JC6	**FlightRes.PName = Accident−Ins.Holder**

Table 2.3: Join Constraints for Example 2.2

view is equivalent, subset of, or superset of the initial view. For two relations R_1 and R_2, the $\mathcal{PC}$ information constraint is given by:

$$\mathcal{PC}_{R_1,R_2} = \quad (\pi_{A_{i_1},\ldots,A_{i_k}}(\sigma_{\mathcal{C}(A_{j_1},\ldots,A_{j_l})}R_1) \;\; \theta \;\; \pi_{A_{n_1},\ldots,A_{n_k}}(\sigma_{\mathcal{C}(A_{m_1},\ldots,A_{m_t})}R_2)) \tag{2.8}$$

where θ 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 $A_{i_1}, \ldots, A_{i_k}$ and $A_{n_1}, \ldots, A_{n_k}$ are such that for $s = \overline{1,k}$ the attributes A_{i_s} and A_{n_s} have the same type.

Example 4 *To give an example, Eq. (2.9) states that the relations* ***Person*** *and* ***Customer*** *have the same data for the attributes* ***Name*** *and* ***Address*** *for customers age* 1 *or older. (This means that the relation* ***Person*** *contains* ***SSN*** *only for persons older than* 1 *year.)*

$$\mathcal{PC}_{\textbf{\textit{Person, Customer}}} = \tag{2.9}$$
$$(\pi_{\textbf{\textit{Name, PermanentAddress}}}(\textbf{\textit{Person}}) \;\; \equiv \pi_{\textbf{\textit{Name, Address}}}(\sigma_{\textbf{\textit{Age}}>1}\; \textbf{\textit{Customer}}))$$

Using the $\mathcal{PC}$ information constraints and type constraints we can, for example, define that two relations are equivalent: (1) they have attributes of the same types (expressed by type constraints); and (2) their extents are the same (expressed by a $\mathcal{PC}$ constraint).

2.3.6 Attribute Function-Of Constraints

The **attribute function-of constraint** relates two attributes by defining a function to transform one of them into another. This constraint is specified by:

$$\mathcal{F}_{R_1.A,R_2.B} = (\ R_1.A = f(R_2.B)\ ) \tag{2.10}$$

where f is a function[3]. The function-of constraint $\mathcal{F}_{R_1.A,R_2.B}$ specifies that if there exists a meaningful way of combining the two relations R_1 and R_2 (e.g., using join constraints) then for any tuple t in the join relation J, we have $t[R_1.A] = f(t[R_2.B])$.

Example 5 *For our running Example 2.2, function-of constraints are given in the table of Table 2.4.*

$\mathcal{F}$	Function-of Constraints
F1	**Customer.Name = FlightRes.PName**
F2	**Customer.Name = Accident−Ins.Holder**
F3	**Customer.Age** = (today - **Accident−Ins.Birthday**)/ 365
F4	**Customer.Name = Participant.Participant**
F5	**Participant.TourID = Tour.TourID**
F6	**Hotels.Address = RentACar.Location**
F7	**Hotels.City = RentACar.City**

Table 2.4: Function-of Constraints for Example 2.2.

2.4 Extending SQL for Flexible View Synchronization

In this section, we present the *EVE* view definition language (E-SQL), which is an extension of SQL augmented with specifications for how the view definition may be synchronized under IS capability changes. Evolution preferences, expressed as *evolution parameters*, allow the user to specify criteria based on which the view will be transparently evolved by the system under capability changes at the ISs. We assume

[3]Note that the inverse of f is not required to exist, and hence if an inverse is available it must be explicitly listed as $R_2.B = f^{-1}(R_1.A)$.

Notation	Definition
$Attr(V(R))_{\mathsf{SELECT}}(d,r)$	$\{R.A \mid R.A$ used by a component $\mathcal{A}$ in SELECT clause of V s.t., $\mathcal{AD}(\mathcal{A}) = d$ AND $\mathcal{AR}(\mathcal{A}) = r\}$
$Attr(V(R))_{\mathsf{SELECT}}$	$\cup_{d,r \in \{false,true\}} Attr(V(R))_{\mathsf{SELECT}}(d,r)$
$Attr(V(R))_{\mathsf{SELECT}}(\_,r)$	$\cup_{d \in \{false,true\}} Attr(V(R))_{\mathsf{SELECT}}(d,r)$
$Attr(V(R))_{\mathsf{SELECT}}(d,\_)$	$\cup_{r \in \{false,true\}} Attr(V(R))_{\mathsf{SELECT}}(d,r)$
$grAttr(V(R))(d,r)$	$\{ R.A \mid R.A$ a group-by attribute in SELECT clause of V s.t. $\mathcal{AD}(R.A) = d$ AND$\mathcal{AR}(R.A) = r \}$
$grAttr(V(R))$	$\cup_{d,r \in \{false,true\}} grAttr(V(R))(d,r)$
$grAttr(V(R))(\_,r)$	$\cup_{d \in \{false,true\}} grAttr(V(R))(d,r)$
$grAttr(V(R))(d,\_)$	$\cup_{r \in \{false,true\}} grAttr(V(R))(d,r)$
$aggAttr(V(R))(d,r)$	$\{R.A \mid R.A$ used by an aggregate attribute f in SELECT clause of V s.t. $\mathcal{AD}(f) = d$ AND $\mathcal{AR}(f) = r\}$
$aggAttr(V(R))$	$\cup_{d,r \in \{false,true\}} aggAttr(V(R))(d,r)$
$aggAttr(V(R))(\_,r)$	$\cup_{d \in \{false,true\}} aggAttr(V(R))(d,r)$
$aggAttr(V(R))(d,\_)$	$\cup_{r \in \{false,true\}} aggAttr(V(R))(d,r)$
$Attr(V(R))_{\mathsf{SELECT}}$	$grAttr(V(R)) \cup aggAttr(V(R))$
$Attr(V(R))_{\mathsf{WHERE}}(d,r)$	$\{ R.A \mid R.A$ in a condition C in WHERE clause of V s.t. $\mathcal{CD}(C) = d$ AND$\mathcal{CR}(C) = r \}$
$Attr(V(R))_{\mathsf{WHERE}}$	$\cup_{d,r \in \{false,true\}} Attr(V(R))_{\mathsf{WHERE}}(d,r)$
$Attr(V(R))_{\mathsf{WHERE}}(\_,r)$	$\cup_{d \in \{false,true\}} Attr(V(R))_{\mathsf{WHERE}}(d,r)$
$Attr(V(R))_{\mathsf{WHERE}}(d,\_)$	$\cup_{r \in \{false,true\}} Attr(V(R))_{\mathsf{WHERE}}(d,r)$
$Attr(V(R))_{\mathsf{WHERE}}^{\mathrm{LOCAL}}$	$\{R.A \mid R.A$ in a *local* condition C in WHERE clause of $V\}$
$Attr(V(R))_{\mathsf{WHERE}}^{\mathrm{JOIN}}$	$\{R.A \mid R.A$ in a *join* condition C in WHERE clause of $V\}$
$Attr(V(R))$	$Attr(V(R))_{\mathsf{SELECT}} \cup Attr(V(R))_{\mathsf{WHERE}}$

Table 2.5: Notations for Attribute Sets.

SELECT-PROJECT-JOIN (denoted by SPJ) or aggregation (denoted by AGGR-SPJ) views defined as in Equation (2.11) and (2.12), respectively, with a conjunction of primitive clauses in the WHERE clause.

As indicated in Table 2.6, each component 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 $\mathcal{A}$, $\mathcal{R}$ or $\mathcal{C}$ for attribute, relation or condition component, respectively) specifies if the component could be dropped (value

$$
\begin{array}{ll}
\text{CREATE VIEW} & V \;\; (B_1, \ldots, B_m) \;\; (\mathcal{VE} = \delta_V) \text{ AS} \\
\text{SELECT} & R_1.A_{1,1}(\mathcal{AD} = \mathcal{AD}_{1,1}, \mathcal{AR} = \mathcal{AR}_{1,1}), \ldots, \\
 & R_1.A_{1,i_1}(\mathcal{AD} = \mathcal{AD}_{1,i_1}, \mathcal{AR} = \mathcal{AR}_{1,i_1}), \ldots \\
 & R_n.A_{n,1}(\mathcal{AD} = \mathcal{AD}_{n,1}, \mathcal{AR} = \mathcal{AR}_{n,1}), \ldots, \\
 & R_n.A_{n,i_n}(\mathcal{AD} = \mathcal{AD}_{n,i_n}, \mathcal{AR} = \mathcal{AR}_{n,i_n}) \\
\text{FROM} & R_1(\mathcal{RD} = \mathcal{RD}_1, \mathcal{RR} = \mathcal{RR}_1), \ldots, R_n(\mathcal{RD} = \mathcal{RD}_n, \mathcal{RR} = \mathcal{RR}_n) \\
\text{WHERE} & C_1(\mathcal{CD} = \mathcal{CD}_1, \mathcal{CR} = \mathcal{CR}_1) \text{ AND } \ldots \text{ AND } C_k(\mathcal{CD} = \mathcal{CD}_k, \mathcal{CR} = \mathcal{CR}_k)
\end{array}
\tag{2.11}
$$

Figure 2.2: Syntax of E-SQL SPJ View Definition.

$$
\begin{array}{ll}
\text{CREATE VIEW} & V \;\; (B_1, \ldots, B_m, F_1, \ldots, F_k) \;\; (\mathcal{VE} = \delta_V) \text{ AS} \\
\text{SELECT} & R_1.A_{1,1}(\mathcal{AD} = \mathcal{AD}_{1,1}, \mathcal{AR} = \mathcal{AR}_{1,1}), \ldots, \\
 & R_1.A_{1,i_1}(\mathcal{AD} = \mathcal{AD}_{1,i_1}, \mathcal{AR} = \mathcal{AR}_{1,i_1}), \ldots \\
 & R_n.A_{n,1}(\mathcal{AD} = \mathcal{AD}_{n,1}, \mathcal{AR} = \mathcal{AR}_{n,1}), \ldots, \\
 & R_n.A_{n,i_n}(\mathcal{AD} = \mathcal{AD}_{n,i_n}, \mathcal{AR} = \mathcal{AR}_{n,i_n}) \\
 & f_1(X_1)(\mathcal{AD} = \mathcal{AD}_{f_1}, \mathcal{AR} = \mathcal{AR}_{f_1}), \ldots, \\
 & f_k(X_k)(\mathcal{AD} = \mathcal{AD}_{f_k}, \mathcal{AR} = \mathcal{AR}_{f_k}) \\
\text{FROM} & R_1(\mathcal{RD} = \mathcal{RD}_1, \mathcal{RR} = \mathcal{RR}_1), \ldots, R_n(\mathcal{RD} = \mathcal{RD}_n, \mathcal{RR} = \mathcal{RR}_n) \\
\text{WHERE} & C_1(\mathcal{CD} = \mathcal{CD}_1, \mathcal{CR} = \mathcal{CR}_1) \text{ AND } \ldots \text{ AND } C_k(\mathcal{CD} = \mathcal{CD}_k, \mathcal{CR} = \mathcal{CR}_k) \\
\text{GROUP BY} & R_1.A_{1,1}, \ldots, R_1.A_{1,i_1} \\
 & R_n.A_{n,1}, \ldots, R_1.A_{n,i_n}
\end{array}
\tag{2.12}
$$

Figure 2.3: Syntax of E-SQL Aggregation View Definition.

true) or must be present in any evolved view definition (value *false*). Two, the *replaceable parameter* (notation $\mathcal{XR}$, where again $\mathcal{X}$ could be $\mathcal{A}$, $\mathcal{R}$ or $\mathcal{C}$ for attribute, relation or condition component, respectively) specifies if the component could be replaced in the process of view evolution (value *true*) or must be left unchanged as defined in the initial view (value *false*).

In Table 2.6, each type of evolution parameter used by E-SQL is represented by a row in that table. Table 2.6 has two columns: column one gives the parameter name and the abbreviation for each parameter, and column two the possible values each parameter can take on plus the associated semantics. The default values are

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

Table 2.6: View Evolution Parameters of E-SQL Language.

underlined for each parameter. When the parameter setting is omitted from the view definition, then the default value is assumed. This means that a conventional SQL view definition (without explicitly specified evolution preferences) has well-defined evolution semantics in our system, i.e., anything the user specified in the original view definition must be preserved exactly as originally defined in order for the view to be well-defined: namely, nothing can be changed. Our extended view definition semantics are thus well-grounded and compatible with current view technology.

The general format of the extended view definition language is given in Equations (2.11) and 2.12). The view interface $\bar{B}_V = (B_1, \ldots, B_m, F_1, \ldots F_k)$ corresponds to the local names given to attributes projected in the view V. For an aggregation view, we call the attributes $B_1, \ldots, B_m$ the *group-by attributes* while the attributes corresponding to the aggregation functions, $F_1, \ldots F_k$, are referred to as the *aggregation attributes*. The set $\{A_{j,1}, \ldots, A_{j,i_j}\}$ is a subset of the attributes of the relation R_j for all $j = \overline{1,n}$; any C_i, $i = \overline{1,k}$, is a primitive clause defined over the attributes of relations in the FROM clause. All parameters $\mathcal{VE}, \mathcal{AD}, \mathcal{AR}, \mathcal{RD}, \mathcal{RR}, \mathcal{CD}$, and $\mathcal{CR}$

are defined as described in Table 2.6. For aggregation views, we assume that all non-aggregation attributes are used as group-by attributes and both sets have the same evolution parameters.

Next, we use one example to demonstrate the usage of and interactions among proposed evolution parameters.

Example 6 *In our Example 2.2, let's assume that the travel agency has a promotion for the customers who travel to Asia. Therefore, the travel agency needs to find the customers' names, addresses, and phone numbers. The travel agency is either going to send promotion letters to these customers or call them by phone. The query for getting the necessary information can be specified as follows:*

$$\begin{array}{ll} \text{CREATE VIEW} & \textbf{\textit{Asia-Customer}} \text{ AS} \\ \text{SELECT} & \textbf{\textit{Name, Address, PhoneNo}} \\ \text{FROM} & \textbf{\textit{Customer C, FlightRes F}} \\ \text{WHERE} & \textbf{\textit{(C.Name = F.PName)}} \\ & \text{AND}\textbf{\textit{(F.Dest = 'Asia')}} \end{array} \tag{2.13}$$

Equation (2.13) is a static SQL query. Next, we incorporate view evolution parameters into Equation (2.13) that indicate restrictions and preferences on how the view ***Asia-Customer*** *may be evolved when the environment changes.*

Assume the travel agency is willing to accept the query results with the customer's names and addresses only. That is, the company is okay to put off the phone marketing strategy, if the customer's phone number attribute ***PhoneNo*** *is deleted from the relation* ***Customer*** *for some reason and a suitable substitute cannot be found. The user can state this preference in the* SELECT *clause (Equation (2.13)) by using the attribute dispensable parameter $\mathcal{AD}$.*

$$\text{SELECT} \quad \textbf{\textit{Name}}\ (\mathcal{AD} = false),\ \textbf{\textit{Address}}\ (\mathcal{AD} = false),\ \textbf{\textit{PhoneNo}}\ (\mathcal{AD} = true)$$

The user may want to guide the system as to whether it is acceptable for an attribute to be obtained from other sources besides the original relation. For example, if the user only accepts the customer name and address to come from the relation ***Customer****, but agrees to have the phone number come from other source(s), then the user can augment the* SELECT *clause (Equation (2.13)) with the attribute replaceable parameter.*

$$\text{SELECT} \quad \textbf{\textit{Name}}\ (\mathcal{AR} = false),\ \textbf{\textit{Address}}\ (\mathcal{AR} = false),\ \textbf{\textit{PhoneNo}}\ \ (\mathcal{AR} = true)$$

Further, let's assume the person who defines the ***Asia-Customer*** *view is willing to accept a view without the second (local) condition specified, as long as the equijoin condition is kept*[4]*. As a consequence (if the local condition is dropped), the promotion invitation letters are sent to all customers traveling by air. The user can specify her preference by adding the condition-dispensable parameter to the conditions in the* WHERE *clause of Equation (2.13).*

$$\text{WHERE}\ \textbf{\textit{(C.Name=F.PName)}}(\mathcal{CD} = false)\ \text{AND}\ \textbf{\textit{(F.Dest='Asia')}}(\mathcal{CD} = true)$$

If the user requires the redefined view extent to be either equivalent to or larger than the original view extent, the user sets the view-extent parameter ($\mathcal{VE}$) to $\supseteq$. This means any substitution of a relation, condition, or attribute should make the new view extent at least as large as the original view extent for the view synchronization

[4]Note that in general dropping a local condition is more acceptable than dropping a join condition, since dropping a join condition may change the view definition dramatically. For example, removing the only join condition between two relations, that returns some subset of tuples, ends up with a Cartesian product of these two relations, which then returns all pairwise combinations of tuples from both relations as the view result.

process to be valid. For example, if originally the ***Asia-Customer*** *view returns the customers who travel to Japan, Korea, or Hong Kong, then the view is still valid if in addition to these customers it also returns the customers who travel to Thailand and Malaysia.*

$$\text{CREATE VIEW} \quad \textbf{\textit{Asia-Customer}} \text{ AS } (\mathcal{VE} \; = \supseteq)$$

Putting together all view evolution parameters proposed above with the initial view definition from Equation (2.13), we get Equation (2.14).

$$\begin{array}{ll} \text{CREATE VIEW} & \textbf{\textit{Asia-Customer}} \ (\mathcal{VE} = \supseteq) \text{ AS} \\ \text{SELECT} & \textbf{\textit{Name, Address, PhoneNo}} \ (\mathcal{AD} = true,\ \mathcal{AR} = true) \\ \text{FROM} & \textbf{\textit{Customer C}} \ (\mathcal{RR} = true), \textbf{\textit{FlightRes F}} \\ \text{WHERE} & \textbf{\textit{(C.Name = F.PName)}} \text{AND } \textbf{\textit{(F.Dest = 'Asia')}} \ (\mathcal{CD} = true) \end{array} \qquad (2.14)$$

In Equation (2.14), wherever no view evolution parameter values are specified for a view component, then the default values are assumed as indicated in Table 2.6. To name a few, **Name** *and* **Address** *attributes in the SELECT clause are indispensable and nonreplaceable, while the relation* **FlightRes** *in the FROM clause is indispensable and nonreplaceable.*

2.5 Formal Foundation for View Synchronization

In this section we give a formal definition of what is considered to be a *legal view rewriting* for a view which became obsolete due to a capability change of an underlying information source. First, we introduce some of the basic notations we are going to use in the rest of this dissertation.

2.5.1 Notations

For two relations R and R' with different attribute sets (i.e., denoted by $Attr(R)$ 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.

Table 2.7 summarizes all π-set operations defined on this common-subset-of-attributes notion, as well as π-equality among tuples.

For relations defined by aggregation views, we have two types of π-set operations: one type when we compare the tuples *only* on common group-by attributes (denoted by π_{gr}- in Table 2.7) and the second type when all common attributes (including the aggregation attributes) are considered (denoted by π- in Table 2.7). We denote by $grAttr(V)$ the group-by attributes of the view V, and by $aggAttr(V)$ its aggregate attributes. A generic aggregation view definition is given in Equation (2.15), where $grAttr(V) = \{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n\}$ and $aggAttr(V) = \{f_1(X_1), \ldots, f_k(X_k)\}$.

$$\begin{array}{l} X_i \text{ an expression using attributes from} \\ \quad Attr(R_1) \cup \ldots \cup Attr(R_n),\ \forall i = \overline{1,k} \\ \bar{W} \subseteq (Attr(R_1) \cup \ldots \cup Attr(R_n)) \\ f_j \in \{MIN,\ MAX,\ SUM,\ AVG,\ COUNT\},\ \forall j = \overline{1,k} \end{array}$$

$$\begin{array}{ll} \text{CREATE VIEW} & V \ \ (\mathcal{VE} = \delta) \text{ AS} \\ \text{SELECT} & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\ & f_1(X_1), \ldots, f_k(X_k) \\ \text{FROM} & R_1, \ldots, R_n \\ \text{WHERE} & \mathcal{CV}(\bar{W}) \\ \text{GROUP BY} & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \end{array} \tag{2.15}$$

We use the notation $Attr(V(R))_{\text{SELECT}}(d, r)$ to denote all attributes of the relation R that are used in a component of the SELECT clause of the view V with the evolution parameters set to d and r (d and r can be *false* (f) or *true* (t)). For the views with no aggregations, $Attr(V(R))_{\text{SELECT}}$are all attributes of the relation R projected in the view V. Similarly, we use $Attr(V(R))_{\text{WHERE}}(d, r)$ for the set of all attributes of

Name	Symbol	Semantics
π-equivalent	$R =_\pi R'$	$\forall\, t' \in R', \exists\, t \in R,\ t'[Attr(R) \cap Attr(R')] = t[Attr(R) \cap Attr(R')]$ $\forall\, t' \in R', \exists\, t \in R,\ t'[Attr(R) \cap Attr(R')] = t[Attr(R) \cap Attr(R')]$
π-subset	$R' \subseteq_\pi R$	$\forall\, t' \in R', \exists\, t \in R,\ t'[Attr(R) \cap Attr(R')] = t[Attr(R) \cap Attr(R')]$
π-superset	$R' \supseteq_\pi R$	$\forall\, t \in R, \exists\, t' \in R'\ t[Attr(R) \cap Attr(R')] = t'[Attr(R) \cap Attr(R')]$
π-intersect	$R \cap_\pi R'$	$\{z \mid \exists\, t \in R\ \exists\, t' \in R',$ $t[Attr(R) \cap Attr(R')] = t'[Attr(R) \cap Attr(R')]$ $z = t[Attr(R) \cap Attr(R')]\}$
π-difference	$R \setminus_\pi R'$	$\{z \mid \exists\, t \in R,\ \nexists\, t' \in R',$ $t[Attr(R) \cap Attr(R')] = t'[Attr(R) \cap Attr(R')]$ $z = t[Attr(R) \cap Attr(R')]\}$
π-equality	$t =_\pi t'$	$t \in R,\ t' \in R',\ t[Attr(R) \cap Attr(R')] = t'[Attr(R) \cap Attr(R')]$
π_{gr}-equivalent	$V =_{\pi_{gr}} V'$	$\forall\, t' \in V', \exists\, t \in V,$ $t'[grAttr(V) \cap grAttr(V')] = t[grAttr(V) \cap grAttr(V')]$ $\forall\, t' \in V', \exists\, t \in V,$ $t'[grAttr(V) \cap grAttr(V')] = t[grAttr(V) \cap grAttr(V')]$
π_{gr}-subset	$V' \subseteq_{\pi_{gr}} V$	$\forall\, t' \in V', \exists\, t \in V,$ $t'[grAttr(V) \cap grAttr(V')] = t[grAttr(V) \cap grAttr(V')]$
π_{gr}-superset	$V' \supseteq_{\pi_{gr}} V$	$\forall\, t \in V, \exists\, t' \in V'$ $t[grAttr(V) \cap grAttr(V')] = t'[grAttr(V) \cap grAttr(V')]$
π_{gr}-intersect	$V \cap_{\pi_{gr}} V'$	$\{z \mid \exists\, t \in V,\ \exists\, t' \in V',$ $t[grAttr(V) \cap grAttr(V')] = t'[grAttr(V) \cap grAttr(V')]$ $z = t[grAttr(V) \cap grAttr(V')]\}$
π_{gr}-difference	$V \setminus_{\pi_{gr}} V'$	$\{z \mid \exists\, t \in V,\ \nexists\, t' \in V',$ $t[grAttr(V) \cap grAttr(V')] = t'[grAttr(V) \cap grAttr(V')]$ $z = t[grAttr(V) \cap grAttr(V')]\}$
π_{gr}-equality	$t =_{\pi_{gr}} t'$	$t \in V,\ t' \in V',$ $t[grAttr(V) \cap grAttr(V')] = t'[grAttr(V) \cap grAttr(V')]$

Table 2.7: π- and π_{gr}-operators on the Common Subset of Attributes.

relation R used in primitive clauses of the WHERE clause of the view V which have the evolution parameters set to d and r, respectively. These notations and others related to the attributes of a relation R used in a view V are summarized in Table 2.5.

2.5.2 *Legal Rewriting*: The Semantics for View Synchronization

In *EVE* we support the following six capability changes:

- "*add-attribute* $IS.R.A$" - the relation R at the site IS was extended with a new attribute A.

- "*change-attribute-name* $IS.R.A,\ IS.R.A'$" - the attribute A of the base relation R residing at the site IS changed its name into A'.

- "*delete-attribute* $IS.R.A$" - the attribute A was dropped from the relation R residing at the site IS.

- "*add-relation* $IS.R$" - the new relation R was added at the site IS.

- "*change-relation-name* $IS.R,\ IS.R'$" - the base relation R residing at the site IS changed its name into R'.

- "*delete-relation* $IS.R$" - the relation R was dropped at the site IS.

In Section 3.2 we present the general EVE synchronization algorithm which treats differently each capability change. From the above capability changes only "$delete\!-\!attribute\ R.A$" and "$delete\!-\!relation\ R$" can potentially make obsolete views defined using the attribute $R.A$ or relation R, thus alternative view rewritings must be found for the affected views.

As we will see in this dissertation (Chapters III and IV), the algorithms for view synchronization could change other components of the original view definition besides the deleted components, i.e., other indirectly affected components. For example, replacing an attribute $R.A$ after the capability change "$delete\!-\!attribute\ R.A$" could result in substituting the entire relation R by some other relation S (even though the relation R without the attribute A is still available). Thus, we distinguish between two types of components of an affected view: the ones that are directly related to the deleted component (we called them *affected components*) and those that are not affected by a capability change (*unaffected components*).

Namely, for the capability change "*delete−attribute* $R.A$" with $R.A$ in the SELECT or WHERE clause of the view V, we have:

- The attribute $R.A$ in the SELECT clause of V (if $R.A \in Attr(V)$) is a ***directly affected* component**;
- An aggregation attribute $f(X)$ in the SELECT clause of V with the expression X referring to the attribute $R.A$, is a ***directly affected* component**;
- The relation R in the FROM clause of V is an ***indirectly affected* component**;
- All attributes of relation R other than $R.A$ that are used in the SELECT clause of V are ***indirectly affected* components**;
- All conditions using the attribute $R.A$ in the WHERE clause of V are ***directly affected* components**;
- All conditions using attributes of relation R (other than $R.A$) in the WHERE clause of V are ***indirectly affected* components**.

And for the capability change "*delete−relation* R" with R in the FROM clause of the view V, we have:

- The relation R in the FROM clause of V is a ***directly affected* component**;
- All components in the SELECT clause of V referring to attributes of the relation R are ***directly affected* components**;
- All conditions using attributes of relation R in the WHERE clause of V are ***directly affected* components**.

Definition 1 *Let ch be a capability change and the E-SQL view V. Let MKB and MKB' be the state of the meta knowledge base containing the IS descriptions right before and right after the change ch, respectively*[5]*. We say that a view V' is a* ***legal rewriting*** *of the view V under capability change ch if the following properties hold:*

P1. The view V' is ***no longer affected*** *by the change ch, i.e., V' has no directly affected components.*

P2. The view V' can be ***evaluated*** *in the new state of the information space (i.e., V' contains only elements defined in MKB').*

P3. The ***view extent parameter*** *$\mathcal{VE} = \delta$ ($\delta \in \{\equiv, \supseteq, \subseteq\}$) of V (Fig. 2.6) is satisfied by the view V'. I.e., $V' \ \delta_{\pi_{gr}} \ V$*[6] *is true for any state of the underlying information sources (δ_π is the π-set operator corresponding to δ as defined in Table 2.7). The relationships between two sets is defined using set semantics (i.e., we don't take into account duplicates).*

P4. All ***evolution parameters*** *attached to the view components such as attributes, relations or conditions of the view V are satisfied by the view V'*[7]*.*

The property P4 from Definition 1 states that any legal rewriting V' of the view V must preserve all indispensable components. For example, all indispensable attributes from the SELECT clause (i.e., the ones having $\mathcal{AD} = false$ in V) of the view V must appear in the SELECT clause of the view V' as well. The semantics of the evolution parameters impose that all indispensable 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

[5]We assume that the meta knowledge base MKB is evolved into MKB' to reflect the change *ch* using the approach described in technical report ([47]). This evolution of the MKB is relatively straightforward, and hence a description of it is omitted here.

[6]Note that $\delta_{\pi_{gr}}$ is equivalent to δ_π if the view V is not an aggregation view. I.e., if V is not an aggregation view, then we have to prove $V' \ \delta_\pi \ V$.

[7]See [47] for rules of how evolution parameters are set for legal rewritings.

capability change, all *directly affected* components *must* be replaced or dropped in order for the view definition to satisfy the evolution parameters. Definition 2 introduces the concept of *evolvable* E-SQL view, one of the essential concepts used in this work.

Definition 2 *An E-SQL view V is said to be* ***evolvable under the capability change*** *ch iff there exists a state of the MKB such that there exists a legal rewriting V' conforming to the Definition 1.*

The following theorem establishes a necessary (but not sufficient) condition imposed to the evolution parameters of the view V for the view to be **evolvable**.

Theorem 1 *If the view V is evolvable under a capability change ch then all the directly affected components are not both indispensable and nonreplaceable, i.e., their evolution parameters are not* $(\mathcal{XD} = false, \mathcal{XR} = false)$.

Proof. Prove by contradiction. Suppose V has a directly affected component C with the evolution parameters $(\mathcal{XD} = false, \mathcal{XR} = false)$. And suppose, there exists a view V', a legal rewriting of it when the state of the meta-knowledge base is MKB. From property $P4$ of the Definition 1, we have that the component C must be preserved unchanged in the view V' (it is indispensable and nonreplaceable). But C was a directly affected component for V and capability change ch.

If ch = "*delete-attribute* $R.A$", then C is a component in the SELECT clause using the attribute $R.A$ or a condition in the WHERE clause referring to $R.A$. In both cases, C cannot be left unchanged in V' as it will continue to be a directly affected component which contradicts $P1$ of the Definition 1.

If ch = "*delete-relation* R", then C is a component in the SELECT clause using an attribute $R.A$ of R, a condition in the WHERE clause referring to an attribute $R.B$

of R or the relation R in the FROM clause. In all cases, C cannot be left unchanged in V' as it continues to be a directly affected component in V' under the capability change "*delete-relation* R". This contradicts $P1$ of the Definition 1.

Q.E.D.

Definition 3 specifies how the evolution parameters are set for the new view definition V'.

Definition 3 ***Evolution Parameter Assignment.*** *When a view component X' is used to replace an affected view component X, the evolution parameters associated with X' are set by the following rules:*

- *Rule 1.* *If X' is used to replace exactly one view component X of the original view V, the new evolution parameters are set to be the same as those of the original component X. If a view component is replaced by more than one new view component, we say that each of the new view components replaces exactly one view component and this Rule 1 applies for each of the new view components.*

- *Rule 2.* *If a new view component X' is used to replace more than one view component of the original view $X_1(\mathcal{XD} = val_{1,1}, \mathcal{XR} = val_{1,2})$, ..., $X_k(\mathcal{XD} = val_{k,1}, \mathcal{XR} = val_{k,2})$ where $\mathcal{XD}$ and $\mathcal{XR}$ are view dispensable and replaceable evolution parameters, respectively, and $val_{i,j} \in \{true, false\}$ their values. We set the evolution parameters of components X' as:*

  $(\mathcal{XD} = val_{1,1}$ AND $\cdots$ AND $val_{k,1},\ \mathcal{XR} = val_{1,2}$ AND $\cdots$ AND $val_{k,2})$.

Definition 1 gives the general semantics the evolution parameters in an E-SQL view definition impose on the view evolution process. Conforming with Definitions

1 and 3, a view V evolved in V' with some unaffected components dropped would be considered legal as long as *all* evolution parameters of V are satisfied and all evolution parameters of V' are appropriately set.

Example 7 *The view given in Equation (2.16)*[8] *could be legally evolved under the capability change "delete-attribute R.A" in different ways: (1) dropping the directly affected component R.A and leaving anything else unchanged (Equation (2.17)); and (2) dropping the directly affected component R.A and, as well, dropping the unaffected component R.C (Equation (2.18)).*

$$\begin{array}{ll} \textit{CREATE VIEW} & V(\mathcal{VE} = \subseteq) \ \textit{AS} \\ \textit{SELECT} & R.A(\mathcal{AD} = true, \mathcal{AR} = true), \\ & R.B(\mathcal{AD} = true) \\ & R.C(\mathcal{AD} = true) \\ \textit{FROM} & R(\mathcal{RR} = true) \\ \textit{WHERE} & R.B > 200 \end{array} \tag{2.16}$$

$$\begin{array}{ll} \textit{CREATE VIEW} & V_1(\mathcal{VE} = \subseteq) \ \textit{AS} \\ \textit{SELECT} & R.B(\mathcal{AD} = true) \\ & R.C(\mathcal{AD} = true) \\ \textit{FROM} & R(\mathcal{RR} = true) \\ \textit{WHERE} & R.B > 200 \end{array} \tag{2.17}$$

$$\begin{array}{ll} \textit{CREATE VIEW} & V_2(\mathcal{VE} = \subseteq) \ \textit{AS} \\ \textit{SELECT} & R.B(\mathcal{AD} = true) \\ \textit{FROM} & R(\mathcal{RR} = true) \\ \textit{WHERE} & R.B > 200 \end{array} \tag{2.18}$$

Because view synchronization algorithms primarily are concerned with affected components, we make a distinction between legal view rewritings that are obtained

[8]Default values for evolution parameters are not shown in the definitions.

by modifying only affected components (leaving all unaffected components intact), called *base rewritings*, and those that have unaffected components modified as well (e.g., replaced or dropped), called *derived rewritings.* Namely, in base rewritings all "linked" affected components are either all *replaced*, all *dropped* or all *unchanged.* For example, if we find a replacement for an attribute $R.A$, then a rewriting is called a base rewriting if **all** occurrences of $R.A$ in replaceable components are replaced (even though the evolution parameters would have allowed for example to drop a condition referring to $R.A$). If a relation S replaces a relation R (i.e., relation S has replacements for some attributes of R) then a base rewriting must have preserved all attributes that could be replaced from S. Formally, we give the following definitions for base and derived rewritings.

Definition 4 ***Base View Rewriting.*** *Let a view V' be a rewriting of the view V after a "delete−relation" or "delete−attribute" capability change. We say that V' is a* ***base view rewriting*** *if*

B0. V' is a legal rewriting of V (by Definition 1).

B1. All unaffected view components of V are still in V'.

B2. If an (directly or indirectly) affected attribute $R.A$ in V has a replacement attribute $S.B$ in V', then any replaceable component referring to $R.A$ (i.e., an attribute in the SELECT *clause or a condition in the* WHERE *clause of V) must appear in V' with $R.A$ replaced by $S.B$.*

B3. If an indirectly affected relation R is still in the FROM *clause of the view V', then all its indirectly affected attributes that are referred to in V and are not replaced in V' must appear in V' as well.*

Definition 5 ***Derived View Rewriting.*** *Let a view V' be a base rewriting of the view V by Definition 4. Then a view V'' is a* ***derived rewriting*** *of the view V' iff:*

D0. V'' is a legal rewriting of V (by Definition 1).

D1. V'' is obtained from V' by dropping dispensable components (affected or unaffected components of V), i.e., components with dispensable evolution parameters set to true ($\mathcal{XD} = true$).

In the following we give examples of base and derived rewritings for different replacement strategies and capability changes.

Example 8 *Let V be defined by the Equation (2.19). Given the capability change "delete-attribute R.A", we give some examples of base (Equations (2.20),(2.21)) and derived (Equations (2.22),(2.23),(2.24)) rewritings of the view V.*

$$\begin{array}{ll} \textsf{CREATE VIEW} & V(\mathcal{VE} = \supseteq) \textsf{ AS} \\ \textsf{SELECT} & R.A(\mathcal{AD} = true, \mathcal{AR} = true), \\ & R.B(\mathcal{AD} = false, \mathcal{AR} = false), \\ & R.C(\mathcal{AD} = true, \mathcal{AR} = false) \\ \textsf{FROM} & R(\mathcal{RD} = false, \mathcal{RR} = true) \\ \textsf{WHERE} & (R.B\ \theta\ value)(\mathcal{CD} = true, \mathcal{CR} = false) \end{array} \tag{2.19}$$

Legal Base Rewritings of the view V.

The affected component R.A is dropped. Since R.A is no longer in R and it is dispensable in V, dropping it from the view gives a base rewriting (one can verify that all the above conditions B0, B1 and B3 are satisfied). Therefore, we get a legal base rewriting V' (V' is π-equivalent to V, i.e., $V' \equiv_\pi V$):

$$\begin{array}{ll} \textsf{CREATE VIEW} & V'(\mathcal{VE} = \supseteq) \textsf{ AS} \\ \textsf{SELECT} & R.B(\mathcal{AD} = false, \mathcal{AR} = false), \\ & R.C(\mathcal{AD} = true, \mathcal{AR} = false) \\ \textsf{FROM} & R(\mathcal{RD} = false, \mathcal{RR} = true) \\ \textsf{WHERE} & (R.B\ \theta\ value)(\mathcal{CD} = true, \mathcal{CR} = false) \end{array} \tag{2.20}$$

The affected component $R.A$ is replaced with $S.A'$ using a join relation between R and S. If $\pi_{S.A',S.B'}S = \pi_{R.A,R.B}R$, then the view V'' defined in Equation (2.21) is a legal based rewriting (e.g., $V'' \equiv_\pi V$), i.e., all the properties from Definition 4 are satisfied[9].

$$\begin{array}{ll} \text{CREATE VIEW} & V''(\mathcal{VE} = \supseteq) \ \text{AS} \\ \text{SELECT} & \underline{S.A'(\mathcal{AD} = true, \mathcal{AR} = true)}, \\ & \underline{R.B(\mathcal{AD} = false, \mathcal{AR} = false)}, \\ & R.C(\mathcal{AD} = true, \mathcal{AR} = false) \\ \text{FROM} & \underline{S(\mathcal{RD} = true, \mathcal{RR} = true)}, R(\mathcal{RD} = false, \mathcal{RR} = true) \\ \text{WHERE} & \underline{(S.B' = R.B)(\mathcal{CD} = true, \mathcal{CR} = true)} \text{AND} \\ & (R.B\ \theta\ value)(\mathcal{CD} = true, \mathcal{CR} = false) \end{array} \tag{2.21}$$

Legal Derived Rewritings of the view V.

The base rewriting V' defined in Equation (2.20) has three derived rewritings: one in which the attribute $R.C$ is dropped from the SELECT *clause (its dispensable evolution parameter is $\mathcal{AD} = true$) (Equation (2.22)); one in which the condition referring the attribute $R.B$ is dropped (its dispensable evolution parameter is $\mathcal{CD} = true$) (Equation (2.23)); and one in which both of these two components are dropped (Equation (2.24)).*

$$\begin{array}{ll} \text{CREATE VIEW} & V_1'(\mathcal{VE} = \supseteq) \ \text{AS} \\ \text{SELECT} & R.B(\mathcal{AD} = false, \mathcal{AR} = false), \\ \text{FROM} & R(\mathcal{RD} = false, \mathcal{RR} = true) \\ \text{WHERE} & (R.B\ \theta\ value)(\mathcal{CD} = true, \mathcal{CR} = false) \end{array} \tag{2.22}$$

$$\begin{array}{ll} \text{CREATE VIEW} & V_2'(\mathcal{VE} = \supseteq) \ AS \\ \text{SELECT} & R.B(\mathcal{AD} = false, \mathcal{AR} = false), \\ & R.C(\mathcal{AD} = true, \mathcal{AR} = false) \\ \text{FROM} & R(\mathcal{RD} = false, \mathcal{RR} = true) \end{array} \tag{2.23}$$

[9] View synchronization algorithms for "*delete−attribute*" are given in Chapters III and IV.

$$\begin{array}{ll} \textsf{CREATE VIEW} & V'(\mathcal{VE} = \supseteq)\ \textsf{AS} \\ \textsf{SELECT} & R.B(\mathcal{AD} = false, \mathcal{AR} = false), \\ \textsf{FROM} & R(\mathcal{RD} = false, \mathcal{RR} = true) \end{array} \tag{2.24}$$

All three rewritings are derived rewritings, i.e., the properties from Definition 5 are satisfied. Note that they don't satisfy conditions imposed to qualify as base rewritings (by Definition 4). For example, the derived rewriting V_1' defined in Equation (2.22) has the attribute R.C available in the relation R from the FROM *clause, but it gets dropped from the* SELECT *clause thus violating condition B3 from Definition 4.*

2.6 Finding Strongest E-SQL View Definition: Resolution of Dependencies between E-SQL Evolution Parameter Settings

Our view definition language E-SQL allows any combination of the evolution parameters to be set for the view components. Because of the close relationship between components, e.g., an attribute in the SELECT clause is coming from a relation in the FROM clause, the synchronization cannot take advantage of the apparent full flexibility of all possible evolution parameter combinations. For example, consider a view V having an attribute $R.A(\mathcal{AD} = false, \mathcal{AR} = false)$ in the SELECT clause and the relation $R(\mathcal{RD} = true, \mathcal{RR} = true)$ in the FROM clause. Even though the parameters associated with the relation apparently permit it, the view could never be synchronized by dropping or replacing the relation R. Both of these actions would not satisfy the evolution parameters for the attribute $R.A$ (even though they are allowed by the evolution parameters of the relation R). Thus, we can safely assume that the view definition V' having the component $R(\mathcal{RD} = false, \mathcal{RR} = false)$ in the

FROM clause is "synch-equivalent" to the original view. Synch-equivalence concept is defined to mean that any synchronization algorithm that fails for V' would have failed for V as well; and any algorithm that succeeds for V' would have succeeded for V as well.

In the following we give some criteria of how to find a "synch-equivalent" definition of a view V that expresses the real evolution flexibility of the initial view definition. Finding the synch-equivalent definitions is essential for helping the view definer to understand the semantics associated with a view definition[10] (it makes the implicit evolution semantics explicit).

Definition 1 *We say that two views V_1 and V_2 are **synch-equivalent** if they differ only in the evolution parameters of their components and for any state of the MKB and a change ch, any synchronization algorithm finds the same set of base rewritings (Definition 4) for the two views.*

Definition 2 *Over the set of all evolution parameters of a component C of a view V, $\{(\mathcal{XD} = false, \mathcal{XR} = false), (\mathcal{XD} = true, \mathcal{XR} = false), (\mathcal{XD} = false, \mathcal{XR} = true), (\mathcal{XD} = true, \mathcal{XR} = true)\}$ (with $\mathcal{X}$ one of $\mathcal{A}$, $\mathcal{R}$ or $\mathcal{C}$ for attribute, relation or condition component, respectively) we introduce a partial order **"stronger than"**, denoted by $\preceq$. We say that one value is "stronger than" another value, i.e., $(\mathcal{XD} = d_1, \mathcal{XR} = r_1) \preceq (\mathcal{XD} = d_2, \mathcal{XR} = r_2)$ if and only if for a view definition V_1 with the evolution parameters for the component C $(\mathcal{XD} = d_1, \mathcal{XR} = r_1)$ and a view definition V_2 identical to V_1 but having the evolution parameters for the component C $(\mathcal{XD} = d_2, \mathcal{XR} = r_2)$, any base rewriting (Definition 4) found by a synchronization*

[10] The process of defining a view should be an interactive process: the user defines an E-SQL view, EVE finds the synch-equivalent definition, the user decides if this is what he meant or not with the process continuing until an agreement is reached.

algorithm for the view V_1 is also found by the same algorithm for the view V_2, for any state of the MKB and capability change ch.

Note that the relation "stronger than" is indeed a partial order: it is reflexive (any pair of evolution parameters is "stronger than" itself by Definition 2), transitive and antisymmetric. From Definition 2, it is easy to see that "stronger than" inequalities from Equations (2.25) and (2.26) are true and they define a partial order over the set of all evolution parameter values for a component C.

$$(\mathcal{XD} = false, \mathcal{XR} = false) \leq (\mathcal{XD} = true, \mathcal{XR} = false) \leq (\mathcal{XD} = true, \mathcal{XR} = true) \tag{2.25}$$

$$(\mathcal{XD} = false, \mathcal{XR} = false) \leq (\mathcal{XD} = false, \mathcal{XR} = true) \leq (\mathcal{XD} = true, \mathcal{XR} = true). \tag{2.26}$$

In Equation (2.25), the evolution parameters $(\mathcal{XD} = false, \mathcal{XR} = false)$ is stronger than the evolution parameter $(\mathcal{XD} = true, \mathcal{XR} = false)$ given that all the rewritings of a view V_1 with a component $C(\mathcal{XD} = false, \mathcal{XR} = false)$ must contain this component unchanged, while the rewritings for a view V_2 differing from V_1 only in the component $C(\mathcal{XD} = true, \mathcal{XR} = false)$ could preserve C or drop it. Thus, any view synchronization algorithm applied to the view V_2 finds all the rewritings corresponding to the view V_1 plus possibly some new rewritings when C is dropped.

Definition 3 *We say that the view V_1 is **"stronger than"** the view V_2, denoted by $V_1 \leq V_2$ if (1) they differ only in the evolution parameters of their components and (2) for any component $C(\mathcal{XD} = d_1, \mathcal{XR} = r_1)$ of the view V_1 and its corresponding component $C(\mathcal{XD} = d_2, \mathcal{XR} = r_2)$ in V_2, we have $(\mathcal{XD} = d_1, \mathcal{XR} = r_1) \leq (\mathcal{XD} = d_2, \mathcal{XR} = r_2)$.*

Our goal is to find the "strongest" view definition that is synch-equivalent to the initial view. This then represents exactly the evolution semantics any view synchronization algorithm attaches to the original view definition (even if not stated explicitly). In other words, an E-SQL view definition is at all times only evolved according to the evolution parameters of its strongest view definition.

Definition 4 *We say that V' is the* ***strongest synch-equivalent*** *definition of the view V iff (1) V' is synch-equivalent to V, (2) $V' \leq V$ and (3) there exists no other view V'' such that $V'' \leq V'$ and V'' is synch-equivalent to V.*

In the Tables 2.8, 2.9, 2.10 and 2.11[11] we give the transformation rules for changing a view definition into a synch-equivalent one. Any transformation applied to a view V gives a "stronger" view definition and at the same time a synch-equivalent one.

$\mathcal{S}(\mathcal{AD},\mathcal{AR})$ \ $(\mathcal{C} = R.A\ \theta\ const)(\mathcal{CD},\mathcal{CR})$	$R.A(false, false)$	$R.A(true, false)$	$R.A(false, true)$	$R.A(true, true)$
$\mathcal{C}(false, false)$	-	$R.A(false, false)$	$R.A(false, false)$	$R.A(false, false)$
$\mathcal{C}(true, false)$	$\mathcal{C}(false, false)$	-	-	-
$\mathcal{C}(false, true)$	$\mathcal{C}(false, false)$	-	-	$R.A(false, true)$
$\mathcal{C}(true, true)$	$\mathcal{C}(false, false)$	-	$\mathcal{C}(false, true)$	-

Table 2.8: Transformation rules for a condition ($\mathcal{C} = R.A\ \theta\ const$) in WHERE clause and a component $\mathcal{S}$ using $R.A$ in SELECT clause.

Example 9 *To illustrate some of the transformation rules, let's consider the view V defined by Equation 2.27.*

$$\begin{array}{ll} \textsf{CREATE VIEW} & V\ \textsf{AS} \\ \textsf{SELECT} & X.A(\mathcal{AD} = true, \mathcal{AR} = true), Y.A(\mathcal{AD} = true, \mathcal{AR} = true), \ldots \\ \textsf{FROM} & X, Y, \ldots \\ \textsf{WHERE} & (X.A > Y.A(\mathcal{CD} = false, \mathcal{CR} = true)), \ldots \end{array} \tag{2.27}$$

[11]Note that in all the definitions related to the strongest synch-equivalent definition associated to a view, we are talking about base rewriting concept defined in Definition 4.

$\mathcal{S}(\mathcal{AD},\mathcal{AR})$ \ $R(\mathcal{RD},\mathcal{RR})$	$\mathcal{S}(false,false)$	$\mathcal{S}(true,false)$	$\mathcal{S}(false,true)$	$\mathcal{S}(true,true)$
$R(false,false)$	-	-	-	-
$R(true,false)$	$R(false,false)$	-	-	-
$R(false,true)$	$R(false,false)$	-	-	-
$R(true,true)$	$R(false,false)$	-	-	-

Table 2.9: Transformation rules for a relation R in FROM clause and a component $\mathcal{S}$ using an attribute $R.A$ in SELECT clause.

$\mathcal{C}(\mathcal{CD},\mathcal{CR})$ \ $R(\mathcal{RD},\mathcal{RR})$	$\mathcal{C}(false,false)$	$\mathcal{C}(true,false)$	$\mathcal{C}(false,true)$	$\mathcal{C}(true,true)$
$R(false,false)$	-	-	-	-
$R(true,false)$	$R(false,false)$	-	-	-
$R(false,true)$	$R(false,false)$	-	-	-
$R(true,true)$	$R(false,false)$	-	-	-

Table 2.10: Transformation rules for a relation R in FROM clause and a condition $\mathcal{C}$ in WHERE clause using an attribute $R.A$.

This case corresponds to column three, row ten in the Table 2.11. If the attribute $X.A$ is deleted, the affected condition in the WHERE clause $C = (X.A > Y.A(\mathcal{AD} = false, \mathcal{AR} = true))$ must be replaced given its evolution parameters. Thus a replacement for the deleted attribute $X.A$ must be found and used in any legal base rewriting V'. Then by Definition 4, all occurrences of the attribute $X.A$ must be substituted by this replacement in V', including the component $X.A(\mathcal{AD} = true, \mathcal{AR} = true)$ in the SELECT clause. Thus, regardless how the replacements are found, any legal base rewriting must have the component $X.A(\mathcal{AD} = true, \mathcal{AR} = true)$ or a replacement of it in the SELECT clause. This behavior never takes advantage of the evolution parameter "$AD = true$", and it is consistent to having "$AD = false$" instead. This explains why the transformation rule from column three, row ten in the Table 2.11 changes the evolution parameters for the component $X.A(\mathcal{AD} = true, \mathcal{AR} = true)$

$\mathcal{C}(\mathcal{CD},\mathcal{CR})$ \ $(\mathcal{X}(\mathcal{AD},\mathcal{AR}),\mathcal{Y}(\mathcal{AD},\mathcal{AR})$ $(\mathcal{X},\mathcal{Y}) \in \{(\mathcal{S}_1,\mathcal{S}_2),(\mathcal{S}_2,\mathcal{S}_1)\}$	$\mathcal{C}(false,false)$	$\mathcal{C}(true,false)$	$\mathcal{C}(false,true)$	$\mathcal{C}(true,true)$
$\mathcal{X}(false,false)$ $\mathcal{Y}(false,false)$	-	$\mathcal{C}(false,false)$	$\mathcal{C}(false,false)$	$\mathcal{C}(false,false)$
$\mathcal{X}(false,false)$ $\mathcal{Y}(true,false)$	$\mathcal{Y}(false,false)$	-	-	-
$\mathcal{X}(false,false)$ $\mathcal{Y}(false,true)$	$\mathcal{Y}(false,false)$	-	-	$\mathcal{C}(false,true)$
$\mathcal{X}(false,false)$ $\mathcal{Y}(true,true)$	$\mathcal{Y}(false,false)$	-	$\mathcal{Y}(false,true)$	-
$\mathcal{X}(true,false)$ $\mathcal{Y}(true,false)$	$\mathcal{X}(\mathcal{Y})(false,false)$	-	-	-
$\mathcal{X}(true,false)$ $\mathcal{Y}(false,true)$	$\mathcal{X}(\mathcal{Y})(false,false)$	-	-	-
$\mathcal{X}(true,false)$ $\mathcal{Y}(true,true)$	$\mathcal{X}(\mathcal{Y})(false,false)$	-	$\mathcal{Y}(false,true)$	-
$\mathcal{X}(false,true)$ $\mathcal{Y}(false,true)$	$\mathcal{X}(\mathcal{Y})(false,false)$	-	-	$\mathcal{C}(false,true)$
$\mathcal{X}(false,true)$ $\mathcal{Y}(true,true)$	$\mathcal{X}(\mathcal{Y})(false,false)$	-	$\mathcal{Y}(false,true)$	-
$\mathcal{X}(true,true)$ $\mathcal{Y}(true,true)$	$\mathcal{X}(\mathcal{Y})(false,false)$	-	$\mathcal{X}(false,true)$ $\mathcal{Y}(false,true)$	-

Table 2.11: Transformation rules for two components $\mathcal{S}_1$ and $\mathcal{S}_2$ using the attributes $R_1.A$ and $R_2.B$ in the SELECT clause a condition $\mathcal{C}$ in WHERE clause using $R_1.A$ and $R_2.B$.

to $X.A(\underline{\mathcal{AD} = false}, \mathcal{AR} = true)$.

To find the strongest synch-equivalent view for a given view definition, we apply the transformation rules for all pairs of related components, until no more modifications could be done.

Theorem 2 *By applying the above transformation rules to an E-SQL view V until no more rules can be applied, the strongest synch-equivalent view definition is obtained.*

Theorem 3 *The strongest synch-equivalent view definition V' of a view V can be obtained by a sequence of transformations defined in the Tables 2.8, 2.9, 2.10 and 2.11.*

Proof The proofs for Theorems 2 and 3 are based on the fact that by applying any transformation rule from Tables 2.8, 2.9, 2.10 and 2.11 to a view definition V, a synch-equivalent view definition V_{st} is obtained with its evolution parameters stronger than those of V. The process of applying the transformation rules must end after a finite number of transformations given that for each evolution parameter $\mathcal{P}$, there exists only a finite number of evolution parameters stronger than $\mathcal{P}$.

Definition 5 *We call two evolution parameter values $\mathcal{P}_1$ and $\mathcal{P}_2$ of a component C incompatible if for any view V having the component C, there don't exist two synch-equivalent definitions V_1 and V_2 stronger than V with the evolution parameters of the component C set to $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively.*

For example, evolution parameters $(\mathcal{XD} = false, \mathcal{XR} = true)$ and $(\mathcal{XD} = true, \mathcal{XR} = false)$ are incompatible: there are no transformation rules that could be applied to an original view definition for a component C such that the two evolution parameters are obtained in stronger synch-equivalent definitions. From our transformation rules, we can conclude that the only incompatible evolution parameters are $(\mathcal{XD} = false, \mathcal{XR} = true)$ and $(\mathcal{XD} = true, \mathcal{XR} = false)$; and $(\mathcal{XD} = true, \mathcal{XR} = true)$ and $(\mathcal{XD} = true, \mathcal{XR} = false)$.

Given the above transformation rules and Theorems 2 and 3, we can now prove that there is only one minimal synch-equivalent definition for any E-SQL view.

Theorem 4 *The strongest synch-equivalent definition is unique for an E-SQL view V.*

Proof. Let's assume that there are two view definitions V_{st1} and V_{st2} qualified to be the strongest synch-equivalent for the same view V such that neither $V_{st1} \leq V_{st2}$ nor $V_{st2} \leq V_{st1}$ hold. Two strongest view definitions of the same view V cannot be compared because a component has in the two views incompatible parameters; or because not all parameters in one of the view are "stronger than" the corresponding parameters in the other view (as required by Definition 3.) Formally, we can have two cases: (I) there must exist a component C in V such that the evolution parameters $\mathcal{P}_{st1}$ of C in V_{st1} is incompatible with the evolution parameters $\mathcal{P}_2$ of C in V_{st2}; or (II) there exist two components C_1 and C_2 in V with the evolution parameters are set to $P_{1,1}$ and $P_{2,1}$ for C_1 in V_{st1} and V_{st2}, respectively; and the evolution parameters are set to $P_{1,2}$ and $P_{2,2}$ for C_2 in V_{st1} and V_{st2}, respectively.

In case (I), the only values of evolution parameters that are incompatible are $(\mathcal{XD} = false, \mathcal{XR} = true)$ and $(\mathcal{XD} = true, \mathcal{XR} = false)$; and $(\mathcal{XD} = true, \mathcal{XR} = true)$ and $(\mathcal{XD} = true, \mathcal{XR} = false)$. Let's assume that $\mathcal{P}_{st1} = (\mathcal{XD} = false, \mathcal{XR} = true)$ and $\mathcal{P}_2 = (\mathcal{XD} = true, \mathcal{XR} = false)$. The only way the component C of V could be transformed to have $\mathcal{P}_1$ and $\mathcal{P}_2$ in minimal synch-equivalent definitions, is for it to have the evolution parameter $(\mathcal{XD} = true, \mathcal{XR} = true)$ in the original view. From Theorem 3 we have that the views V_{st1} and V_{st2} are obtained by applying only the transformations rules from Tables 2.8, 2.9, 2.10 and 2.11 which don't contain any transformation that would change an evolution parameter in $\mathcal{P}_2 = (\mathcal{XD} = true, \mathcal{XR} = false)$. Thus, this case cannot occur while applying the transformation rules. Case (II) could be proven never to happen using the transformation rules given in Tables 2.8, 2.9, 2.10 and 2.11.

Q.E.D.

2.7 Summary

In this Chapter we propose the Evolvable View Environment (*EVE*) architecture as a generic framework within which to solve view adaptation when underlying ISs change their capabilities. The main contributions of this introductory chapter are:

- The characterization of an open new problem with current view technology in the context of dynamic large-scale environments such as the WWW, which we coin the *view synchronization problem.*
- The development of a general solution approach (and architecture), called the *EVE* framework, for addressing this view evolution problem based on the concept of view synchronization.
- The proposal of an extended view definition language, called E-SQL, that is capable of defining flexible SPJ and aggregation views by incorporating view change preferences into the view definition.
- The design of an IS description model, called MISD, that can capture capabilities of diverse ISs, and thus serves as foundation for the view synchronization process.
- The development of formal foundations for view evolution and correctness criteria for the replacement of affected components of a view definition with alternate components.
- The specification of the concept of *synch-equivalence* between E-SQL specifications to express the real evolution flexibility of an E-SQL view definition, i.e., making the implicit semantics explicit.
- The outline of a general strategy for finding the strongest synch-equivalent E-SQL definition for a given view specification, and prove this transformation process to always generate the unique strongest solution.

CHAPTER III

View Synchronization Algorithms based on Containment Information

In Chapter I we identify a new view evolution problem, namely *view synchronization*, by establishing a taxonomy of view evolution problems for views defined in data warehouses. The proposed taxonomy identifies alternate dimensions of the problem space, and hence serves as a framework for characterizing and distinguishing our view synchronization problem from other (previously studied) view adaptation problems, such as maintenance after data updates [16] or after view redefinition [17, 37].

While we assume that the evolution of the affected view definitions is triggered by capability changes of ISs, previous work, e.g., by Gupta et al. [17] and Mohania et al. [37], typically assumed that view redefinition was explicitly requested by the user at the view site. Hence, previous work on view redefinition did not deal with the problem of how to salvage the affected view definitions themselves (at the schema level) but only dealt with efficiently managing changes at the data level. Our problem and solution thus is complimentary to work by others as once our *EVE* (Evolvable View Environment) system has determined an acceptable view redefinition then algorithms proposed by others [17, 37] on how most efficiently to maintain the materialized view (i.e., at the data level) could be applied to our system.

In previous Chapter II we set forth the *EVE* approach by introducing the overall *EVE* solution framework. One key contribution of our *EVE* solution [60, 30, 40] is the design of 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. E-SQL allows the view definer to express preferences about the view evolution process, i.e., about the the criticality and dispensability of the different components of a view definition. A second contribution of the *EVE* approach is the design of the model for information source description (MISD) [45, 44, 30], for capturing the capabilities of each IS as well as the interrelationships between ISs, such as containment constraints, join constraints, etc. These MISD descriptions then can be exploited when searching for an appropriate substitution for the components of views that are affected by an IS capability change. E-SQL and the MISD then represent the basis for our strategies for rewriting views based on changing IS. Our proposed view rewriting process, i.e., *view synchronization*, is designed to find a view redefinition that meets all view preservation constraints specified by the original E-SQL view definition. That is, it identifies and extracts appropriate information from other ISs as replacement of the affected components of the view and produces an alternative view definition. This selection effectively corresponds to a query planning process that extracts knowledge from the MKB, such as when two relations can be joined or information kept in one IS is a subset of information kept in a second IS site.

In this chapter, we now present the first view synchronization algorithms that require only the availability of *containment information* among information sources in order to evolve views. Unlike the strategies proposed for query rewriting using views in the database literature ([33, 61]), our techniques address three new issues:

(1) finding view rewritings that are not necessarily *equivalent* to the original view definition, (2) preserving only indispensable attributes from the SELECT clause if preserving all is not possible; and (3) using semantic *containment information* for replacing the deleted view components. (1) and (2) are possible because our E-SQL view definition language allows view users to explicitly define evolution preferences such as what attributes are to be kept in the SELECT clause (Section 2.4). (3) is possible because the containment information, expressed as MISD constraints (Section 2.3), models possibly complex select and/or project based containment between ISs instead of just simple subset relationships among complete relations.

The first proposed technique, *PrOject Containment* view synchronization (POC) algorithm finds a replacement for an entire relation or only one of the attributes used in a view based on containment constraints between the projections of the affected relation and replacement relation. In the second algorithm, we consider the select-project containment information between the to-be-replaced relation and its replacement, thus we call it *Select-PrOject Containment* view synchronization (SPOC) algorithm. We present the algorithms and give formal proofs that the rewritings found are consistent with our semantics for legal view rewritings.

The remainder of this chapter is structured as follows. In Section 3.1 we characterize the problem space for the view synchronization and describe the coordinates for which we give solutions to this problem in this dissertation. Sections 3.2 present the general view synchronization process in *EVE* for all the capability change operators we support in *EVE*. Sections 3.4 and 3.5 give the specific POC and SPOC algorithms, respectively, along with sufficiency conditions, and the proofs of correctness. In Section 3.6 we describe *EVE* system prototype that currently implements some of the algorithms presented in this chapter. Section 3.7 presents our conclusions.

The work presented in this chapter has been published by the author and her adviser in several papers in [48, 50] and also can be found in [46, 49].

3.1 Problem Space of View Synchronization Problem

For the *View Synchronization* problem identified in Section 1.2.3 as a new view adaptation problem, we further describe three dimensions along which the complexity of view synchronization algorithms can be characterized (Figure 3.1). Selections of design choices along these dimensions results in specific subproblems, each worth future investigation.

IS Capability Changes dimension: This corresponds to capability changes in databases such as *delete attribute*, *delete relation*, etc. Generally, these types of changes make the views derived from using these components undefined, leaving the view user with no choice but redefining the view completely. It is this problem that we address in this dissertation: what kind of extra information we must have at the view site in order to be able to redefine the view so that it survives after such changes.

Complexity of Meta Knowledge dimension: This corresponds to the description of capabilities and content of each IS plus of relationships between ISs. Such knowledge allows our system to reason about finding appropriate replacements of view components when evolving a view definition. One example of meta knowledge is that an identical duplicate of a relation R at IS_1 may also be available in another site IS_2, i.e., "$\mathrm{IS}_1.R \equiv \mathrm{IS}_2.S$". When the relation R is deleted from IS_1, the view synchronization process is able to preserve the affected view definitions that reference R by replacing $\mathrm{IS}_1.R$ with $\mathrm{IS}_2.S$. The more knowledge our system has about the information space, the higher chance it has to preserve the affected views when a capability change takes place at an IS - though of course the more complicated the algorithm for synchronization may become.

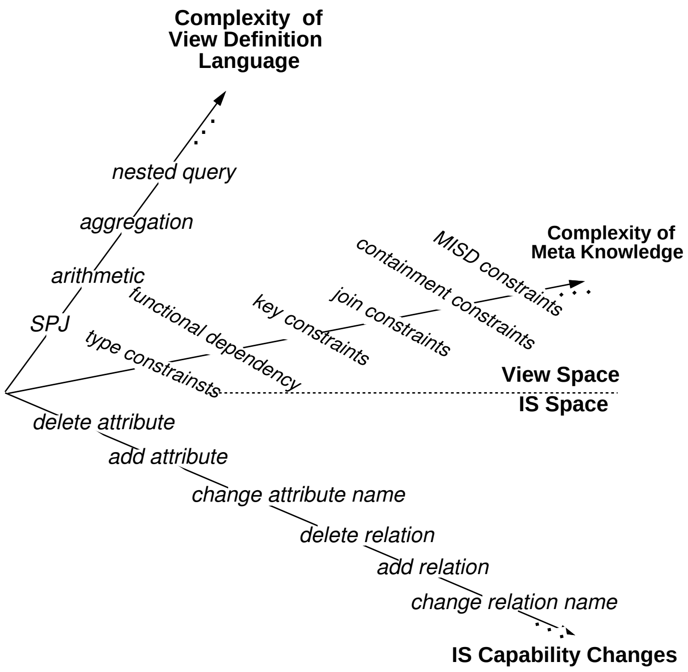


Figure 3.1: Problem Space of View Synchronization.

Complexity of View Definition Language dimension: This can be classified according to the view components allowed in the view definitions. The simplest view definition is specified by using the conventional SPJ structure with WHERE clause containing conjunctive clauses only, denoted by SPJ in Figure 3.1. More complex views allow aggregation, negation, nested queries, and so on in their view definitions. If more complex language constructs are used as part of a view definition, then it is more likely that sophisticated algorithms are required to preserve the view interface (and the view extent) under capability changes.

In this chapter in Section 3.3, we describe a one-step view synchronization algorithm that addresses the points in this problem space using the following coordinates:

(1)"*delete-attribute* " and "*delete-relation* " values on the **IS Capability Changes** dimension; (2) **Containment Constraints** value on the **Complexity of Meta Knowledge** dimension (see Section 2.3); and (3) SPJ and aggregation points on the **Complexity of View Definition Language** dimension. In the next Chapter IV we present synchronization strategies that make full use of the semantics expressed in the MISD description language that address the following coordinates from the problem space depicted in Figure 3.1. First, the coordinates include all standard capability change values from the **IS Capability Changes** dimension, i.e., *delete attribute*, *add attribute*, *change attribute-name*, *delete relation*, *add relation*, and *change relation name*. The second value *MISD constraints* is selected from the **Complexity of Meta Knowledge** dimension, and the third value *SPJ* and *aggregation* from the **Complexity of View Definition Language** dimension.

3.2 The General View Synchronization Process in EVE

While the overall view synchronization algorithm is summarized in Figure 3.2, we review here the basic steps taken for any capability change operator.

Step 1. Given a capability change *ch*, our system will first evolve the meta knowledge base (MKB) itself by detecting and modifying the affected MISD descriptions found in the MKB. Column 1 in Figure 3.3 summarizes the types of capability changes *EVE* can handle. Columns 2, 3 and 4 indicate if a given MISD integrity constraint in the MKB may need to be evolved under a capability change ($\star$ denotes yes, - denotes no).

Step 2. Given a capability change *ch* of an underlying IS, *EVE* detects if the definition of the view V is affected by *ch* when the state of the knowledge base is MKB. *EVE* will return one of three results:

<u>The General *EVE* View Synchronization Algorithm</u>

```
INPUT = capability change ch, VKB, MKB
OUTPUT = VKB', MKB'
BEGIN
VKB' = VKB
  IF ch is "change-attribute-name (R.A,R.A')" THEN
     Step 1.
            MKB' = CHANGE − ATTRIBUTE − NAME(R.A, R.A', MKB)
  IF ch is "change-relation-name (R,R')" THEN
     Step 1.
            MKB' = CHANGE − RELATION − NAME(R, R', MKB)
  IF ch is "add-attribute (R.A)" THEN
     Step 1.
            MKB' = ADD − ATTRIBUTE(R.A, MKB)
 IF ch is "add-relation (R)" THEN
     Step 1.
            MKB' = ADD − RELATION(R, MKB)
 IF ch is "delete-attribute (R.A)" or "delete-relation (R)" THEN
     Step 1.
            MKB' = DELETE − ATTRIBUTE(R.A, MKB)
              if ch is "delete-attribute (R.A)"
            MKB' = DELETE − RELATION(R, MKB)
              if ch is "delete-relation (R)"
     Step 2.
            \overline{evV} = set of evolvable, affected E-SQL views from VKB under ch
            \overline{obV} = set of non-evolvable affected E-SQL views from VKB under ch[1];
            // * delete all views in \overline{obV} from VKB' * //
            VKB' = VKB' \ \overline{obV}
     Step 3. // * find all rewritings of V * //
            FOR EACH V in \overline{evV} DO
                 rewritingsV = POC(ch, V) ∪ SPOC(ch, V) ∪ CVS(ch, V)
                 IF ch is "delete-attribute (R.A)" and
               V is evolvable under capability change ch' = "delete-relation R"
                     THEN rewritingsV = rewritingsV ∪ POC(ch', V)
                                                ∪ SPOC(ch', V) ∪ CVS(ch', V)
                 rewritingsV = rewritingsV ∪ ComputeDerived(rewritingsV, V)
                 IF rewritingsV = ∅
                     THEN // * delete V from VKB' * //
                              VKB' = VKB' \ {V}
                     ELSE V' = the best of all rewritingsV
                              // * replace V with V' in VKB' * //
                              VKB' = VKB' \ {V} ∪ {V'}
     Step 4. // * recompute the extents of the synchronized views * //
            MAINTENANCE − AFTER − SYNCH(V, V', ch, MKB, MKB')
 END
```

Figure 3.2: General View Synchronization Algorithm in the *EVE* System.

capability \ constraint change	type integrity constraint	join constraint	partial\complete info constraint
"*delete-attribute*"	$\star$	$\star$	$\star$
"*add-attribute*"	-	-	-
"*change-attribute-name*"	-	-	-
"*delete-relation* "	$\star$	$\star$	$\star$
"*add-relation*"	-	-	-
"*change-relation-name*"	-	-	-

Figure 3.3: MKB Evolution under Capability Changes: $\star$ denotes the fact that the constraint must be evolved.

- *The view definition cannot be synchronized* - the view V is affected by the change *ch* and cannot be evolved under any circumstances. This would occur if there exists at least one view component (i.e., attribute, relation or condition) affected by the change that is indispensable and nonreplaceable (Theorem 1).

- *The view definition can be synchronized* - the view V is affected by the change *ch* but it may be possible to find a legal rewriting. Note that even if the view is *evolvable* (Theorem 1) then the *EVE* system may still fail to evolve the view if replacements for affected components cannot be found.

- *The view definition is not affected by the change ch* - In this case, Step 3 below is skipped, and the view synchronization process terminates.

Step 3. For affected yet potentially evolvable views (those corresponding to Step2, second case above) we apply some view synchronization algorithm to find legal rewritings for view definitions guided by constraints imposed by the E-SQL view evolution preferences as well as by the knowledge captured in the evolved MKB.

Step 4. For all synchronized views, if they are materialized, we apply an incremental view maintenance algorithm for computing the new view extents. The algorithms for rematerialization are presented in Chapter V.

Figure 3.2 outlines the general *EVE* synchronization algorithm for all capability changes supported in *EVE*. The functions *CHANGE-ATTRIBUTE-NAME* (line 4), *CHANGE-RELATION-NAME* (line 7), *ADD-ATTRIBUTE* (line 10), *ADD-RELATION* (line 13), *DELETE-ATTRIBUTE* (line 16) and *DELETE-RELATION* (line 16) used in Step 1 are the MKB evolution functions [32]. Step 2 and Step 3 are only applicable for the capability changes "*delete-relation* " and "*delete-attribute* ", the only changes that can make a view definition obsolete. In these two cases, the view synchronization algorithms are applied to affected and yet evolvable views of the VKB. The functions *POC* (defined in this chapter), *SPOC* (defined in this chapter) and *CVS* (defined in Chapter IV) are different synchronization algorithms developed in the EVE system that can handle different sets of constraints. The set of legal rewritings returned by these algorithms could be an empty set if the original view definition doesn't have the properties required by the algorithms or the algorithms fail to find specific MKB constraints. The function *ComputeDerived* takes a view and a set of base rewritings and compute all legal *derived* rewritings corresponding to it, i.e., all legal rewritings of the view that *don't* use replacements, assuming the E-SQL semantics allow to simply delete a view component.

In the general *EVE* algorithm, Step 3 computes *all* rewritings for a view and then it chooses the best one based on some cost model, namely Quality-Cost model, that we introduce in Chapter VI. Optimized versions of this algorithm could apply heuristic techniques to find the *best* rewriting without first computing all legal rewritings by pruning the search space based on the cost model [25]. Finally, the function *MAINTENANCE-AFTER-SYNCH* (line 41) is the maintenance process that brings the view extent up to date after synchronization. In Chapter V we introduce incremental maintenance techniques that could be applied for maintenance after synchronization.

3.3 View Synchronization Strategies based on Containment Constraints

Because view synchronization algorithms primarily are concerned with *replacing* affected components, we make distinction between legal view rewritings that are obtained by modifying only affected components (leaving all unaffected components intact), called *base rewritings*, and those having unaffected components modified as well (e.g., replaced or dropped), called *derived rewritings* (Section 2.5.2). In base rewritings "linked" affected components are either all *replaced*, all *dropped* or all *unchanged*. For example, if we find a replacement for an attribute $R.A$, then in a base rewriting *all* occurrences of $R.A$ are replaced (even though the evolution parameters would have allowed, for example, to drop a condition referring to $R.A$). If a relation S replaces a relation R (i.e., relation S has replacements for at least indispensable attributes of R in V) then a base rewriting must have preserved all replaceable attributes that could be replaced from S.

We propose in the following two view synchronization strategies that find base rewritings after the capability changes "*delete-attribute* " and "*delete-relation* "[2]. Thus, the third step described in Section 3.2 is the topic of this section.

When a relation $IS_1.R$ referred in the FROM clause of a view V is deleted from its site and the view is evolvable under that change, the view synchronizer will attempt to find a legal base rewriting for it, process that we call *view synchronization under capability change "delete-relation $IS_1.R$"*. Similarly, when only one attribute $IS_1.R.A$ used in the view V in the WHERE or SELECT clause is deleted from relation R, EVE system finds replacements for it, and uses them to rewrite the view V. This process

[2]The other capability changes EVE system can handle are treated as shown in Figure 3.2.

is called *view synchronization under capability change "delete-attribute $IS_1.R.A$".*

The view synchronization strategies presented below are based only on containment constraints found in MKB. First strategy uses $\mathcal{PC}$-constraints in PROJECT form while the second deal with the more general expressions, namely $\mathcal{PC}$-constraints in SELECT-PROJECT form. Unlike the strategies proposed for query rewriting using views ([33, 61]), our proposed techniques address new issues: (1) finding view rewritings that are not necessarily *equivalent* to the original view definition, (2) using semantic containment information expressed using $\mathcal{PC}$-constraints (Section 2.3) for replacing the deleted relation; and (3) preserving only indispensable attributes from the SELECT clause if preserving all is not possible.

$$
\begin{array}{ll}
\text{CREATE VIEW} & V \quad (\mathcal{VE} = \delta) \ \text{AS} \\
\text{SELECT} & Attr(V(R))_{\text{SELECT}}(\_, t), Attr(V(R))_{\text{SELECT}}(t, f), \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
\text{FROM} & R(\mathcal{RR} = true), R_1, \ldots, R_n \\
\text{WHERE} & \mathcal{CV}(Attr(V(R))_{\text{WHERE}}(\_, t), Attr(V(R))_{\text{WHERE}}(t, f), \bar{W})
\end{array}
\tag{3.1}
$$

$$
\begin{array}{ll}
\text{CREATE VIEW} & V \quad (\mathcal{VE} = \delta) \ \text{AS} \\
\text{SELECT} & grAttr(V(R))(f, t), grAttr(V(R))(t, t), grAttr(V(R))(t, f), \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n, \\
 & \mathcal{F}(aggAttr(V(R))(\_, t), aggAttr(V(R))(t, f), \bar{F}) \\
\text{FROM} & R(\mathcal{RR} = true), R_1, \ldots, R_n \\
\text{WHERE} & \mathcal{CV}(Attr(V(R))_{\text{WHERE}}(\_, t), Attr(V(R))_{\text{WHERE}}(t, f), \bar{W}) \\
\text{GROUP BY} & grAttr(V(R))(f, t), grAttr(V(R))(t, t), grAttr(V(R))(t, f), \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n
\end{array}
\tag{3.2}
$$

When considering the capability change "*delete-relation R*", we assume a generic E-SQL view V defined as in Equation (3.1) for SPJ type and as in Equation (3.2) for AGGR-SPJ view type. The relation R is a replaceable relation used in the FROM clause of the view V. Moreover, the view V is evolvable (Theorem 1) under this capability change. Namely, (1) there are no attributes of R used in the SELECT clause in a component (i.e., a group-by attribute, an aggregate attribute for AGGR-SPJ views, or a projected in attribute for SPJ views) with evolution parameters $(false, false)$, i.e., $Attr(V(R))_{\mathsf{SELECT}}(f, f) = \emptyset$[3] and (2) attributes of R used in the WHERE clause are referred to in conditions with evolution parameters different than $(false, false)$, i.e., $Attr(V(R))_{\mathsf{WHERE}}(f, f) = \emptyset$[4].

$\mathcal{CV}()$ is a conjunction of primitive clauses in the WHERE clause of V referring to attributes from relation R ($Attr(V(R))_{\mathsf{WHERE}}(f, t)$, $Attr(V(R))_{\mathsf{WHERE}}(t, t)$, $Attr(V(R))_{\mathsf{WHERE}}(t, f)$) and attributes from the rest of the relations ($\bar{W} \subseteq Attr(R_1) \cup \ldots \cup Attr(R_n)$). For the aggregation view V defined in Equation (3.2), $\mathcal{F}$ is a set of aggregate attributes $f_1(X_1), \ldots, f_k(X_k)$, using the attributes $aggAttr(V(R))$of relation R and the attributes $\bar{F}$ from the relations $R_1, \ldots, R_n$. Note that for the view V to be evolvable under capability change "*delete-relation R*", all aggregate attributes using attributes from $aggAttr(V(R))$cannot be both indispensable and nonreplaceable (following from Theorem 1).

[3] Note that the attributes of relation R in the GROUP BY clause are assumed to be exactly the attributes from the SELECT clause of relation R and thus they inherit the evolution parameters from the SELECT clause attributes. For the view V defined in Equation (3.2), $Attr(V(R))_{\mathsf{SELECT}}(d, r) = grAttr(V(R))(d, r) \cup aggAttr(V(R))(d, r)$.

[4] Actually, if the view V is the strongest synch-equivalent of its own (see Section 2.6) (the property which we assume any view in the VKB has) and the relation R is replaceable then all attributes of R must be replaceable or dispensable or both.

$$
\begin{array}{ll}
\text{CREATE VIEW} & V \quad (\mathcal{VE} = \delta) \text{ AS} \\
\text{SELECT} & Attr(V(R))_{\text{SELECT}}, R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
\text{FROM} & R, R_1, \ldots, R_n \\
\text{WHERE} & \mathcal{CV}(Attr(V(R))_{\text{WHERE}}, \bar{W})
\end{array}
\tag{3.3}
$$

$$
\begin{array}{ll}
\text{CREATE VIEW} & V \quad (\mathcal{VE} = \delta) \text{ AS} \\
\text{SELECT} & grAttr(V(R)), R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n, \\
 & \mathcal{F}(aggAttr(V(R)), \bar{F}) \\
\text{FROM} & R, R_1, \ldots, R_n \\
\text{WHERE} & \mathcal{CV}(Attr(V(R))_{\text{WHERE}}, \bar{W}) \\
\text{GROUP BY} & grAttr(V(R)), R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n
\end{array}
\tag{3.4}
$$

When the capability change is "*delete-attribute* $R.A$", we assume the view V be defined as in Equation (3.3) for SPJ views and as in Equation (3.4) for AGGR-SPJ views. The attribute $R.A$ is used in the SELECT and/or WHERE clauses. The view V must be evolvable (Theorem 1) under this capability change and at least one component referring to the attribute $R.A$ is replaceable in V. For the view V to be evolvable under the capability change "*delete-attribute* $R.A$", $R.A$ must belong only to $Attr(V(R))_{\text{SELECT}}(\_, t)$, $Attr(V(R))_{\text{SELECT}}(t, f)$, $Attr(V(R))_{\text{WHERE}}(\_, t)$, or $Attr(V(R))_{\text{WHERE}}(t, f)$ (and not to $Attr(V(R))_{\text{SELECT}}(f, f)$ or $Attr(V(R))_{\text{WHERE}}(f, f)$), i.e., $R.A$ cannot be used in a view component that is nonreplaceable and indispensable in the same time.

3.4 The POC View Synchronization Strategy

We discuss here rewriting strategies for E-SQL views using project $\mathcal{PC}$-constraints in PROJECT form (Section 2.3). The algorithm is applicable for the two capability changes that require substitution of view components: "*delete-relation*" and "*delete-attribute*".

We give first a common condition for finding a suitable substitution for either the deleted relation or the deleted attribute. Then, for each capability change considered, we give some extra conditions that a replacement must have and how a rewriting could be obtained using it. Lastly, we prove that using this strategy legal rewritings conforming to Definition 1 are found.

3.4.1 Conditions for Replacement Used by the POC Algorithm

For both capability changes requiring replacements for the directed affected components, POC strategy uses $\mathcal{PC}$-constraints to find candidate relations for such substitutions. The properties that a candidate relation must have are different if we have to replace an entire relation (in the case of "*delete-relation* " capability change) or only one of its attributes (for "*delete-attribute* " capability change). Hence, we structure the rest of this section as following: first we give Condition 1 that must hold for both capability changes, and then for each "*delete-relation* " and "*delete-attribute* ", we give the rest of the conditions and the rewriting techniques used to find legal rewritings.

A relation $IS_2.S$ is said to be a <u>*Project Substitution*</u> for $IS_1.R$ if the following condition is satisfied:

Condition 1: Project-Relation Containment. Assume that the Equation (3.5) is true for any states of relations R and S, i.e., the following $\mathcal{PC}$-constraint containing only projections is defined in the MKB:

$$\pi_{\bar{B}}(S) \ \phi \ \pi_{\bar{A}}(R) \tag{3.5}$$

The attributes $S.\bar{B} \subseteq Attr(S)$ are called *replacements* for the attributes $R.\bar{A} \subseteq Attr(R)$. That is, for any pair $(R.A_i, S.B_i)$ of attributes having the same position in the vectors $R.\bar{A}$ and $S.\bar{B}$, respectively, the following type constraints exist in MKB: $\mathcal{TC}(R.A_i) = (\ R(A_i) \ \subseteq \ Type_i(A_i) \ )$ and $\mathcal{TC}(S.B_i) = (\ S(B_i) \ \subseteq \ Type_i(B_i) \ )$. 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 $R.C$ is not among the attributes with replacements in S (i.e., $R.C \notin R.\bar{A}$) then $S(R.C) = \emptyset$.

3.4.2 Conditions for Replacement Used by the POC Algorithm: "*delete-relation* " Capability Change

The conditions 2 through 4 given below are the extra conditions a candidate project-substitution relation S must have such that the rewriting obtained by replacing the entire relation R with S (view V' defined in Equations (3.10) and (3.11)) to be a legal rewriting. Theorem 5 proves the sufficiency of the conditions 1 through 4 and thus, that the rewriting V' is legal conforming to Definition 1. The Tables 3.1, 3.2, and 3.3 summarize the conditions and rewriting strategies for the SPJ and AGGR-SPJ views, respectively.

Condition 2: Indispensable Evolution Parameters Satisfaction. This condition requires that relation S must contain replacements for *at least* attributes of the relation R that are indispensable and replaceable in the view V. That is, all attributes of R used in the SELECT clause with $\mathcal{AD} = false$ and $\mathcal{AR} = true$ and all attributes

Condition 1	Condition 2	Condition 3
$\mathcal{VE}\ = \subseteq$		
Condition 1: $\pi_{S.\bar{B}}(S) \subseteq \mid \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $Attr(V(R))_{\mathsf{SELECT}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\mathsf{WHERE}}(f,t) \subseteq R.\bar{A}$	Condition 3: $Attr(V(R))_{\mathsf{WHERE}}(t,f) = \emptyset$ $Attr(V(R))_{\mathsf{WHERE}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\mathsf{WHERE}}(t,t) \subseteq R.\bar{A}$
CREATE VIEW V' $(\mathcal{VE} = \subseteq)$ AS SELECT $\underline{S(Attr(V(R))_{\mathsf{SELECT}}(f,t)), S(Attr(V(R))_{\mathsf{SELECT}}(t,t)),}$ $\underline{R_1.D_1, \ldots, R_n.D_n}$ FROM $\underline{S}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))_{\mathsf{WHERE}}(f,t), Attr(V(R))_{\mathsf{WHERE}}(t,t)), \bar{W})}$		
Theorem 5: $V' \subseteq_\pi \mid \equiv_\pi V$		
$\mathcal{VE}\ = \supseteq$		
Condition 1: $\pi_{S.\bar{B}}(S) \supseteq \mid \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $Attr(V(R))_{\mathsf{SELECT}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\mathsf{WHERE}}(f,t) \subseteq R.\bar{A}$	
CREATE VIEW V' $(\mathcal{VE} = \supseteq)$ AS SELECT $\underline{S(Attr(V(R))_{\mathsf{SELECT}}(f,t)), S(Attr(V(R))_{\mathsf{SELECT}}(t,t)),}$ $\underline{R_1.D_1, \ldots, R_n.D_n}$ FROM $\underline{S}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))_{\mathsf{WHERE}}(f,t), Attr(V(R))_{\mathsf{WHERE}}(t,t)), \bar{W})}$		
Theorem 5: $V' \supseteq_\pi V$		
$\mathcal{VE}\ = \equiv$		
Condition 1: $\pi_{S.\bar{B}}(S) \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $Attr(V(R))_{\mathsf{SELECT}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\mathsf{WHERE}}(f,t) \subseteq R.\bar{A}$	Condition 3: $Attr(V(R))_{\mathsf{WHERE}}(t,f) = \emptyset$ $Attr(V(R))_{\mathsf{WHERE}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\mathsf{WHERE}}(t,t) \subseteq R.\bar{A}$
CREATE VIEW V' $(\mathcal{VE} = \equiv)$ AS SELECT $\underline{S(Attr(V(R))_{\mathsf{SELECT}}(f,t)), S(Attr(V(R))_{\mathsf{SELECT}}(t,t)),}$ $\underline{R_1.D_1, \ldots, R_n.D_n}$ FROM $\underline{S}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))_{\mathsf{WHERE}}(f,t), Attr(V(R))_{\mathsf{WHERE}}(t,t)), \bar{W})}$		
Theorem 5: $V' \equiv_\pi V$		
$\mathcal{VE}\ = \approx$		
Condition 1: $\pi_{S.\bar{B}}(S)\ \approx\ \pi_{R.\bar{A}}(R)$	Condition 2: $Attr(V(R))_{\mathsf{SELECT}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\mathsf{WHERE}}(f,t) \subseteq R.\bar{A}$	
CREATE VIEW V' $(\mathcal{VE} = \approx)$ AS SELECT $\underline{S(Attr(V(R))_{\mathsf{SELECT}}(f,t)), S(Attr(V(R))_{\mathsf{SELECT}}(t,t)),}$ $\underline{R_1.D_1, \ldots, R_n.D_n}$ FROM $\underline{S}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))_{\mathsf{WHERE}}(f,t), Attr(V(R))_{\mathsf{WHERE}}(t,t)), \bar{W})}$		

Table 3.1: POC Synchronization Strategy for SPJ View V, Eq. (3.24): "*delete-relation R*".

Condition 1	Condition 2	Condition 3
$\boxed{\mathcal{VE} \ = \subseteq}$		
Condition 1: $\pi_{S.\bar{B}}(S) \subseteq \mid \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $grAttr(V(R))(f,t) \subseteq R.\bar{A}$ $aggAttr(V(R))(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$	Condition 3: $Attr(V(R))_{\text{WHERE}}(t,f) = \emptyset$ $Attr(V(R))_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\text{WHERE}}(t,t) \subseteq R.\bar{A}$
CREATE VIEW V' $(\mathcal{VE} = \subseteq)$ AS SELECT $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t))},$ $\underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n},$ $\underline{\mathcal{F}(S(aggAttr(V(R))(f,t)), S(aggAttr(V(R))(t,t)), \bar{F})}$ FROM $\underline{S(\mathcal{RR} = true)}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))_{\text{WHERE}}(\_,t)), \bar{W})}$ GROUP BY $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t))},$ $\underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n}$		
Theorem 5: $V' \subseteq_{\pi_{gr}} \mid \equiv_{\pi_{gr}} V$		
$\boxed{\mathcal{VE} \ = \supseteq}$		
Condition 1: $\pi_{S.\bar{B}}(S) \supseteq \mid \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $grAttr(V(R))(f,t) \subseteq R.\bar{A}$ $aggAttr(V(R))(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$	
CREATE VIEW V' $(\mathcal{VE} = \supseteq)$ AS SELECT $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t))},$ $\underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n},$ $\underline{\mathcal{F}(S(aggAttr(V(R))(f,t)), S(aggAttr(V(R))(t,t)), \bar{F})}$ FROM $\underline{S(\mathcal{RR} = true)}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))_{\text{WHERE}}(\_,t)), \bar{W})}$ GROUP BY $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t))},$ $\underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n}$		
Theorem 5: $V' \supseteq_{\pi_{gr}} V$ $\forall t \in V \exists t' \in V'$ $s.t.$ $t' =_{\pi_{gr}} t$ $t'[f(X)] \geq t[f(X)], f \in \{MAX, COUNT(DISTINCT)\},$ $t'[f(X)] \leq t[f(X)], f \in \{MIN\},$ f an aggregate attribute in V and V'		

Table 3.2: POC Synchronization Strategy for Aggregation View V, Eq. (3.25): "*delete-relation R*".

Condition 1	Condition 2	Condition 3
$\boxed{\mathcal{VE} \ = \equiv}$		
Condition 1: $\pi_{S.\bar{B}}(S) \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $grAttr(V(R))(f,t) \subseteq R.\bar{A}$ $aggAttr(V(R))f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$	Condition 3: $Attr(V(R))_{\text{WHERE}}(t,f) = \emptyset$ $Attr(V(R))_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\text{WHERE}}(t,t) \subseteq R.\bar{A}$
CREATE VIEW $V' \ \ (\mathcal{VE} = \equiv)$ AS SELECT $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t)),}$ $\underline{R_1.D_1, \ldots, R_n.D_n,}$ $\underline{\mathcal{F}(S(aggAttr(V(R))(f,t)), S(aggAttr(V(R))(t,t)), \bar{F})}$ FROM $\underline{S(\mathcal{RR} = true)}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))_{\text{WHERE}}(f,t)), S(Attr(V(R))_{\text{WHERE}}(t,t)), \bar{W})}$ GROUP BY $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t)),}$ $\underline{R_1.D_1, \ldots, R_n.D_n}$		
Theorem 5: $V' \equiv_{\pi_{gr}} V$ $\forall t \in V \exists t' \in V' \ s.t. \ t' =_{\pi_{gr}} t$ $t'[f(X)] \geq t[f(X)], f \in \{MAX, COUNT(DISTINCT)\}$, $t'[f(X)] \leq t[f(X)], f \in \{MIN\}$, $\forall t' \in V' \exists t \in V \ s.t. \ t =_{\pi_{gr}} t' \quad t[f(X)] \leq t'[f(X)], f \in \{MAX, COUNT(DISTINCT)\}$, $t[f(X)] \geq t'[f(X)], f \in \{MIN\}$, f an aggregate attribute in V and V'		
$\boxed{\mathcal{VE} \ = \approx}$		
Condition 1: $\pi_{S.\bar{B}}(S) \ \approx \ \pi_{R.\bar{A}}(R)$	Condition 2: $grAttr(V(R))(f,t) \subseteq R.\bar{A}$ $aggAttr(V(R))(f,t) \subseteq R.\bar{A}$ $Attr(V(R))_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$	
CREATE VIEW $V' \ \ (\mathcal{VE} = \approx)$ AS SELECT $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t)),}$ $\underline{R_1.D_1, \ldots, R_n.D_n,}$ $\underline{\mathcal{F}(S(aggAttr(V(R))(f,t)), S(aggAttr(V(R))(t,t)), \bar{F})}$ FROM $\underline{S(\mathcal{RR} = true)}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))_{\text{WHERE}}(\_,t)), \bar{W})}$ GROUP BY $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t)),}$ $\underline{R_1.D_1, \ldots, R_n.D_n}$		

Table 3.3: POC Synchronization Strategy for Aggregation View V, Eq. (3.25): "*delete-relation* R" (cont.).

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 the Equation (3.6) (note that for the aggregate views, $Attr(V(R))_{\mathsf{SELECT}}(f,t) = grAttr(V(R))(f,t) \cup aggAttr(V(R))(f,t)$).

$$Attr(V(R))_{\mathsf{SELECT}}(f,t) \ \cup \ Attr(V(R))_{\mathsf{WHERE}}(f,t) \ \subseteq \ R.\bar{A}. \tag{3.6}$$

In short, the Indispensable Evolution Parameters Satisfaction condition states that all attributes of R that are essential for the view (i.e., the indispensable attributes) and replaceable (i.e., their attribute-replaceable evolution parameter values are set to $true$) must be obtained from S. Clearly, this condition alone is necessary but not sufficient to obtain a legal rewriting (Definition 1) by replacing the relation R by relation S.

Condition 3: View-Extent Evolution Parameter Satisfaction. This condition expresses the properties that must hold for the $\mathcal{PC}$-constraint defined by Equation (3.5) in order for the view-extent evolution parameter 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> These case requires that all conditions in the WHERE clause of the view V be preserved regardless to their evolution parameters (Equations (3.8), (3.7)) which imposes that all directly affected attributes used in the WHERE clause be preserved (i.e., they need to have *replacements* in the relation S). Because the attributes from $Attr(V(R))_{\mathsf{WHERE}}(t,f)$ cannot be replaced ($\mathcal{CR} = false$) the conditions referring to them can only be dropped given their evolution parameters. But in this case we cannot drop any primitive clause from the WHERE clause, hence we have to impose that the set $Attr(V(R))_{\mathsf{WHERE}}(t,f)$ is empty (Equation (3.7)). Note that dropping a condition in the WHERE clause of the view V makes, in general, a

bigger extent, which is not allowed in this case when the view-extent parameter $\mathcal{VE}$ is $\subseteq$.

$$Attr(V(R))_{\mathsf{WHERE}}(t,f) = \emptyset \tag{3.7}$$
$$Attr(V(R))_{\mathsf{WHERE}}(f,t) \ \cup \ Attr(V(R))_{\mathsf{WHERE}}(t,t) \ \subseteq \ R.\bar{A} \tag{3.8}$$

<u>Case 2. $\mathcal{VE} = \supseteq$.</u> In this case, the view can be evolved by dropping conditions from the WHERE clause and still have satisfied the view extent evolution parameter $\mathcal{VE} = \supseteq$.

<u>Case 3. $\mathcal{VE} = \approx$.</u> 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 $\mathcal{VE} = \approx$.

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 (Equation (3.5)) must satisfy the property from Equation (3.9).

$$\begin{array}{llll} \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \equiv, & \text{then } \phi \text{ must be } \equiv; \\ \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \subseteq, & \text{then } \phi \text{ must be } \subseteq \text{ or } \equiv; \text{ and} \\ \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \supseteq, & \text{then } \phi \text{ must be } \supseteq \text{ or } \equiv. \end{array} \tag{3.9}$$

3.4.3 The POC Algorithm for "*delete-relation*" Capability Change

Let the view V be defined as in Equation (3.1) for SPJ view type and as in Equation (3.2) for aggregation views. S is a relation that satisfies Conditions 1 through 4. Let the view V' in Equations (3.10) and (3.11) be obtained from V (in Equation (3.1)) 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 Equations (3.10) and (3.11) where the new view components are underlined.

$$
\begin{array}{ll}
\text{CREATE VIEW} & V' \ \ (\mathcal{VE} = \delta) \text{ AS} \\
\text{SELECT} & \underline{S(Attr(V(R))_{\text{SELECT}}(f,t)), S(Attr(V(R))_{\text{SELECT}}(t,t))}, \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
\text{FROM} & \underline{S}, R_1, \ldots, R_n \\
\text{WHERE} & \underline{\mathcal{CV}(S(Attr(V(R))_{\text{WHERE}}(\_, t)), \bar{W})}
\end{array}
\tag{3.10}
$$

$$
\begin{array}{ll}
\text{CREATE VIEW} & V' \ \ (\mathcal{VE} = \delta) \text{ AS} \\
\text{SELECT} & \underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t))}, \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n, \\
 & \underline{\mathcal{F}(S(aggAttr(V(R))(\_, t)), \bar{F})} \\
\text{FROM} & \underline{S(\mathcal{RR} = true)}, R_1, \ldots, R_n \\
\text{WHERE} & \underline{\mathcal{CV}(S(Attr(V(R))_{\text{WHERE}}(\_, t)), \bar{W})} \\
\text{GROUP BY} & \underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t))}, \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n
\end{array}
\tag{3.11}
$$

In both rewriting V' of Equations (3.10) and (3.11), $S(Attr(V(R))_{\text{SELECT}}(f,t))$, $S(grAttr(V(R))(f,t))$ and $S(aggAttr(V(R))(f,t))$ are the attribute replacements from $S.\bar{B}$ corresponding to attributes in the SELECT clause with the evolution parameters (f,t). By Condition 2, all these attributes must have replacements in S, thus $|S(Attr(V(R))_{\text{SELECT}}(f,t))| = |Attr(V(R))_{\text{SELECT}}(f,t)|$, $|S(aggAttr(V(R))(f,t))| = |aggAttr(V(R))(f,t)|$, and $|S(grAttr(V(R))(f,t))| = |grAttr(V(R))(f,t)|$.

The attributes of relation R used in the SELECT clause with evolution parameters (t,t) were replaced in the view V' if replacements have been found in $S.\bar{B}$. However, some of the attributes with evolution parameters (t, t) without replacements and all attributes from $Attr(V(R))_{\mathsf{SELECT}}(t, f)$ were dropped in V'.

$S(Attr(V(R))_{\mathsf{WHERE}}(f, t))$ are the attributes from $S.\bar{B}$ replacing attributes from $Attr(V(R))_{\mathsf{WHERE}}(f, t)$ ($\subseteq R.\bar{A}$), with $|S(Attr(V(R))_{\mathsf{WHERE}}(f, t))| = |Attr(V(R))_{\mathsf{WHERE}}(f, t)|$ (by Condition 2). As stated by Condition 3, it is not necessarily to have replacements for all attributes of R in the WHERE clause used in the conditions with evolution parameters $(true, true)$, hence $|S(Attr(V(R))_{\mathsf{WHERE}}(t, t))| \leq |Attr(V(R))_{\mathsf{WHERE}}(t, t)|$. $\mathcal{CV}(S(Attr(V(R))_{\mathsf{WHERE}}(f, t)), S(Attr(V(R))_{\mathsf{WHERE}}(t, t)), \bar{W})$ is the conjunction of primitive clauses in the WHERE clause of the view V (Equation (3.1)) where attributes of relation R were replaced by the corresponding attributes in $S(Attr(V(R))_{\mathsf{WHERE}}(f, t))$ and $S(Attr(V(R))_{\mathsf{WHERE}}(t, t)$. If not all attributes from $Attr(V(R))_{\mathsf{WHERE}}(t, t)$ have replacements in $S.\bar{B}$, then the primitive clauses using them where dropped in $\mathcal{CV}$, as well as the primitive clauses referring to the attributes in $Attr(V(R))_{\mathsf{WHERE}}(t, f)$ which cannot be replaced given their evolution parameters. For the aggregation view V' (Equation (3.11)), the aggegate functions from $\mathcal{F}$ using attributes from R were preserved if a replacement was found in $S.\bar{B}$.

The above conditions are *sufficient* (but not *necessary*) to obtain a legal rewriting by using project substitution $IS_2.S$ to replace the relation $IS_1.R$. The following theorems prove the sufficiency of Conditions 1 through 4 to obtain a legal base rewriting V' by replacing the relation R with the replacement relation S.

Theorem 5 *Let an E-SQL view V (Equations (3.1) and (3.2)) be evolvable (Theorem 1) under the capability change "delete-relation R", with R a replaceable relation in the* FROM *clause. Then the rewriting V' obtained by applying POC Algorithm (Equations (3.10) and (3.11), respectively) is a legal rewriting (Definition 1) under capability change "delete-relation R".*

For aggregate functions in the view V' (Equation (3.2)), we can prove relationships from Table (3.2) for common tuples of the old view V and the legal rewriting V'.

Proof. See Appendix A.1.

3.4.4 Illustrative Example for POC Algorithm: "*delete-relation*" Capability Change

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** (Equation (3.12)).

$$
\begin{array}{ll}
\text{CREATE VIEW} & \textbf{Customer-Passengers}\ (\mathcal{VE} = \delta)\ \text{AS} \\
\text{SELECT} & \textbf{C.Name}\ (\mathcal{AD} = false, \mathcal{AR} = true), \\
 & \textbf{C.Age}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\
 & \textbf{C.Phone}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\
\text{FROM} & \textbf{Customer C}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
 & \textbf{FlightRes F}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
\text{WHERE} & (\textbf{C.Name} = \textbf{F.PName})\ (\mathcal{CD} = false, \mathcal{CR} = true)\ \text{AND} \\
 & (\textbf{F.Dest} = \text{'}\textbf{Asia}\text{'})\ \text{AND} \\
 & (\ \textbf{C.Age} < 20)(\mathcal{CD} = false, \mathcal{CR} = true)
\end{array}
\tag{3.12}
$$

We assume the capability change delete-relation **Customer**. In the MKB a $\mathcal{PC}$-constraint is defined between the relations **Customer** and **Accidental–Ins** shown in Equation (3.13).

$$
\begin{array}{l}
\mathcal{PC}_{\textbf{Accident-Ins,Customer}} = (\pi_{\textbf{Accident-Ins.Holder,Accident-Ins.Age}}(\textbf{Accident-Ins}) \\
\quad \subseteq \pi_{\textbf{Customer.Name,Customer.Age}}(\ \textbf{Customer}))
\end{array}
\tag{3.13}
$$

The elements of the view **Customer–Passengers** relative to relation **Customer** are given in Figure 3.4.

The relation **Accident–Ins** has all properties required by Conditions 1 to 4 falling into Case 1 in Condition 3: ϕ in our $\mathcal{PC}$-constraint from Equation (3.13) is $\subseteq$, and

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

Figure 3.4: Elements of the **Customer-Passengers** view relative to **Customer** relation.

all attributes from $Attr(V(R))_{\mathsf{WHERE}}$have replacements in **Accident−Ins** (Figure 3.4). 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 Equations (3.12) and (3.14) must be $\subseteq$). Finally, applying the POC Algorithm we obtain the legal view rewriting **Customer−Passengers'** defined in Equation (3.14).

$$
\begin{array}{ll}
\textsf{CREATE VIEW} & \textbf{Customer-Passengers'}\ (\mathcal{VE} = \subseteq)\ \textsf{AS} \\
\textsf{SELECT} & \underline{\textbf{A.Holder}\ (\mathcal{AD} = false, \mathcal{AR} = true),} \\
 & \underline{\textbf{A.Age}\ (\mathcal{AD} = true, \mathcal{AR} = true)} \\
\textsf{FROM} & \underline{\textbf{Accident−Ins A}\ (\mathcal{RD} = true, \mathcal{RR} = true),} \\
 & \textbf{FlightRes F}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
\textsf{WHERE} & \underline{(\textbf{A.Holder} = \textbf{F.PName})\ (\mathcal{CD} = false, \mathcal{CR} = true)}\textsf{AND} \\
 & (\textbf{F.Dest} = \textbf{'Asia'})\textsf{AND} \\
 & \underline{(\textbf{A.Age} < 20)(\mathcal{CD} = false, \mathcal{CR} = true)}
\end{array}
\tag{3.14}
$$

3.4.5 Conditions for Replacement Used by the POC Algorithm: "*delete-attribute* " Capability Change

For the capability change "*delete-attribute* $R.A$", we need only the deleted attribute $R.A$ to have a replacement in the relation S. Our goal is to replace relation R in the view V by a join relation defined in Equation (3.15). We denote by R' the relation R after the attribute $R.A$ was dropped. The attribute sets $S.\bar{B}'$ and $R.\bar{A}'$ are defined by: $S.\bar{B}' = S.\bar{B} \ \setminus \ \{S(R.A)\}$, $R.\bar{A}' = R.\bar{A} \ \setminus \ \{R.A\}$. (where the attributes $S.\bar{B}$ and $R.\bar{A}$ are the attributes projected in the $\mathcal{PC}$-constraint defined in Equation (3.5)).

$$R' \bowtie_{R'.\bar{A}'=S.\bar{B}'} S \tag{3.15}$$

The attribute $R.A$ will be replaced by its replacement $S(R.A)$ from S. Moreover, because this rewriting must satisfy the view evolution parameters, the relation S must satisfied other properties as well, given in Conditions 2 through 4 below. The Tables 3.4, 3.5, 3.6, and 3.7 summarize the conditions and the POC synchronization strategies we detail in the followings.

Condition 1	Condition 2	Condition 3
$\boxed{\mathcal{VE} \ = \subseteq}$		
Condition 1: $\pi_{S.\bar{B}}(S) \subseteq \mid \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $R.A \ \in \ R.\bar{A}$	Condition 3: $Attr(V(R))_{\text{WHERE}} \ \subseteq \ R.\bar{A}$ $R.A \ \notin \ Attr(V(R))_{\text{WHERE}}(\_, f)$ $(Attr(V(R))_{\text{SELECT}} \ \setminus \ R.\bar{A})$ $\subseteq \ Attr(V(R))_{\text{SELECT}}(t, \_)$
CREATE VIEW $V' \ \ (\mathcal{VE} = \subseteq)$ AS SELECT $\underline{(Attr(V(R))_{\text{SELECT}} \ \cap \ R.\bar{A}) \setminus \{R.A\}}$, $\underline{\{S.B \mid S.B = S(R.A) \ if \ R.A \text{ is replaceable}\}}$, $R_1.\bar{D}_1 \ldots, R_n.\bar{D}_n$ FROM $\underline{S}, R', R_1, \ldots, R_n$ WHERE $\underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))}$ AND $\underline{\mathcal{CV}(S(R.A), Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}, W)}$		
Theorem 6: $V' \subseteq_\pi \mid \equiv_\pi V$		
$\boxed{\mathcal{VE} \ = \supseteq}$		
Condition 1: $\pi_{S.\bar{B}}(S) \supseteq \mid \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $R.A \ \in \ R.\bar{A}$	
CREATE VIEW $V' \ \ (\mathcal{VE} = \supseteq)$ AS SELECT $\underline{(Attr(V(R))_{\text{SELECT}} \ \setminus \ \{R.A\})}$, $\underline{\{S.B \mid S.B = S(R.A) \ if \ R.A \text{ is replaceable}\}}$, $R_1.\bar{D}_1 \ldots, R_n.\bar{D}_n$ FROM $\underline{S}, R', R_1, \ldots, R_n$ WHERE $\underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))}$ AND $\underline{\mathcal{CV}(S(R.A), Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}, W)}$		
Theorem 6: $V' \supseteq_\pi V$		

Table 3.4: POC Synchronization Strategy for SPJ View V, Eq. (3.49): "*delete-attribute* $R.A$".

Condition 1	Condition 2	Condition 3
$\boxed{\mathcal{VE} \;\; = \equiv}$		
Condition 1: $\pi_{S.\bar{B}}(S) \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $R.A \; \in \; R.\bar{A}$	Condition 3: $Attr(V(R))_{\text{WHERE}} \;\subseteq\; R.\bar{A}$ $R.A \;\not\in\; Attr(V(R))_{\text{WHERE}}(\_, f)$ $(Attr(V(R))_{\text{SELECT}} \;\setminus\; R.\bar{A})$ $\subseteq\; Attr(V(R))_{\text{SELECT}}(t, \_)$
CREATE VIEW $V' \;\; (\mathcal{VE} =\equiv)$ AS SELECT $\underline{(Attr(V(R))_{\text{SELECT}} \;\cap\; R.\bar{A}) \setminus \{R.A\}}$, $\underline{\{S.B \mid S.B = S(R.A) \; if \; R.A \text{ is replaceable}\}}$, $R_1.D_1 \ldots, R_n.D_n$ FROM $\underline{S, R'}, R_1, \ldots, R_n$ WHERE $\underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))}$ AND $\underline{\mathcal{CV}(S(R.A), Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}, W)}$		
Theorem 6: $V' \equiv_\pi V$		
$\boxed{\mathcal{VE} \;\; == \approx}$		
Condition 1: $\pi_{S.\bar{B}}(S) \;\approx\; \pi_{R.\bar{A}}(R)$	Condition 2: $R.A \; \in \; R.\bar{A}$	
CREATE VIEW $V' \;\; (\mathcal{VE} =\approx)$ AS SELECT $\underline{(Attr(V(R))_{\text{SELECT}} \;\setminus\; \{R.A\})}$ $\underline{\{S.B \mid S.B = S(R.A) \; if \; R.A \text{ is replaceable}\}}$, $R_1.D_1 \ldots, R_n.D_n$ FROM $\underline{S}, R', R_1, \ldots, R_n$ WHERE $\underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))}$ AND $\underline{\mathcal{CV}(S(R.A), Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}, W)}$		

Table 3.5: POC Synchronization Strategy for SPJ View V, Eq. (3.49): "*delete-attribute* $R.A$" (cont.).

Condition 1	Condition 2	Condition 3
$\boxed{\mathcal{VE} \ = \subseteq}$		
Condition 1: $\pi_{S.\bar{B}}(S) \subseteq \mid \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $R.A \ \in \ R.\bar{A}$	Condition 3: $Attr(V(R))_{\text{WHERE}} \ \subseteq \ R.\bar{A}$ $R.A \ \notin \ Attr(V(R))_{\text{WHERE}}(\_, f)$ $(grAttr(V(R)) \ \setminus \ R.\bar{A})$ $\subseteq \ grAttr(V(R))(t, \_)$
CREATE VIEW V' $(\mathcal{VE} = \subseteq)$ AS SELECT $\underline{(grAttr(V(R)) \ \cap \ R.\bar{A}) \setminus \{R.A\}}$, $\underline{\{S.B \mid S.B = S(R.A) \text{if R.A is replaceable}\}}$ $R_1.D_1, \ldots, R_n.D_n$, $\underline{\mathcal{F}(S(R.A), (aggAttr(V(R)) \setminus \{R.A\}), \bar{F})}$ FROM $\underline{S, R'}, R_1, \ldots, R_n$ WHERE $\underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))}$, $\underline{\mathcal{CV}(S(R.A), (Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}), \bar{W})}$ GROUP BY $\underline{(grAttr(V(R)) \cap R.\bar{A}) \setminus \{R.A\}}$, $\underline{\{S.B \mid S.B = S(R.A) \text{if R.A is replaceable}\}}$ $R_1.D_1, \ldots, R_n.D_n$		
Theorem 6: $V' \subseteq_{\pi_{gr}} \mid \equiv_{\pi_{gr}} V$		
$\boxed{\mathcal{VE} \ = \supseteq}$		
Condition 1: $\pi_{S.\bar{B}}(S) \supseteq \mid \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $R.A \ \in \ R.\bar{A}$	
CREATE VIEW V' $(\mathcal{VE} = \supseteq)$ AS SELECT $\underline{grAttr(V(R)) \setminus \{R.A\}}$, $\underline{\{S.B \mid S.B = S(R.A) \text{if R.A is replaceable}\}}$ $R_1.D_1, \ldots, R_n.D_n$, $\underline{\mathcal{F}(S(R.A), (aggAttr(V(R)) \setminus \{R.A\}), \bar{F})}$ FROM $\underline{S, R'}, R_1, \ldots, R_n$ WHERE $\underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))}$, $\underline{\mathcal{CV}(S(R.A), (Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}), \bar{W})}$ GROUP BY $\underline{grAttr(V(R)) \setminus \{R.A\}}$, $\underline{\{S.B \mid S.B = S(R.A) \text{if R.A is replaceable}\}}$ $R_1.D_1, \ldots, R_n.D_n$		
Theorem 6: $V' \supseteq_{\pi_{gr}} V$ $\forall t \in V \exists t' \in V' \ s.t. \ t' =_{\pi_{gr}} t$ $t'[f(X)] \geq t[f(X)], f \in \{MAX, COUNT\}$, $t'[f(X)] \leq t[f(X)], f \in \{MIN\}$, f an aggregate attribute in V and V'		

Table 3.6: POC Synchronization Strategy for Aggregation View V, Eq. (3.50): "*delete-attribute* $R.A$".

Condition 1	Condition 2	Condition 3
$\boxed{\mathcal{VE} \;=\equiv}$		
Condition 1: $\pi_{S.\bar{B}}(S) \equiv \pi_{R.\bar{A}}(R)$	Condition 2: $R.A \;\in\; R.\bar{A}$	Condition 3: $Attr(V(R))_{\text{WHERE}} \;\subseteq\; R.\bar{A}$ $R.A \;\notin\; Attr(V(R))_{\text{WHERE}}(\_, f)$ $(grAttr(V(R)) \;\setminus\; R.\bar{A})$ $\subseteq\; grAttr(V(R))(t, \_)$
CREATE VIEW $V' \;\; (\mathcal{VE} =\equiv)$ AS SELECT $\underline{(grAttr(V(R)) \;\cap\; R.\bar{A}) \setminus \{R.A\}}$, $\underline{\{S.B \mid S.B = S(R.A)\text{ if R.A is replaceable}\}}$ $\underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n}$, $\underline{\mathcal{F}(S(R.A), (aggAttr(V(R)) \setminus \{R.A\}), \bar{F})}$ FROM $\underline{S, R'}, R_1, \ldots, R_n$ WHERE $\underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))}$, $\underline{\mathcal{CV}(S(R.A), (Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}), \bar{W})}$ GROUP BY $\underline{(grAttr(V(R)) \cap R.\bar{A}) \setminus \{R.A\}}$, $\underline{\{S.B \mid S.B = S(R.A)\text{if R.A is replaceable}\}}$ $\underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n}$		
Theorem 6: $V' \equiv_{\pi_{gr}} V$ $\forall t \in V \exists t' \in V' \;\; s.t. \;\; t' =_{\pi_{gr}} t$ $t'[f(X)] \geq t[f(X)], f \in \{MAX, COUNT\}$, $t'[f(X)] \leq t[f(X)], f \in \{MIN\}$, $\forall t' \in V' \exists t \in V \;\; s.t. \;\; t =_{\pi_{gr}} t'$ $t[f(X)] \leq t'[f(X)], f \in \{MAX, COUNT\}$, $t[f(X)] \geq t'[f(X)], f \in \{MIN\}$, f an aggregate attribute in V and V'		
$\boxed{\mathcal{VE} \;=\approx}$		
Condition 1: $\pi_{S.\bar{B}}(S) \;\approx\; \pi_{R.\bar{A}}(R)$	Condition 2: $R.A \;\in\; R.\bar{A}$	
CREATE VIEW $V' \;\; (\mathcal{VE} =\approx)$ AS SELECT $\underline{grAttr(V(R)) \setminus \{R.A\}}$, $\underline{\{S.B \mid S.B = S(R.A)\text{if R.A is replaceable}\}}$ $\underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n}$, $\underline{\mathcal{F}(S(R.A), (aggAttr(V(R)) \setminus \{R.A\}), \bar{F})}$ FROM $\underline{S, R'}, R_1, \ldots, R_n$ WHERE $\underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))}$, $\underline{\mathcal{CV}(S(R.A), (Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}), \bar{W})}$ GROUP BY $\underline{grAttr(V(R)) \setminus \{R.A\}}$, $\underline{\{S.B \mid S.B = S(R.A)\text{if R.A is replaceable}\}}$ $\underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n}$		
Theorem 6: $V' \supseteq_{\pi_{gr}} V$		

Table 3.7: POC Synchronization Strategy for Aggregation View V, Eq. (3.50): "*delete-attribute* $R.A$" (cont.).

Condition 2: Deleted Attribute Replacement. At least the attribute $R.A$ must have a replacement in relation S. That is $R.A \in R.\bar{A}$ which implies that $S(R.A) \in S.\bar{B}$.

Condition 3: View-Extent Evolution Parameter Satisfaction. The view-extend evolution parameter defined for the view V must be satisfied if the view rewriting is done by using the expression defined in Equation (3.15) for replacing relation R. The properties we impose on the $\mathcal{PC}$-constraint (Equation (3.5)) guarantee that the relationship between the old view V and its rewriting is the one imposed by the view-extend parameter. The relationship between the original view V and such a rewriting V' is determined from the relationship between the relation R and $R' \bowtie_{R'.\bar{A}'=S.\bar{B}'} S$, on one side, and, on the other side, the relationship between $Attr(V(R))$ (the attributes used in the view V) and $R.\bar{A}$. For example, if dropping a dispensable attribute from $Attr(V(R))_{\mathsf{SELECT}}$ helps proving the view extend parameter, POC will drop it.

Case 1. $\mathcal{VE} = \subseteq$ or $\equiv$. In this case all conditions in the WHERE clause must be preserved in any legal rewriting, thus we have this property expressed in Equations (3.16) and (3.17). And, all attributes from $Attr(V(R))_{\mathsf{SELECT}}$ not in $R.\bar{A}$ should be used in dispensable components in the SELECT clause (and we will drop them in the view V'). This is expressed in Equation (3.18). For the aggregation views (Equation (3.4)), Equation (3.18) states that the group-by attributes of relation R (i.e., $grAttr(V(R))$) *not* in $R.\bar{A}$ must be dispensable in V.

$$Attr(V(R))_{\mathsf{WHERE}} \subseteq R.\bar{A} \tag{3.16}$$

$$R.A \notin Attr(V(R))_{\mathsf{WHERE}}(\_, f) \tag{3.17}$$

$$\text{SPJ Views:} \quad (Attr(V(R))_{\mathsf{SELECT}} \setminus R.\bar{A}) \subseteq Attr(V(R))_{\mathsf{SELECT}}(t, \_)$$

$$\text{AGGR-SPJ Views:} \quad (grAttr(V(R)) \setminus R.\bar{A}) \subseteq grAttr(V(R))(t, \_) \tag{3.18}$$

<u>Case 2. $\mathcal{VE} = \supseteq$</u>. In this case, there is no restriction on the attributes of R in the view and those in $R.\bar{A}$. We can drop the nonreplaceable conditions using the attribute $R.A$ and still have the view-extent parameter satisfied[5].

<u>Case 3. $\mathcal{VE} = \approx$</u>. No restriction in this case either.

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 (Equation (3.5)) must satisfy the property from Equation (3.19).

$$\begin{array}{llll} \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \equiv, & \text{then } \phi \text{ must be } \equiv; \\ \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \subseteq, & \text{then } \phi \text{ must be } \subseteq \text{ or } \equiv; \text{ and} \\ \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \supseteq, & \text{then } \phi \text{ must be } \supseteq \text{ or } \equiv. \end{array} \tag{3.19}$$

3.4.6 The POC Algorithm for "*delete-attribute*" Capability Change

Let an SPJ view V be defined as in Equation (3.3) and S be a relation that satisfies Conditions 1 through 4. The the view V' in Equations (3.22) and (3.23) is obtained from V (in Equation (3.3)) by replacing R with $R' \bowtie_{R'.\bar{A}'=S.\bar{B}'} S$. For the aggregation view V defined in Equation (3.4), the view V' obtained by using the relation S for replacing the attribute $R.A$, is given in Equations (3.20) and (3.21).

Note that it is sufficient (see the proof for Theorem 6) to restrict the join condition only on attributes of R used in the view, i.e., the new join condition added in the WHERE clause is actually $(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))$. Equations (3.22) and (3.20) are the legal rewritings when the view-extent evolution parameter is $\subseteq$ or $\equiv$. For the view-extent evolution parameter either $\supseteq$ or $\approx$, the legal rewritings

[5]Note that if a condition in the WHERE clause of the view V referring to the attribute $R.A$ is nonreplaceable, then it must be dispensable following from the Theorem 1 and the fact that the view V is assumed to be evolvable under "*delete-attribute* $R.A$".

for the two types of views (SPJ and aggregation) are given in Equations (3.23) and (3.21), respectively. The differences between the two forms of rewritings is that for the cases $\mathcal{VE} \in \{\subseteq, \equiv\}$, all conditions from the WHERE clause are kept (with the attribute $R.A$ replaced by its replacement $S(R.A)$), and the dispensable attributes of the relation R from the SELECT clause *not* in the set $R.\bar{A}$ are dropped from the view interface. These constraints are necessarily to prove the view-extent evolution parameter satisfaction (see the proof for Theorem 6).

$$
\begin{array}{ll}
\text{CREATE VIEW} & V' \ \ (\mathcal{VE} = \subseteq \mid \equiv) \ \text{AS} \\
\text{SELECT} & \underline{(grAttr(V(R)) \ \cap \ R.\bar{A}) \setminus \{R.A\}}, \\
 & \underline{\{S.B | S.B = S(R.A) \text{if R.A is replaceable}\}} \\
 & \underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n}, \\
 & \underline{\mathcal{F}(S(R.A), (aggAttr(V(R)) \setminus \{R.A\}), \bar{F})} \\
\text{FROM} & \underline{S, R', R_1, \ldots, R_n} \\
\text{WHERE} & \underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))}, \\
 & \underline{\mathcal{CV}(S(R.A), (Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}), \bar{W})} \\
\text{GROUP BY} & \underline{(grAttr(V(R)) \cap R.\bar{A}) \setminus \{R.A\}}, \\
 & \underline{\{S.B | S.B = S(R.A) \text{if R.A is replaceable}\}} \\
 & \underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n}
\end{array}
\tag{3.20}
$$

$$
\begin{array}{ll}
\text{CREATE VIEW} & V' \ \ (\mathcal{VE} = \supseteq \mid \approx) \ \text{AS} \\
\text{SELECT} & \underline{grAttr(V(R)) \setminus \{R.A\}}, \\
 & \underline{\{S.B | S.B = S(R.A) \text{if R.A is replaceable}\}} \\
 & \underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n}, \\
 & \underline{\mathcal{F}(S(R.A), (aggAttr(V(R)) \setminus \{R.A\}), \bar{F})} \\
\text{FROM} & \underline{S, R', R_1, \ldots, R_n} \\
\text{WHERE} & \underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))}, \\
 & \underline{\mathcal{CV}(S(R.A), (Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}), \bar{W})} \\
\text{GROUP BY} & \underline{grAttr(V(R)) \setminus \{R.A\}}, \\
 & \underline{\{S.B | S.B = S(R.A) \text{if R.A is replaceable}\}} \\
 & \underline{R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n}
\end{array}
\tag{3.21}
$$

In all rewritings, the attribute $R.A$ is replaced in the SELECT clause only if in components using it that are replaceable (if not, the the aggregate attribute or the group-by attribute must be dispensable, in which case we just drop it from the view interface). In Equations (3.23) and (3.21) we can drop nonreplaceable conditions referring $R.A$: $\mathcal{CV}$ thus denotes the conjunction $\mathcal{CV}$ from where we dropped nonreplaceable conditions referring to $R.A$.

$$
\begin{array}{ll}
\text{CREATE VIEW} & V' \;\; (\mathcal{VE} = \subseteq \mid \equiv) \text{ AS} \\
\text{SELECT} & \underline{(Attr(V(R))_{\text{SELECT}} \;\cap\; R.\bar{A}) \setminus \{R.A\}}, \\
 & \underline{\{S.B \mid S.B = S(R.A) \; if \; R.A \text{ is replaceable}\}}, \\
 & R_1.\bar{D}_1 \ldots, R_n.\bar{D}_n \\
\text{FROM} & \underline{S}, \underline{R'}, R_1, \ldots, R_n \\
\text{WHERE} & \underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))} \text{ AND} \\
 & \underline{\mathcal{CV}(S(R.A), Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}, \bar{W})}
\end{array}
\tag{3.22}
$$

$$
\begin{array}{ll}
\text{CREATE VIEW} & V' \;\; (\mathcal{VE} = \supseteq \mid \approx) \text{ AS} \\
\text{SELECT} & \underline{(Attr(V(R))_{\text{SELECT}} \;\setminus\; \{R.A\})}, \\
 & \underline{\{S.B \mid S.B = S(R.A) \; if \; R.A \text{ is replaceable}\}}, \\
 & R_1.\bar{D}_1 \ldots, R_n.\bar{D}_n \\
\text{FROM} & \underline{S}, R', R_1, \ldots, R_n \\
\text{WHERE} & \underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))} \text{ AND} \\
 & \underline{\mathcal{CV}(S(R.A), Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}, \bar{W})}
\end{array}
\tag{3.23}
$$

We make the observation here that in the case of the "*delete-attribute* $R.A$" capability change a strategy for view synchronization is to replace the whole relation R (using for example, the POC algorithm for the capability change "*delete-relation* R"). However, there are cases when replacing only the attribute being deleted is more efficient or the only solution given a $\mathcal{PC}$-constraint. In the case the view-extent parameter is $\supseteq$ or $\approx$, the POC algorithm for "*delete-attribute* " obtains a legal rewriting (Equations (3.23) and (3.21)) with more attributes preserved in the SELECT clause as opposed to the POC algorithm for "*delete-relation* " which must dispose the attributes not in $R.\bar{A}$. The cost model for view synchronization introduced in [27] has one dimension that measures the degree of preservation of the view interface of legal rewritings. In the case of "*delete-attribute* " capability change, replacing only the affected attribute will, in general, have a higher preservation degree than replacing the entire affected relation[6].

The conditions that must be satisfied for applying the POC algorithm for "*delete-attribute* " are less restrictive than for "*delete-relation* " capability change. For example, to replace the entire relation R requires that all attributes of R used in the view are dispensable or replaceable, condition that is not required to be satisfied in the case only the affected attribute is replaced.

As in the case of capability change "*delete-relation* ", we can prove that the above conditions are *sufficient* (but not *necessary*) to obtain a legal rewriting by using the replacement $S(R.A)$ for the deleted attribute. Theorem 6 proves that the rewritings described in Equations (3.22) and (3.23) for SPJ views and Equations (3.20) and (3.21) for aggregation views are legal rewriting conforming to Definition 1.

Theorem 6 *Let the view V (Equations (3.3) and (3.4) for* SPJ *and aggregation views, respectively) be an E-SQL view that is evolvable (Theorem 1) under capability change "delete-attribute $R.A$". The attribute $R.A$ is replaceable in at least one of the view components (i.e., a condition in the* WHERE *clause or in the* SELECT *clause). Then the rewriting V' obtained by applying POC Algorithm (Equations (3.22), (3.23), and (3.20), (3.21)) is a legal rewriting (Definition 1) under capability change "delete-attribute $R.A$".*

For the aggregation view V' defined in Equations (3.20) and (3.21), the aggregate attributes have the properties stated in the Tables 3.6 and 3.7.

Proof. See Appendix A.2.

[6]However, when replacing only the attribute, the number of relations involved in the view is increased by one which, in the long run, increases the cost of view maintenance (another dimension in our cost model), such as maintenance after data update.

3.5 The SPOC View Synchronization Strategy

In this section, we describe our SPOC algorithm for rewriting E-SQL views based on exploiting $\mathcal{PC}$-constraints in the SELECT-PROJECT general form. First, we give the sufficiency conditions for finding a suitable substitution for the two types of capability changes "*delete-relation* " and "*delete-attribute* ". Then we describe how SPOC obtains a rewriting using these conditions. Lastly, we prove that the rewritings found using SPOC are legal, i.e., they conform to Definition 1.

Our goal is to find the "best" rewriting of the view V using some alternate relation S once R itself or one of its attributes has been dropped. A "best" rewriting preserves the view extent such that the difference between the old view extent and the new one is minimal. In the previous work in the literature done on rewriting queries using views (see [61]), the rewritings are obtained such that the old and new views are *equivalent in extents* and that the later preserves *all attributes* of the former (the rewriting process fails if this is not possible). These techniques match (syntactically rather than semantically) parts of the query with view definitions, whereas we now focus on semantic matches. That is, instead of requiring necessarily equivalence, we attempt to establish some (subset, superset, equivalent) relationship with the old view extent preserving *at least* indispensable attributes. Our strategies replace affected view components with other elements obtained by using containment constraints. The need to rewrite the view definition in EVE is imposed by capability changes that render the views undefined, while rewriting a query using views has been proposed in the literature [33, 61] in order to optimize the query execution.

In the following we assume a generic E-SQL view V for the capability change "*delete-relation* R", defined as in Equation (3.24) for SPJ view type and as in Equation

(3.25) for AGGR-SPJ view type (see Table 2.5 for the notation conventions). The relation R is a replaceable relation used in the FROM clause of the view V since its replaceable evolution parameter $\mathcal{RR}$ is set to $true$. Moreover, the view V is evolvable (Theorem 1) under this capability change. Namely, (1) there are no attributes of R used in the SELECT clause in a component (e.g., a group-by attribute, an aggregate attribute for AGGR-SPJ views) with evolution parameters set to $(false, false)$, i.e., $Attr(V(R))_{\text{SELECT}}(f, f) = \emptyset$[7] and (2) the attributes of R used in the WHERE clause are referred to in conditions with evolution parameters different than $(false, false)$, i.e., $Attr(V(R))_{\text{WHERE}}(f, f) = \emptyset$.

$$
\begin{array}{ll}
\text{CREATE VIEW} & V \ \ (\mathcal{VE} = \delta) \ \text{AS} \\
\text{SELECT} & Attr(V(R))_{\text{SELECT}}(\_, t), Attr(V(R))_{\text{SELECT}}(t, f), \bar{T} \\
\text{FROM} & R(\mathcal{RR} = true), R_1, \ldots, R_n \\
\text{WHERE} & \mathcal{CV}(Attr(V(R))_{\text{WHERE}}(\_, t), Attr(V(R))_{\text{WHERE}}(t, f), \bar{W})
\end{array}
\tag{3.24}
$$

$$
\begin{array}{ll}
\text{CREATE VIEW} & V \ \ (\mathcal{VE} = \delta) \ \text{AS} \\
\text{SELECT} & grAttr(V(R))(\_, t), grAttr(V(R))(t, f), \bar{T}, \\
 & \mathcal{F}(aggAttr(V(R))(\_, t), aggAttr(V(R))(t, f), \bar{F}) \\
\text{FROM} & R(\mathcal{RR} = true), R_1, \ldots, R_n \\
\text{WHERE} & \mathcal{CV}(Attr(V(R))_{\text{WHERE}}(\_, t), Attr(V(R))_{\text{WHERE}}(t, f), \bar{W}) \\
\text{GROUP BY} & grAttr(V(R))(\_, t), grAttr(V(R))(t, f), \bar{T}
\end{array}
\tag{3.25}
$$

In Equations (3.24) and (3.25), $\bar{T}$ is the set of attributes from the relations

[7] Note that the attributes of relation R in the GROUP BY clause are assumed to be exactly the attributes from the SELECT clause of relation R and thus they inherit the evolution parameters from the SELECT clause attributes. For the view V defined in Equation (3.25), $Attr(V(R))_{\text{SELECT}}(d, r) = grAttr(V(R))(d, r) \cup aggAttr(V(R))(d, r)$ (see Table 2.5).

$R_1, \ldots, R_n$ used in the SELECT clause. $\mathcal{CV}()$ denotes a conjunction of primitive clauses in the WHERE clause of V referring to attributes from relation R (i.e., $Attr(V(R))_{\text{WHERE}}(\_, t)$, $Attr(V(R))_{\text{WHERE}}(t, f)$) and attributes from the rest of the relations ($\bar{W} \subseteq \text{Attr}(R_1) \cup \ldots \cup \text{Attr}(R_n)$). For the aggregation view V defined in Equation (3.25), the expression $\mathcal{F}(aggAttr(V(R)), \bar{F})$ stands for a set of aggregate functions $f_1(X_1), \ldots, f_k(X_k)$ using the attributes $aggAttr(V(R))$ of relation R and the attributes $\bar{F}$ from the relations $R_1, \ldots, R_n$. Note that for the view V to be evolvable under capability change "*delete-relation R*", all aggregate functions using attributes from $aggAttr(V(R))$ cannot be both indispensable and nonreplaceable (following from Theorem 1).

Example 10 *To illustrate the SPOC algorithm, we assume the view* ***Customer-Passengers*** *defined to select all customers that fly in "Asia" and are younger than* 20 *as given in Eq. (3.26) (the MISD relation descriptions are given in Figure 2.1). We will use this example throughout this paper to show how the view definition is synchronized under the capability change "delete-relation* ***Customer****".*

$$
\begin{array}{ll}
\textit{CREATE VIEW} & \textbf{\textit{Customer-Passengers}}\ (\mathcal{VE} = \subseteq)\ \textit{AS} \\
\textit{SELECT} & \textbf{\textit{C.Name}}\ (\mathcal{AD} = false, \mathcal{AR} = true), \\
 & \textbf{\textit{C.Age}}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\
\textit{FROM} & \textbf{\textit{Customer C}}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
 & \textbf{\textit{FlightRes F}}\ (\mathcal{RD} = true, \mathcal{RR} = true) \\
\textit{WHERE} & (\textbf{\textit{C.Name}} = \textbf{\textit{F.PName}})\ (\mathcal{CD} = false, \mathcal{CR} = true)\ \textit{AND} \\
 & (\textbf{\textit{F.Dest}} =' Asia')\textit{AND} \\
 & (\ \textbf{\textit{C.Age}} < 20)(\mathcal{CD} = false, \mathcal{CR} = true)
\end{array}
\tag{3.26}
$$

The elements of the view V relative to R are given in Figure 3.8 where for simplicity we use V to stand for the view ***Customer–Passengers*** *and we use R to represent the relation* ***Customer****. Note that elements of the generic view definition given in Equation (3.24) could be empty sets (for example,* $\bar{T}$ *and* $Attr(V(R))_{\text{SELECT}}(t, f)$*).*

Element of V	*Definition*
$Attr(V(R))_{SELECT}(f,f)$, $Attr(V(R))_{SELECT}(t,f)$	$\emptyset$
$Attr(V(R))_{SELECT}(f,t)$	***Customer.Name***
$Attr(V(R))_{SELECT}(t,t)$	***Customer.Age***
T	$\emptyset$
R_1	***FlightRes F***
$Attr(V(R))_{WHERE}(f,f)$, $Attr(V(R))_{WHERE}(t,f)$, $Attr(V(R))_{WHERE}(t,t)$	$\emptyset$
$Attr(V(R))^{JOIN}_{WHERE}(f,t)$	***Customer.Name***
$Attr(V(R))^{LOCAL}_{WHERE}(f,t)$	***Customer.Age***
W	***F.PName, F.Dest***

*Figure 3.5: Elements of the view $V =$ **Customer-Passengers** relative to relation $R =$ **Customer**.*

3.5.1 Select-Project Information in $\mathcal{PC}$-constraints

Let V be an evolvable E-SQL view as defined in Eqs. (3.24) and (3.25) for SPJ and aggregation views, respectively. Let $\mathcal{PC}_{S,R}$ be a containment constraint in the SELECT-PROJECT form defined by Eq. (3.27) with the relation R one of the relations used in the FROM clause of the view V.

$$\mathcal{PC}_{R,S} = \left(\underbrace{\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S))}_{\mathcal{PC}(S)} \ \phi \ \underbrace{\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))}_{\mathcal{PC}(R)} \right) \tag{3.27}$$

In Eq. (3.27), $\mathcal{C}_S(S.\bar{B}_1)$ and $\mathcal{C}_R(R.\bar{A}_1)$ are conjunctions of primitive clauses over the attributes of the relations S (i.e, $S.\bar{B}_1$) and R (i.e, $R.\bar{A}_1$), respectively. We denote the select-project expression of the $\mathcal{PC}$-constraint referring to the relation S by $\mathcal{PC}(S)$ and the one referring to R by $\mathcal{PC}(R)$.

Definition 6 ***$\mathcal{PC}$-constraint Normal Form.*** *We say that a $\mathcal{PC}$-constraint is in normal form if all select conditions using the attributes projected out (i.e., $S.\bar{A}$ or $R.\bar{B}$ in Equation (3.27))) appear on both sides of the $\mathcal{PC}$-constraint.*

Example 11 *If the $\mathcal{PC}$-constraint is defined as*

$$\mathcal{PC}_{R,S} = (\pi_{A_1,A_2}(\sigma_{A_1>10}(R)) \ \phi \ \pi_{B_1,B_2}(S)) \tag{3.28}$$

The normal form of the $\mathcal{PC}$-constraint from Equation (3.28) is given in Equation (3.29).

$$\mathcal{PC}_{R,S} = (\pi_{A_1,A_2}(\sigma_{A_1>10}(R)) \ \phi \ \pi_{B_1,B_2}(\sigma_{B_1>10}(S))) \tag{3.29}$$

Note, that the two containment constraints are equivalent in the sense that one implies the other for any $\phi \in \{\subseteq, \equiv, \supseteq\}$.

Example 12 *In our running Example 10, we consider the capability change "delete-relation* ***Customer****". Let's further assume that the MKB contains a containment constraint between the relation* ***Customer*** *and a relation* ***Accident–Ins****, as defined in Equation (3.30)). Note that this $\mathcal{PC}$-constraint is not in normal form: the condition $(_{Accident-Ins.Age\,<\,50})$ refers to the projected attribute $_{Accident-Ins.Age}$ but a similar condition on the corresponding attribute $_{Customer.Age}$ doesn't appear in the right hand side of the constraint (Definition 6). To normalize this containment constraint, it is sufficient to add the condition $(_{Customer.Age\,<\,50})$ to the select conditions for the relation $_{Customer}$. We obtain in this way a new equivalent $\mathcal{PC}$-constraint shown in Equation (3.31). The elements of these $\mathcal{PC}$-constraints are shown in Figure 3.6.*

$$\begin{aligned} &\mathcal{PC}_{Customer,Accident-Ins} = \\ &\quad (\pi_{Accident-Ins.Holder,Accident-Ins.Age} \\ &\quad\quad (\sigma_{(Accident-Ins.Amount > 1,000,000)\ AND\ (Accident-Ins.Age < 50)} \textbf{\textit{Accident–Ins}})) \\ &\quad\quad \subseteq \pi_{Customer.Name,Customer.Age}(\textbf{\textit{Customer}}) \end{aligned} \tag{3.30}$$

$$\begin{aligned} &\mathcal{PC}_{Customer,Accident-Ins} = \\ &\underbrace{\pi_{Accident-Ins.Holder,Accident-Ins.Age}}_{S.\bar{B}} \\ &\quad (\sigma_{\underbrace{(Accident-Ins.Amount > 1,000,000)\ AND\ (Accident-Ins.Age < 50)}_{\mathcal{C}_S(S.\bar{B}_1)}} \textbf{\textit{Accident–Ins}}) \\ &\quad \subseteq \ \underbrace{\pi_{Customer.Name,Customer.Age}}_{R.\bar{A}}(\sigma_{\underbrace{(Customer.Age < 50)}_{\mathcal{C}_R(R.\bar{A}_1)}} \textbf{\textit{Customer}}) \end{aligned} \tag{3.31}$$

Element of $\mathcal{PC}$	Definition
$R.A$	{ **Customer.Name, Customer.Age** }
$R.A_1$	{ **Customer.Age** }
$\mathcal{C}_R(R.A_1)$	(**Customer.Age** < 50)
$S.B$	{ **Accident-Ins.Holder, Accident-Ins.Age** }
$S.B_1$	{ **Accident-Ins.Amount, Accident-Ins.Age** }
$\mathcal{C}_S(S.B_1)$	(**Accident-Ins.Amount** > 1,000,000) AND (**Accident-Ins.Age** < 50)

Figure 3.6: Elements of the $\mathcal{PC}_{\textbf{Customer, Accident-Ins}}$-constraint relative to the relations R = **Customer** and S = **Accident-Ins**.

The SPOC algorithm introduced below makes use of a $\mathcal{PC}$-constraint between the to-be-deleted relation R (such as **Customer**) and another relation S (e.g., **Accident-Ins**) to replace elements of the view V referring to the relation R. We introduce the concept of the maximal expression on R in the view V, denoted by $V(R)$, to represent those elements of the view definition V that are affected by the capability change "*delete-relation* R" and hence need to be replaced by elements of S. Intuitively, any view V referring to the relation R is obtained from some fragment of the relation R, namely only those rows that satisfy the local conditions on R and those columns that are needed in the SELECT clause or in some join conditions. Definition 7 formally defines this concept while Example 13 gives an illustrative example of how $V(R)$ could be found for an SPJ view.

Definition 7 ***Maximal Expression on*** R ***in a View*** V***.*** *Let the view V be defined as either an SPJ view in Eq. (3.24) or as an aggregation view in Eq. (3.25). V can be written by highlighting the "maximal" expression on R used in V, denoted by $V(R)$. Formally, let $\mathcal{C}_V(Attr(V(R))^{LOCAL}_{WHERE})$[8] be the conjunction of all local select conditions in the WHERE clause of the view V referring only to attributes of R ($\mathcal{C}_V$ contains only a subset of the primitive clauses from $\mathcal{CV}$). Let $\mathcal{C}_{Rest}(Attr(V(R))^{JOIN}_{WHERE}, \bar{W})$ denotes the rest of the primitive clauses in $\mathcal{CV}$, i.e., it contains all join conditions*

[8]See Figure 2.5 for definitions of $Attr(V(R))_{\mathsf{WHERE}}$and $Attr(V(R))_{\mathsf{SELECT}}$.

using attributes from the relation R and all join and select conditions not referring to attributes from R. That is, the condition $\mathcal{CV}$ could be written as following: $\mathcal{CV}(Attr(V(R))_{\textsf{WHERE}}(\_, t), Attr(V(R))_{\textsf{WHERE}}(t, f), \bar{W}) \equiv \mathcal{C}_V(Attr(V(R))^{LOCAL}_{\textsf{WHERE}})$ AND $\mathcal{C}_{Rest}(Attr(V(R))^{JOIN}_{\textsf{WHERE}}, \bar{W})$*. Then the view V could be represented as in Eq. (3.32) where the maximal expression of R in V is the expression $V(R)$.*

$$V = \Pi_{\mathrm{Attr}(V)} \left(\underbrace{\pi_{Attr(V(R))_{\mathsf{SELECT}} \cup Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}}(\sigma_{\mathcal{C}_V(Attr(V(R))^{\mathrm{LOCAL}}_{\mathsf{WHERE}})}(R))}_{V(R)} \right.$$
$$\left. \bowtie_{\mathcal{C}_{Rest}(Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}, \bar{W})} \; (R_1 X \ldots X R_n) \right) \tag{3.32}$$

The operator $\Pi_{\bar{Y}}$ in Eq. (3.32) is the *generalized project operator* introduced in [14]. This operator is an extension of duplicate-eliminating projection, where the projected attributes $\bar{Y}$ can include aggregate functions and group-by attributes. The generalized projection operator Π without aggregate functions is identical to the duplicate-eliminating projection operator π.

Example 13 *In our running Example 10, the view* ***Customer-Passengers*** *(Equation (3.26)) could be written as shown in Eq. (3.33) using the maximal expression on the relation* ***Customer****,* $V(R) =$ ***Customer-Passengers(Customer)****.*

$$\textbf{\textit{Customer-Passengers}} \; = $$
$$\pi_{Customer.Name,\; Customer.Age} \left(\underbrace{(\pi_{Customer.Name, Customer.Age}(\sigma_{(Customer.Age < 20)}(\textbf{\textit{Customer}})))}_{V(R)} \right.$$
$$\left. \bowtie_{\underbrace{(Customer.Name = \; FlightRes.PName) \; (AND \;\; FlightRes.Dest \; =' Asia')}_{\mathcal{C}_{Rest}}} \; (\textbf{\textit{FlightRes}}) \right) \tag{3.33}$$

The expressions $V(R)$, $\mathcal{C}_V()$ and $\mathcal{C}_{Rest}()$ are also summarized in Figure 3.7.

Element of V	Definition
$V(R)$	$\pi_{\text{Customer.Name,Customer.Age}}(\sigma_{(\text{Customer.Age}<20)}(\textbf{Customer}))$
$\mathcal{C}_V$	(Customer.Age < 20)
$\mathcal{C}_{Rest}$	(Customer.Name = FlightRes.PName) AND (FlightRes.Dest = 'Asia')

Figure 3.7: Elements of the maximal expression $V(R)$ on the relation $R =$ **Customer** in the view $V =$ **Customer-Passengers.**

When the relation R or one of its attributes is deleted, the replacement algorithm must find a substitution for the expression $V(R)$ defined in Eq. (3.32). Below we now define the conditions that must hold between the expression $V(R)$ and the expression $\mathcal{PC}(R) = \pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$ from the $\mathcal{PC}$-constraint given in Eq. (3.27), such that the expression $\mathcal{PC}(S)$ could be used to replace $V(R)$ in V. Note that, the SPOC algorithm obtains a legal rewriting V' by replacing the expression $V(R)$ with the expression $\mathcal{PC}(S)$ for the "*delete-relation R*" capability change.

3.5.2 Condition for Replacement Used by the SPOC Algorithm

We give first the general condition for a relation to be a suitable replacement and then we treat separately the two capability changes that require substitution for synchronization. We discuss in each case the extra properties a candidate substitution must have and then give the SPOC algorithm for obtaining legal rewritings using these candidate relations.

A relation $IS_2.S$ is said to be a *Select-Project Substitution* for a to-be-deleted relation R, if the following conditions are satisfied.

Condition 1: Select-Project-Relation Containment. The containment constraint in normal form (Definition 6) from Eq. (3.27) is satisfied for all states of the relations S and R. Put differently, the $\mathcal{PC}$-constraint defined in Eq. (3.27) is available in MKB. The attributes $S.\bar{B} \subseteq a(S)$ are called *replacements* for the attributes $R.\bar{A} \subseteq a(R)$. That is, for any pair $(R.A_i, S.B_i)$ of attributes having the same position in the vectors $R.\bar{A}$ and $S.\bar{B}$, respectively, the following type constraints exist in MKB.

Example 14 *In our running Example 10, for the containment constraint $\mathcal{PC}_{\textit{Customer, Accident-Ins}}$ from the Equation (3.31), the attributes $S.\bar{B} = \{$ **Accident-Ins.Holder, Accident-Ins.Age** $\}$ of the relation **Accident-Ins** are replacements for the attributes $R.\bar{A} = \{$ **Customer.Name, Customer.Age** $\}$ of the relation **Customer**.*

3.5.3 Conditions for Replacement Used by the SPOC Algorithm: "*delete-relation*" Capability Change

The conditions 2 through 4 given below are the extra conditions a candidate select-project-substitution relation S must fulfill in order for the rewriting obtained by replacing the entire relation R with S (view V' defined in Eqs. (3.45) and (3.46)) to be a legal rewriting. In Theorem 7 we prove the sufficiency of these conditions, i.e., that the view V' is a legal rewriting conforming to Definition 1. Tables 3.8, 3.9, and 3.10 and 3.11 summarize the conditions and rewriting strategies for the SPJ and AGGR-SPJ views, respectively.

Condition 2: Indispensable Evolution Parameter Satisfaction. This condition requires that relation S must contain replacements for *at least* all attributes of the relation R that are indispensable and replaceable in the view V. This is formally stated in the Eq. (3.34)[9]:

[9]Note that for the aggregate views, $Attr(V(R))_{\mathsf{SELECT}}(f,t) = grAttr(V(R))(f,t) \cup$

$$Attr(V(R))_{\mathsf{SELECT}}(f,t) \ \cup \ Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(f,t) \ \subseteq \ R.\bar{A}. \tag{3.34}$$

Note that here we don't impose that all attributes of relation R used in the WHERE clause in indispensable and replaceable conditions $(Attr(V(R))_{\mathsf{WHERE}}(f,t))$ have replacements in $S.\bar{B}$. Given a select-project $\mathcal{PC}$-constraint, it is possible to replace the whole indispensable and replaceable *local* conditions in V (i.e., the ones with attributes in $Attr(V(R))^{\mathrm{LOCAL}}_{\mathsf{WHERE}}(f,t))$ by conditions in $\mathcal{C}_S(S.\bar{B}_1)$ from $\mathcal{PC}(S)$ expression in Eq. (3.27). The extra condition for applying this strategy is given by Condition 3 below.

Example 15 *Let's check this condition for our example. The elements of the relation* ***Customer*** *in the view* ***Customer-Passengers*** *are summarized in Figure 3.5 while the elements of the containment constraint* $\mathcal{PC}_{\textit{Customer, Accident-Ins}}$ *are shown in Figure 3.6. Thus, the condition from Equation (3.34) becomes:*

$\underbrace{\{\textbf{\textit{Customer.Name}}\}}_{Attr(V(R))_{SELECT}(f,t)} \cup \underbrace{\{\textbf{\textit{Customer.Name}}\}}_{Attr(V(R))^{JOIN}_{WHERE}(f,t)} \subseteq \underbrace{\{\textbf{\textit{Customer.Name, Customer.Age}}\}}_{R.\bar{A}}$. *Thus, so far, our candidate replacement relation* $S =$ ***Accident-Ins*** *is satisfying both Condition 1 and 2.*

Condition 3: View-Extent Satisfaction. This condition expresses the properties that must hold for the $\mathcal{PC}$-constraint defined by Eq. (3.27) in order for the *view-extent evolution parameter* $\mathcal{VE}$ to be satisfied. We must make a distinction among different possible values of the parameter $\mathcal{VE}$, hence three distinct cases are discussed below.

$aggAttr(V(R))(f,t))$.

<u>Case 1. $\mathcal{VE} = \subseteq$</u>. This case requires that all conditions in the WHERE clause of V be preserved regardless of their evolution parameters, since we must have $V' \subseteq V$. Because the attributes from $Attr(V(R))_{\mathsf{WHERE}}(t,f)$ cannot be replaced (due to the corresponding conditions in the WHERE clause being nonreplaceable ($\mathcal{CR} = false$)) the conditions referring to them can only be dropped. But when $\mathcal{VE} = \subseteq$, we cannot drop any primitive clause from the WHERE clause. Hence, we have to impose that the set $Attr(V(R))_{\mathsf{WHERE}}(t,f)$ is empty. This fact is formalized in Eq. (3.35).

$$Attr(V(R))_{\mathsf{WHERE}}(t,f) = \emptyset \tag{3.35}$$

For the rest of the replaceable WHERE conjuncts using attributes of R in the WHERE clause (referring to attributes in $Attr(V(R))^{\mathrm{LOCAL}}_{\mathsf{WHERE}}(\_,t)$ and $Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(\_,t)$) we have two options:

(1) replace their affected attributes with replacements from $S.\bar{B}$; or

(2) replace the whole condition with select conditions from the $\mathcal{PC}$-constraint given in Eq. (3.27).

For replaceable join primitive clauses in the WHERE clause of V (using attributes in $Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(f,t)$ and $Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(t,t)$), we cannot apply option (2) as all select conditions in the $\mathcal{PC}$-constraint are *local* conditions. For this reason, we have to impose that these attributes have replacements in $S.\bar{B}$ (option (1)). Thus the inclusion given in Eq. (3.36) must be true:

$$Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(f,t) \cup Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(t,t) \subseteq R.\bar{A} \tag{3.36}$$

For the rest of replaceable conjuncts, i.e., the ones referring to attributes in $Attr(V(R))^{\mathrm{LOCAL}}_{\mathsf{WHERE}}(\_,t)$, we can impose either option (1) or (2). Eq. (3.37) describes this condition.

$$\forall R.C \in Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\_,t) \text{ used in a local conjunct of } CV(R.C) \text{ in } \mathcal{C}_V,$$
$$\text{one of the properties holds:} \tag{3.37}$$
$$(i)\ R.C \text{ has a replacement in } S.\bar{B}, \text{ or}$$
$$(ii)\ \exists C_R(R.C) \text{ a condition in } \mathcal{C}_R(R.\bar{A}_1) \text{ of } \mathcal{PC}(R) \text{ s.t. } C_R(R.C) \rightarrow CV(R.C)^{10}.$$

Example 16 *Our running example falls in this particular Case 1. That is, with the elements of the containment constraint* $\mathcal{PC}_{\textit{Customer, Accident-Ins}}$ *and of the view* $V =$ ***Customer-Passengers*** *(Figures 3.6 and 3.7, respectively), we can prove that the conditions required by Condition 3, Case 1 are satisfied. The condition from Equation (3.35) is fulfilled since* $Attr(V(R))_{\textit{WHERE}}(t,f) = \emptyset$.

The condition from Equation (3.36) is translated into:

$$\underbrace{\{\textbf{\textit{Customer.Name}}\}}_{Attr(V(R))^{JOIN}_{WHERE}(f,t)} \cup \underbrace{\emptyset}_{Attr(V(R))^{JOIN}_{WHERE}(t,t)} \subseteq \underbrace{\{\textbf{\textit{Customer.Name, Customer.Age}}\}}_{R.\bar{A}}.$$

Moreover, the condition from Equation (3.37) is satisfied: $Attr(V(R))^{LOCAL}_{WHERE}(\_,t) = \{$ ***Customer.Age*** $\}$. *The condition (i) from Equation (3.37) is true for the attribute* ***Customer.Age*** *because the attribute* ***Accident-Ins.Age*** *is its replacement (from Condition 1).*

Case 2. $\mathcal{VE} = \supseteq$. In this case, the view can be evolved by dropping dispensable conditions from the WHERE clause and still have the view extent evolution parameter $\mathcal{VE} = \supseteq$ satisfied (i.e., $V' \supseteq V$). Thus, dispensable conditions referring to attributes in $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(t,\_)$ and $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(t,\_)$ could be dropped in the legal rewritings of V.

For indispensable and replaceable conditions (referring to attributes in $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(f,t)$), we again have two options as given in Eq. (3.38).

$$\forall R.C \in Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(f,t) \text{ used in a local conjunct } CV(R.C) \text{ in } \mathcal{C}_V\text{:}$$
$$\text{one of the properties holds:} \tag{3.38}$$
$$(i)\ R.C \text{ has a replacement in } S.\bar{B}, \text{ or}$$
$$(ii)\ \exists C_R(R.C) \text{ a condition in } \mathcal{C}_R(R.\bar{A}_1) \text{ of } \mathcal{PC}(R) \text{ s.t. } CV(R.C) \rightarrow C_R(R.C).$$

In this case, we have to make sure that the conjunction of conditions $\mathcal{C}_R(R.\bar{A}_1)$ (Eq. (3.27)) isn't more restrictive than the local conditions on R in V, i.e., $\mathcal{C}_V(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})$ (Eq. (3.32)). Thus we impose that any condition in $\mathcal{C}_R(R.\bar{A}_1)$ must be implied by some condition in $\mathcal{C}_V(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})$:

$$\forall C_R(R.C) \text{ a condition in } \mathcal{C}_R(R.\bar{A}_1),\ \exists CV(R.C) \text{ a condition in}$$
$$\mathcal{C}_V(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}) \text{ s.t. } CV(R.C) \rightarrow C_R(R.C). \tag{3.39}$$

<u>Case 3. $\mathcal{VE} = \equiv$</u>. In this case, all of the conditions imposed in Case 1 (Eqs. (3.35), (3.36), (3.37)) and Case 2 (Eqs. (3.38), (3.39)) must be satisfied in order to have the view extent evolution parameter $\mathcal{VE} =\equiv$ satisfied.

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. (3.27)) must satisfy the property from Eq. (3.40).

$$\begin{array}{llll} \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \equiv, & \text{then } \phi \text{ must be } \equiv; \\ \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \subseteq, & \text{then } \phi \text{ must be } \subseteq \text{ or } \equiv \text{; and} \\ \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \supseteq, & \text{then } \phi \text{ must be } \supseteq \text{ or } \equiv. \end{array} \tag{3.40}$$

Example 17 *Note that in our running example, the view* ***Customer-Passengers*** *has* *the view-extent evolution parameter* $\mathcal{VE}$ *set to "*$\subseteq$*" while for the containment constraint* $\mathcal{PC}_{\textit{Customer, Accident-Ins}}$*,* $\phi = \subseteq$*.*

3.5.4 The SPOC Algorithm for the "*delete-relation*" Capability Change

Let the view V be defined as in Eq. (3.24) for SPJ views and as in Eq. (3.25) for aggregation views. Let S be a relation that satisfies the four conditions from above. We give here the steps for finding a view rewriting for V under the capability change "*delete−relation* R" using select-project containment information.

Step 1. Projecting out from the $\mathcal{PC}$-constraint the not needed attributes. First, let's rewrite the $\mathcal{PC}$-constraint defined in Eq. (3.27) into a more restrictive form (implied by it) where the projected attributes are the ones needed in the view V, i.e., all replaceable attributes in $Attr(V(R))_{\text{SELECT}} \cup Attr(V(R))_{\text{WHERE}}$. If an affected attribute in the SELECT clause is not replaceable it will be dropped in any legal rewriting of V. Any nonreplaceable condition in the WHERE clause using attributes of R will be dropped as well. In the revised $\mathcal{PC}$-constraint in Eq. (3.41) only the replaceable attributes are projected, i.e., the ones in $Attr(V(R)) \setminus Attr(V(R))(t,f)$ (which we denote by $Attr(V(R))(\neg(t,f))$, see Table 2.5)[11].

$$\mathcal{PC}'_{R,S} = \underbrace{\pi_{S.\bar{B}'}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S))}_{\mathcal{PC}'(S)} \;\phi\; \underbrace{\pi_{R.\bar{A}\cap(Attr(V(R))(\neg(t,f))}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))}_{\mathcal{PC}'(R)} \tag{3.41}$$

In Eq. (3.41), $S.\bar{B}'$ is the vector of attributes from $S.\bar{B}$ (defined in Eq. (3.27)) containing replacements for the attributes from R, i.e., in $R.\bar{A} \cap Attr(V(R))(\neg(t,f))$.

Step 2. Augmenting the $\mathcal{PC}$-constraint with select conditions on R in V.

We can further rewrite the $\mathcal{PC}$-constraint defined in Eq. (3.41) by adding to the both sides all replaceable select conditions on relation R from the WHERE clause of

[11]There are no attributes in $Attr(V(R))(f,f)$ as the view is evolvable under delete relation R.

V (referring to attributes in $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}$) whose attributes are among $R.\bar{A} \cap Attr(V(R))(\neg(t,f))$. Note that common attributes of $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}$ and $R.\bar{A} \cap Attr(V(R))(\neg(t,f))$ are among $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\neg(t,f)) \cap R.\bar{A}$. The containment defined in Eq. (3.42) is hence implied by the $\mathcal{PC}$-constraint in Eq. (3.41)[12]. We denote by $\mathcal{C}_V(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\neg(t,f)) \cap R.\bar{A})$ the conjunction of select conditions from $\mathcal{C}_V(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})$ (defined in Eq. (3.32)) from which we have dropped the nonreplaceable dispensable primitive clauses, and replaceable dispensable clauses with no replacements found. The rewritten $\mathcal{PC}$ from Eq. (3.41) now becomes $\mathcal{PC}''$ given in Eq. (3.42):

$$\begin{aligned} &\mathcal{PC}''_{R,S} = \\ &\underbrace{\sigma_{\mathcal{C}_V(S.\bar{B}_2)}(\pi_{S.\bar{B}'}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)))}_{\mathcal{PC}''(S)}\ \phi \\ &\underbrace{\sigma_{\mathcal{C}_V(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\neg(t,f))\cap R.\bar{A})}(\pi_{R.\bar{A}\cap Attr(V(R))(\neg(t,f))}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R)))}_{\mathcal{PC}''(R)} \end{aligned} \tag{3.42}$$

where $S.\bar{B}_2$ are the replacement attributes in $S.\bar{B}$ corresponding to the attributes in $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\neg(t,f)) \cap R.\bar{A}$.

Example 18 *The first step of the SPOC Algorithm doesn't change the containment constraint $\mathcal{PC}_{\text{Customer,Accident-Ins}}$ (Eq. (3.31)) as all the attributes projected are also needed in the view* ***Customer-Passengers****. For the second step, we add to the left and right sides of this $\mathcal{PC}$-constraint the select condition (**Accident–Ins.Age** < 20) and (**Customer.Age** < 20), respectively (shown in Eq. (3.43)).*

$$\begin{aligned} &\mathcal{PC}''_{Customer,Accident-Ins} = \\ &\underbrace{\sigma_{Accident-Ins.Age<20}\left(\pi_{Holder,Age}(\sigma_{(Amount>1,000,000)\ AND\ (Age<50)}\boldsymbol{Accident\text{–}Ins})\right)}_{\mathcal{PC}''(S)} \\ &\subseteq \underbrace{\sigma_{Customer.Age<20}\left(\pi_{Customer.Name,Customer.Age}(\sigma_{(Customer.Age<50)}\ \boldsymbol{Customer})\right)}_{\mathcal{PC}''(R)} \end{aligned} \tag{3.43}$$

[12]We can further optimize this expression by adding *only* select conditions that are not already among the select conditions in the $\mathcal{PC}$-constraint.

Step 3. Obtaining the view rewriting.

The view V' defined in Eq. (3.44) is a final rewriting of the view V (Eq. (3.32)) obtained by replacing relation R with relation S, the attributes of R in V with their replacements in S, and the select conditions on R in V with conditions from the $\mathcal{PC}$-constraint. Namely, the maximal expression on R in V, which is $V(R)$ defined in Eq. (3.32), was replaced by the expression $\mathcal{PC}''(S)$ defined in Eq. (3.42).

$$V' = \Pi_{a(V)} \Bigg(\underbrace{\sigma_{\mathcal{C}_V(S.\bar{B_2})}(\pi_{S.\bar{B}'}(\sigma_{\mathcal{C}_S(S.\bar{B_1})}(S)))}_{\mathcal{PC}''(S)} \\ \bowtie_{\mathcal{C}_{Rest}(S(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(\neg(t,f))),\bar{W})} (R_1 X \ldots X R_n) \Bigg) \tag{3.44}$$

Eqs. (3.45) and (3.46) depict the SQL view definitions corresponding to the relational algebra expression of V' in Eq. (3.44) (for SPJ and aggregation views, respectively). $S(Attr(V(R))_{\text{SELECT}}(f,t))$ are the attribute replacements from $S.\bar{B}$ corresponding to all attributes in $Attr(V(R))_{\text{SELECT}}(f,t)$ ($\subseteq R.\bar{A}$). By Condition 2, $|S(Attr(V(R))_{\text{SELECT}}(f,t))| = |Attr(V(R))_{\text{SELECT}}(f,t)|$[13]). $S(Attr(V(R))_{\text{SELECT}}(t,t))$ are the replacements for $Attr(V(R))_{\text{SELECT}}(t,t)$ in $S.\bar{B}$ noting that not necessarily all attributes from $Attr(V(R))_{\text{SELECT}}(t,t)$ have replacements. In other words, $|S(Attr(V(R))_{\text{SELECT}}(t,t))| \leq |Attr(V(R))_{\text{SELECT}}(t,t)|$. Note that the attributes $Attr(V(R))_{\text{SELECT}}(t,f)$ were dropped in V' as they are nonreplaceable in V.

$$\begin{array}{ll} \text{CREATE VIEW} & V' \;\; (\mathcal{VE} = \delta) \text{ AS} \\ \text{SELECT} & \underline{S(\; Attr(V(R))_{\text{SELECT}}(f,t)\;), S(\; Attr(V(R))_{\text{SELECT}}(t,t)\;)}, \bar{T} \\ \text{FROM} & \underline{S}, R_1, \ldots, R_n \\ \text{WHERE} & \underline{\mathcal{CV}(S(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(\neg(t,f))),\; \bar{W})} \\ & \underline{\text{AND } \mathcal{C}_V(S.\bar{B}_2) \text{ AND } \mathcal{C}_S(S.\bar{B}_1)} \end{array} \tag{3.45}$$

[13]We denote by $|X|$ the cardinality of relation X.

Case 1. $\mathcal{VE} = \subseteq$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \subseteq \mid \equiv$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $Attr(V(R))_{\text{SELECT}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$	Condition 3: $Attr(V(R))_{\text{WHERE}}(t,f) = \emptyset$ $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(t,t) \subseteq R.\bar{A}$ $\forall CV(R.C)\ in\ \mathcal{CV}:$ $(i)\ R.C\ in\ R.\bar{A}\ or$ $(ii)\ \exists C_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1)$ $C_R(R.C) \rightarrow CV(R.C)$
CREATE VIEW V' $(\mathcal{VE} = \subseteq)$ AS SELECT $\underline{S(Attr(V(R))_{\text{SELECT}}(f,t)), S(Attr(V(R))_{\text{SELECT}}(t,t))}, \bar{T}$ FROM $\underline{S}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(\neg(t,f))), \bar{W})}$ $\underline{\text{AND } \mathcal{C}_V(S.B_2) \text{ AND } \mathcal{C}_S(S.B_1)}$		
Theorem 7: $V' \subseteq_\pi \mid \equiv_\pi V$		
Case 2. $\mathcal{VE} = \supseteq$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \supseteq \mid \equiv$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $Attr(V(R))_{\text{SELECT}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$	Condition 3: $\forall CV(R.C)\ in\ \mathcal{CV}:$ $(i)\ R.C\ in\ R.\bar{A}\ or$ $(ii)\ \exists C_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1)$ $CV(R.C) \rightarrow C_R(R.C)$ $\forall C_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1),$ $\exists CV(R.C)\ in\ \mathcal{CV}\ s.t.$ $CV(R.C) \rightarrow C_R(R.C)$
CREATE VIEW V' $(\mathcal{VE} = \supseteq)$ AS SELECT $\underline{S(Attr(V(R))_{\text{SELECT}}(f,t)), S(Attr(V(R))_{\text{SELECT}}(t,t))}, \bar{T}$ FROM $\underline{S}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(\neg(t,f))), \bar{W})}$ $\underline{\text{AND } \mathcal{C}_V(S.B_2) \text{ AND } \mathcal{C}_S(S.B_1)}$		
Theorem 7: $V' \supseteq_\pi V$		

Table 3.8: SPOC Synchronization Strategy for SPJ View V, Eq. (3.24): "*delete-relation R*".

Condition 1	Condition 2	Condition 3
Case 3. $\boxed{\mathcal{VE}\ = \equiv}$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \equiv$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $Attr(V(R))_{\mathsf{SELECT}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(f,t) \subseteq R.\bar{A}$	Condition 3: $Attr(V(R))_{\mathsf{WHERE}}(t,f) = \emptyset$ $Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(t,t) \subseteq R.\bar{A}$ $\forall CV(R.C)\ in\ \mathcal{CV}:$ $(i)R.C\ in\ R.\bar{A}\ or$ $((ii)\exists C_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1)$ $C_R(R.C) \rightarrow CV(R.C)\ and$ $(ii')\exists C'_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1)$ $CV(R.C) \rightarrow C'_R(R.C))$ $\forall C_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1),$ $\exists CV(R.C)\ in\ \mathcal{CV}\ s.t.$ $CV(R.C) \rightarrow C_R(R.C)$
CREATE VIEW $V'\ \ (\mathcal{VE} = \equiv)$ AS SELECT $\underline{S(Attr(V(R))_{\mathsf{SELECT}}(f,t)), S(Attr(V(R))_{\mathsf{SELECT}}(t,t))}, \bar{T}$ FROM $\underline{S}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(\neg(t,f))), \bar{W})}$ $\underline{\text{AND } \mathcal{C}_V(S.\bar{B}_2) \text{ AND } \mathcal{C}_S(S.\bar{B}_1)}$		
Theorem 7: $V' \equiv_\pi V$		

Table 3.9: SPOC Synchronization Strategy for SPJ View V, Eq. (3.24): "*delete-relation R*" (cont.).

$$
\begin{array}{ll}
\textsf{CREATE VIEW} & V' \ \ (\mathcal{VE} = \delta) \ \textsc{as} \\
\textsf{SELECT} & \underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t)), \bar{T}} \\
 & \underline{\mathcal{F}(S(aggAttr(V(R))(f,t)), S(aggAttr(V(R))(t,t)), \bar{F})} \\
\textsf{FROM} & \underline{S(\mathcal{RR} = true), R_1, \ldots, R_n} \\
\textsf{WHERE} & \underline{\mathcal{CV}(S(Attr(V(R))^{\text{JOIN}}_{\textsf{WHERE}}(\neg(t,f))), \bar{W})} \\
 & \underline{\textsc{and}\ \mathcal{C}_V(S.\bar{B}_2)\ \textsc{and}\ \mathcal{C}_S(S.\bar{B}_1)} \\
\textsf{GROUP BY} & \underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t)), \bar{T}}
\end{array}
\tag{3.46}
$$

We can make a similar observation for the WHERE clause. $S(Attr(V(R))^{\text{JOIN}}_{\textsf{WHERE}}(\neg(t,f)))$ are the attributes from $S.\bar{B}$ replacing attributes from $Attr(V(R))^{\text{JOIN}}_{\textsf{WHERE}}$. By Condition 2, all attributes in $Attr(V(R))^{\text{JOIN}}_{\textsf{WHERE}}(f,t)$ have replacements in $S.\bar{B}$, but this is not necesarely the case for all in $Attr(V(R))^{\text{JOIN}}_{\textsf{WHERE}}(t,t)$.

$\mathcal{CV}(S(Attr(V(R))^{\text{JOIN}}_{\textsf{WHERE}}(\neg(t,f))), \bar{W})$ is the conjunction of primitive clauses in the WHERE clause of the view V where attributes of relation R were replaced by the corresponding attributes in $S(Attr(V(R))_{\textsf{WHERE}}(\neg(t,f)))$. If not all the attributes from $Attr(V(R))^{\text{JOIN}}_{\textsf{WHERE}}(t,t)$ have replacements in $S.\bar{B}$ then the primitive clauses using them were dropped together with primitive clauses referring to the attributes in $Attr(V(R))_{\textsf{WHERE}}(t,f)$. Select conditions on R used in the WHERE clause of V (i.e., the ones referring to ttributes in $Attr(V(R))^{\text{LOCAL}}_{\textsf{WHERE}}$) were replaced by local conditions in $\mathcal{C}_V(S.\bar{B}_2)$ or $\mathcal{C}_S(S.\bar{B}_1)$, or were dropped (if dispensable).

Example 19 *We conclude that the relation* ***Accident–Ins*** *has all the properties required by Conditions 1, 2, 3 and 4, and falls into Case 1 in Condition 3:* $\phi = \subseteq$ *(Condition 4) in our* $\mathcal{PC}$*-constraint from Eq. (3.30), and all attributes from* $Attr(V(R))_{\textsf{WHERE}}$*have replacements in* ***Accident–Ins*** *(Figure 3.8).*

Finally, applying the last step of the SPOC Algorithm we obtain the legal view rewriting (cf. Theorem 7) ***Customer–Passengers'*** *defined in Eqs. (3.47) and (3.48) as a relational algebra expression and E-SQL expression, respectively.*

Case 1. $\mathcal{VE} \; = \subseteq$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \subseteq \mid \equiv$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $grAttr(V(R))(f,t) \subseteq R.\bar{A}$ $aggAttr(V(R))(f,t) \subseteq R.\bar{A}$ $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$	Condition 3: $Attr(V(R))_{\text{WHERE}}(t,f) = \emptyset$ $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$ $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(t,t) \subseteq R.\bar{A}$ $\forall CV(R.C)\ in\ \mathcal{CV}:$ $(i)\ R.C\ in\ R.\bar{A}\ or$ $(ii)\ \exists C_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1)$ $C_R(R.C) \rightarrow CV(R.C)$
CREATE VIEW $V' \;\; (\mathcal{VE} = \subseteq)$ AS SELECT $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t))}, \bar{T}$ $\underline{\mathcal{F}(S(aggAttr(V(R))(f,t)), S(aggAttr(V(R))(t,t)), \bar{F})}$ FROM $\underline{S(\mathcal{RR} = true)}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(\neg(t,f))), \bar{W})}$ $\underline{\text{AND } \mathcal{C}_V(S.\bar{B}_2) \text{ AND } \mathcal{C}_S(S.\bar{B}_1)}$ GROUP BY $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t)),}, \bar{T}$		
Theorem 7: $V' \subseteq_{\pi_{gr}} \mid \equiv_{\pi_{gr}} V$		
Case 2. $\mathcal{VE} \; = \supseteq$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \supseteq \mid \equiv$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $grAttr(V(R))(f,t) \subseteq R.\bar{A}$ $aggAttr(V(R))(f,t) \subseteq R.\bar{A}$ $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$	Condition 3: $\forall CV(R.C)\ in\ \mathcal{CV}:$ $(i)\ R.C\ in\ R.\bar{A}\ or$ $(ii)\ \exists C_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1)$ $CV(R.C) \rightarrow C_R(R.C)$ $\forall C_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1),$ $\exists CV(R.C)\ in\ \mathcal{CV}\ s.t.$ $CV(R.C) \rightarrow C_R(R.C)$
CREATE VIEW $V' \;\; (\mathcal{VE} = \supseteq)$ AS SELECT $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t))}, \bar{T}$ $\underline{\mathcal{F}(S(aggAttr(V(R))(f,t)), S(aggAttr(V(R))(t,t)), \bar{F})}$ FROM $\underline{S(\mathcal{RR} = true)}, R_1, \ldots, R_n$ WHERE $\underline{\mathcal{CV}(S(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(\neg(t,f))), \bar{W})}$ $\underline{\text{AND } \mathcal{C}_V(S.\bar{B}_2) \text{ AND } \mathcal{C}_S(S.\bar{B}_1)}$ GROUP BY $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t)),}, \bar{T}$		
Theorem 7: $V' \supseteq_{\pi_{gr}} V$ $\forall t \in V\ \exists t' \in V'\ s.t.\ t' =_{\pi_{gr}} t$ $t'[f(X)] \geq t[f(X)], f \in \{MAX, COUNT(DISTINCT)\},$ $t'[f(X)] \leq t[f(X)], f \in \{MIN\},$ f an aggregate attribute in V and V'		

Table 3.10: SPOC Synchronization Strategy for Aggregation View V, Eq. (3.25): "*delete-relation R*".

<table>
<tr><td colspan="3">Case 3. $\boxed{\mathcal{VE}\ = \equiv}$</td></tr>
<tr>
<td>Condition 1:
$\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B_1})}(S)) \equiv$
$\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A_1})}(R))$</td>
<td>Condition 2:
$grAttr(V(R))(f,t) \subseteq R.\bar{A}$
$aggAttr(V(R))(f,t) \subseteq R.\bar{A}$
$Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$</td>
<td>Condition 3:
$Attr(V(R))_{\text{WHERE}}(t,f) = \emptyset$
$Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(f,t) \subseteq R.\bar{A}$
$Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(t,t) \subseteq R.\bar{A}$
$\forall CV(R.C)\ in\ \mathcal{CV}:$
$(i) R.C\ in\ R.\bar{A}\ or$
$((ii)\exists C_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1)$
$C_R(R.C) \rightarrow CV(R.C)\ and$
$(ii')\exists C'_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1)$
$CV(R.C) \rightarrow C'_R(R.C))$
$\forall C_R(R.C)\ in\ \mathcal{C}_R(R.\bar{A}_1),$
$\exists CV(R.C)\ in\ \mathcal{CV}\ s.t.$
$CV(R.C) \rightarrow C_R(R.C)$</td>
</tr>
<tr><td colspan="3">CREATE VIEW $V'\ \ (\mathcal{VE} = \equiv)$ AS
SELECT $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t)), \bar{T}}$
$\underline{\mathcal{F}(S(aggAttr(V(R))(f,t)), S(aggAttr(V(R))(t,t)), \bar{F})}$
FROM $\underline{S(\mathcal{RR} = true)}, R_1, \ldots, R_n$
WHERE $\underline{\mathcal{CV}(S(Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}(\neg(t,f))), \bar{W})}$
$\underline{\text{AND } \mathcal{C}_V(S.\bar{B}_2) \text{ AND } \mathcal{C}_S(S.\bar{B}_1)}$
GROUP BY $\underline{S(grAttr(V(R))(f,t)), S(grAttr(V(R))(t,t)), \bar{T}}$</td></tr>
<tr><td colspan="3">Theorem 7: $V' \equiv_{\pi_{gr}} V$
$\forall t \in V\ \exists t' \in V'\ s.t.\ t' =_{\pi_{gr}} t$
$t'[f(X)] \geq t[f(X)], f \in \{MAX, COUNT(DISTINCT)\},$
$t'[f(X)] \leq t[f(X)], f \in \{MIN\},$
$\forall t' \in V'\ \exists t \in V\ s.t.\ t =_{\pi_{gr}} t'$
$t[f(X)] \leq t'[f(X)], f \in \{MAX, COUNT(DISTINCT)\},$
$t[f(X)] \geq t'[f(X)], f \in \{MIN\},$
f an aggregate attribute in V and V' f an aggregate attribute in V and V'</td></tr>
</table>

Table 3.11: SPOC Synchronization Strategy for Aggregation View V, Eq. (3.25): "*delete-relation* R" (cont.).

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** $\}$
$S(Attr(V(R))_{SELECT}(t,t))$	$\{$**Accident−Ins.Age**$\}$
$Attr(V(R))^{JOIN}_{WHERE}(f,t)$	$\{$**Customer.Name**$\}$
$S(Attr(V(R))^{JOIN}_{WHERE}(f,t))$	**Accident−Ins.Holder**
$Attr(V(R))^{LOCAL}_{WHERE}(f,t)$	$\{$**Customer.Age**$\}$
$V(R)$	$\pi_{Customer.Name, Customer.Age}(\sigma_{(Customer.Age<20)}(\textbf{Customer}))$
$\mathcal{C}_{Rest}$	$(Customer.Name = FlightRes.PName)\ AND (FlightRes.Dest = \text{'Asia'})$
$\mathcal{C}_V(S.B_2)$	$(Accident{-}Ins.Age < 20)$
$\mathcal{C}_S(S.B_1)$ AND $(Accident{-}Ins.Age < 50)$	$(Accident{-}Ins.Amount > 1,000,000)$

*Figure 3.8: Replacements for the elements of $V(R)$ = **Customer-Passengers(Customer)** found in the relation S = **Accident-Ins**.*

$$\begin{aligned}&\textbf{Customer-Passengers'} = \\ &\underbrace{\pi_{Accident-Ins.Holder,\ Accident-Ins.Age}(\sigma_{Accident-Ins.Age<20}\ (\pi_{Accident-Ins.Holder, Accident-Ins.Age}\ (\sigma_{(Accident-Ins.Amount>1,000,000)\ AND\ (Accident-Ins.Age<50)}\textbf{Accident−Ins}))}_{replaces\ V(R)} \\ &\bowtie_{(Accident-Ins.Holder=\ FlightRes.PName)\ (AND\ FlightRes.Dest='Asia')}\ (\textbf{FlightRes}))\end{aligned} \tag{3.47}$$

$$\begin{array}{ll}
\text{CREATE VIEW} & \textbf{Customer-Passengers'}\ (\mathcal{VE} = \subseteq)\ \text{AS} \\
\text{SELECT} & \underline{\textbf{A.Holder}\ (\mathcal{AD} = false, \mathcal{AR} = true),\ \textbf{A.Age}\ (\mathcal{AD} = true, \mathcal{AR} = true)} \\
\text{FROM} & \underline{\textbf{Accident−Ins A}\ (\mathcal{RD} = true, \mathcal{RR} = true)}, \\
 & \underline{\textbf{FlightRes F}\ (\mathcal{RD} = true, \mathcal{RR} = true)}, \\
\text{WHERE} & \underline{(\textbf{A.Holder} = \textbf{F.PName})\ (\mathcal{CD} = false, \mathcal{CR} = true)}_{AND} \\
 & \underline{(\textbf{F.Dest} = \textbf{'Asia'})}_{AND} \\
 & \underline{(\textbf{A.Age} < 20)(\mathcal{CD} = false, \mathcal{CR} = true)}_{AND} \\
 & \underline{(\textbf{A.Amount} > 1,000,000)(\mathcal{CD} = false, \mathcal{CR} = true)}_{AND} \\
 & \underline{(\textbf{A.Age} < 50)(\mathcal{CD} = false, \mathcal{CR} = true)}
\end{array} \tag{3.48}$$

The above four conditions are *sufficient* (but not always *necessary*) to obtain a legal rewriting by using a select-project substitution of $IS_2.S$ to replace relation

$IS_1.R$. The following theorem states the sufficiency of Conditions 1, 2, 3 and 4 for the SPOC algorithm to obtain a legal base rewriting for a view V.

Theorem 7 *Let an E-SQL view V (Eqs. (3.24) and (3.25)) be evolvable (Theorem 1) under the capability change "delete-relation R", with R a replaceable relation in the* FROM *clause. Then the rewriting V' obtained by applying the SPOC Algorithm (Eqs. (3.45) and (3.46)) is a legal rewriting under capability change "delete-relation R".*

For aggregation view V' defined in Equation (3.46), the aggregation attributes using duplicate-insensitive aggregation functions (e.g., MAX, MIN, COUNT(DISTINCT)) have the properties from Tables (3.10) and (3.11).

Proof. See Appendix B.1.

3.5.5 Conditions for Replacement Used by the SPOC Algorithm: "*delete-attribute*" Capability Change

When the capability change is "*delete-attribute R.X*", we assume the view V is defined as in Equation (3.49) for SPJ views and as in Equation (3.50) for AGGR-SPJ views. The to-be-deleted attribute $R.X$ is used in the SELECT and/or WHERE clauses. The view V must be evolvable (Theorem 1) under this capability change and at least one component referring to the attribute $R.X$ is replaceable in V.

$$\begin{array}{ll} \text{CREATE VIEW} & V \quad (\mathcal{VE} = \delta) \text{ AS} \\ \text{SELECT} & Attr(V(R))_{\text{SELECT}}, \bar{T} \\ \text{FROM} & R, R_1, \ldots, R_n \\ \text{WHERE} & \mathcal{CV}(Attr(V(R))_{\text{WHERE}}, \bar{W}) \end{array} \tag{3.49}$$

$$
\begin{array}{ll}
\textsf{CREATE VIEW} & V \quad (\mathcal{VE} = \delta) \ \textsf{AS} \\
\textsf{SELECT} & grAttr(V(R)), \bar{T}, \\
 & \mathcal{F}(aggAttr(V(R)), \bar{F}) \\
\textsf{FROM} & R, R_1, \ldots, R_n \\
\textsf{WHERE} & \mathcal{CV}(Attr(V(R))_{\textsf{WHERE}}, \bar{W}) \\
\textsf{GROUP BY} & grAttr(V(R)), \bar{T}
\end{array}
\tag{3.50}
$$

In the following we give the extra properties the relation S must have to be a good replacement when the capability change "*delete-attribute* $R.X$" is considered. The Condition 1 given in Section 3.5.2 must still hold for this capability change. Hence, we assume that a candidate containment constraint as defined in Equation (3.51) between the relation R and a replacement relation S exists in the MKB:

$$
\mathcal{PC}_{R,S} = (\underbrace{\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S))}_{\mathcal{PC}(S)} \ \phi \ \underbrace{\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))}_{\mathcal{PC}(R)}) \tag{3.51}
$$

We make a distinction between two types of the candidate containment constraints based on the relationship of the to-be-deleted attribute $R.X$ with the $\mathcal{PC}$-constraint. Namely, Type (I) corresponds to the class of $\mathcal{PC}$-constraints for which the attribute $R.X$ is *not* among the attributes of the relation R that has replacements in S, i.e., $R.X \notin R.\bar{A}$. A Type (II) $\mathcal{PC}$-constraint has the property that the attribute $R.X$ has a replacement in S, i.e., $R.X \in R.\bar{A}$.

If the candidate $\mathcal{PC}$-constraint is of type (I), the view definition can be synchronized only if the attribute $R.X$ appears in dispensable components or it appears in local WHERE conditions of the view V that are implied by the select conditions found in the $\mathcal{PC}$-constraint. For the case when the candidate $\mathcal{PC}$-constraint is of type (II) however, the attribute $R.X$ has an attribute replacement $S.B$ in the relation S and the view is synchronized by simply replacing all the instances of the attribute $R.X$ with this replacement. We formally define below the sufficient conditions the view definition V (Equations (3.49) and (3.50)) and the $\mathcal{PC}$-constraint (Equation (3.51)) must satisfied for each of the types (I) and (II) such that a legal rewriting can be found for the view V. The Tables 3.12, 3.13, 3.14, 3.15 and 3.16, 3.17, 3.18, 3.19 summarize the conditions and rewriting strategies for the SPJ and AGGR-SPJ views, respectively.

Case 1. $\mathcal{VE} = \subseteq$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \subseteq$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $(D1) if\ R.X \in Attr(V(R))_{\mathsf{SELECT}}$ $then\ R.X \in Attr(V(R))_{\mathsf{SELECT}}(t, \_)$, $R.X \notin Attr(V(R))_{\mathsf{SELECT}}(f, \_)$ $(D2) if\ R.X \in Attr(V(R))_{\mathsf{WHERE}}$ $then\ R.X \in (Attr(V(R))^{\mathrm{LOCAL}}_{\mathsf{WHERE}}(\_, t)$ $\cup\ Attr(V(R))_{\mathsf{WHERE}}(t, \_))$, $R.X \notin Attr(V(R))_{\mathsf{WHERE}}(f, f)$	Condition 3: $(E0) Attr(V(R))_{\mathsf{SELECT}} \setminus R.\bar{A}$ $\subseteq Attr(V(R))_{\mathsf{SELECT}}(t, \_)$ $(E1) Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.X\}$ $\subseteq R.\bar{A}$, $R.X \notin (Attr(V(R))_{\mathsf{WHERE}}$ $\setminus Attr(V(R))^{\mathrm{LOCAL}}_{\mathsf{WHERE}}(\_, t))$ $(E2) \forall CV(R.X)\ in\ \mathcal{CV}$ $\exists C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)\ s.t.$ $C_R(R.X) \rightarrow CV(R.X)$
Eq. (3.66):	CREATE VIEW V' $(\mathcal{VE} = \subseteq)$ AS SELECT $\underline{(Attr(V(R))_{\mathsf{SELECT}} \cap R.\bar{A})}, \bar{T}$ FROM $\underline{S, R'}, R_1, \ldots, R_n$ WHERE $\underline{(R'.\bar{A} \cap Attr(V(R))) = (S.\bar{B} \cap S(Attr(V(R))))}$, $\underline{\mathcal{C}_S(S.\bar{B}_1)}$, $\underline{\mathcal{CV}(Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.X\}, \bar{W})}$	
Theorem 8: $V' \subseteq_\pi \mid \equiv_\pi V$		
Case 2. $\mathcal{VE} = \supseteq$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \supseteq$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $(D1) if\ R.X \in Attr(V(R))_{\mathsf{SELECT}}$ $then\ R.X \in Attr(V(R))_{\mathsf{SELECT}}(t, \_)$, $R.X \notin Attr(V(R))_{\mathsf{SELECT}}(f, \_)$ $(D2) if\ R.X \in Attr(V(R))_{\mathsf{WHERE}}$ $then\ R.X \in (Attr(V(R))^{\mathrm{LOCAL}}_{\mathsf{WHERE}}(\_, t)$ $\cup\ Attr(V(R))_{\mathsf{WHERE}}(t, \_))$, $R.X \notin Attr(V(R))_{\mathsf{WHERE}}(f, f)$	Condition 3: $(E3) \forall C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)$, $\exists CV(R.X)\ in\ \mathcal{CV}\ s.t.$ $CV(R.X) \rightarrow C_R(R.X)$ $(E4) \forall$ indisp. $CV(R.X)\ in\ \mathcal{CV}$ $\exists C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)\ s.t.$ $CV(R.X) \rightarrow C_R(R.X)$
Eq. (3.67):	CREATE VIEW V' $(\mathcal{VE} = \supseteq)$ AS SELECT $\underline{(Attr(V(R))_{\mathsf{SELECT}} \setminus \{R.X\})}, \bar{T}$ FROM $\underline{S, R'}, R_1, \ldots, R_n$ WHERE $\underline{(R'.\bar{A} \cap Attr(V(R))) = (S.\bar{B} \cap S(Attr(V(R))))}$, $\underline{\mathcal{C}_S(S.\bar{B}_1)}$, $\underline{\mathcal{CV}'(Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.X\}, \bar{W})}$	
Theorem 8: $V' \supseteq_\pi V$		

Table 3.12: SPOC Synchronization Strategy for SPJ View V, Eq. (3.49), "*delete-attribute* $R.X$" and Type (I) $\mathcal{PC}$-constraint ($R.X \notin R.\bar{A}$).

Case 3. $\mathcal{VE} = \equiv$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \equiv$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $(D1) if\ R.X \in Attr(V(R))_{\mathsf{SELECT}}$ $then\ R.X \in Attr(V(R))_{\mathsf{SELECT}}(t,\_)$, $R.X \notin Attr(V(R))_{\mathsf{SELECT}}(f,\_)$ $(D2) if\ R.X \in Attr(V(R))_{\mathsf{WHERE}}$ $then\ R.X \in (Attr(V(R))^{\mathrm{LOCAL}}_{\mathsf{WHERE}}(\_,t)$ $\cup\ Attr(V(R))_{\mathsf{WHERE}}(t,\_))$, $R.X \notin Attr(V(R))_{\mathsf{WHERE}}(f,f)$	Condition 3: $(E0) Attr(V(R))_{\mathsf{SELECT}} \setminus R.\bar{A}$ $\subseteq Attr(V(R))_{\mathsf{SELECT}}(t,\_)$ $(E1) Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.X\}$ $\subseteq R.\bar{A}$, $R.X \notin (Attr(V(R))_{\mathsf{WHERE}}$ $\setminus Attr(V(R))^{\mathrm{LOCAL}}_{\mathsf{WHERE}}(\_,t))$ $(E2) \forall CV(R.X)\ in\ \mathcal{CV}$ $\exists C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)\ s.t.$ $C_R(R.X) \rightarrow CV(R.X)$ $(E3) \forall C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)$, $\exists CV(R.X)\ in\ \mathcal{CV}\ s.t.$ $CV(R.X) \rightarrow C_R(R.X)$ $(E4) \forall$ indisp. $CV(R.X)\ in\ \mathcal{CV}$ $\exists C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)\ s.t.$ $CV(R.X) \rightarrow C_R(R.X)$
Eq. (3.66):	CREATE VIEW SELECT FROM WHERE	V' $(\mathcal{VE} = \equiv)$ AS $\underline{(Attr(V(R))_{\mathsf{SELECT}} \cap R.\bar{A})}, \bar{T}$ $\underline{S, R'}, R_1, \ldots, R_n$ $\underline{(R'.\bar{A} \cap Attr(V(R))) = (S.\bar{B} \cap S(Attr(V(R))))}$, $\underline{\mathcal{C}_S(S.\bar{B}_1)}$, $\underline{\mathcal{CV}(Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.X\}, \bar{W})}$
Theorem 8: $V' \equiv_\pi V$		

Table 3.13: SPOC Synchronization Strategy for SPJ View V, Eq. (3.49), "*delete-attribute* $R.X$" and Type (I) $\mathcal{PC}$-constraint ($R.X \notin R.\bar{A}$) (cont.).

Condition 2: Deleted Attribute Replacement. The attribute $R.X$ has a replacement in the relation S only for containment constraints of type (II). Each view component using the attribute $R.X$ is synchronized by replacing this attribute with its replacement $S(R.X)$. It is legal to do so as long as these view components are replaceable. For a non-replaceable component, we have to drop it from the view definition. This condition is formalized in Equation 3.52.

Using a $\mathcal{PC}$-constraint of type (I) it is still possible to evolve the view *without* replacing the attribute $R.X$ but rather replacing the local conditions referring to it. Eqs. (3.52) and 3.54 are formally expressing these requirements.

$$\begin{aligned}
&(I)\,(D1) \quad && if\ R.X \in Attr(V(R))_{\mathsf{SELECT}}\ then && (3.52)\\
& && R.X \in Attr(V(R))_{\mathsf{SELECT}}(t,_),\ and && \\
& && R.X \notin Attr(V(R))_{\mathsf{SELECT}}(f,_) && \\
&(I)\,(D2) \quad && if\ R.X \in Attr(V(R))_{\mathsf{WHERE}}\ then && (3.53)
\end{aligned}$$

<table>
<tr><td colspan="3">Case 1. $\mathcal{VE} = \subseteq$</td></tr>
<tr><td>Condition 1:
$\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \subseteq \mid \equiv$
$\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$</td><td>Condition 2:
$(D3) R.X \notin Attr(V(R))(f,f)$</td><td>Condition 3:
$(E0) Attr(V(R))_{\text{SELECT}} \setminus R.\bar{A}$
$\subseteq Attr(V(R))_{\text{SELECT}}(t, \_)$
$(E5) Attr(V(R))_{\text{WHERE}} \subseteq R.\bar{A}$,
$R.X \notin Attr(V(R))_{\text{WHERE}}(\_, f)$</td></tr>
<tr><td>Eq. (3.68):</td><td colspan="2">CREATE VIEW V' $(\mathcal{VE} = \subseteq)$ AS
SELECT $\underline{(Attr(V(R))_{\text{SELECT}} \cap R.\bar{A}) \setminus \{R.X\}}$,
$\underline{\{S(R.X) \mid if\ R.X \text{ is replaceable}\}}$,
$\underline{T}$
FROM $\underline{S}, R', R_1, \ldots, R_n$
WHERE $\underline{((R'.\bar{A} \cap Attr(V(R))) \setminus \{R.X\}) = ((S.\bar{B} \cap S(Attr(V(R)))) \setminus \{S(R.X)\})}$,
$\underline{\mathcal{C}_S(S.\bar{B}_1)}$,
$\underline{\mathcal{CV}(S(R.X), Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}, \bar{W})}$</td></tr>
<tr><td colspan="3">Theorem 8: $V' \subseteq_\pi \mid \equiv_\pi V$</td></tr>
<tr><td colspan="3">Case 2. $\mathcal{VE} = \supseteq$</td></tr>
<tr><td>Condition 1:
$\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \supseteq \mid \equiv$
$\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$</td><td>Condition 2:
$(D3) R.X \notin Attr(V(R))(f,f)$</td><td>Condition 3:
$(E3) \forall C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)$,
$\exists CV(R.X)\ in\ \mathcal{CV}\ s.t.$
$CV(R.X) \rightarrow C_R(R.X)$</td></tr>
<tr><td>Eq. (3.69):</td><td colspan="2">CREATE VIEW V' $(\mathcal{VE} = \supseteq)$ AS
SELECT $\underline{Attr(V(R))_{\text{SELECT}} \setminus \{R.X\}}$,
$\underline{\{S(R.X) \mid if\ R.X \text{ is replaceable}\}}$,
$\underline{T}$
FROM $\underline{S}, R', R_1, \ldots, R_n$
WHERE $\underline{((R'.\bar{A} \cap Attr(V(R))) \setminus \{R.X\}) = ((S.\bar{B} \cap S(Attr(V(R)))) \setminus \{S(R.X)\})}$,
$\underline{\mathcal{C}_S(S.\bar{B}_1)}$,
$\underline{\mathcal{CV}'(S(R.X), Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}, \bar{W})}$</td></tr>
<tr><td colspan="3">Theorem 8: $V' \supseteq_\pi V$</td></tr>
</table>

Table 3.14: SPOC Synchronization Strategy for SPJ View V, Eq. (3.50), "*delete-attribute* $R.X$" and Type (II) $\mathcal{PC}$-constraint ($R.X \in R.\bar{A}$).

Case 3. $\boxed{\mathcal{VE}\ = \equiv}$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \equiv \pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $(D3) R.X \notin Attr(V(R))(f,f)$	Condition 3: $(E0) Attr(V(R))_{\text{SELECT}} \setminus R.\bar{A} \subseteq Attr(V(R))_{\text{SELECT}}(t,\_)$ $(E5) Attr(V(R))_{\text{WHERE}} \subseteq R.\bar{A}$, $R.X \notin Attr(V(R))_{\text{WHERE}}(\_,f)$ $(E3) \forall C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)$, $\exists CV(R.X)\ in\ \mathcal{CV}\ s.t.$ $CV(R.X) \rightarrow C_R(R.X)$
Eq. (3.68):	CREATE VIEW $V'\ (\mathcal{VE} = \subseteq)$ AS SELECT $\underline{(Attr(V(R))_{\text{SELECT}} \cap R.\bar{A}) \setminus \{R.X\}}$, $\underline{\{S(R.X) \mid if\ R.X \text{ is replaceable}\}}$, $\underline{\bar{T}}$ FROM $\underline{S, R', R_1, \ldots, R_n}$ WHERE $\underline{((R'.\bar{A} \cap Attr(V(R))) \setminus \{R.X\}) = ((S.\bar{B} \cap S(Attr(V(R)))) \setminus \{S(R.X)\})}$, $\underline{\mathcal{C}_S(S.\bar{B}_1)}$, $\underline{\mathcal{CV}(S(R.X), Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}, \bar{W})}$	
Theorem 8: $V' \equiv_\pi V$		

Table 3.15: SPOC Synchronization Strategy for SPJ View V, Eq. (3.49), "*delete-attribute* $R.X$" and Type (II) $\mathcal{PC}$-constraint ($R.X \in R.\bar{A}$) (cont.).

$$\begin{array}{lll} & R.X \in & (Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}(t,\_)),\ and \\ & R.X \notin & Attr(V(R))_{\text{WHERE}}(f,f) \\ (II)\ (D3) & R.X \notin & Attr(V(R))(f,f) \end{array} \quad (3.54)$$

Example 20 *Let the relations* ***Person(ID, Name, Address, Age, Status)*** *and* ***Student(Name, Address, Age)*** *be used in the view* ***Students*** *defined in Eq. (3.55).*

$$\begin{array}{ll} \textit{CREATE VIEW} & \textbf{\textit{Students}}\ \textit{AS} \\ \textit{SELECT} & \textbf{\textit{P.Name, P.Address, P.ID}} \\ \textit{FROM} & \textbf{\textit{Person P}} \\ \textit{WHERE} & \textbf{\textit{(P.Status = 'student')}}(\mathcal{CD} = false, \mathcal{CR} = true)_{\textit{AND}}\textbf{\textit{(P.Age}} > 35\textbf{\textit{)}} \end{array} \quad (3.55)$$

The view ***Students*** *must be synchronized after the capability change "delete-attribute* ***Person.Status****". The candidate $\mathcal{PC}$-constraint given in Eq. (3.56) belongs to the Type (I) class of containment constraints. That is, the attribute* ***Person.Status*** *doesn't have a replacement in the relation* ***Student****. Moreover, Condition 2 (Eq. (3.52)) is* *satisfied: (D1) is satisfied because* $\textbf{\textit{Person.Status}} \notin Attr(V(R))_{\textsf{SELECT}}$*; and, (D2) is satisfied because* $\textbf{\textit{Person.Status}} \in Attr(V(R))^{LOCAL}_{\textsf{WHERE}}(f,t)$.

$$\mathcal{PC}_{Student,Person} = \tag{3.56}$$
$$\pi_{Name,\ Address}(\sigma_{(Age\ >\ 30)}\textbf{\textit{Student}})) \supseteq \pi_{Name,\ Address}(\sigma_{(Status\ =\ \text{``student''})AND(Age\ >\ 30)}\textbf{\textit{Person}}$$

Condition 3: View-Extent Satisfaction. As in the case of the "*delete-relation* " capability change, here we give properties that must hold for the $\mathcal{PC}$-constraint defined by Eq. (3.27) in order for the view-extent evolution parameter $\mathcal{VE}$ to be satisfied.

<u>Case 1. $\mathcal{VE} = \subseteq$.</u> For both types (I) and (II) of $\mathcal{PC}$-constraints, all conditions affected by the "*delete-attribute* $R.X$" capability change in the WHERE clause of V must be preserved. Thus all of them must be replaceable which is expressed by the conditions (E1) and (E5) in Eqs. (3.59) and (3.61) below. Moreover, to prove $V' \subseteq_{\pi} V$, all attributes of R used in the WHERE clause must be in $R.\bar{A}$ as given in Eqs. (3.58) and (3.60), while all attributes of R in the SELECT clause not in $R.\bar{A}$ are to be dropped thus they must be dispensable, ((E0) in Eq. (3.57)).

$$\begin{array}{rll}
(I)\&(II)\,(E0) & Attr(V(R))_{\textsf{SELECT}} \setminus R.\bar{A} \subseteq Attr(V(R))_{\textsf{SELECT}}(t,_) & (3.57)\\
(I)\,(E1) & Attr(V(R))_{\textsf{WHERE}} \setminus \{R.X\} \subseteq R.\bar{A},\ and & (3.58)\\
 & R.X \notin (Attr(V(R))_{\textsf{WHERE}} \setminus Attr(V(R))^{\text{LOCAL}}_{\textsf{WHERE}}(_,t)) & (3.59)\\
(II)\,(E5) & Attr(V(R))_{\textsf{WHERE}} \subseteq R.\bar{A}\ and & (3.60)\\
 & R.X \notin Attr(V(R))_{\textsf{WHERE}}(_,f) & (3.61)
\end{array}$$

For the type (II) $\mathcal{PC}$-constraints, the affected conditions will have the attribute $R.X$ replaced by $S.B = S(R.X)$ in any legal rewriting. For the type (I), we replace them by the conditions in $\mathcal{C}_S(S.\bar{B}_1)$ of the $\mathcal{PC}$-constraint from Eq. (3.27). This is possible only if conditions referring to $R.X$ in V are implied by conditions in $\mathcal{C}_R(R.\bar{A}_1)$. This fact is expressed in Eq. (3.62).

$$(I)\ (E2) \quad \begin{array}{l} \forall CV(R.X)\ in\ \mathcal{CV} \\ \exists C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)\ s.t.\ C_R(R.X) \rightarrow CV(R.X) \end{array} \tag{3.62}$$

Case 2. $\mathcal{VE} = \supseteq$. In this case, the view can be evolved by dropping conditions from the WHERE clause and still have the view extent evolution parameter $\mathcal{VE} = \supseteq$ satisfied. For type (II), we can drop nonreplaceable conditions referring to the attribute $R.X$, and replace the replaceable ones using the attribute $S(R.X)$. For type (I), we can drop nonreplaceable conditions referring to $R.X$ as well, and replace the replaceable conditions with the expression $\mathcal{C}_S(S.\bar{B}_1)$. Thus in both cases, we have to impose that any indispensable condition must be implied by a condition in $\mathcal{C}_R(R.\bar{A}_1)$, as in (E4) of Eq. (3.63).

$$(I)\&(II)\ (E4) \quad \begin{array}{l} \forall \text{ indispensable } CV(R.X)\ in\ \mathcal{CV} \\ \exists C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)\ s.t.\ CV(R.X) \rightarrow C_R(R.X) \end{array} \tag{3.63}$$

For both types (I) and (II), we have to make sure that the conjunction of conditions $\mathcal{C}_R(R.\bar{A}_1)$ (Eq. (3.27)) isn't more restrictive than the local conditions on R in V. Thus we impose that any condition in $\mathcal{C}_R(R.\bar{A}_1)$ must be implied by some condition in $\mathcal{CV}(Attr(V(R))_{\text{WHERE}}, \bar{W})$:

$$(I)\&(II)\ (E3) \quad \begin{array}{l} \forall C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1), \\ \exists CV(R.X)\ in\ \mathcal{CV}()\ s.t.\ CV(R.X) \rightarrow C_R(R.X) \end{array} \tag{3.64}$$

Case 3. $\mathcal{VE} = \equiv$. All conditions imposed above in Case 1 and Case 2 must be satisfied in order to have the view extent evolution parameter $\mathcal{VE} = \equiv$ satisfied.

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. (3.27)) must satisfy the property from Eq. (3.65).

$$\begin{array}{llll} \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \equiv, & \text{then } \phi \text{ must be } \equiv; \\ \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \subseteq, & \text{then } \phi \text{ must be } \subseteq \text{ or } \equiv; \text{ and} \\ \text{if } \textit{view-extent} \text{ parameter } \mathcal{VE} & = & \supseteq, & \text{then } \phi \text{ must be } \supseteq \text{ or } \equiv. \end{array} \tag{3.65}$$

3.5.6 The SPOC Algorithm for the "*delete-attribute*" Capability Change

Below we give the query rewriting for the "*delete-attribute R.X*" capability change. Note that for type (I) $\mathcal{PC}$-constraints, $S(R.X)$ is empty set indicating that the attribute $R.X$ doesn't have a replacement in S.

$$\begin{array}{ll} \text{CREATE VIEW} & V' \ \ (\mathcal{VE} = \subseteq \mid \equiv) \text{ AS} \\ \text{SELECT} & \underline{(Attr(V(R))_{\text{SELECT}} \ \cap \ R.\bar{A}) \setminus \{R.X\}}, \bar{T} \\ \text{FROM} & \underline{S, R'}, R_1, \ldots, R_n \\ \text{WHERE} & \underline{(R'.\bar{A} \cap Attr(V(R))) = (S.\bar{B} \cap S(Attr(V(R))))} \text{ AND} \\ & \underline{\mathcal{C}_S(S.\bar{B}_1)} \text{ AND} \\ & \underline{\mathcal{CV}(Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}, \bar{W})} \end{array} \tag{3.66}$$

$$\begin{array}{ll} \text{CREATE VIEW} & V' \ \ (\mathcal{VE} = \supseteq) \text{ AS} \\ \text{SELECT} & \underline{(Attr(V(R))_{\text{SELECT}} \ \setminus \ \{R.X\}}, \bar{T} \\ \text{FROM} & \underline{S, R'}, R_1, \ldots, R_n \\ \text{WHERE} & \underline{(R'.\bar{A} \cap Attr(V(R))) = (S.\bar{B} \cap S(Attr(V(R))))} \text{ AND} \\ & \underline{\mathcal{C}_S(S.\bar{B}_1)} \text{ AND} \\ & \underline{\mathcal{CV}(Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}, \bar{W})} \end{array} \tag{3.67}$$

Theorem 8 *Let an E-SQL view V be defined as in Equations (3.49) for* SPJ *type and (3.50) for aggregation type views. V is evolvable (Theorem 1) under capability change* *"delete-attribute R.X", and the attribute R.X is replaceable in at least one of the view components. Then the rewriting V′ (see Table 3.12, 3.13, 3.14, 3.15 and 3.16, 3.17, 3.18, 3.19) obtained by applying the SPOC algorithm is a* ***legal rewriting*** *(Definition 1) under capability change "delete-attribute R.X".*

Proof. See Appendix B.2.

$$
\begin{array}{ll}
\text{CREATE VIEW} & V' \ \ (\mathcal{VE} = \subseteq \mid \equiv) \ \text{AS} \\
\text{SELECT} & \underline{(Attr(V(R))_{\text{SELECT}} \ \cap \ R.\bar{A}) \setminus \{R.X\},} \\
 & \underline{\{S(R.X) \mid if \ R.X \text{ is replaceable}\}}, \bar{T} \\
\text{FROM} & \underline{S, R'}, R_1, \ldots, R_n \\
\text{WHERE} & \underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))} \ \text{AND} \\
 & \underline{\mathcal{C}_S(S.\bar{B}_1)} \ \text{AND} \\
 & \underline{\mathcal{CV}(S(R.X), Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}, \bar{W})}
\end{array}
\tag{3.68}
$$

$$
\begin{array}{ll}
\text{CREATE VIEW} & V' \ \ (\mathcal{VE} = \supseteq) \ \text{AS} \\
\text{SELECT} & \underline{(Attr(V(R))_{\text{SELECT}} \ \setminus \ \{R.X\}),} \\
 & \underline{\{S(R.X) \mid if \ R.X \text{ is replaceable}\}}, \bar{T} \\
\text{FROM} & \underline{S, R'}, R_1, \ldots, R_n \\
\text{WHERE} & \underline{(R'.\bar{A}' \cap Attr(V(R))) = (S.\bar{B}' \cap S(Attr(V(R))))} \ \text{AND} \\
 & \underline{\mathcal{C}_S(S.\bar{B}_1)} \ \text{AND} \\
 & \underline{\mathcal{CV}(S(R.X), Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}, \bar{W})}
\end{array}
\tag{3.69}
$$

Example 21 *For our Example 20, the rewritten view* ***Students'*** *is given in Equation (3.70). Note that a new condition has been added to the* WHERE *clause: namely,* ***((P.Address=St.Address)*** AND ***(P.Name=St.Name))*** *and the replaceable condition* ***((P.Status = 'student')*** *) from Equation (3.55), was replaced by the condition $\mathcal{C}_S(S.\bar{B}_1)$. In other words, we applied Theorem 8 for the view* ***Students*** *and capability change "delete-attribute* ***Person.Status****". We can prove that* ***Students'*** $\supseteq$ ***Students*** *and that all evolution parameters are satisfied by this rewriting.*

$$
\begin{array}{ll}
\textit{CREATE VIEW} & \textbf{\textit{Students'}}\ \textit{AS} \\
\textit{SELECT} & \textbf{\textit{P.Name, P.Address, P.Id}} \\
\textit{FROM} & \textbf{\textit{Person P, Student St}} \\
\textit{WHERE} & \underbrace{\textbf{\textit{(P.Address=St.Address)}}\textit{AND}\textbf{\textit{(P.Name=St.Name)}}}_{R.\bar{A}=S.\bar{B}'}\ \textit{AND} \\
& \underbrace{\textbf{\textit{(St.Age}} > 30\textit{)}(\mathcal{CD} = false, \mathcal{CR} = true)}_{\mathcal{C}_S(S.\bar{B}_1)}\ \textit{AND} \\
& \textbf{\textit{(P.Age}} > 35\textit{)}
\end{array}
\tag{3.70}
$$

3.6 *EVE* System Prototype

A prototype of the *EVE* system has been implemented at Worcester Polytechnic Institute in collaboration with the Database Systems Research Group from there [10]. The system is implemented in Java 1.1 using Java Foundation Classes and has a client-server architecture. The server connects to several SQL-Databases through JDBC and contains an RMI Interface to connect to the Meta Knowledge and Schema Change Request applets. The GUI applets can run remotely and communicate with the server through RMI remote procedure calls. The server contains modules such as Meta Knowledge Base, View Knowledge Base, Incremental Maintenance, database interfaces, E-SQL parser, RMI interface and data warehouse functionality. The whole system currently consists of over 120 Java classes and runs under Windows NT 4.0 and JDK 1.1.6. See [10] for an online demonstration of the system. The *EVE* system has been demonstrated at the IBM-CASCON'97 Technology Showcase, International Conference of Extended Database Technology'98 [59] and SIGMOD'99 [4].

While URL = `http://davis.wpi.edu/dsrg/EVE` provides for a complete demo and tutorial of the system, we now discuss few sample screen dumps. Figure 3.9 shows a screendump of the VKB applet displaying a view definition **PotentialCustomer**

Case 1. $\boxed{\mathcal{VE} \ = \subseteq}$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \subseteq$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $(D1) if\ R.X \ \in \ Attr(V(R))_{\text{SELECT}}$ $then\ R.X \ \in \ Attr(V(R))_{\text{SELECT}}(t, \_),$ $R.X \ \notin \ Attr(V(R))_{\text{SELECT}}(f, \_)$ $(D2) if\ R.X \ \in \ Attr(V(R))_{\text{WHERE}}$ $then\ R.X \ \in \ (Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\_, t)$ $\cup\ Attr(V(R))_{\text{WHERE}}(t, \_)),$ $R.X \ \notin \ Attr(V(R))_{\text{WHERE}}(f, f)$	Condition 3: $(E0) Attr(V(R))_{\text{SELECT}} \ \setminus \ R.\bar{A}$ $\subseteq \ Attr(V(R))_{\text{SELECT}}(t, \_)$ $(E1) Attr(V(R))_{\text{WHERE}} \ \setminus \ \{R.X\}$ $\subseteq \ R.\bar{A},$ $R.X \ \notin (Attr(V(R))_{\text{WHERE}}$ $\setminus Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\_, t))$ $(E2) \forall CV(R.X)\ in\ \mathcal{CV}$ $\exists C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)\ s.t.$ $C_R(R.X) \rightarrow CV(R.X)$

CREATE VIEW	$V' \ \ (\mathcal{VE} = \subseteq)$ AS
SELECT	$\underline{(grAttr(V(R)) \ \cap \ R.\bar{A}), \bar{T}}$
	$\underline{\mathcal{F}(S(R.X), (aggAttr(V(R)) \setminus \{R.X\}), \bar{F})}$
FROM	$\underline{S, R', R_1, \ldots, R_n}$
WHERE	$\underline{(R'.\bar{A} \cap Attr(V(R))) = (S.\bar{B} \cap S(Attr(V(R)))),}$
	$\underline{\mathcal{C}_S(S.\bar{B}_1),}$
	$\underline{\mathcal{CV}(S(R.X), (Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}), \bar{W})}$
GROUP BY	$\underline{(grAttr(V(R)) \cap R.\bar{A}), \bar{T}}$
	Theorem 8: $V' \subseteq_{\pi_{gr}} \mid \equiv_{\pi_{gr}} V$

Case 2. $\boxed{\mathcal{VE} \ = \supseteq}$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \supseteq$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $(D1) if\ R.X \ \in \ Attr(V(R))_{\text{SELECT}}$ $then\ R.X \ \in \ Attr(V(R))_{\text{SELECT}}(t, \_),$ $R.X \ \notin \ Attr(V(R))_{\text{SELECT}}(f, \_)$ $(D2) if\ R.X \ \in \ Attr(V(R))_{\text{WHERE}}$ $then\ R.X \ \in \ (Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\_, t)$ $\cup\ Attr(V(R))_{\text{WHERE}}(t, \_)),$ $R.X \ \notin \ Attr(V(R))_{\text{WHERE}}(f, f)$	Condition 3: $(E3) \forall C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1),$ $\exists CV(R.X)\ in\ \mathcal{CV}\ s.t.$ $CV(R.X) \rightarrow C_R(R.X)$ $(E4) \forall$ indisp. $CV(R.X)\ in\ \mathcal{CV}$ $\exists C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)\ s.t.$ $CV(R.X) \rightarrow C_R(R.X)$

CREATE VIEW	$V' \ \ (\mathcal{VE} = \supseteq)$ AS
SELECT	$\underline{grAttr(V(R)) \setminus \{R.X\}, \bar{T}}$
	$\underline{\mathcal{F}(S(R.X), (aggAttr(V(R)) \setminus \{R.X\}), \bar{F})}$
FROM	$\underline{S, R', R_1, \ldots, R_n}$
WHERE	$\underline{(R'.\bar{A} \cap Attr(V(R))) = (S.\bar{B} \cap S(Attr(V(R)))),}$
	$\underline{\mathcal{C}_S(S.\bar{B}_1),}$
	$\underline{\mathcal{CV}(S(R.X), (Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}), \bar{W})}$
GROUP BY	$\underline{grAttr(V(R)) \setminus \{R.X\}, \bar{T}}$
	Theorem 8: $V' \supseteq_{\pi_{gr}} V$

Table 3.16: SPOC Synchronization Strategy for the Aggregation View V, Eq. (3.50), "*delete-attribute* $R.X$" and Type (I) $\mathcal{PC}$-constraint ($R.X \notin R.\bar{A}$).

Case 3. $\mathcal{VE} \; = \equiv$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \equiv$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $(D1) if\ R.X \in Attr(V(R))_{\text{SELECT}}$ $then\ R.X \in Attr(V(R))_{\text{SELECT}}(t,\_)$, $R.X \notin Attr(V(R))_{\text{SELECT}}(f,\_)$ $(D2) if\ R.X \in Attr(V(R))_{\text{WHERE}}$ $then\ R.X \in (Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\_,t)$ $\cup\ Attr(V(R))_{\text{WHERE}}(t,\_))$, $R.X \notin Attr(V(R))_{\text{WHERE}}(f,f)$	Condition 3: $(E0) Attr(V(R))_{\text{SELECT}} \setminus R.\bar{A}$ $\subseteq Attr(V(R))_{\text{SELECT}}(t,\_)$ $(E1) Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}$ $\subseteq R.\bar{A}$, $R.X \notin (Attr(V(R))_{\text{WHERE}}$ $\setminus Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\_,t))$ $(E2) \forall CV(R.X)\ in\ \mathcal{CV}$ $\exists C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)\ s.t.$ $C_R(R.X) \rightarrow CV(R.X)$ $(E3) \forall C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)$, $\exists CV(R.X)\ in\ \mathcal{CV}\ s.t.$ $CV(R.X) \rightarrow C_R(R.X)$ $(E4) \forall$ indisp. $CV(R.X)\ in\ \mathcal{CV}$ $\exists C_R(R.X)\ in\ \mathcal{C}_R(R.\bar{A}_1)\ s.t.$ $CV(R.X) \rightarrow C_R(R.X)$
CREATE VIEW V' $(\mathcal{VE} = \equiv)$ AS SELECT $\underline{(grAttr(V(R)) \cap R.\bar{A}), \bar{T}}$ $\underline{\mathcal{F}(S(R.X), (aggAttr(V(R)) \setminus \{R.X\}), \bar{F})}$ FROM $\underline{S, R', R_1, \ldots, R_n}$ WHERE $\underline{(R'.\bar{A} \cap Attr(V(R))) = (S.\bar{B} \cap S(Attr(V(R))))}$, $\underline{\mathcal{C}_S(S.\bar{B}_1)}$, $\underline{\mathcal{CV}(S(R.X), (Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}), \bar{W})}$ GROUP BY $\underline{(grAttr(V(R)) \cap R.\bar{A}), T}$		
Theorem 8: $V' \equiv_{\pi_{gr}} V$		

Table 3.17: SPOC Synchronization Strategy for the Aggregation View V, Eq. (3.50), "*delete-attribute* $R.X$" and Type (I) $\mathcal{PC}$-constraint ($R.X \notin R.\bar{A}$) (cont.).

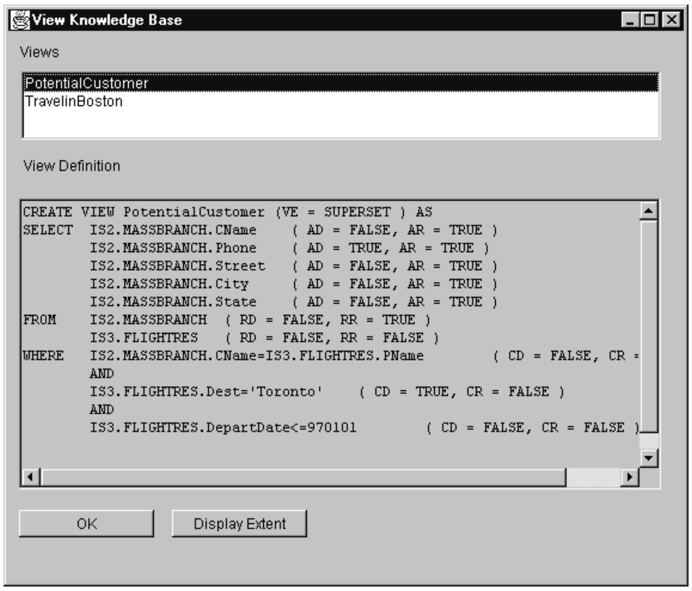


Figure 3.9: *EVE* System Prototype: View Knowledge Base *before* Synchronization.

Case 1. $\mathcal{VE} \ = \subseteq$

Condition 1:	Condition 2:	Condition 3:
$\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \subseteq \mid \equiv$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	$(D3) R.X \notin Attr(V(R))(f, f)$	$(E0) Attr(V(R))_{\text{SELECT}} \setminus R.\bar{A} \subseteq Attr(V(R))_{\text{SELECT}}(t, \_)$ $(E5) Attr(V(R))_{\text{WHERE}} \subseteq R.\bar{A}$, $R.X \notin Attr(V(R))_{\text{WHERE}}(\_, f)$

CREATE VIEW	$V' \ (\mathcal{VE} = \subseteq)$ AS
SELECT	$\underline{(grAttr(V(R)) \ \cap \ R.\bar{A}) \setminus \{R.X\}}$,
	$\underline{\{S(R.X) \text{ if R.X is replaceable}\}}, \bar{T}$
	$\underline{\mathcal{F}(S(R.X), (aggAttr(V(R)) \setminus \{R.X\}), \bar{F})}$
FROM	$\underline{S, R'}, R_1, \ldots, R_n$
WHERE	$\underline{((R'.\bar{A} \cap Attr(V(R))) \setminus \{R.X\}) = ((S.\bar{B} \cap S(Attr(V(R)))) \setminus \{S(R.X)\})}$,
	$\underline{\mathcal{C}_S(S.\bar{B}_1)}$,
	$\underline{\mathcal{CV}(S(R.X), (Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}), \bar{W})}$
GROUP BY	$\underline{(grAttr(V(R)) \cap R.\bar{A}) \setminus \{R.X\}}$,
	$\underline{\{S(R.X) \text{if R.X is replaceable}\}}, \bar{T}$

Theorem 8: $V' \subseteq_{\pi_{gr}} V$

Case 2. $\mathcal{VE} \ = \supseteq$

Condition 1:	Condition 2:	Condition 3:
$\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \supseteq \mid \equiv$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	$(D3) R.X \notin Attr(V(R))(f, f)$	$(E3) \forall C_R(R.X) \ in \ \mathcal{C}_R(R.\bar{A}_1)$, $\exists CV(R.X) \ in \ \mathcal{CV} \ s.t.$ $CV(R.X) \rightarrow C_R(R.X)$

CREATE VIEW	$V' \ (\mathcal{VE} = \supseteq)$ AS
SELECT	$\underline{grAttr(V(R)) \setminus \{R.X\}}$,
	$\underline{\{S(R.X) \text{ if R.X is replaceable}\}}, \bar{T}$
	$\underline{\mathcal{F}(S(R.X), (aggAttr(V(R)) \setminus \{R.X\}), \bar{F})}$
FROM	$\underline{S, R'}, R_1, \ldots, R_n$
WHERE	$\underline{((R'.\bar{A} \cap Attr(V(R))) \setminus \{R.X\}) = ((S.\bar{B} \cap S(Attr(V(R)))) \setminus \{S(R.X)\})}$,
	$\underline{\mathcal{C}_S(S.\bar{B}_1)}$,
	$\underline{\mathcal{CV}(S(R.X), (Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}), \bar{W})}$
GROUP BY	$\underline{grAttr(V(R)) \setminus \{R.X\}}$,
	$\underline{\{S(R.X) \text{if R.X is replaceable}\}}, \bar{T}$

Theorem 8: $V' \supseteq_{\pi_{gr}} V$
$\forall t \in V, \exists \ t' \in V' \ s.t. \ t' =_{\pi_{gr}} t$
$t'[f(X)] \geq t[f(X)], f \in \{MAX, COUNT\}$,
$t'[f(X)] \leq t[f(X)], f \in \{MIN\}$

Table 3.18: SPOC Synchronization Strategy for the Aggregation View V, Eq. (3.49), "*delete-attribute* $R.X$" and Type (II) $\mathcal{PC}$-constraint $(R.X \in R.\bar{A})$.

Case 3. $\mathcal{VE} \; = \equiv$		
Condition 1: $\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}(S)) \equiv$ $\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}(R))$	Condition 2: $(D3) R.X \notin Attr(V(R))(f, f)$	Condition 3: $(E0) Attr(V(R))_{\mathsf{SELECT}} \setminus R.\bar{A}$ $\subseteq Attr(V(R))_{\mathsf{SELECT}}(t, \_)$ $(E5) Attr(V(R))_{\mathsf{WHERE}} \subseteq R.\bar{A}$, $R.X \notin Attr(V(R))_{\mathsf{WHERE}}(\_, f)$ $(E3) \forall C_R(R.X)$ in $\mathcal{C}_R(R.\bar{A}_1)$, $\exists CV(R.X)$ in $\mathcal{CV}$ $s.t.$ $CV(R.X) \rightarrow C_R(R.X)$
CREATE VIEW V' $(\mathcal{VE} = \subseteq)$ AS SELECT $\underline{(grAttr(V(R)) \cap R.\bar{A}) \setminus \{R.X\}}$, $\underline{\{S(R.X) \text{ if R.X is replaceable}\}}, \bar{T}$ $\underline{\mathcal{F}(S(R.X), (aggAttr(V(R)) \setminus \{R.X\}), \bar{F})}$ FROM $\underline{S, R'}, R_1, \ldots, R_n$ WHERE $\underline{((R'.\bar{A} \cap Attr(V(R))) \setminus \{R.X\}) = ((S.\bar{B} \cap S(Attr(V(R)))) \setminus \{S(R.X)\})}$, $\underline{\mathcal{C}_S(S.\bar{B}_1)}$, $\underline{\mathcal{CV}(S(R.X), (Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.X\}), \bar{W})}$ GROUP BY $\underline{(grAttr(V(R)) \cap R.\bar{A}) \setminus \{R.X\}}$, $\underline{\{S(R.X) \text{if R.X is replaceable}\}}, \bar{T}$		
Theorem 8: $V' \equiv_{\pi_{gr}} V$ $\forall t \in V \exists t' \in V'$ $s.t.$ $t' =_{\pi_{gr}} t$ $t'[f(X)] \geq t[f(X)], f \in \{MAX, COUNT\}$, $t'[f(X)] \leq t[f(X)], f \in \{MIN\}$, $\forall t' \in V' \exists t \in V$ $s.t.$ $t =_{\pi_{gr}} t'$ $t[f(X)] \leq t'[f(X)], f \in \{MAX, COUNT\}$, $t[f(X)] \geq t'[f(X)], f \in \{MIN\}$, f an aggregate attribute in V and V'		

Table 3.19: SPOC Synchronization Strategy for the Aggregation View V, Eq. (3.50), "*delete-attribute* $R.X$" and Type (II) $\mathcal{PC}$-constraint $(R.X \in R.\bar{A})$ (cont.).

specified over two relations, **MASSBRANCH** and **FLIGHTRES**, residing at sites IS2 and IS3, respectively. Under the capability change "*delete-relation* **MASSBRANCH**" this view becomes undefined. The system applies the SPOC algorithm (Section 3.5) and, based on the meta knowledge recorded in the MKB, the view synchronizer finds a legal rewriting of **PotentialCustomer** (shown in Figure 3.10) by replacing the deleted relation **MASSBRANCH** by the relation **CUSTOMER**, which is provided by the provider for site IS1.

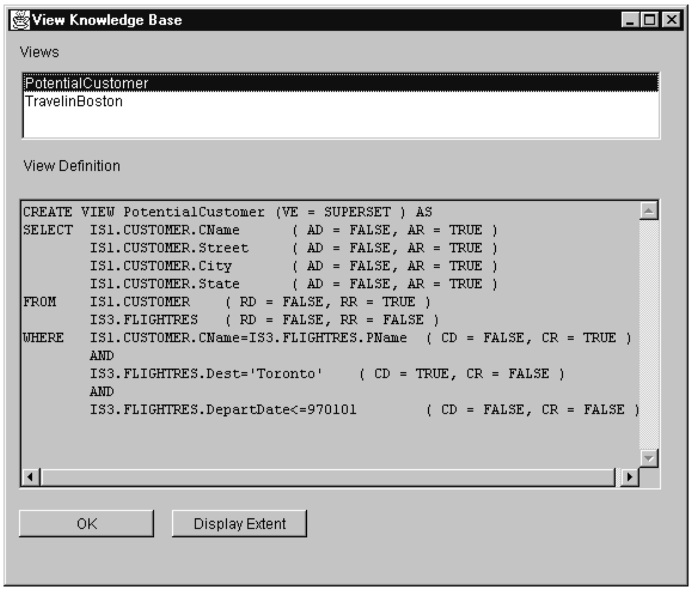


Figure 3.10: *EVE* System Prototype: View Knowledge Base *after* Synchronization.

3.7 Summary

Equipped with E-SQL and the MKB defined in the previous Chapter II, we propose in this chapter strategies for the view synchronization process that exploit *containment* relationships between ISs captured in the MKB.

The main contributions of this chapter can be summarized as following:

- For the *View Synchronization* problem identified in Section 1.2.3 as a new view adaptation problem, we identify three dimensions along which the complexity of view synchronization algorithms can be characterized.

- We present the first view synchronization algorithms that require only the availability of *containment information* among information sources in order to evolve views. Synchronization techniques are defined for a subset of coordinates of the view synchronization problem space, namely, for each dimension of the problem space, we completely study the following points: { Project $\mathcal{PC}$-constraints, Select-Project $\mathcal{PC}$-constraints } X { E-SQL SPJ views, E-SQL aggregation views } X { "*delete-attribute* ", "*delete-relation* " }.

- We identify the properties the replacement relations must have, and for each of the values of the view-extent evolution parameter, we define the synchronization strategies applied to obtained legal rewritings.

- We give formal proofs that the rewritings obtained by each strategy are legal rewritings satisfying all E-SQL evolution parameters attached to a view definition.

CHAPTER IV

View Synchronization Algorithms based on Join Constraints

Based on the formal foundation for view synchronization presented in Chapter II, we propose in the following a strategy for solving the view synchronization problem which uses the semantic constraints captured in the MKB such as *join and function-of constraints.* Rather than just providing so-called 'one-step-away' view rewriting as we have presented in Chapter III [31, 30, 48], the replacements found may correspond to possibly complex pieces of information from several ISs. For this, our solution succeeds in determining view rewritings through multiple join constraints given in the MKB. To demonstrate our approach, we present algorithms for handling the capability change operators that require replacements, namely, "*delete-attribute* " and "*delete-relation* " operators, in depth in this chapter. The proposed strategy is shown to find a new valid definition of a view in many cases where current view technology would have simply disabled the view, and where our proposed one-step view synchronization strategies POC and SPOC described in Chapter III would have failed to locate a suitable solution. The complex synchronization process is referred to as *Complex View Synchronization* (or short, CVS) as solution approach for the third step in the synchronization process.

In the remainder of this chapter we introduce in Section 4.1 the concept of a *legal rewriting* when the affected components have replacements in more than one relation, i.e., the replacements are found based on join constraints among ISs. In Section 4.2, we then present CVS algorithm for view synchronization and give examples when CVS succeeds in finding replacements. Section 4.3 concludes this chapter.

The work presented in this chapter has been published initially in the paper [40] as joint work with other members of the EVE project and also could be found in [39, 47]. The author subsequently developed the synchronization strategies for "*delete-attribute* " capability change as presented in Section 4.2.4.

4.1 Correctness Criteria for Complex View Synchronization

We assume views defined by an E-SQL view definition as shown in Equation (2.11) with a conjunction of primitive clauses in the WHERE clause. In the following we use the term *view component* to refer to a pair composed of a *view element* such as an attribute, relation or condition used in the view definition together with the set of *evolution parameters* attached to it. If an attribute, relation or condition has no evolution parameters associated with it in the view definition, we assume that the view element corresponding to it has the default parameters as defined in Table 2.6, Chapter II.

Definition 8 *Let ch be one of the capability changes "delete-relation " or "delete-attribute ", and let MKB and MKB' be the state of the meta knowledge base containing the IS descriptions right before and right after the change ch, respectively*[1]*. Then a view V' is a* ***legal rewriting*** *with complex replacements of the view V under*

[1] We assume that the meta knowledge base MKB is evolved into MKB' to reflect the change *ch* using the approach described in [39]. This evolution of the MKB is relatively straightforward, and hence a description of it is omitted here.

capability change ch if the following properties hold:

M1. V' *is a base legal rewriting (Definition 1).*

M2. The definition of the view V' *is* ***consistent*** *with the constraints of the evolved MKB'. That is, any new element in* V' *(e.g., new condition in the* WHERE *clause) is backed up by the existence of a supporting constraint in MKB'. Put differently, new elements appear in the view* V' *only if they are required to replace existing elements that, after a delete-relation or delete-attribute change, must be replaced by other elements.*

Thus, we have the following cases:

<u>Case 1.</u> A new element $g(S.A')(\mathcal{AD} = \mathcal{AD}_A, \mathcal{AR} = true)$ *could appear in the* SELECT *clause of* V' *if it replaces exactly one affected element (Section 2.5.2)* $R.A(\mathcal{AD} = \mathcal{AD}_A, \mathcal{AR} = true)$ *from the* SELECT *clause of* V[2]*. The attributes* $g(S.A')$ *and* $R.A$ *must have the same type and the following condition must hold:*

$\exists R_1, R_2, \ldots, R_n$ *such that* $R_1 = R$ *and* $R_n = S$,

$\exists \; \{\mathcal{JC}_{R_i,R_{i+1}} \mid \mathcal{JC}_{R_i,R_{i+1}}$ *a join constraint in MKB,* $i = \overline{1, n-1}\}$ *and it exists a function-of constraint* $\mathcal{F}_{R.A,S.A'} = (R.A = g(S.A'))$ *in MKB. That is, for any tuple* t *in the join relation:*

$$R_1 \bowtie_{\mathcal{JC}_{R_1,R_2}} R_2 \bowtie_{\mathcal{JC}_{R_2,R_3}} \cdots \bowtie_{\mathcal{JC}_{R_{n-1},R_n}} R_n \tag{4.1}$$

we have $t[R.A] = g(t[S.A'])$.

[2]In Section 2.5.2 we established what are the affected components for a given capability change. Note that the attribute $R.A$ must be replaceable, i.e., its attribute-replaceable parameter $\mathcal{AR}$ must be set to $true$.

All relations from the expression (4.1) (except relation R if R is to be dropped) must appear in the FROM *clause of the view V' and all primitive clauses (except the ones involving attributes of R if R is to be dropped) from the set of join constraints used in expression (4.1) must appear in the* WHERE *clause of the view V'. Moreover, all occurrences of the attribute $R.A$ in the conditions of the* WHERE *clause of V must be replaced by $g(S.A')$ in V'.*

<u>*Case 2.*</u> *A new element $C(\mathcal{CD} = \mathcal{CD}_C, \mathcal{CR} = \mathcal{CR}_C)$*[3] *could appear in the* WHERE *clause of V' if it is a primitive clause in one of the sequences of join constraints used to replace an attribute $R.A$ as in <u>Case 1</u>.*

<u>*Case 3.*</u> *A new element $R(\mathcal{RD} = \mathcal{RD}_R, \mathcal{RR} = \mathcal{RR}_R)$ could appear in the* FROM *clause of V' if it appears in one of the sequences of joins used to replace an attribute $R.A$ as in <u>Case 1</u>.*

M3. The definition of the view V' is ***minimal*** *with respect to the set of new relations in the* FROM *clause and the set of new conditions in the* WHERE *clause. That is, if we drop a newly added relation from the* FROM *clause or a newly added condition from the* WHERE *clause of V', the modified definition doesn't satisfy properties M1 and M2 any longer. M3 imposes that the new view definition cannot have new extraneous elements added that are not needed in the view, such as an extra attribute in the* SELECT *clause or an extra relation in the* FROM *clause that serves no purpose*[4]*.*

[3] The evolution parameters are set as defined in Definition 3.

[4] We impose the minimality property only for avoiding the introduction of new elements into V. Conforming to this definition, two view rewritings V' of V one containing all attributes from V that are marked as dispensable and the other one with all dispensable attributes dropped are both legal. On this venue, we incorporated into EVE a cost model to differentiate between the "quality" of these two alternative yet legal rewritings [27].

Example 22 *Let a view V be defined in Equation (4.2).*

$$\begin{array}{ll}
\textsf{CREATE VIEW} & \textit{\textbf{Asia-Customer}}\ (\ \textit{\textbf{AsiaName, AsiaAddress, AsiaPhone}})\ (\mathcal{VE} = \supseteq)\ \textsc{as} \\
\textsf{SELECT} & \textit{\textbf{C.Name, C.Address}}(\mathcal{AD} = false, \mathcal{AR} = true),\ \textit{\textbf{C.PhoneNo}} \\
\textsf{FROM} & \textit{\textbf{Customer C, FlightRes F}} \\
\textsf{WHERE} & \textit{\textbf{(C.Name = F.PName)}}\ \textsc{and}\ \textit{\textbf{(F.Dest = 'Asia')}}
\end{array} \tag{4.2}$$

And let's assume that change ch is "delete-attribute ***Customer.Address***". *We have to find a replacement for this attribute that could be obtained from a chain of join constraints defined in MKB'. Let's assume we have the following constraints in MKB:*

(i) The relation ***Person*** *is defined by* ***Person(Name, SSN, PAddress);***

(ii) $\mathcal{JC}_{\textit{Customer, Person}} = (\ \textit{\textbf{Customer.Name}} = \textit{\textbf{Person.Name}})$ *;*

(iii) $\mathcal{F}_{\textit{Customer.Address, Person.PAddress}} = (\ \textit{\textbf{Customer.Address}} = \ \textit{\textbf{Person.PAddress}})$ *;*

(iv) $\mathcal{PC}_{\textit{Customer, Person}} = (\pi_{\textit{Name, PAddress}}(\ \textit{\textbf{Person}}) \supseteq \pi_{\textit{Name, Address}}(\ \textit{\textbf{Customer}}))$.

It is easily verifiable that the new view definition ***Asia-Customer'*** *defined in Equation (4.3) is a legal rewriting (new elements are underlined) of Equation (4.2):*

$$\begin{array}{ll}
\textsf{CREATE VIEW} & \textit{\textbf{Asia-Customer'}}\ (\ \textit{\textbf{AsiaName, AsiaAddress, AsiaPhone}})\ (\mathcal{VE} = \supseteq)\ \textsc{as} \\
\textsf{SELECT} & \textit{\textbf{C.Name}},\ \underline{\textit{\textbf{P.PAddress}}\ (\mathcal{AD} = false, \mathcal{AR} = true)},\ \textit{\textbf{C.PhoneNo}} \\
\textsf{FROM} & \textit{\textbf{Customer C, FlightRes F}},\ \underline{\textit{\textbf{Person P}}} \\
\textsf{WHERE} & \textit{\textbf{(C.Name = F.PName)}}\ \textsc{and}\ \textit{\textbf{(F.Dest = 'Asia')}} \\
 & \underline{\textsc{and}\ \textit{\textbf{(P.Name = C.Name)}}}
\end{array} \tag{4.3}$$

This legal rewriting uses the join constraint $\mathcal{JC}_{\textit{Customer, Person}}$ *(defined in (ii)) to obtain the address from the relation* ***Person*** *by using the join relation* $\left(\textit{\textbf{Customer}} \bowtie_{\mathcal{JC}_{\textit{Customer, Person}}} \textit{\textbf{Person}}\right)$ *in the WHERE clause, and the function-of constraint defined in (iii). Then the evolved view definition given in Equation (4.3) has* *all the properties M1 to M3 from Definition 1. Thus it is a legal rewriting. Note, that we can prove that for the evolved view defined by Equation (4.3), the extent parameter "$\mathcal{VE} = \supseteq$" is satisfied given the $\mathcal{PC}$ constraint from (iv): i.e., for any state of the relations* ***Customer***, ***Person*** *and* ***FlightRes***, *we have* ***Asia-Customer'*** $\supseteq$ ***Asia-Customer***.

Example 23 *Queries that violate at least one of the properties from Definition 1 are given below:*

(A) The evolution parameters from the initial query V are not satisfied by attribute ***Address*** *being dropped. Thus property M1 is violated (i.e., the view obtained is not a legal rewriting (Definition 1)):*

$$\begin{array}{ll} \text{CREATE VIEW} & \textbf{\textit{Asia-Customer}}(\mathcal{VE} = \supseteq) \ \text{AS} \\ \text{SELECT} & \textbf{\textit{C.NameC.PhoneNo}} \\ \text{FROM} & \textbf{\textit{Customer C, FlightRes F}} \\ \text{WHERE} & \textbf{\textit{(C.Name = F.PName)}} \ \text{AND} \ \textbf{\textit{(F.Dest = 'Asia')}} \end{array} \tag{4.4}$$

(B) The expression used to obtain a replacement for the attribute ***Customer.Address*** *is not merged into the new definition by failing to add the join condition in the* WHERE *clause, thus violating M2:*

$$\begin{array}{ll} \text{CREATE VIEW} & \textbf{\textit{Asia-Customer}}(\mathcal{VE} = \supseteq) \ \text{AS} \\ \text{SELECT} & \textbf{\textit{C.Name,}} \ \underline{\textbf{\textit{P.PAddress}} \ (\mathcal{AD} = false, \mathcal{AR} = true)}, \ \textbf{\textit{C.PhoneNo}} \\ \text{FROM} & \textbf{\textit{Customer C, FlightRes F,}} \ \underline{\textbf{\textit{Person P}}} \\ \text{WHERE} & \textbf{\textit{(C.Name = F.PName)}} \ \text{AND} \ \textbf{\textit{(F.Dest = 'Asia')}} \end{array} \tag{4.5}$$

This is an example when failing to add the join constraint between the relations ***Customer*** *and* ***Person*** *in the* WHERE *clause results in the new view including meaningless tuples (coming from the Cartesian product between the new relation* ***Person*** *and the rest of the view).*

4.2 View Synchronization: The CVS Algorithm

We now describe our solution for the third step of the view synchronization process given in Section 3.2, namely, the actual rewriting of an affected view definition. As already indicated in Figure 3.3 of Section 3.2, four of the six capability change operations we consider can be handled in a straightforward manner. Namely, the *add-relation* and *add-attribute* capability changes do not cause any changes to existing (and hence valid) views, and we assume that our current system will not further optimize existing views based on this new knowledge. The two change-name capability change operators, *change-relation-name* and *change-attribute-name*, are caught by the name mapping service in the MKB and hence also do not require any synchronization of views.

However, the two remaining capability change operators ("*delete-attribute* " and "*delete-relation* ") cause existing views to become invalid and hence need to be addressed by the view synchronization algorithm. Below, we present the algorithm for handling these two most difficult operators, namely, "*delete-attribute* " and "*delete-relation* " operators.

We make the following assumption in order to keep the discussion simple, however, our described solution can be easily extended to handle less restrictive cases. Given a "*delete-relation* $IS.R$" request, we assume that an affected view V uses R only once in the FROM clause. Our **CVS** algorithm could be easily adapted for a more general case when the relation R appears more than once in the FROM clause.

Example 24 *To illustrate the steps of our approach for rewriting, we will use the view defined in Equation 4.6 below and the change operator "delete-relation **Customer**" both over the schema of our running example (Example 2.2). The view*

Customer-Passengers-Asia *defines (passenger, participant) pairs of passengers flying to Asia and participants to a tour in Asia that fly and start the tour at the same day, respectively. Such a view could be used to see what participants of a tour are flying to "Asia" on the same day as the tour starts.*

CREATE VIEW ***Customer-Passengers-Asia*** $(\mathcal{VE}_V)$ AS
SELECT ***C.Name*** $(\mathcal{AD} = false, \mathcal{AR} = true)$, ***C.Age*** $(\mathcal{AD} = true, \mathcal{AR} = true)$,
P.Participant $(\mathcal{AD} = true, \mathcal{AR} = true)$,
P.TourID $(\mathcal{AD} = true, \mathcal{AR} = true)$
FROM ***Customer C*** $(\mathcal{RD} = true, \mathcal{RR} = true)$,
FlightRes F $(\mathcal{RD} = true, \mathcal{RR} = true)$,
Participant P $(\mathcal{RD} = true, \mathcal{RR} = true)$
WHERE ***(C.Name = F.PName)*** $(\mathcal{CD} = false, \mathcal{CR} = true)$ AND
(F.Dest = 'Asia') AND
(P.StartDate = F.Date) AND ***(P.Location = 'Asia')*** (4.6)

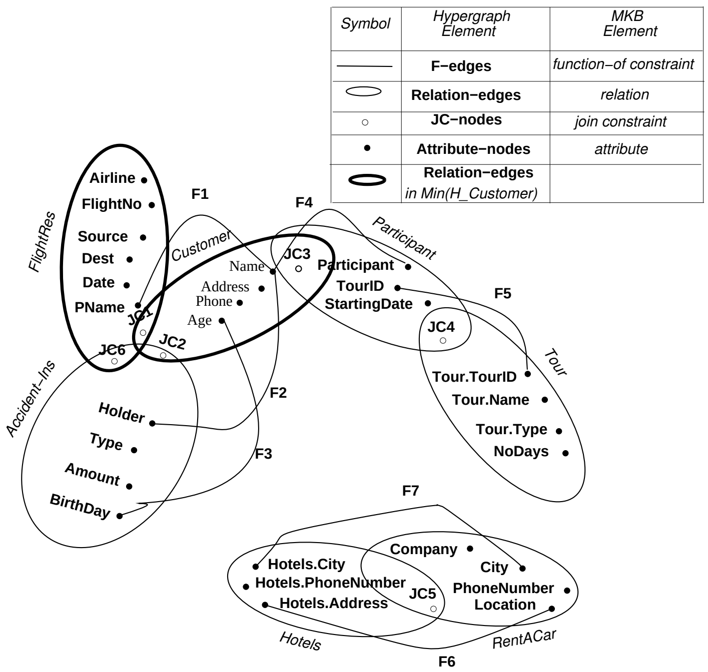


Figure 4.1: The Hypergraph $\mathcal{H}$(MKB) for Example 2.2 and View (4.6).

4.2.1 Mapping the MKB to a Hypergraph Representation

Generally, a database schema can be represented as a hypergraph whose nodes are the attributes and whose hyperedges are the relations [65, 11]. Inspired by this representation, we represent the set of attributes and the set of relations described in MKB by a hypergraph that is extended with extra nodes corresponding to the join constraints in the MKB and extra edges corresponding to the function-of constraints. Below we give the formal definition of a hypergraph representation of a MKB and in Figure 4.1 we give an example of its graphical representation.

Definition 9 ***Meta Knowledge Base Hypergraph*** $\mathcal{H}(MKB)$. *Given a meta knowledge base MKB, we define the hypergraph associated with the MKB as a four-tuple*

$\mathcal{H}(MKB) = \{\mathcal{A}(MKB), \mathcal{J}(MKB), \mathcal{S}(MKB), \mathcal{F}(MKB)\}$, *where:*

- *(*$\mathcal{A}(MKB) \cup \mathcal{J}(MKB)$ *) is the set of nodes of the hypergraph where:*

  (1) $\mathcal{A}(MKB)$ *are the attribute-nodes corresponding to the attributes defined in MKB; and*

  (2) $\mathcal{J}(MKB)$ *are the* $\mathcal{JC}$*-nodes corresponding to the join constraints defined in MKB.*

- *(*$\mathcal{S}(MKB) \cup \mathcal{F}(MKB)$*) is the set of edges for the hypergraph where:*

  (1) $\mathcal{S}(MKB)$ *are the relation-edges corresponding to the set of relations defined in MKB. For R a relation in MKB, the relation-edge R contains the set of attribute-nodes and* $\mathcal{JC}$*-nodes corresponding to the attributes of relation R and to the join constraints defined for relation R, respectively.*

  (2) $\mathcal{F}(MKB)$ *are the* $\mathcal{F}$*-edges corresponding to the set of function-of constraints defined in MKB. For* $\mathcal{F}$ *a function-of constraint in MKB, the* $\mathcal{F}$*-edge*

representing $\mathcal{F}$ contains two attribute-nodes corresponding to the attributes used in $\mathcal{F}$.

Example 25 *Figure 4.1 depicts the hypergraph for the MKB of our travel agency example (Example 2.1) where:*

$\mathcal{A}(MKB) = \{$ ***Name, Address, Phone, Age, Tour.TourID, TourName, Tour.Type, NoDays, Participant, TourID, StartDate, Participant.Location, PName, Airline, FlightNo, Source, Dest, Data, Holder, Type, Amount, Birthday, Hotels.City, Hotels.Address, Hotels.PhoneNumber, Company, City, PhoneNumber, RentACar.Location*** $\}$ *(see Figure 2.1);*

$\mathcal{J}(MKB) = \{$ ***JC1, JC2, JC3, JC4, JC5, JC6*** $\}$ *(see Figure 2.3);*

$\mathcal{S}(MKB) = \{$ ***Customer, Tour, Participant, FlightRes, Accident-Ins, Hotels, RentACar*** $\}$ *(see Figure 2.1);*

$\mathcal{F}(MKB) = \{$ ***F1, F2, F3, F4, F5, F6, F7*** $\}$ *(see Figure 2.4);*

The notations used for the graphical representation in Figure 4.1 and their connections with the elements of $\mathcal{H}(MKB)$ are given in the table below.

Note that two relation-edges can share a $\mathcal{JC}$-node if and only if it corresponds to a join constraint defined in MKB for the two relations.

We say that a hypergraph is disconnected if one can partition its hyperedges into nonempty sets such that no hypernode appears in hyperedges of different sets. If such partition doesn't exist, then we say that the hypergraph is connected. Using these definitions, one can define connected sub-hypergraphs of a disconnected hypergraph as being its maximal connected components. For our problem, we are interested in finding the connected sub-hypergraph that contains a given relation R.

$\mathcal{H}(MKB)$ element	MKB Concept	graphical symbol
attribute-node A	attribute A	filled circle labeled with the attribute name **A**
$\mathcal{JC}$-node $\mathcal{JC}_i$	join constraint $\mathcal{JC}_i$	empty circle labeled with $\mathcal{JC}$-name **JCi**
relation-edge R	relation R	a circle with attribute-nodes and $\mathcal{JC}$-nodes inside; labeled with relation name **R**
$\mathcal{F}$-edge $\mathcal{F}_j$	function-of constraint $\mathcal{F}_j$	a line connecting two attribute-nodes; labeled with $\mathcal{FC}$-name **Fj**

Table 4.1: $\mathcal{H}(MKB)$ Graphical Representation.

Definition 10 ***Connected Sub-Hypergraph*** $\mathcal{H}_R$***(MKB).*** *For a relation R, we define the sub-hypergraph* $\mathcal{H}_R$*(MKB) of the hypergraph* $\mathcal{H}$*(MKB) as being the connected sub-hypergraph that contains the relation-edge R plus all other hypernodes and hyperedges connected to it. Namely,* $\mathcal{H}_R$*(MKB) is defined by:*

$$\mathcal{H}_R(MKB) = \{\mathcal{A}_R(MKB), \mathcal{J}_R(MKB), \mathcal{S}_R(MKB), \mathcal{F}_R(MKB)\} \tag{4.7}$$

where (1) $\mathcal{A}_R(MKB)$*,* $\mathcal{J}_R(MKB)$*,* $\mathcal{S}_R(MKB)$ *and* $\mathcal{F}_R(MKB)$ *are subsets of* $\mathcal{A}(MKB)$*,* $\mathcal{J}(MKB)$*,* $\mathcal{S}(MKB)$ *and* $\mathcal{F}(MKB)$*, respectively; (2) R is a relation-edge in* $\mathcal{S}_R(MKB)$*; and (3)* $\mathcal{H}_R(MKB)$ *is a connected sub-hypergraph of* $\mathcal{H}(MKB)$*.*

Note that because $\mathcal{JC}$-nodes are the only shared nodes between relation-edges in $\mathcal{H}(MKB)$ and because $\mathcal{H}_R(MKB)$ is a connected sub-hypergraph, we have: $\forall$ S_1, $S_2 \in \mathcal{S}_R(MKB)$, there exists a sequence of join constraints $\mathcal{JC}_{S_1,R_1}, \ldots, \mathcal{JC}_{R_n,S_2}$ defined in MKB, with $R_1, \ldots, R_n \in \mathcal{S}_R(MKB)$ such that the following join relation can be defined $S_1 \bowtie_{\mathcal{JC}_{S_1,R_1}} R_1 \cdots \bowtie \cdots \bowtie_{\mathcal{JC}_{R_n,S_2}} S_2$.

Example 26 *Figure 4.1 depicts two connected sub-hypergraphs for the hypergraph* $\mathcal{H}$*(MKB) for Example 2.2.*
For R = ***Customer****, the connected sub-hypergraph* $\mathcal{H}_{\textbf{Customer}}(MKB)$ *is the connected sub-hypergraph drawn on the top left of the Figure 4.1.*

4.2.2 R-mapping from a View into MKB Hypergraph

Given a view definition referring to a relation R and an MKB, we want to determine which parts of the affected view definition need to be replaced when R is dropped. To find possible replacements, we must look in the MKB for join constraints related to the relation R that are also used in the view definition. That is, the view could be seen as a join between a join relation defined using only join constraints from MKB and some other relations (the rest of the view definition). As we will show later, if R is to be dropped, our synchronization algorithm will try to substitute the affected part of the view definition with another join relation defined using join constraints from MKB. Definition 11 formally defines this relationship between a view definition and the (default) join constraints in MKB related to the relation R that could potentially be exploited to locate replacements for R.

Definition 11 ***R-mapping of a view V into sub-hypergraph $\mathcal{H}_R(MKB)$.*** *For a view definition V defined as in Equation (2.11) and a relation R from the* FROM *clause of the view V, we define the R-mapping of V into $\mathcal{H}_R(MKB)$ by R-mapping(V, $\mathcal{H}_R(MKB)$) $= (Max(V_R), Min(\mathcal{H}_R))$ to be a pair of two subexpressions one constructed from the view V and the second one constructed from the connected sub-hypergraph $\mathcal{H}_R(MKB)$ such that the following must hold:*

(I) The expression $Max(V_R)$ is of the form:

$$Max(V_R) \ = \ R_{v_1} \bowtie_{\mathcal{C}_{R_{v_1},R_{v_2}}} \cdots \bowtie_{\mathcal{C}_{R_{v_{l-1}},R_{v_l}}} R_{v_l} \tag{4.8}$$

such that relations $\{R_{v_1},\ldots,R_{v_l}\}(\ni R)$ are from the FROM *clause of V, and $\{\mathcal{C}_{R_{v_1},R_{v_2}},\ldots,\mathcal{C}_{R_{v_{l-1}},R_{v_l}}\}$ are conjunctions of primitive clauses from the* WHERE *clause of V. A conjunction $\mathcal{C}_{R_{v_{s-1}},R_{v_s}}$ contains all the primitive clauses that use only attributes of relations $R_{v_{s-1}}$ and R_{v_s} (both local and join conditions).*

(II) The expression $Min(\mathcal{H}_R)$ is of the from:

$$Min(\mathcal{H}_R) = R_{v_1} \bowtie_{\mathcal{JC}_{R_{v_1},R_{v_2}}} \cdots\cdots \bowtie_{\mathcal{JC}_{R_{v_{l-1}},R_{v_l}}} R_{v_l} \tag{4.9}$$

where relations $\{R_{v_1},\ldots,R_{v_l}\} \subseteq \mathcal{S}_R(MKB)$, and $\{\mathcal{JC}_{R_{v_1},R_{v_2}},\ldots,\mathcal{JC}_{R_{v_{l-1}},R_{v_l}}\} \subseteq \mathcal{J}_R(MKB)$.

(III) The relation defined by $Max(V_R)$ is contained in the relation defined by $Min(\mathcal{H}_R)$:

$$\underbrace{\left(R_{v_1} \bowtie_{\mathcal{C}_{R_{v_1},R_{v_2}}} \cdots \bowtie \cdots \bowtie_{\mathcal{C}_{R_{v_{l-1}},R_{v_l}}} R_{v_l}\right)}_{Max(V_R)} \subseteq \underbrace{\left(R_{v_1} \bowtie_{\mathcal{JC}_{R_{v_1},R_{v_2}}} \cdots \bowtie \cdots \bowtie_{\mathcal{JC}_{R_{v_{l-1}},R_{v_l}}} R_{v_l}\right)}_{Min(\mathcal{H}_R)} \tag{4.10}$$

(IV) The expression $Max(V_R)$ is maximal with the properties (I) and (III). I.e., there is no other relations from the FROM *clause and primitive clauses from the* WHERE *clause of the view V that could be added to it and still be able to find a subexpression in $\mathcal{H}_R(MKB)$ such that (II) and (III) are satisfied.*

(V) The expression $Min(\mathcal{H}_R)$ is minimal with the properties (II) and (III). I.e., we cannot drop a relation or a join condition from it and still have (II) and (III) satisfied.

Note that Equation 4.10 implies that there exists a conjunction of primitive clauses $\mathcal{C}_{Max/Min}$ such that

$$Max(V_R) = \sigma_{\mathcal{C}_{Max/Min}}(Min(\mathcal{H}_R)) \tag{4.11}$$

The goal of Definition 11 is to find the expressions $Max(V_R)$ and $Min(\mathcal{H}_R)$ such that the view V could be written as:

$$V = \pi_{\bar{B}_V}\left(\underbrace{\left(\sigma_{\mathcal{C}_{Max/Min}}(Min(\mathcal{H}_R))\right)}_{Max(V_R)} \bowtie_{\mathcal{C}_{Rest}} Rest\right) \tag{4.12}$$

where $\bar{B}_V$ is the view interface (see Equation (2.11)), $\mathcal{C}_{Rest}$ and $Rest$ are the rest of the primitive clauses and relations in V, respectively. $Rest$ is a join relation containing relations from the FROM clause that don't appear in $Min(\mathcal{H}_R)$.

Example 27 *In Figure 4.1, the minimal subexpression $Min(\mathcal{H}_{Customer})$ of $\mathcal{H}_{Customer}(MKB)$ is marked by bold lines and corresponds to:*

$$Min(\mathcal{H}_{Customer}) \ = \ \textbf{\textit{FlightRes}} \bowtie_{\underbrace{(FlightRes.PName=Customer.Name)}_{JC1}} \textbf{\textit{Customer}} \tag{4.13}$$

*The maximal subexpression $Max($**Customer-Passenger-Asia**$_{Customer})$ of the view defined by Equation (4.6) and the relation **Customer** is:*

$$Max(\textbf{\textit{Customer-Passenger-Asia}}_{Customer}) \ = \tag{4.14}$$

$$\textbf{\textit{FlightRes}} \bowtie_{\underbrace{\left(\begin{array}{c}\textbf{\textit{(FlightRes.PName = Customer.Name)}} \\ \textbf{\textit{AND (FlightRes.Dest = 'Asia')}}\end{array}\right)}_{\mathcal{C}_{FlightRes,\ Customer}}} \textbf{\textit{Customer}}$$

$$= \ \sigma_{\underbrace{FlightRes.Dest\ =\ 'Asia'}_{\mathcal{C}_{Max/Min}}}(Min(\mathcal{H}_R)) \tag{4.15}$$

The relation defined by Equation(4.14) is contained in the relation defined by Equation (4.13) and they are maximal and minimal, respectively, with this property. I.e., the relation defined by Equation (4.14) is the maximal subexpression in the view defined in Equation (4.6) having the properties (I) to (III): if we add a new relation from the expression given in Equation (4.6) to it, we cannot find any longer a subexpression of $\mathcal{H}$(MKB) so that these properties hold. The same we can say for the subexpression defined by Equation (4.13) to be minimal with the properties (I) to (III).

Observation. To find two expressions $Max(V_R)$ and $Min(\mathcal{H}_R)$ with the property (III), it is sufficient ([65]) to have that each join constraint $\mathcal{JC}_{S,S'}$ of expression $Min(\mathcal{H}_R)$ (Equation 4.9) is implied by the corresponding join condition $\mathcal{C}_{S,S'}$ of expression $Max(V_R)$ (Equation 4.8), where $S, S' \in \{R_{v_1}, \ldots, R_{v_l}\}$.

Computing R-mapping(V, $\mathcal{H}_R(MKB)$) = $(Max(V_R), Min(\mathcal{H}_R))$.

To find $Max(V_R)$ and $Min(\mathcal{H}_R)$ having the above property, we start by selecting all relations S that join with R in V with a join condition $\mathcal{C}_{R,S}$ such that $\exists \mathcal{JC}_{R,S}$ in MKB, and $\mathcal{C}_{R,S}$ implies $\mathcal{JC}_{R,S}$. Then for the relations found by this first step, we recursively find others that are joined with them in V with join conditions that imply the corresponding join constraints in MKB, until we cannot find any new relation to add. $Min(\mathcal{H}_R)$ is the join of the relations found using the join constraints from MKB. $Max(V_R)$ is the join of the relations found using the join conditions from the WHERE clause of V.

4.2.3 R-replacements Associated With R-mappings

Intuitively, we now have found the maximal part of the view definition that "relates" to our MKB (Definition 11). So now we can ask how this part (i.e., $Max(V_R)$) is affected by the relation R being dropped. And, further, we need to determine how we can find new join relations from the MKB that can replace affected view components in the view definition (i.e., $Max(V_R)$). The next definition identifies what are the most useful candidates for such replacement that we can construct using join constraints defined in MKB. Note that at this point we don't worry about the relationship between the R-mapping and the potential candidates (e.g., subset, equivalent or superset). Our goal is to find all possible replacements for the relation $Max(V_R)$ (Equation 4.12). Only after that, when given the view-extent parameter $\mathcal{VE}_V$ (Section 2.4) and the $\mathcal{PC}$ constraints from MKB (Section 2.3), we want to choose the ones that satisfy the properties of legal rewritings (see Definition 1).

Definition 12 ***R-replacement($V, \mathcal{H}_R(MKB)$).*** *For a given view V and the MKB, we compute a set of expressions constructed from $\mathcal{H}_R(MKB)$ that don't contain R*

and could be used to meaningfully replace the maximal subexpression $Max(V_R)$ *in* V. *Let MKB' be the meta knowledge base evolved from MKB (see Section 3.2) when relation R is dropped; and* $\mathcal{H}'_R(MKB')$ *be the sub-hypergraph of* $\mathcal{H}_R(MKB)$ *obtained by erasing relation-edge R. We define R-replacement(*$V, \mathcal{H}_R(MKB)$*)* $=$ $\{Max(V_{1,R}), \ldots, Max(V_{l,R})\}$ *to be a set of subexpressions constructed from* $\mathcal{H}'_R(MKB')$ *and* $Max(V_R)$ *such that a subexpression* $Max(V_{j,R})$ *has the following properties:*

(I) $Max(V_{j,R}) = \sigma_{\mathcal{C}'_{Max/Min}}\left(R_1 \ltimes_{\mathcal{JC}_{R_1,R_2}} \cdots \ltimes_{\mathcal{JC}_{R_{k-1},R_k}} R_k\right)$ *with* $R_1, \ldots, R_k$ *and* $\mathcal{JC}_{R_1,R_2}, \ldots, \mathcal{JC}_{R_{k-1},R_k}$ *in* $\mathcal{H}'_R(MKB')$.

(II) R doesn't appear in $Max(V_{j,R})$. *I.e., R not among* $R_1, \ldots, R_k$.

(III) The expression $Min(\mathcal{H}_R)$ *without R,* $Min(\mathcal{H}'_R)$*, could be mapped into* $Max(V_{j,R})$. *That is, if* $Min(\mathcal{H}_R)$ *is given by the Equation (4.9) then:*

$\{R_{v_1}, \ldots, R_{v_l}\} \setminus \{R\} \subseteq \{R_1, \ldots, R_k\}$ *and*

$(\{\mathcal{JC}_{R_{v_1},R_{v_2}}, \ldots \mathcal{JC}_{R_{v_{l-1}},R_{v_l}}\} \setminus \{\mathcal{JC}_{S,S'} \mid S = R \text{ or } S' = R\}) \subseteq \{\mathcal{JC}_{R_1,R_2}, \ldots \mathcal{JC}_{R_{k-1},R_k}\}$.

I.e., the expression $Max(V_{j,R})$ *must contain all the elements of the expression* $Min(\mathcal{H}_R)$ *unaffected by dropping relation R.*

(IV) For any attribute $A \in R$ *that is indispensable and replaceable in the view definition, the expression* $Max(V_{j,R})$ *contains a relation* $S \in \{R_1, \ldots, R_k\}$ *such that there exists a function-of constraint* $\mathcal{F}_{R.A,S.B} = (R.A = f(S.B))$ *in MKB. We call the relation S a* ***cover*** *for the attribute A and the attribute* $f(S.B)$ *a* ***replacement*** *for the attribute A in* $Max(V_{j,R})$.

(V) The conjunction $\mathcal{C}'_{Max/Min}$ *is obtained from conjunction* $\mathcal{C}_{Max/Min}$ *by substituting the attributes of R with their* ***replacements*** *(see (IV)) if any, or dropping primitive clauses that are dispensable and for which no replacement was found for their attributes.*

Erasing R from the connected sub-hypergraph $\mathcal{H}_R(MKB)$ could lead to a disconnected sub-hypergraph $\mathcal{H}'_R(MKB')$. If $\mathcal{H}'_R(MKB')$ is disconnected and the relations left in $Min(\mathcal{H}'_R)$ are in disconnected components then the set R-replacement$(V, \mathcal{H}_R(MKB))$ is empty.

Example 28 *In Figure 4.2, the expression* $Min(\mathcal{H}'_{\textit{Customer}})$ *defined by Equation (4.13) is marked with bold lines:* $Min(\mathcal{H}'_{\textit{Customer}}) = ($ ***FlightRes***$)$.

If relations left in $Min(\mathcal{H}'_R)$ are in a connected component of $\mathcal{H}'_R(MKB')$, we construct the set $\{Max(V_{1,R}), \ldots, Max(V_{k,R})\}$ as in the following algorithm.

Computing R-replacement$(V, \mathcal{H}_R(MKB))$.

Step 1. First, we find all the relations that could qualify as **covers** for the indispensable and replaceable attributes of V as required by property (IV) of Definition 12. That is, for any indispensable attribute A of R from the SELECT clause, we find $Cover(A)$, a set of pairs (relation, function-of) from $\mathcal{H}'_R(MKB')$ such that $\forall(S, (R.A = f(S.B))) \in Cover(A)$, S is in $\mathcal{H}'_R(MKB')$ and there exists a function-of constraint $\mathcal{F}_{R.A,S.B}$ in MKB such that $\mathcal{F}_{R.A,S.B} = (R.A = f(S.B))$.
If there exists an indispensable attribute A of R such that $Cover(A) = \emptyset$, then the set R-replacement$(V, \mathcal{H}_R(MKB))$ is empty. In other words, no replacement can be found.

Step 2. Using the covers found at Step 1, we construct the expressions $Max(V_{j,R})$ as required by properties (I) and (III) of Definition 12. In $\mathcal{H}'_R(MKB')$, an expression $Max(V_{j,R})$ corresponds to a connected "path" that must contain all the join constraints (i.e., $\mathcal{JC}$-nodes) and relations (i.e., relation-edges) left in $Min(\mathcal{H}'_R)$ plus for each indispensable attribute A of R, one relation (i.e., the relation-edge) from the $Cover(A)$. The expression $Max(V_{j,R})$ is obtained by adding to this connected path, the conjunction $\mathcal{C}'_{Max/Min}$ defined by property (V) of Definition 12.

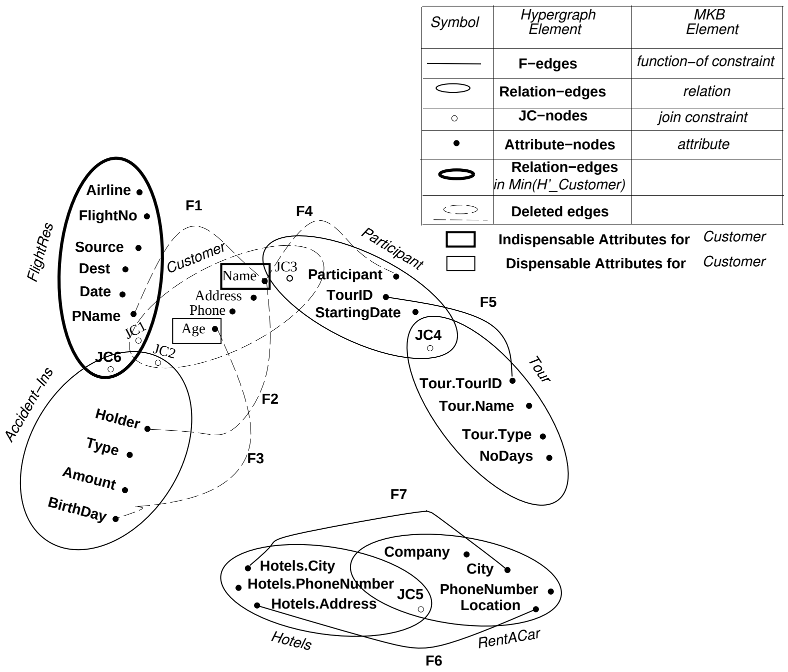


Figure 4.2: The Hypergraph $\mathcal{H}'$(MKB') for Example 2.2, View (4.6) and "*delete-relation* **Customer**" operator.

4.2.4 $R.A$-replacements Associated With R-mappings

As for the "*delete-relation R*" capability change, we can similarly find a replacement for the expression $Max(V_R)$ when *only one* attribute of the relation R has been deleted, i.e., the capability change is "*delete-attribute R.A*". The following definition identifies what are the most useful candidates for replacement that we can construct using join constraints defined in the MKB under the capability change "*delete-attribute R.A*".

Definition 13 $R.A$***-replacement***$(V, \mathcal{H}_R(MKB))$. *For a given view V and the MKB, we compute a set of expressions constructed from $\mathcal{H}_R(MKB)$ that don't contain $R.A$ and could be used to meaningfully replace the maximal subexpression $Max(V_R)$*

in V. Let MKB' be the meta knowledge base evolved from MKB (see Section 3.2) when the attribute R.A is deleted; and $\mathcal{H}'_R(MKB')$ be the sub-hypergraph of $\mathcal{H}_R(MKB)$ obtained by deleting the attribute-node corresponding to R.A.

A R.A-replacement($V, \mathcal{H}_R(MKB)$) $= \{Max(V_{1,R}), \ldots, Max(V_{l,R})\}$ is defined to be a set of subexpressions constructed from $\mathcal{H}'_R(MKB')$ and $Max(V_R)$ such that a subexpression $Max(V_{j,R})$ has the following properties:

(I) $Max(V_{j,R}) = \sigma_{\mathcal{C}'_{Max/Min}} \left(R_1 \bowtie_{\mathcal{JC}_{R_1,R_2}} \cdots \bowtie_{\mathcal{JC}_{R_{k-1},R_k}} R_k\right)$ with $R_1, \ldots, R_k$ and $\mathcal{JC}_{R_1,R_2}, \ldots, \mathcal{JC}_{R_{k-1},R_k}$ in $\mathcal{H}'_R(MKB')$.

(II) The expression $Min(\mathcal{H}_R)$ without the attribute R.A, $Min(\mathcal{H}'_R)$, could be mapped into $Max(V_{j,R})$. That is, if $Min(\mathcal{H}_R)$ is given by the Equation (4.9) then:

$\{R_{v_1}, \ldots, R_{v_l}\} \setminus \{R\} \cup \{R'\} \subseteq \{R_1, \ldots, R_k\}$ (R' is the relation R without the deleted attribute R.A) and

$(\{\mathcal{JC}_{R_{v_1},R_{v_2}}, \ldots \mathcal{JC}_{R_{v_{l-1}},R_{v_l}}\} \setminus \{\mathcal{JC}_{S,S'} \mid S = R \text{ or } S' = R\} \cup \{\mathcal{JC}'_{S,S'} \mid S = R \text{ or } S' = R\})$

$\subseteq \{\mathcal{JC}_{R_1,R_2}, \ldots \mathcal{JC}_{R_{k-1},R_k}\}$.

I.e., the expression $Max(V_{j,R})$ must contain all the elements of the expression $Min(\mathcal{H}_R)$ unaffected by dropping the attribute R.A. The join conditions on R are affected only if they refer to the attribute R.A, in which case they are evolved.

(III) For the attribute R.A that is replaceable in at least one component of the view V, the expression $Max(V_{j,R})$ contains a relation $S \in \{R_1, \ldots, R_k\}$ such that there exists a function-of constraint $\mathcal{F}_{R.A,S.B} = (R.A = f(S.B))$ in MKB. We call the relation S a ***cover*** *for the attribute R.A and the attribute $f(S.B)$ a* ***replacement*** *for the attribute R.A in $Max(V_{j,R})$.*

(IV) The conjunction $\mathcal{C}'_{Max/Min}$ is obtained from conjunction $\mathcal{C}_{Max/Min}$ by substituting the attribute R.A with its ***replacement****, or dropping primitive clauses that are*

dispensable and non-replaceable.

The computation of the $R.A$-replacements is done similarly to the computation of R-replacements where only one attribute, namely $R.A$, must have a cover.

Computing $R.A$**-replacement**$(V, \mathcal{H}_R(MKB))$.

Step 1. The first step involves finding all relations that could qualify as **covers** for the attribute $R.A$ of V as required by property (III) of Definition 13. $Cover(R.A)$ is a set of pairs (relation, function-of) from $\mathcal{H}'_R(MKB')$ such that $\forall(S, (R.A = f(S.B))) \in Cover(R.A)$, S is in $\mathcal{H}'_R(MKB')$ and there exists a function-of constraint $\mathcal{F}_{R.A,S.B}$ in MKB such that $\mathcal{F}_{R.A,S.B} = (R.A = f(S.B))$.

If $Cover(R.A) = \emptyset$, then the $R.A$-replacement$(V, \mathcal{H}_R(MKB))$ set is empty and the algorithm ends by failing to find any legal rewriting.

Step 2. Using the covers found at Step 1, we construct the expressions $Max(V_{j,R})$ as required by properties (I) and (II) of Definition 13. In $\mathcal{H}'_R(MKB')$, an expression $Max(V_{j,R})$ corresponds to a connected "path" that must contain all the join constraints (i.e., $\mathcal{JC}$-nodes) and relations (i.e., relation-edges) from $Min(\mathcal{H}'_R)$[5] plus a relation from the $Cover(R.A)$. The expression $Max(V_{j,R})$ is obtained by adding to this connected path the conjunction $\mathcal{C}'_{Max/Min}$ defined by property (IV) of Definition 12.

Example 29 *We give here an example of how R-replacements are constructed for the view defined by Equation (4.6) and* $R =$ ***Customer***. $\mathcal{H}'(MKB')$ *is depicted in Figure 4.2.*

Step 1. In our example, the only indispensable attribute of relation ***Customer*** *is* ***Customer.Name***. *Using the hypergraph depicted in Figure 4.2, we find:*

[5]In the case of "*delete-attribute* $R.A$", $\mathcal{H}'_R$ has the relation R' replacing R and possibly some of the join constraints evolved under "*delete-attribute* $R.A$".

$Cover(\textbf{\textit{Customer.Name}}) =$

$\{$ (***Accident–Ins***, ***F2*** = (***Customer.Name*** = ***Accident–Ins.Holder***)),

(***Participant***, ***F4*** = (***Customer.Name*** = ***Participant.Participant***)),

(***FlightRes***, ***F1*** = (***Customer.Name*** = ***FlightRes.PName***)) $\}$.

Step 2. *Following property (V) of Definition 12, we have that*

$\mathcal{C}'_{Max/Min}$ =***(FlightRes.Dest = 'Asia')***.

Let us now construct the candidate expressions

$Max($ ***Customer-Passenger-Asia***$_{j,}$ ***Customer***$)$,

$j \in \{1, 2, 3\}$, *and define what is the replacement for the attribute* ***Customer.Name***.

(1) For the cover (***Accident–Ins***, *(* ***Customer.Name*** = ***Accident–Ins.Holder*** *))* $\in$ $Cover($ ***Customer.Name***$)$, *we find the following expression that has all the properties from Definition 12.*

$$Max(\textbf{\textit{Customer-Passenger-Asia}}_{1,\textbf{\textit{Customer}}}) = \tag{4.16}$$

$$\underbrace{\sigma_{(\textbf{\textit{FlightRes.Dest = 'Asia'}})}}_{\mathcal{C}'_{Max/Min}} \left(\underbrace{\textbf{\textit{FlightRes}}}_{Min(\mathcal{H}'_{\textbf{\textit{Customer}}})} \bowtie_{\underbrace{(\textbf{\textit{FlightRes.PName = Accident–Ins.Holder}})}_{\textbf{\textit{JC6}}}} \underbrace{\textbf{\textit{Accident–Ins}}}_{in\ Cover(\textbf{\textit{Customer.Name}})} \right)$$

(2) For the cover (Participant, (Customer.Name = Participant.Participant)) $\in$ $Cover($ ***Customer.Name***$)$ *we see that there is no connected path in* $\mathcal{H}'(MKB')$ *(Figure 4.2) that contains this cover and the relation* ***FlightRes*** *(from* $Min(\mathcal{H}'_{\textbf{\textit{Customer}}})$*). Thus we cannot generate any replacement for*

$Max(\textbf{\textit{Customer-Passenger-Asia}}_{\textbf{\textit{Customer}}})$ *using this cover.*

(3) For the cover (***FlightRes***, *(* ***Customer.Name*** = ***FlightRes.PName*** *))* $\in$ $Cover($***Customer.Name***$)$, *we find the following expression that has all the properties*

from Definition 12.

$$Max(\textbf{\textit{Customer-Passenger-Asia}}_{2,\textbf{\textit{Customer}}}) = \underbrace{\sigma_{(\textbf{\textit{FlightRes.Dest}} = \textbf{\textit{'Asia'}})}}_{\mathcal{C}'_{Max/Min}} \left(\underbrace{\textbf{\textit{FlightRes}}}_{Min(\mathcal{H}'_{\textbf{\textit{Customer}}}), in\ Cover(\textbf{\textit{Customer.Name}})} \right) \tag{4.17}$$

4.2.5 Putting it all Together: The CVS Algorithm

Now we are ready to give the **Complex View Synchronization (CVS)** algorithm that has as input a view definition V, the MKB and a change ch one of "*delete-relation* R" and "*delete-attribute* $R.A$". It returns all legal rewritings conforming to Definition 1 of the view V.

Complex View Synchronization (CVS) Algorithm :

CVS(V, ch, MKB, MKB')

INPUT:

view definition V defined as in Equation (2.11);

change ch;

MKB represented by the hypergraph $\mathcal{H}(MKB)$;

evolved MKB' represented by the hypergraph $\mathcal{H}'(MKB')$.

OUTPUT:

A set of legal rewritings $V_1, \ldots V_l$ of V.

Step 1. Construct the sub-hypergraph $\mathcal{H}_R(MKB)$ as defined in Definition 10.

Step 2. Compute R-mapping(V, $\mathcal{H}_R(MKB)) = (Max(V_R), Min(\mathcal{H}_R))$ as defined in Definition 11.

Step 3. If ch is "*delete-relation* R" then compute R-replacement(V, $\mathcal{H}'_R(MKB')$) $= \{Max(V_{1,R}), \ldots, Max(V_{k,R})\}$ as defined in Definition 12. If R-replacement(V, $\mathcal{H}'_R(MKB') = \emptyset$ then the algorithm fails to find an evolved view definition for the view V.

If ch is "*delete-attribute* $R.A$" then compute $R.A$-replacement(V, $\mathcal{H}'_R(MKB')) = \{Max(V_{1,R}), \ldots, Max(V_{k,R})\}$ as defined in Definition 13. If $R.A$-replacement(V, $\mathcal{H}'_R(MKB') = \emptyset$ then the algorithm fails.

Step 4. A legal rewriting V' is found by replacing $Max(V_R)$ with $Max(V_{j,R})$ in Equation 4.12. Because some more conditions are added in the WHERE clause (corresponding to the join conditions in $Max(V_{j,R})$), we have to check if there are no inconsistencies in the WHERE clause. Example 30 below gives some examples of evolved view definitions generated by Step 4 for the view defined by the Equation (4.6).

Step 5. Set the E-SQL evolution parameters for all V' obtained at Step 4 as defined in Section 2.5, Definition 3.

Step 6. All the rewritings obtained by Step 4 have properties M2, and M3 from Definition 8, Section 4.1. At this step, we have to check for which rewriting V' obtained in Step 4 the extent parameter $\mathcal{VE}_V$ of the view V is satisfied in order to see if the property P3 from Definition 1 is satisfied and hence property M1 (Definition 8) is satisfied. This issue is similar to the problem of answering queries using views which was extensively studied in the database community [7, 35]. However, in our problem domain, we have an added issue of the availability of the set of partial/complete information constraints defined in MKB' that could be used to compare the extent of the initial view V and the extent of the evolved view V'.

Example 30 *For our view **Customer-Passenger-Asia** defined by Equation (4.6), we now show how to apply Steps 4 and 5 from the CVS algorithm and find replacements under the change "delete-relation **Customer**".*

The expression $Max($ **Customer-Passenger-Asia**$_{\text{Customer}}) =$

$\left(\textbf{\textit{FlightRes}} \bowtie_{\left(\substack{\textbf{\textit{(FlightRes.PName = Customer.Name) AND}} \\ \textbf{\textit{(FlightRes.Dest = 'Asia')}}}\right)} \textbf{\textit{Customer}} \right)$ *(Example 27, Equation (4.14)) could be replaced by one of the following expressions found at Step 3 of the **CVS** algorithm:*

(1) $Max($ ***Customer-Passenger-Asia***$_{1,Customer})$ $=$

$\sigma_{\textbf{\textit{(FlightRes.Dest = 'Asia')}}}($ ***FlightRes*** $\bowtie_{\textbf{\textit{(PName = Holder)}}}$ ***Accident–Ins***$)$.

For this particular case, we see that the attribute ***Customer.Age*** *is also covered by the relation* ***Accident–Ins*** *with the function-of constraint* ***F3*** $=$ *(****Customer.Age*** $=$ $($*today* $-$ ***Accident–Ins.Birthday***$)/365$*). In this case, we can replace the attribute* ***Customer.Age*** *in the view, too. A new rewriting of Equation (4.6) using this substitution as well is given by Equation (4.18). There are no contradictions in the* WHERE *clause after the replacements are done.*

$$
\begin{array}{ll}
\text{CREATE VIEW} & \textbf{\textit{Customer-Passengers-Asia}}_1 \text{ AS} \\
\text{SELECT} & \underline{\textbf{\textit{AI.Holder}}\ (\mathcal{AD} = false, \mathcal{AR} = true)}, \\
 & \underline{\textbf{\textit{f(AI.Birthday)}}\ (\mathcal{AD} = true, \mathcal{AR} = true)}, \\
 & \textbf{\textit{P.Participant}}\ (\mathcal{AD} = true, \mathcal{AR} = true), \\
 & \textbf{\textit{P.TourID}}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\
\text{FROM} & \underline{\textbf{\textit{Accident–Ins AI}}\ (\mathcal{RD} = true, \mathcal{RR} = true)}, \\
 & \textbf{\textit{FlightRes F}}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
 & \textbf{\textit{Participant P}}\ (\mathcal{RD} = true, \mathcal{RR} = true) \\
\text{WHERE} & \underline{\textbf{\textit{(F.PName = AI.Holder)}}\ (\mathcal{CD} = false, \mathcal{CR} = true)} \text{ AND} \\
 & \textbf{\textit{(F.Dest = 'Asia')}} \text{AND} \\
 & \textbf{\textit{(P.StartDate = F.Date)}} \text{ AND } \textbf{\textit{(P.Location = 'Asia')}}
\end{array}
\tag{4.18}
$$

(2) $Max($ ***Customer-Passenger-Asia***$_{2,Customer})$ $=$

$\sigma_{\textbf{\textit{(FlightRes.Dest = 'Asia')}}}$ ***FlightRes***.

A new rewriting of the view from Equation (4.6) is given in Equation (4.19). There are no contradictions in the WHERE *clause after the replacement is done.*

$$\begin{array}{ll}
\text{CREATE VIEW} & \textbf{\textit{Customer-Passengers-Asia}}_2 \text{ AS} \\
\text{SELECT} & \underline{\textbf{\textit{F.PName}} \ (\mathcal{AD} = false, \mathcal{AR} = true),} \\
 & \textbf{\textit{P.Participant}} \ (\mathcal{AD} = true, \mathcal{AR} = true), \\
 & \textbf{\textit{P.TourID}} \ (\mathcal{AD} = true, \mathcal{AR} = true) \\
\text{FROM} & \textbf{\textit{FlightRes F}} \ (\mathcal{RD} = true, \mathcal{RR} = true), \\
 & \textbf{\textit{Participant P}} \ (\mathcal{RD} = true, \mathcal{RR} = true) \\
\text{WHERE} & (\textbf{\textit{F.Dest}} = \text{'}\textbf{\textit{Asia}}\text{'}) \\
 & \text{AND } (\textbf{\textit{P.StartDate}} = \textbf{\textit{F.Date}}) \text{ AND } (\textbf{\textit{P.Location}} = \text{'}\textbf{\textit{Asia}}\text{'})
\end{array} \tag{4.19}$$

Step 6 in the CVS algorithm requires to verify that the view-extent evolution parameter is satisfied by rewritings found in the previous steps of the algorithm. Unlike the approach proposed for query rewriting using materialized views [33, 61, 7, 36] our proposed techniques address new issues: (1) finding view rewritings that are not necessarily *equivalent* to the original view definition ($\mathcal{VE} \in \{\subseteq, \equiv, \supseteq\}$), (2) using semantic containment information expressed using $\mathcal{PC}$-constraints for proving that candidate rewritings satisfy the view-extent evolution parameter, and (3) preserving at least indispensable attributes from the SELECT clause if preserving all is not possible. We give an example for when the view-extent parameter could be shown to be satisfied by a $\mathcal{PC}$-constraint.

Example 31 *Let's assume that the view* ***Customer-Passengers-Asia*** *has the view-extent evolution parameter* $\mathcal{VE}$ *set to "*$\subseteq$*". And the* $\mathcal{PC}$*-constraint shown in Equation (4.20) is defined in MKB between the relation* ***Customer*** *and the relation* ***FlightRes*** *(i.e., the* ***Customer*** *relation has all the passenger names from* ***FlightRes*** *relation). Then we can prove that the view* ***Customer-Passengers-Asia***$_2$ *defined in Equation (4.19) is satisfying the view-extent evolution parameter. I.e.,* ***Customer-Passengers-Asia***$_2$ $\subseteq_\pi$ ***Customer-Passengers-Asia*** *for any states of the relations involved in the views (as required by Definition 1, Section 2.5).*

$$\mathcal{PC}_{Customer, FlightRes} = (\pi_{Name}(\textbf{\textit{Customer}}) \supseteq \pi_{PName}(\textbf{\textit{FlightRes}})) \tag{4.20}$$

*Indeed, let $t' \in$ **Customer-Passengers-Asia**$_2$ be a tuple in the new view. Then there must exist two tuples $t_F \in$ **FlightRes** and $t_P \in$ **Participant** that generate the tuple t'. But from the $\mathcal{PC}$-constraint defined in Equation (4.20) there must exist a tuple $t_C \in$ **Customer** having the same value for the attribute **Name** as the tuple t_F has for the attribute **PName**. One can easily see that the tuples t_C, t_F and t_P generate a tuple $t \in$ **Customer-Passengers-Asia** such that $t =_\pi t'$.*

4.3 Summary

Equipped with E-SQL and the MKB introduced in Chapter II, we propose in this chapter strategies for the view synchronization process. We present the Complex View Synchronization (CVS) algorithm that fully exploits the constraints defined in MISD by allowing relation substitution to be done by a sequence of joins among candidate relations. It can be shown that CVS meets all preservation constraints in the view definition, while finding appropriate information from other ISs as replacement for affected components. CVS finds a new valid rewriting of a view in many cases where current view technology would have simply disabled the view, and where our previous one-step view synchronization (POC/SPOC from Chapter III) would have failed to locate a suitable solution. Examples to illustrate the main ideas are given throughout this part of the dissertation. In particular, we treat in depth view evolution caused by the "*delete-relation* " and "*delete-attribute* " capability changes. To summarize, the main contributions of this chapter are:

- We have presented a formal description of when a view rewriting generated in response to an IS capability change is considered to be *legal rewriting* when the affected components are replaced by *more than one* component.

- We have designed a solution approach for view synchronization that achieves view rewriting by exploiting chains of *multiple join constraints* given in the MKB.

- To demonstrate our solution approach, we have presented the **Complex View Synchronization (CVS)** algorithm for handling the most difficult capability change operators, namely, "*delete-relation* " and "*delete-attribute* " operators.

CHAPTER V

View Maintenance after View Synchronization

In the previous Chapters III and IV, we presented different strategies for evolving view definitions under capability changes supported in *EVE*. The proposed view rewriting process finds a view redefinition that meets all view preservation constraints specified by the E-SQL view definition including the constraint imposed on the relationship between the old and new view extent in case of POC/SPOC algorithms. Unlike the strategies proposed for query rewriting using views in the database literature ([33, 61]), our proposed techniques address three new issues: (1) finding view rewritings that are not necessarily *equivalent* to the original view definition, (2) using semantic *containment information* for replacing the deleted relation; and (3) partially preserving the view interface (i.e., at least indispensable attributes from the SELECT clause) if preserving all is not possible.

In this chapter, 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 view maintenance process corresponds to the step *MAINTENANCE-AFTER-SYNCH* of the general *EVE* synchronization algorithm presented in Chapter III, Section 3.2,

Figure 3.2.

On the first sight, one could attempt to map this problem to the explicit view redefinition problem studied by others in [37, 17]: the view definition changes done by the view synchronization process could be seen as a sequence of simple local changes of different types assumed to be explicitly requested by a user as in [37, 17]. In Section 5.2 we give an explicit example of such mapping after view synchronization process. Once this mapping is completed, one can apply the maintenance-after-redefinition strategies from [37, 17] for each such local change. However, as we will show here, treating the possible complex rewriting achieved by synchronization as a sequence of local view changes is a very inefficient approach in terms of keeping the view extent up-to-date. Most importantly, some of the techniques proposed in [37, 17] 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 is that the deleted attribute or deleted relation that is to be queried during maintenance is no longer available at the base relation site. Maintenance strategies for view redefinition in general assume (including those given in [37, 17]) that base relations *don't* change and are available during the process of maintenance.

To overcome these limitations of previous techniques, 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) to the view definition as atomic operations and handle associated maintenance tasks, such as querying ISs, as efficiently as possible. One key concept here is that the maintenance strategies take advantage of the knowledge of how the view rewritings were obtained, especially the containment information between the old view and the new view, which is available in our system due to the evolution being performed by the system itself. The proposed techniques are general enough to be also applicable when complex (simultaneous) view redefinitions are *explicitly* requested by the user and *directly applied* to the view definition. These cases were not treated in the previous work [37, 17], and thus our results are an immediate extension of this previous work now covering a larger class of complex changes explicitly applied to a view definition. Moreover, our solution also applied for view maintenance after ISs in the underlying information space have schema level changes.

To validate our solution, we present the results of a systematic experimental evaluation done using the TPC-D benchmark [64], comparing our proposed techniques with recomputation and redefinition [17] strategies. For certain types of view synchronization, our view maintenance techniques perform four times better for both SPJ and aggregation views.

In summary, the contributions of this current chapter are: (1) we introduce a new view adaptation problem, namely *view maintenance after view synchronization* and we distinguish it from the view redefinition problem treated in [37, 17] demonstrating that our techniques succeed in cases when those previous techniques would fail; (2) we present view maintenance 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; and (3) we perform a systematic experimental evaluation of our proposed algorithms using the TPC-D benchmark, including a comparison of our techniques with recomputation and redefinition strategies. For certain types of view synchronization, our view maintenance techniques perform 400% faster for both SPJ and aggregation views. Hence, a view maintainer can achieve significant performance improvements when incorporating our proposed algorithms into its repertoire of maintenance strategies.

The remainder of this chapter is structured as follows. In Section 5.1 we discuss the dimensions that characterize the view maintenance after synchronization problem and define the subproblems for which we give solutions in this chapter. Section 5.2 gives a detailed example of mapping view maintenance after view synchronization problem into a maintenance-after-redefinition problem [37, 17]. In Section 5.3 we introduce solution strategies for view maintenance after synchronization and compare our technique with existing view redefinition strategies. Then, we present experimental results in Section 5.4 by comparing our techniques with both recomputation as well as the more recent redefinition strategies. Section 5.5 concludes the chapter.

The work presented in this chapter has been published by the author and her adviser in the paper [50] and also can be found in [49].

5.1 Problem Space of Materialized View Maintenance After Synchronization

After the view synchronization process, the view, if materialized, must be updated accordingly. In the reminder of this thesis we coin this problem the *materialized view maintenance after view synchronization*, or *maint-synch* for short. In Chapter I (Figure 1.3) we have proposed a taxonomy of the view adaptation problems in which we classified this new problem as a view adaptation problem. The synchronization process is triggered by capability changes in the underlying information sources and has the goal of *adapting* view definitions affected by these changes.

In the following, we describe the dimensions that characterize *maint-synch* problem space depicted in Figure 5.1.

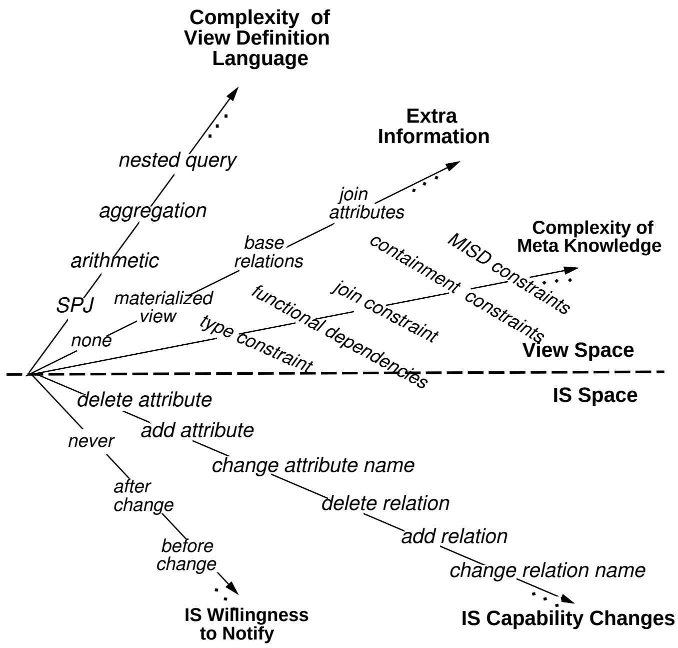


Figure 5.1: Problem Space of View Maintenance after View Synchronization.

Complexity of View Definition Language dimension: This dimension identifies the types of the view components allowed in the view definitions. The simplest view definition is specified by using the conventional SELECT-PROJECT-JOIN structure with WHERE clause containing conjunctive clauses only, denoted by SPJ in Figure 5.1. More complex views would allow aggregation, negation, nested queries, and so on in view definitions. If more complex language constructs are used as part of a view definition, then it is more likely that sophisticated algorithms are required to preserve the view extent after synchronization.

Extra Information dimension: This dimension is concerned with the issue of what information besides the old and new views is available for the purpose of facilitating view maintenance in general (and after view synchronization in particular). The amount of information available for view maintenance can range from *fully duplicated* base relations, *partially duplicated* base relations (such as preserving join attributes, computing derive-counts) to *no data* from base relations being available at the view site.

Complexity of Meta Knowledge dimension: This corresponds to the description of the capabilities and content of each IS plus of relationships between ISs. Such capabilities could describe query templates for a certain IS, schema definition, or functional dependencies.

One example of meta knowledge is that an identical duplicate of a relation R at IS_1 may also be available in another site IS_2, i.e., "$\mathrm{IS}_1.R \equiv \mathrm{IS}_2.S$". When the relation R is deleted from IS_1, the view synchronization process is able to preserve the affected view definitions that reference R by replacing $\mathrm{IS}_1.R$ with $\mathrm{IS}_2.S$ without doing any maintenance as the old and new views are equivalent. The more knowledge our system has about the information space, the higher chance it has to efficiently maintain the view extents with the minimum cost.

IS Capability Changes dimension: This corresponds to capability changes that need to be supported such as *delete attribute*, *delete relation*, or more complex changes.

IS Willingness to Notify dimension: This dimension considers whether, and if so, when ISs notify the view site of their changes. Choices here are that the IS does *not notify*, *notifies only after*, or *notifies the view site even before* the change actually takes place. A more optimal re-organization may be able to take place at a view site if it gets informed of a potential change (especially a deletion) before a change takes effect. For example, advance notification could trigger copying the affected data over to the view site, or utilizing meta knowledge about the to-be-deleted IS component to identify alternate sources for information supply.

Incremental view maintenance strategies proposed in this chapter for efficient update of the extent of the affected views address the problem that corresponds to the following coordinates in Figure 5.1:

- on the dimension **Complexity of Meta Knowledge**, the value considered is *containment constraints*;
- on the dimension **Complexity of View Definition Language**, the values considered are *SPJ* and *aggregation*;
- on the dimension **IS Willingness to Notify**, the values considered are *after change* and *before change*[1].
- on the dimension **Extra Information** the values considered are *materialized view*, and *base relations*. The assumption is that the base relations used in a view definition are available at their sites for querying (as long as their schema is advertise at the datawarehouse site). When a base relation is deleted (e.g., after a "*delete-relation* " capability change), the dimension **IS Willingness to Notify** coordinates determines if that base relation is still available during the maintenance process. The view extents are available at the datawarehouse site (i.e., the views are materialized). These assumptions are similar to assumptions made in [37, 17] (and as a matter of fact, those are general assumptions for most view maintenance techniques studied in the literature).
- on the dimension **IS Capability Changes**, the values are *delete-attribute* and *delete-relation*[2].

[1]The value *never* on this coordinate means that the IS would not announce any changes, thus some type of pooling need to be performed for detecting changes. Then, this situation is equivalent to *after change* notification.

[2]Note that these two capability changes are the only ones requiring extent update after synchronization.

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

Example 32 *In the rest of this chapter we use as working example the view* ***Customer-Passengers*** *defined to select all customers that fly in "Asia" and are younger than 20. The information space descriptions are given in a previous chapter in Section 2.2, Table 2.1. The E-SQL view definition for* ***Customer-Passengers*** *is given in Equation (5.1)* [3].

$$
\begin{array}{ll}
\text{CREATE VIEW} & \textbf{\textit{Customer-Passengers}}\ (\mathcal{VE} = \delta)\ \text{AS} \\
\text{SELECT} & \textbf{\textit{C.Name}}\ (\mathcal{AR} = true),\ \textbf{\textit{C.Age}}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\
 & \textbf{\textit{C.Phone}}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\
\text{FROM} & \textbf{\textit{Customer C}}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
 & \textbf{\textit{FlightRes F}}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
\text{WHERE} & (\textbf{\textit{C.Name}} = \textbf{\textit{F.PName}})\ (\mathcal{CR} = true)\ \text{AND}\ (\textbf{\textit{F.Dest}} = \textbf{\textit{'Asia'}})\ \text{AND} \\
 & (\ \textbf{\textit{C.Age}} < 20)(\mathcal{CR} = true)
\end{array}
\tag{5.1}
$$

We assume the capability change "delete-relation ***Customer****". In the MKB a $\mathcal{PC}$-constraint is defined between the relations* ***Customer*** *and* ***Accidental–Ins*** *shown in Equation (5.2).*

$$
\begin{aligned}
\mathcal{PC}_{Accident-Ins,Customer} \ &= \ (\pi_{Accident-Ins.Holder,Accident-Ins.Age}(\textbf{\textit{Accident–Ins}}) \\
&\subseteq \pi_{Customer.Name,Customer.Age}(\ \textbf{\textit{Customer}}))
\end{aligned}
\tag{5.2}
$$

The elements of the view ***Customer–Passengers*** *(as defined in Section 2.5, Table 2.5) relative to the relation* ***Customer*** *are given in Figure 5.2.*

The relation ***Accident–Ins*** *has all the properties required by POC algorithm (Section 3.4) Conditions 1 to 4 falling into Case 1 in Condition 3: ϕ in our $\mathcal{PC}$-constraint*

[3]The evolution parameters not specified have the default values as defined in Table 2.6 in Chapter II.

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 5.2: Elements of the ***Customer-Passengers*** *view relative to the relation* ***Customer.***

from Equation (5.2) is $\subseteq$*, and all attributes from* $Attr(V(R))_{WHERE}$*have replacements in* ***Accident–Ins*** *(Figure 5.2). Based on Theorem 5 (Section 3.4), we can conclude that the view-extent evolution parameter value for the view* ***Customer–Passengers*** *must be* $\mathcal{VE} = \subseteq$ *(thus* δ *in Equations (5.1) and (5.3) must be* $\subseteq$*). Finally, applying the POC algorithm we obtain the legal view rewriting* ***Customer–Passengers'*** *defined in Equation (5.3) (the replaced components are underlined).*

$$
\begin{array}{ll}
\text{CREATE VIEW} & \textbf{\textit{Customer-Passengers'}}\ (\mathcal{VE} = \subseteq)\ \text{AS} \\
\text{SELECT} & \underline{\textbf{\textit{A.Holder}}\ (\mathcal{AR} = true),\ \textbf{\textit{A.Age}}\ (\mathcal{AD} = true, \mathcal{AR} = true)} \\
\text{FROM} & \underline{\textbf{\textit{Accident–Ins A}}\ (\mathcal{RD} = true, \mathcal{RR} = true)}, \\
 & \textbf{\textit{FlightRes F}}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
\text{WHERE} & \underline{(\textbf{\textit{A.Holder = F.PName}})\ (\mathcal{CR} = true)}\ \text{AND} (\textbf{\textit{F.Dest = 'Asia'}}) \\
 & \underline{\text{AND} (\textbf{\textit{A.Age}} < 20)(\mathcal{CR} = true)}
\end{array}
\tag{5.3}
$$

Let's assume that we could apply the techniques for view maintenance after redefinition [17, 37] 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. [17] 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. (3.24)) 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.

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.

[4]In [17], 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.

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 33).

Example 33 *Let's now apply the steps S1 through S6 listed above for Example 32. Assume that the cardinalities of the relations are* |***Customer***| = 40000*,* |***FlightRes***| = 20000*,* |***Accident−Ins***| = 30000 *and* |***Customer-Passengers***| = 1000*. In Table 5.1, 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* $\boldsymbol{C.Age} < 20$	*8000*	*40000*	*10000*	*0*
S2	*Drop* $\boldsymbol{C.Name} = \boldsymbol{F.PName}$	*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* $\boldsymbol{A.Age} < 20$	*6000*	*0*	*0*	*6000*
S6	*Add* $\boldsymbol{A.Holder} = \boldsymbol{F.PName}$	*400*	*0*	*10000*	*0*

Table 5.1: *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 [17] 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 [17] assumes that nothing changes at the IS site.

Example 34 *Consider the step S1 in Example 33 that has as goal to drop the local condition (*$\boldsymbol{C.Age} < 20$*) 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 [17] the new view after this one local change would be computed as* $\boldsymbol{Customer\text{-}Passengers}^{S1}$ $= \boldsymbol{Customer\text{-}Passengers} \cup \boldsymbol{Customer\text{-}Passengers}^{+}$*, where* $\boldsymbol{Customer\text{-}Passengers}^{+}$ *is defined in Eq. (5.4):*

$$\begin{array}{ll} & \textbf{\textit{Customer-Passengers}}^{+} = \\ (\text{SELECT} & \textbf{\textit{C.Name, C.Age}}, \textbf{\textit{C.Phone}} \\ \text{FROM} & \textbf{\textit{Customer C , FlightRes F}} \\ \text{WHERE} & \textbf{\textit{(C.Name = F.PName)}} \text{ AND } \textbf{\textit{(F.Dest = 'Asia')}} \text{ AND} \\ & (NOT\ (\textbf{\textit{C.Age}} < 20))) \end{array} \tag{5.4}$$

Note that the expression ***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.*

5.3 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 5.2.

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. 5.3 and 5.4, respectively. The rest of the cases summarized in Figs. 5.5 and 5.6 are similar and hence are not further discussed.

	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 5.2: Cases of Maintenance after View Synchronization.

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. (3.10));

(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. (3.5) 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 5.2).

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. (3.5): $\mathcal{PC}_{S,R} = (\pi_{\bar{B}}(S)\ \phi\ \pi_{\bar{A}}(R))$. 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 [16, 6, 2, 53] 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 5.3.1, 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 5.3.2, 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.

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

Cases 1 through 8 are summarized in Figs. 5.3 and 5.4. 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.

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 3.4.1 is $\pi_{S.\bar{B}}(S) \phi \pi_{R.\bar{A}}(R)$ (Eq. (3.5). ϕ has to be either $\equiv$ or $\subseteq$ as given in Eq. (3.40). 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. (3.24) and (3.10), respectively. The proof can be found in Chapter III 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. (3.5)) 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. (5.5).

$$V' = \pi_{(Attr(V(R))_{\text{SELECT}}(f,t),(Attr(V(R))_{\text{SELECT}}(t,t) \cap R.\bar{A}), R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n)}(V) \tag{5.5}$$

#	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 $\mathcal{CV}(Attr(V(R))_{\text{WHERE}}^{\text{LOCAL}}))$ EXCEPT (SELECT $S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM S WHERE $\mathcal{CV}(S(Attr(V(R))_{\text{WHERE}}^{\text{LOCAL}})))$ $\boxed{2.}$ $\Delta V_-^0 =$ (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 $\mathcal{CV}(Attr(V(R))_{\text{WHERE}}^{\text{JOIN}}, \bar{W}))$ $\boxed{3.}$ $V_-^0 = V^0 \setminus \Delta V_-^0$ $\boxed{4.}$ 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 $\mathcal{CV}(S(Attr(V(R))_{\text{WHERE}}^{\text{LOCAL}})))$ EXCEPT (SELECT $R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})$ FROM R WHERE $\mathcal{CV}(Attr(V(R))_{\text{WHERE}}^{\text{LOCAL}})$ $\boxed{6.}$ $\Delta V_+^0 =$ (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 $\mathcal{CV}(Attr(V(R))_{\text{WHERE}}^{\text{JOIN}}, \bar{W}))$ $\boxed{7.}$ $V_+^0 = V^0 \cup \Delta V_+^0$ $\boxed{8.}$ 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 5.3: SYNCMAB Strategy: SPJ View Maintenance after View Synchronization (Cases 1-4 as Defined in Table 5.2).

#	Parameters	Maintenance Strategy
5.1 6 7.1	$\mathcal{VE} = \equiv \mid \subseteq \mid \supseteq$ $V' \equiv_{\pi_{gr}} V$ $R\ avail.$ $\phi\ = \equiv$ $\pi_{S.\bar{B}}(S) =$ $\pi_{R.\bar{A}}(R)$	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	$\mathcal{VE}\ = \subseteq$ $V' \subseteq_{\pi_{gr}} V$ $R\ avail.$ $\phi\ = \subseteq$ $\pi_{S.\bar{B}}(S) \subseteq$ $\pi_{R.\bar{A}}(R)$	$\boxed{0.}$ 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^-" [53] $\boxed{4.}$ INSERT INTO V' (SELECT $*$ FROM V^0)
7.2	$\mathcal{VE}\ = \supseteq$ $V' \supseteq_{\pi_{gr}} V$ $R\ avail.$ $\phi\ = \supseteq$ $\pi_{S.\bar{B}}(S)$ $\supseteq \pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{5.}$ $\Delta R^+ =$ (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^+" [53] $\boxed{8.}$ INSERT INTO V' (SELECT $*$ FROM V^0)
8	$\mathcal{VE}\ = \approx$ $V' \approx_{\pi} V$ $R\ avail.$ $\phi\ = \approx$ $\pi_{S.\bar{B}}(S) \approx$ $\pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{1.}$, $\boxed{5.}$, $\boxed{8.}$ Update the AGGR-SPJ view V^0 after base relation data update "*add-tuples* ΔR^+" AND "*delete-tuples* ΔR^-" [53] $\boxed{10.}$ INSERT INTO V' (SELECT $*$ FROM V^0)

Figure 5.4: SYNCMAB Strategy: AGGR-SPJ View Maintenance after View Synchronization (Cases 5-8 as Defined in Table 5.2).

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. (5.6).

$$\begin{array}{ll} \text{INSERT INTO } V' & \\ \quad (\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (Attr(V(R))_{\text{SELECT}}(t,t) \cap R.\bar{A}), \\ & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\ \quad \text{FROM} & V) \end{array} \tag{5.6}$$

<u>SYNCMAB - Case 1.2: $\phi = \subseteq$.</u> 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. (3.5) 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. (5.7).

$$\begin{array}{ll} \text{CREATE VIEW} & V^0 \text{ AS} \\ \text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)) \\ & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\ \text{FROM} & R, R_1, \ldots, R_n \\ \text{WHERE} & \mathcal{CV}(R.\bar{A} \cap Attr(V(R))_{\text{WHERE}}(\_,t), \bar{W}) \end{array} \tag{5.7}$$

V^0 is defined in Eq. (5.7) and is initialized as defined in Eq. (5.8). 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. (5.7)) is different than the WHERE clause of

[5]The full proof is given in Chapter III.

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.

$$\begin{array}{ll} \text{INSERT INTO } V^0 & \\ (\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)), \\ & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\ \text{FROM} & V) \end{array} \tag{5.8}$$

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. (5.9) and (5.10) in algebraic and SQL notation, respectively:

$$\begin{aligned} \Delta R^- &= \pi_{R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})}(\sigma_{\mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})}(R)) \\ &\setminus \pi_{S(R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}))}(\sigma_{\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))}(S)) \end{aligned} \tag{5.9}$$

$$\begin{array}{ll} \Delta R^- = & \\ (\text{SELECT} & R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}) \\ \text{FROM} & R \\ \text{WHERE} & \mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})) \\ \text{EXCEPT} & \\ (\text{SELECT} & S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}) \\ \text{FROM} & S \\ \text{WHERE} & \mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))) \end{array} \tag{5.10}$$

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 3.4.1). Then, V^0_- is defined as in Eq. (5.11).

$$
\begin{array}{ll}
\textsf{CREATE VIEW} & V^0_- \ \textsf{AS} \\
\textsf{SELECT} & Attr(V(R))_{\textsf{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\textsf{SELECT}}(t,t)) \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
\textsf{FROM} & R^-, R_1, \ldots, R_n \\
\textsf{WHERE} & \mathcal{CV}(R.\bar{A} \cap Attr(V(R))_{\textsf{WHERE}}(\_, t), \bar{W})
\end{array}
\tag{5.11}
$$

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 [16, 6, 2]. For example, one of the techniques [6] is to first compute relation ΔV^0_- (defined in Eq. (5.12)) to consist of tuples of V^0 generated by the "deleted" tuples of ΔR^-, and then to compute the rest of the tuples as $V^0_- = V^0 \setminus \Delta V^0_-$.

$$
\begin{array}{rl}
\Delta V^0_- = & \\
(\textsf{SELECT} & Attr(V(R))_{\textsf{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\textsf{SELECT}}(t,t)), \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
\textsf{FROM} & \Delta R^-, R_1, \ldots, R_n \\
\textsf{WHERE} & \mathcal{CV}(Attr(V(R))^{\mathrm{JOIN}}_{\textsf{WHERE}}, \bar{W}))
\end{array}
\tag{5.12}
$$

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. (5.13).

$$\begin{aligned}
&\text{INSERT INTO } V' \\
&\quad (\text{SELECT} \quad Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)), \\
&\qquad\qquad\quad R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
&\qquad\ \ \text{FROM} \quad V^0_-)
\end{aligned} \tag{5.13}$$

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 3.4.1.

SYNCMAB - Case 2: SPJ views, $\mathcal{VE} = \equiv$, relation R is available. 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. 5.3.

SYNCMAB - Case 3: SPJ views, $\mathcal{VE} = \supseteq$, relation R is available. 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. (3.5)) must be $\equiv$ or $\supseteq$ as imposed by Condition 4, Eq. (3.40). Thus we have to discuss the two cases of $\phi = \equiv$ and $\phi = \supseteq$:

SYNCMAB - Case 3.1: $\phi = \equiv$. 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. (5.6) (see also Fig. 5.3).

SYNCMAB - Case 3.2: $\phi = \supseteq$. We proved in Chapter III 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. (3.5). Let the view V^0 be obtained as above (Eq. (5.7))

[6]The full proof is given in Chapter III.

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. (5.14) and (5.15), respectively in relational algebra and in SQL notation:

$$\begin{aligned}\Delta R^+ = {} & \pi_{S(R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}))}(\sigma_{\mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))}(S)) \\ & \setminus \pi_{R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}})}(\sigma_{\mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})}(R))\end{aligned} \quad (5.14)$$

$$\begin{array}{rl} \Delta R^+ = & \\ (\text{SELECT} & S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}) \\ \text{FROM} & S \\ \text{WHERE} & \mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))) \\ \text{EXCEPT} & \\ (\text{SELECT} & R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_,t) \cup Attr(V(R))_{\text{WHERE}}) \\ \text{FROM} & R \\ \text{WHERE} & \mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})) \end{array} \quad (5.15)$$

Note that in Case 3.2, the set ΔR^- as defined in Eq. (5.10) is empty. And vice versa, in Case 1.2, the set ΔR^+ defined by Eq. (5.15) 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. (5.16)) 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 3.4.1 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 $Attr(V(R))^{\text{JOIN}}_{\text{WHERE}}$.

$$
\begin{array}{ll}
\text{CREATE VIEW} & V^0_+ \text{ AS} \\
\text{SELECT} & Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)) \\
 & R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
\text{FROM} & R^+, R_1, \ldots, R_n \\
\text{WHERE} & \mathcal{CV}(R.\bar{A} \cap Attr(V(R))_{\text{WHERE}}(\_, t), \bar{W})
\end{array}
\tag{5.16}
$$

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 [16, 6, 2]. To give an example, we can compute $V^0_+ = V^0 \cup \Delta V^0_+$ [6], where ΔV^0_+ (Eq. (5.17)) 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 [6].

$$
\begin{aligned}
&\Delta V_{+}^{0} = \\
&\quad (\text{SELECT} \quad Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)), \\
&\qquad\qquad R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
&\qquad \text{FROM} \quad \Delta R^{+}, R_1, \ldots, R_n \\
&\qquad \text{WHERE} \quad \mathcal{CV}(Attr(V(R))_{\text{WHERE}}^{\text{JOIN}}, \bar{W}))
\end{aligned}
\tag{5.17}
$$

After V_{+}^{0} is computed, we then can use it to (locally) set the view V' using the maintenance strategy given by Eq. (5.18).

$$
\begin{aligned}
&\text{INSERT INTO } V' \\
&\quad (\text{SELECT} \quad Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)), \\
&\qquad\qquad R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
&\qquad \text{FROM} \quad V_{+}^{0})
\end{aligned}
\tag{5.18}
$$

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 $\mathcal{PC}_{S,R} = (\pi_{S.\bar{B}}(S) \supseteq \pi_{R.\bar{A}}(R))$.

<u>SYNCMAB - Case 4: SPJ views, $\mathcal{VE} = \approx$, relation R is available.</u> 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. (5.12) and (5.17), respectively. If we define $V_{-,+}^{0} = V^{0} \setminus V_{-}^{0} \cup V_{+}^{0}$, then the view V' is computed as expressed in Eq. (5.19) (see Fig. 5.3):

$$\begin{aligned}
&\text{INSERT INTO } V' \\
&\quad (\text{SELECT} \quad Attr(V(R))_{\text{SELECT}}(f,t), (R.\bar{A} \cap Attr(V(R))_{\text{SELECT}}(t,t)), \\
&\qquad\qquad\quad R_1.\bar{D}_1, \ldots, R_n.\bar{D}_n \\
&\quad\quad \text{FROM} \quad V^0_{-,+})
\end{aligned} \tag{5.19}$$

Example 35 *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. (5.20) and the $\mathcal{PC}$-constraint defined in Eq. (5.21). The capability change is "delete-relation* ***Customer****". A legal rewriting found by applying the POC algorithm is the view* ***Customer-East'*** *(Eq. (5.22)), 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).*

$$\begin{aligned}
&\textit{CREATE VIEW} \quad \textbf{\textit{Customer-East}}\ (\mathcal{VE} =\approx)\ \textit{AS} \\
&\textit{SELECT} \quad \textbf{\textit{C.Name}}\ (\mathcal{AR} = true),\ \textbf{\textit{C.Age}}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\
&\qquad\qquad \textbf{\textit{C.Phone}}\ (\mathcal{AD} = true, \mathcal{AR} = true) \\
&\textit{FROM} \quad \textbf{\textit{Customer C}}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
&\qquad\qquad \textbf{\textit{FlightRes F}}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
&\textit{WHERE} \quad (\textbf{\textit{C.Name}} = \textbf{\textit{F.PName}})\ (\mathcal{CR} = true)\ \textit{AND}\ (\textbf{\textit{F.Dest}} = \textbf{\textit{'Asia'}})\ \textit{AND} \\
&\qquad\qquad (\textbf{\textit{C.Age}} < 20)(\mathcal{CR} = true)\textit{AND} \\
&\qquad\qquad (\textbf{\textit{C.State}} = \text{``}MA'')(\mathcal{CD} = true, \mathcal{CR} = true)
\end{aligned} \tag{5.20}$$

$$\mathcal{PC}_{Customer,Accident-Ins} = (\pi_{Accident-Ins.Holder,Accident-Ins.Age}(\boldsymbol{Accident-Ins}) \approx \pi_{Customer.Name,Customer.Age}(\boldsymbol{Customer})) \tag{5.21}$$

$$
\begin{array}{ll}
\text{CREATE VIEW} & \boldsymbol{Customer\text{-}East'}\ (\mathcal{VE} = \approx)\ \text{AS} \\
\text{SELECT} & \underline{\boldsymbol{A.Holder}\ (\mathcal{AR} = true),\ \boldsymbol{A.Age}\ (\mathcal{AD} = true, \mathcal{AR} = true)} \\
\text{FROM} & \underline{\boldsymbol{Accident-Ins\ A}\ (\mathcal{RD} = true, \mathcal{RR} = true)}, \\
 & \boldsymbol{FlightRes\ F}\ (\mathcal{RD} = true, \mathcal{RR} = true), \\
\text{WHERE} & \underline{(\boldsymbol{A.Holder} = \boldsymbol{F.PName})\ (\mathcal{CR} = true)}\ \text{AND}(\boldsymbol{F.Dest} = \text{'Asia'}) \\
 & \underline{\text{AND}(\boldsymbol{A.Age} < 20)(\mathcal{CR} = true)}
\end{array}
\tag{5.22}
$$

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 5.3: Data for the Relations Used in Example 35.

The steps in the SYNCMAB strategy (see Fig. 5.3) presented above are for this example the following:

$\boxed{0.}$ *The view* ***Customer-East***^{0} *is defined as in Eq. (5.23) and its extent is initialized with the set* $\{(John, 19), (Mary, 18)\}$ *given the extents of the relations* ***Customer*** *and* ***FlightRes*** *from Table 5.3. Note that this extent is not up-to-date given the definition of the view* ***Customer-East***^{0}.

$\boxed{1.}$ *The expression* $\Delta\boldsymbol{Customer}^-$ *is* $\{(John, 19)\}$ *(Fig. 5.3).*

$\boxed{5.}$ *The expression* $\Delta\boldsymbol{Customer}^+$ *computes* $\{(Gill, 16), (Bob, 17)\}$ *(Fig. 5.3). Note that the relation* $\Delta\boldsymbol{Customer}^+$ *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'****.*

$$\begin{array}{ll} \text{CREATE VIEW} & \textbf{\textit{Customer-East}}^0 \ \text{AS} \\ \text{SELECT} & \textbf{\textit{C.Name, C.Age}} \\ \text{FROM} & \textbf{\textit{Customer C, FlightRes F}} \\ \text{WHERE} & \textbf{\textit{(C.Name = F.PName)}} \ \text{AND} \ \textbf{\textit{(F.Dest = 'Asia')}} \text{AND} \ (\textbf{\textit{C.Age}} < 20) \end{array} \tag{5.23}$$

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. (5.22).*

5.3.2 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 2.5 using set semantics. We make the assumption that in any rewriting of the view V,

#	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 5.5: SYNCMAA Strategy: SPJ View Maintenance after View Synchronization (Cases 11-14 as Defined in Table 5.2).

#	Parameters	Maintenance Strategy
15.1 16 17.1	$\mathcal{VE} = \equiv \mid \subseteq \mid \supseteq$ $V' \equiv_{\pi_{gr}} V$ R *NOT avail.* $\phi\ = \equiv$ $\pi_{S.\bar{B}}(S) =$ $\pi_{R.\bar{A}}(R)$	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) $\forall f(X)(=F)\ in\ \mathcal{F} \exists f'(Y)\ in\ \mathcal{F}',\ s.t.\ (f(X), f'(Y)) \in$ $\{(MAX(X), MAX(F)), (MIN(X), MIN(F)), (SUM(X), SUM(F)),$ $(COUNT(*), SUM(F))(AVG(X), SUM(FxF')/SUM(F')) \mid F' = COUNT(*)\}$
15.2	$\mathcal{VE}\ = \subseteq$ $V' \subseteq_{\pi_{gr}} V$ R *NOT avail.* $\phi\ = \subseteq$ $\pi_{S.\bar{B}}(S) \subseteq$ $\pi_{R.\bar{A}}(R)$	$\boxed{0.}$ 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 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^-" [53] (delete whole groups) $\boxed{4.}$ INSERT INTO V' (SELECT $*$ FROM V^0)
17.2	$\mathcal{VE}\ = \supseteq$ $V' \supseteq_{\pi_{gr}} V$ R *NOT avail.* $\phi\ = \supseteq$ $\pi_{S.\bar{B}}(S) \supseteq$ $\pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{5.}$ $\Delta R^+ =$ (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^+" [53] (only new groups) $\boxed{8.}$ INSERT INTO V' (SELECT $*$ FROM V^0)
18	$\mathcal{VE}\ = \approx$ $V' \approx_{\pi} V$ R *NOT avail.* $\phi\ = \approx$ $\pi_{S.\bar{B}}(S) \approx$ $\pi_{R.\bar{A}}(R)$	$\boxed{0.}$, $\boxed{1.}$, $\boxed{5.}$, $\boxed{8.}$ Update the AGGR-SPJ view V^0 after base relation data update "*add-tuples* ΔR^+" AND "*delete-tuples* ΔR^-" [53] (add/delete whole groups only) $\boxed{10.}$ INSERT INTO V' (SELECT $*$ FROM V^0)

Figure 5.6: SYNCMAA Strategy: AGGR-SPJ View Maintenance after View Synchronization (Cases 15-18 as Defined in Table 5.2).

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. 5.5 summarizes the cases we present below.

<ins>SYNCMAA - Case 11: SPJ views, $\mathcal{VE} = \subseteq$, relation R is NOT available.</ins> This is the case when Condition 4 imposes that ϕ in Equation (3.5) is either $\equiv$ or $\subseteq$ (Eq. (3.40)). Thus we have the two cases of $\phi = \equiv$ and $\phi = \subseteq$:

<ins>SYNCMAA - Case 11.1: $\phi = \equiv$.</ins> 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. 5.5.

<ins>SYNCMAA - Case 11.2: $\phi = \subseteq$.</ins> 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 5.3.1, Eq. (5.7) and be initialized

as in Eq. (5.8). 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. (5.10) for ΔR^-. The new definition of the set-difference ΔR^- (given in Eq. (5.24)) computes the set of tuples from the view V that cannot be generated by any tuples in the relation S.

$$
\begin{array}{rl}
\Delta R^- = & \\
(\text{SELECT} & R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_, t) \cup Attr(V(R))_{\text{WHERE}}) \\
\text{FROM} & V) \\
\text{EXCEPT} & \\
(\text{SELECT} & S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_, t) \cup Attr(V(R))_{\text{WHERE}}) \\
\text{FROM} & S \\
\text{WHERE} & \mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})))
\end{array}
\tag{5.24}
$$

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 [16, 6, 2]. In Fig. 5.5 we give one of the techniques (same as in SYNCMAB strategy, Case 1.2).

<u>SYNCMAA - Case 12: SPJ views, $\mathcal{VE} = \equiv$, relation R is NOT available.</u> This case is treated exactly as Case 11.1 and is summarized in Fig. 5.5.

<u>SYNCMAA - Case 13: SPJ views, $\mathcal{VE} = \supseteq$, relation R is NOT available.</u> 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. (3.40). ϕ used in the $\mathcal{PC}$-constraint (Eq. (3.5)) must be $\equiv$ or $\supseteq$ which we treat separately in the two Cases 13.1 and 13.2 below.

<u>SYNCMAA - Case 13.1: $\phi = \equiv$.</u> In this case, we can prove that $V' =_\pi V$ (Chapter III), thus the appropriate maintenance equation is given in Fig. 5.5. This is the same as for the Cases 11.1 and 12.

<u>SYNCMAA - Case 13.2: $\phi = \supseteq$.</u> We proved in Chapter III 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. (5.25) computes the set of tuples in S that are potential candidates to generate new tuples in V'.

$$
\begin{aligned}
&\Delta R^+ = \\
&\quad (\text{SELECT} \quad S.\bar{B} \cap S(Attr(V(R))_{\text{SELECT}}(\_, t) \cup Attr(V(R))_{\text{WHERE}}) \\
&\quad\quad \text{FROM} \quad S \\
&\quad\quad \text{WHERE} \quad \mathcal{CV}(S(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}))) \\
&\quad\quad \text{EXCEPT} \\
&\quad (\text{SELECT} \quad R.\bar{A} \cap (Attr(V(R))_{\text{SELECT}}(\_, t) \cup Attr(V(R))_{\text{WHERE}}) \\
&\quad\quad \text{FROM} \quad V
\end{aligned}
\tag{5.25}
$$

If we propagate these "insertions" to the view V^0, the view extent obtained

[7] See Chapter III for the proof.

(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 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 [16, 6, 2]. 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. 5.5 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. 5.5, Case 14.

Example 36 *Let's consider again Example 35 from Section 5.3.1 and apply the SYNCMAA strategy defined above when the relation* ***Customer*** *is not available. The view* ***Customer-East*** *is defined in Eq. (5.20), while the view rewriting under the capability change "delete-relation* ***Customer****"(found by the POC algorithm is the view* ***Customer-East'*** *(Eq. (5.22)). 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. (5.21). We now go over the steps defined in Table 5.5 for this particular example.*

In step $\boxed{0.}$, the view ***Customer-East***^{0} *is defined as in Eq. (5.23) 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 35. Without using the deleted relation* ***Customer***, *the expression Δ**Customer**$^-$ (Eq. (5.26)) generates the tuple $\{(John, 19)\}$ (step $\boxed{1.}$), while the expression Δ**Customer**$^+$ (Eq. (5.27)) 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. (5.22).*

$$
\begin{array}{ll}
 & \Delta \textbf{\textit{Customer-East}}^{-} = \\
(\text{SELECT} & \textbf{\textit{Name, Age}} \\
\text{FROM} & \textbf{\textit{Customer-East}}) \\
\text{EXCEPT} & \\
(\text{SELECT} & \textbf{\textit{Holder, Age}} \\
\text{FROM} & \textbf{\textit{Accident-Ins}} \\
\text{WHERE} & (\textbf{\textit{Age}} < 20))
\end{array}
\tag{5.26}
$$

$$
\begin{array}{ll}
 & \Delta \textbf{\textit{Customer-East}}^{+} = \\
(\text{SELECT} & \textbf{\textit{Holder, Age}} \\
\text{FROM} & \textbf{\textit{Accident-Ins}} \\
\text{WHERE} & (\textbf{\textit{Age}} < 20)) \\
\text{EXCEPT} & \\
(\text{SELECT} & \textbf{\textit{Name, Age}} \\
\text{FROM} & \textbf{\textit{Customer-East}})
\end{array}
\tag{5.27}
$$

5.4 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.

5.4.1 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

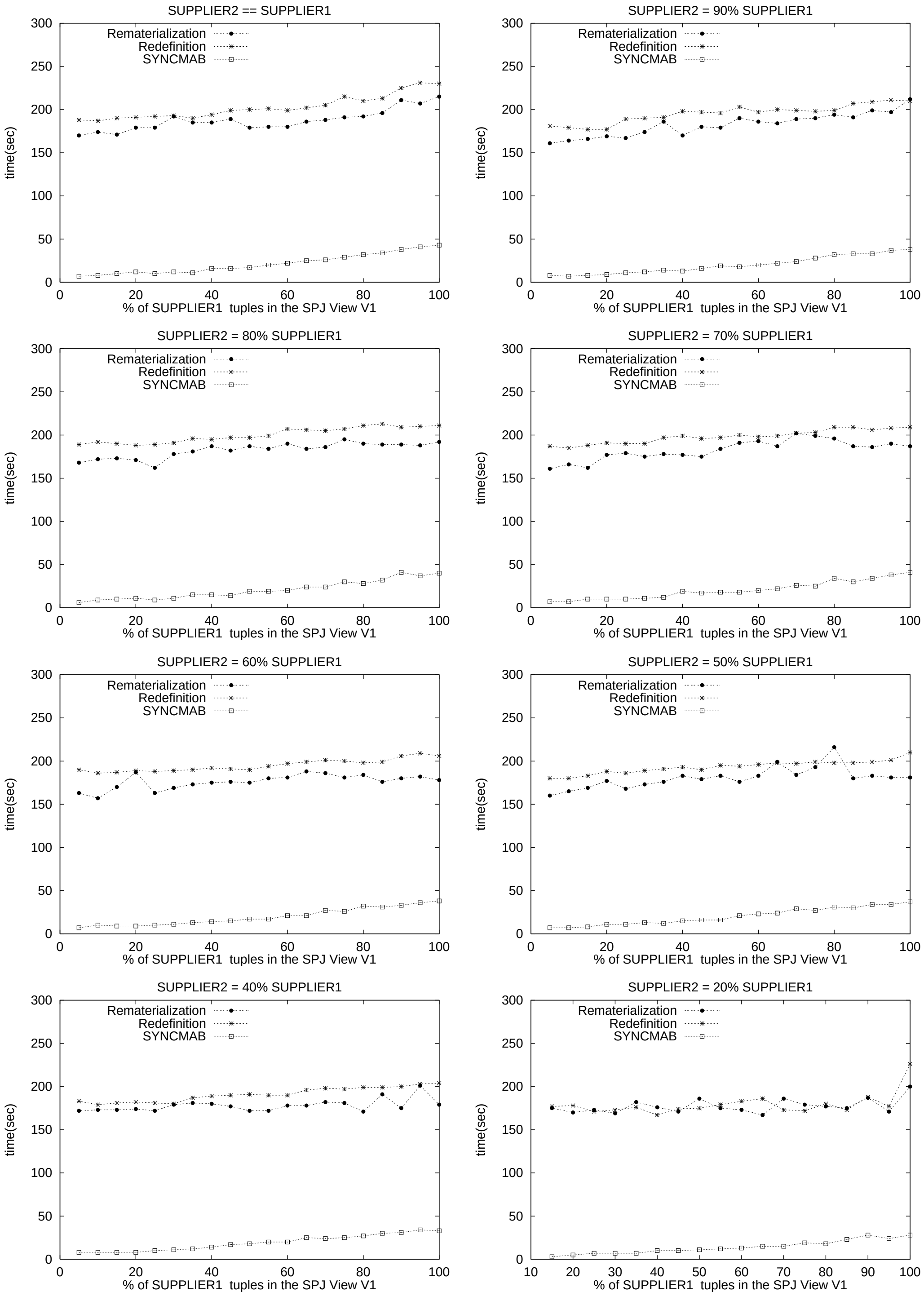


Figure 5.7: Case 1, for the SPJ view $V1$, the legal rewriting $V2$ when $V2 \subseteq V1$ using $\mathcal{PC}$-constraint SUPPLIER2 $\subseteq$ SUPPLIER1: the experimental results for maintenance after synchronization with *deletion* by varying the size of the SUPPLIER2 when SUPPLIER2 $\subseteq$ SUPPLIER1.

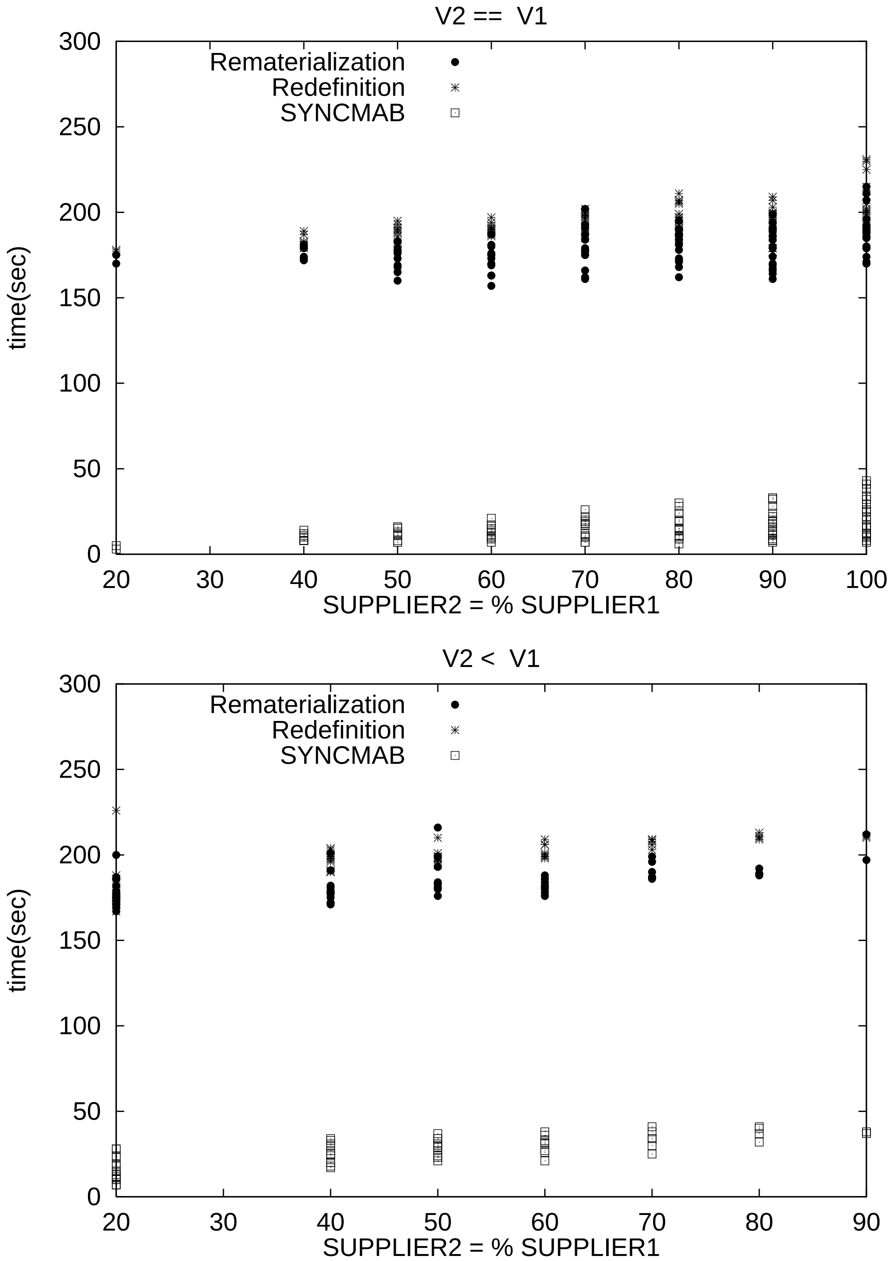


Figure 5.8: Case 1, for the SPJ view $V1$, the legal rewriting $V2$ when $V2 \subseteq V1$ using $\mathcal{PC}$-constraint SUPPLIER2 $\subseteq$ SUPPLIER1: the experimental results for maintenance after synchronization with *deletion* by varying the difference between $V2$ and $V1$: Cases 1.1 (upper chart) and 1.2 (bottom chart).

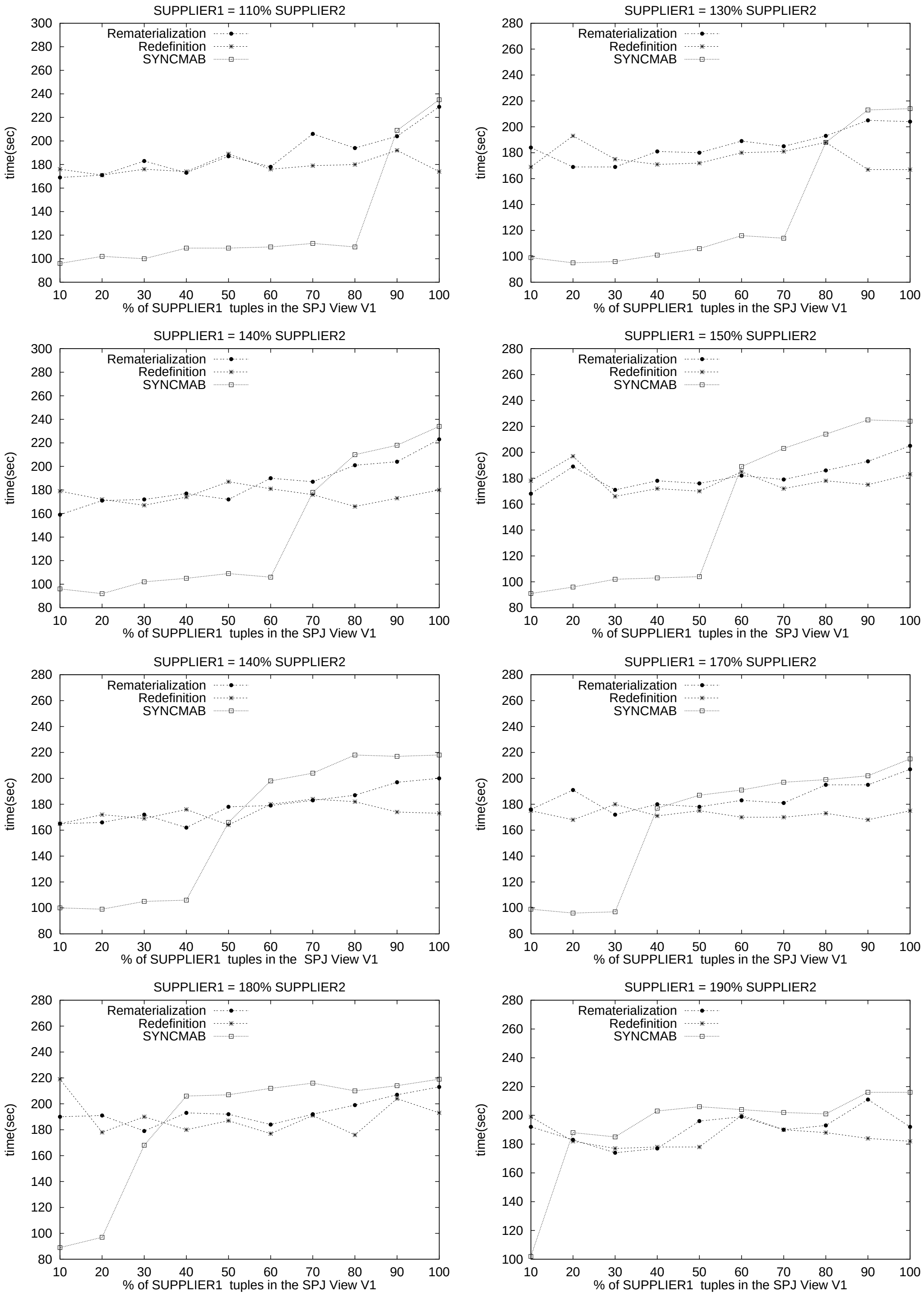
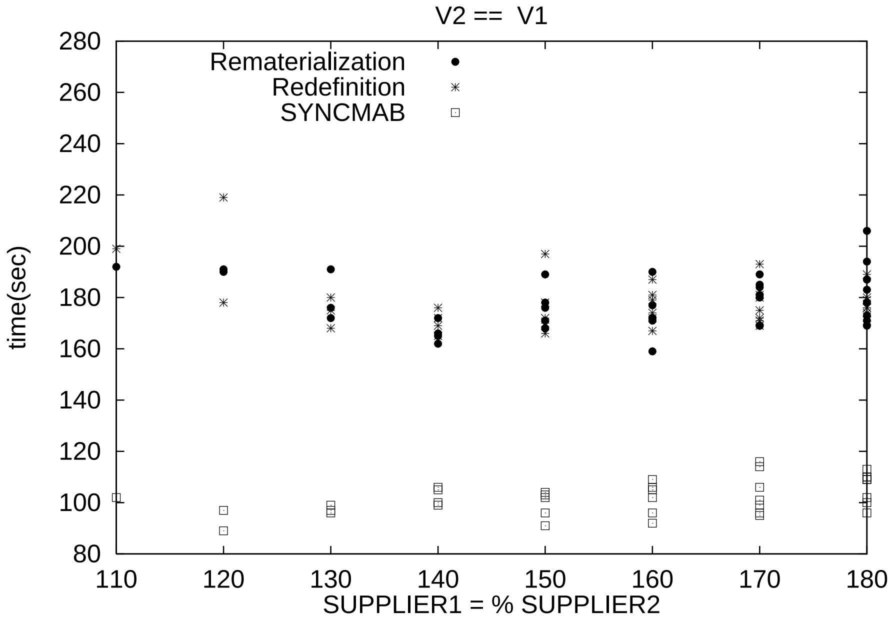


Figure 5.9: Case 3, for the SPJ view $V2$, the legal rewriting $V1$ when $V1 \supseteq V2$ using $\mathcal{PC}$-constraint SUPPLIER1 $\supseteq$ SUPPLIER2: the experimental results for maintenance after synchronization with *insertion* by varying the size of the SUPPLIER1 when SUPPLIER1 $\supseteq$ SUPPLIER2.



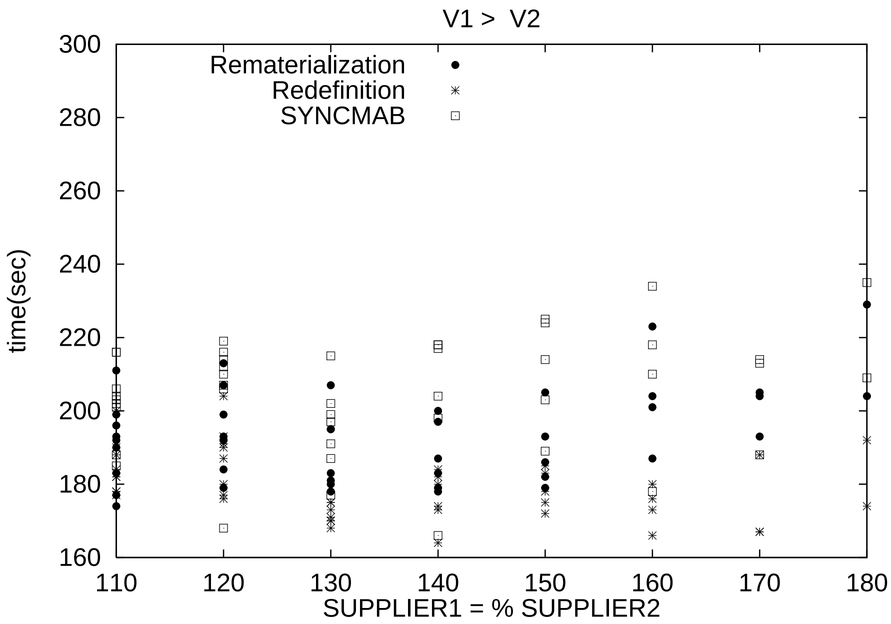


Figure 5.10: Case 3, for the SPJ view $V2$, the legal rewriting $V1$ when $V1 \supseteq V2$ using $\mathcal{PC}$-constraint SUPPLIER1 $\supseteq$ SUPPLIER2: the experimental results for maintenance after synchronization with *insertion* by varying the difference between $V1$ and $V2$: Cases 3.1 (upper chart) and 3.2 (bottom chart)

.

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 5.3, namely the SYNCMAB strategy 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);

(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.

5.4.2 Description of the TPC-D Benchmark Schema and Database

The sample database used for these experiments is conforming to the TPC-D benchmark [64]. The TPC-D benchmark database schema which we use unchanged is described in Figure 5.4.2. 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-)).

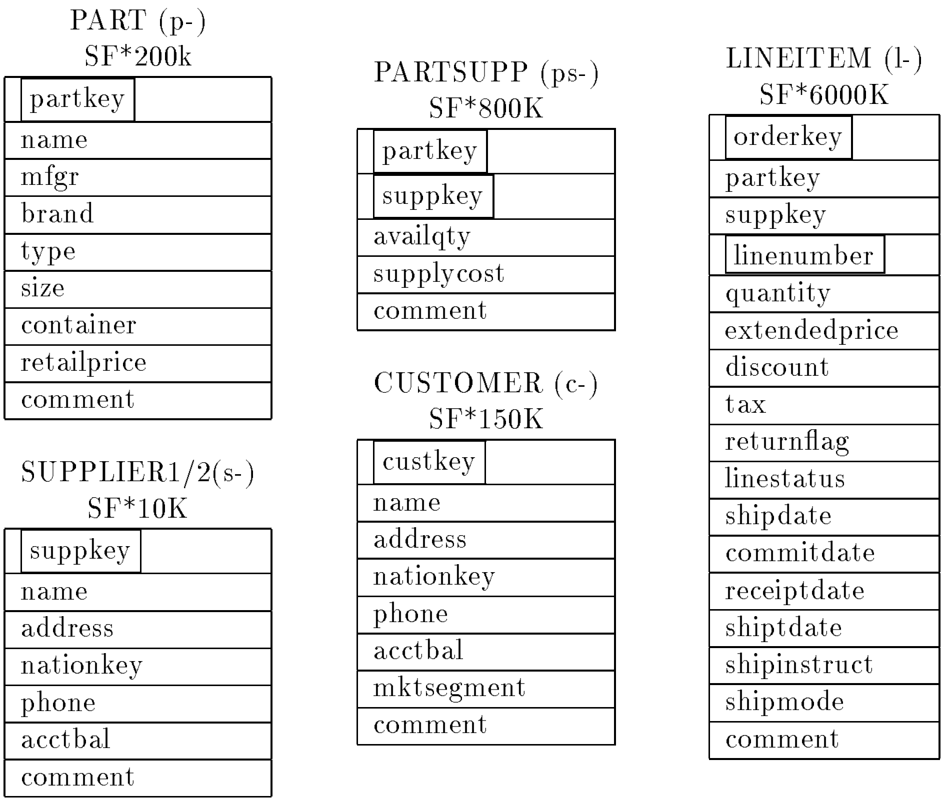


Figure 5.11: The TPC-D Benchmark Database Schema.

5.4.3 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. 5.3 and 5.4. Figs. 5.12 and 5.13 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 5.2 that need to be applied for maintenance. For all four view definitions, the 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. 5.4.2 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
```

$\xrightarrow{\text{Case 1}}$

$\xleftarrow{\text{Case 3}}$

```
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 5.12: Cases 1 and 3 defined in Table 5.2 for SPJ Views: View Definitions for $V1$ and $V2$.

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	$s\_suppkey, s\_name,$ $s\_address, s\_nationkey,$ $s\_acctbal, s\_comment,$ $ps\_availqty$

$\xrightarrow{\text{Case 5}}$ $\xleftarrow{\text{Case 7}}$

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 5.13: Cases 5 and 7 defined in Table 5.2 for Aggregation Views: View Definitions for $V3$ and $V4$.

We use the test cases depicted in Fig. 5.12 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 5.4.4 and 5.4.5.

For the views $V3$ and $V4$ shown in Fig. 5.13 that contain aggregate attributes, we measure the performance 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. 5.14 and 5.15 for Case 5, and 5.16 and 5.17 for Case 7.

5.4.4 Maintenance after Synchronization for Views with Deletion

The charts in Figs. 5.7 and 5.8 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. 5.3).

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. 5.7 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. 5.8 we have the same experimental results represented in graphs from Fig. 5.7, 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%.

5.4.5 Maintenance after Synchronization for Views with Insertion

The charts in Figs. 5.9 and 5.10 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. 5.9 and 5.10, the x-axis and y-axis are defined as explained above in Section 5.4.4.

Our SYNCMAB strategy must perform insertion under capability change "*delete-relation* SUPPLIER2" (see Fig. 5.3). 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. 5.9 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. 5.3). 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. 5.9).

In Fig. 5.10 we have the same experimental results represented in graphs from Fig. 5.9, 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.

5.4.6 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.

5.5 Summary of Contributions

To summarize, the main contributions of this chapter are:

- In this chapter work we have presented solutions for view maintenance after synchronization, and in particular demonstrate their effectiveness when applied to view rewritings obtained by one of the *EVE* view synchronization algorithms. The proposed strategies are however general enough to be used in other framework of complex (simultaneous) view redefinitions specified by the user and directly applied to the view definition.

- We describe a new view adaptation problem arising from view synchronization under capability changes, which we coin *view maintenance after view synchronization.*

- We motivate our proposed solution by distinguishing this new problem from the explicit view redefinition problem, and demonstrate that the existing techniques for view redefinition from the literature [17, 37] cannot in general be applied for view maintenance after view synchronization due to unavailability of the old data or schema (for example, after a "*delete-relation R*" capability change, the data of the relation R is not longer available).

- We propose a series of maintenance strategies after view synchronization based on different system parameters such as the availability of the base relations and the types of evolution parameters. The proposed solution regards changes to the view definition after synchronization as complex changes and handles them in one optimized batch strategy. We exploit knowledge of how the view

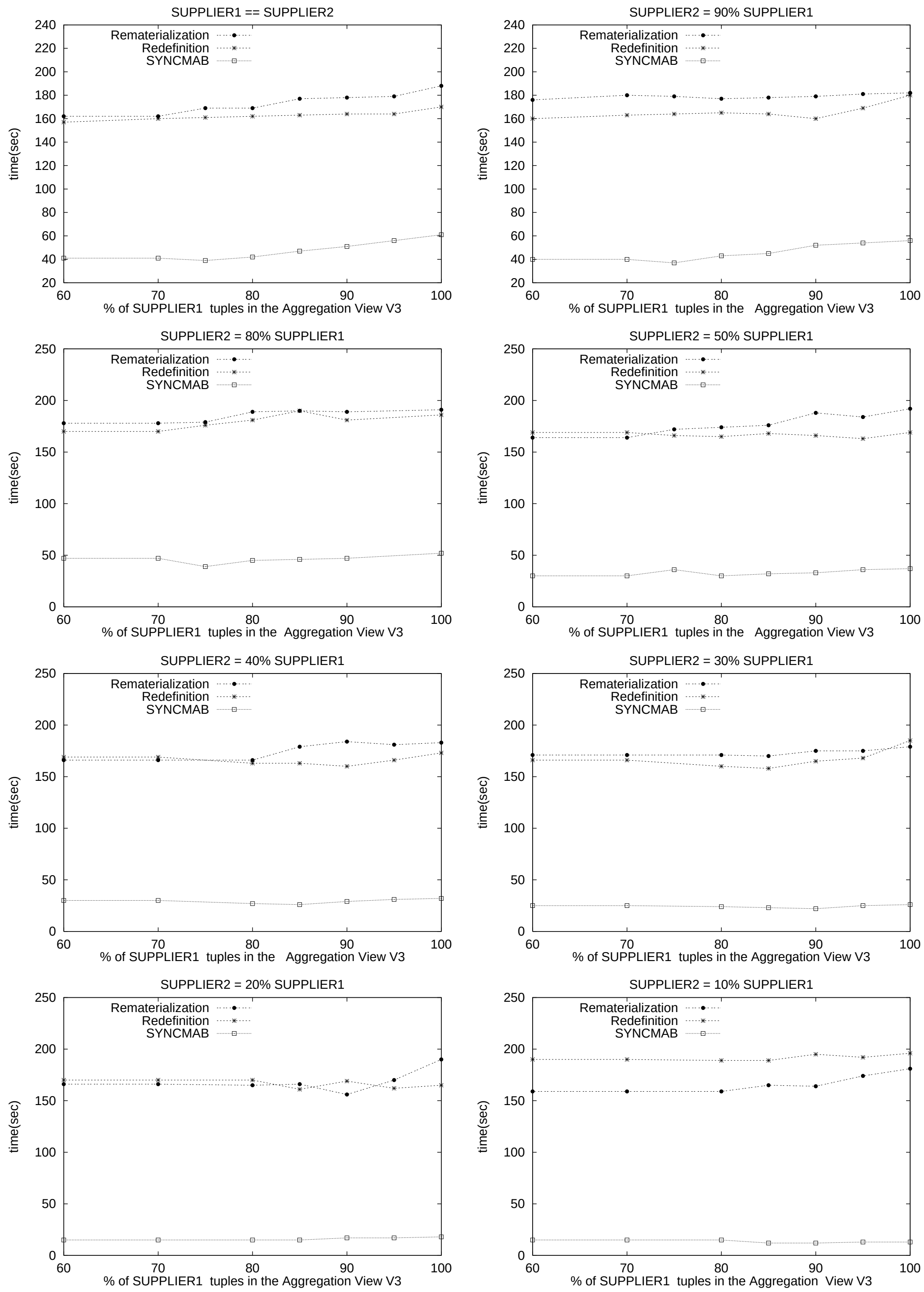


Figure 5.14: Case 5, for the AGGR-SPJ view $V3$, the legal rewriting $V4$ when $V4 \subseteq V3$ using $\mathcal{PC}$-constraint SUPPLIER2 $\subseteq$ SUPPLIER1: the experimental results for maintenance after synchronization with *deletion* by varying the size of the SUPPLIER2 when SUPPLIER2 $\subseteq$ SUPPLIER1.

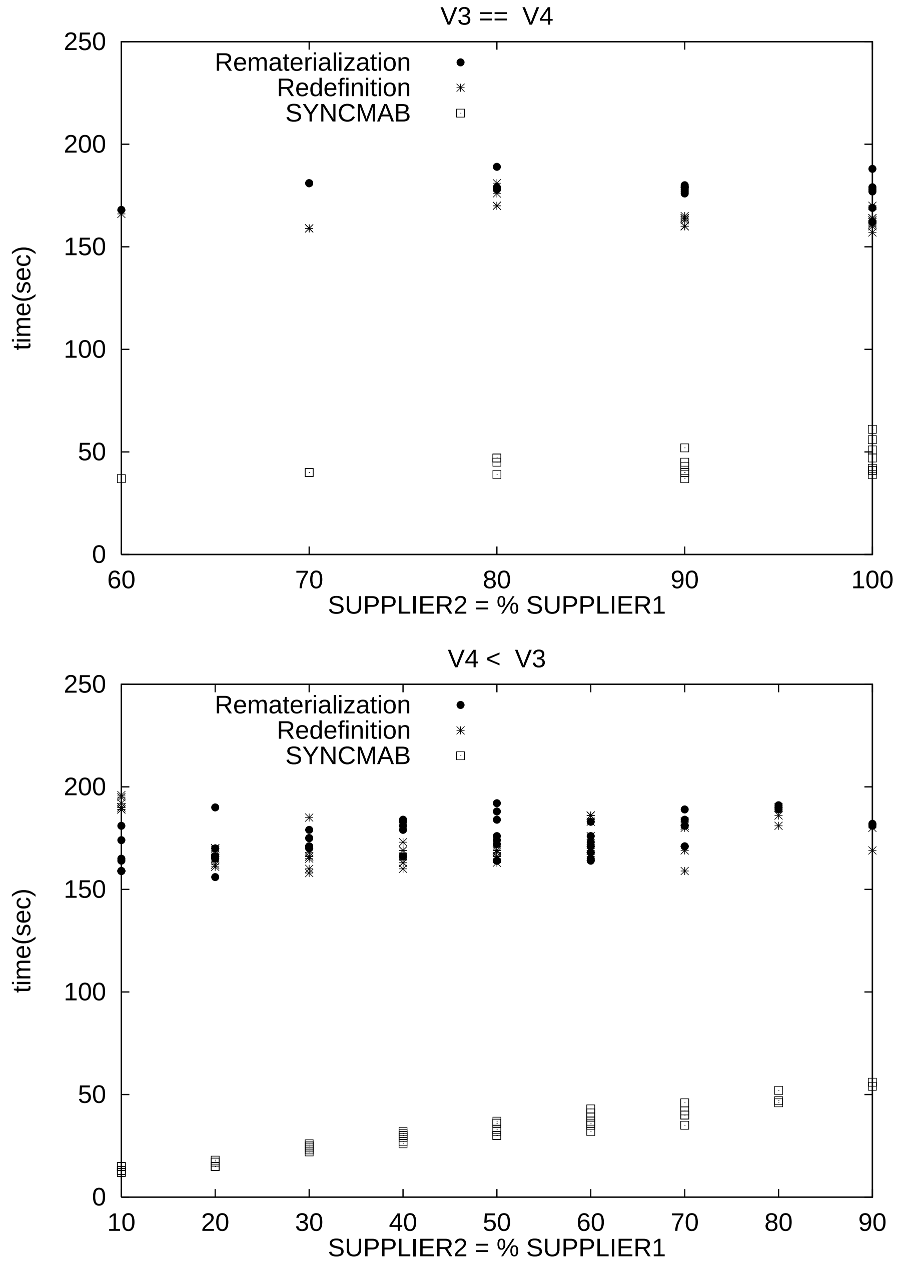


Figure 5.15: Case 5, for the AGGR-SPJ view $V3$, the legal rewriting $V4$ when $V4 \subseteq V3$ using $\mathcal{PC}$-constraint SUPPLIER2 $\subseteq$ SUPPLIER1: the experimental results for maintenance after synchronization with *deletion* by varying the difference between $V4$ and $V3$: Cases 5.1 (upper chart) and 5.2 (bottom chart).

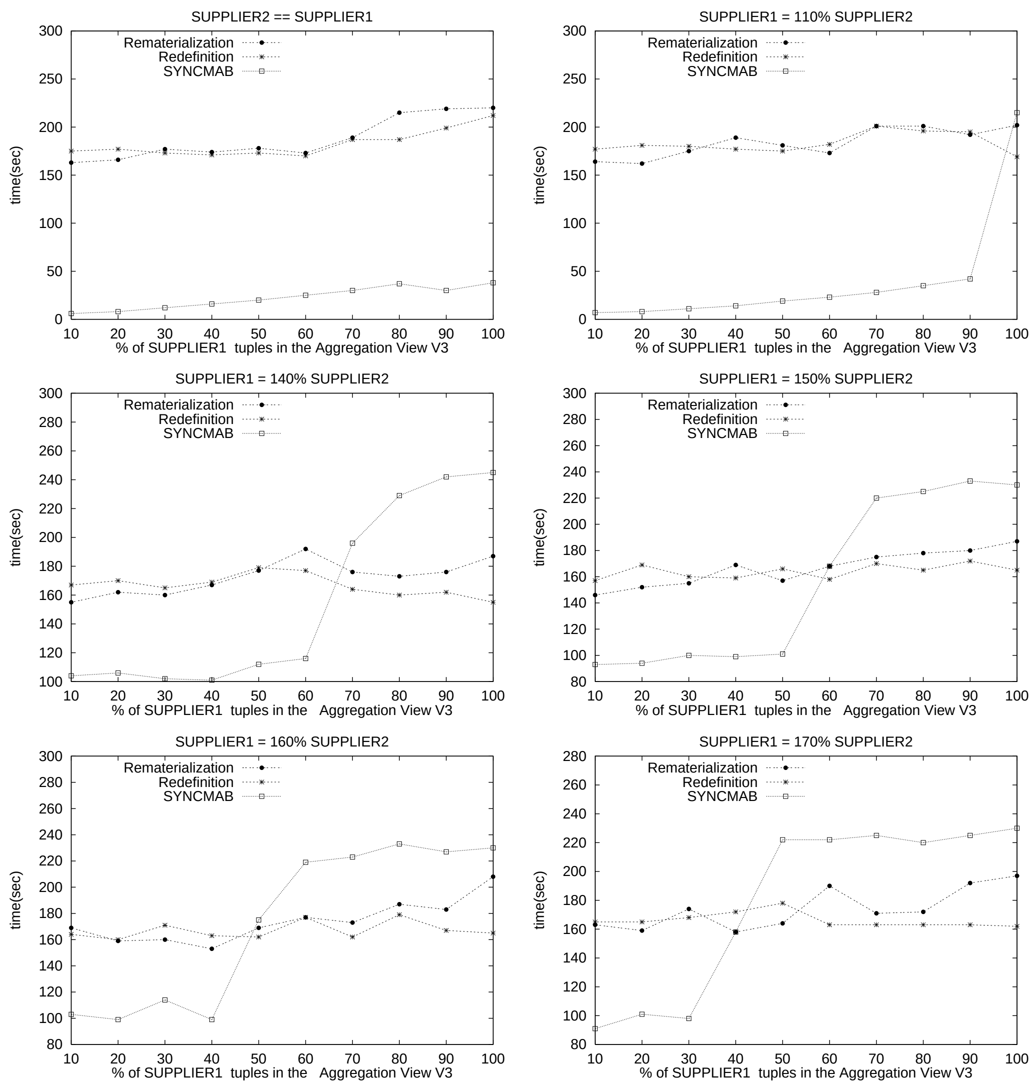
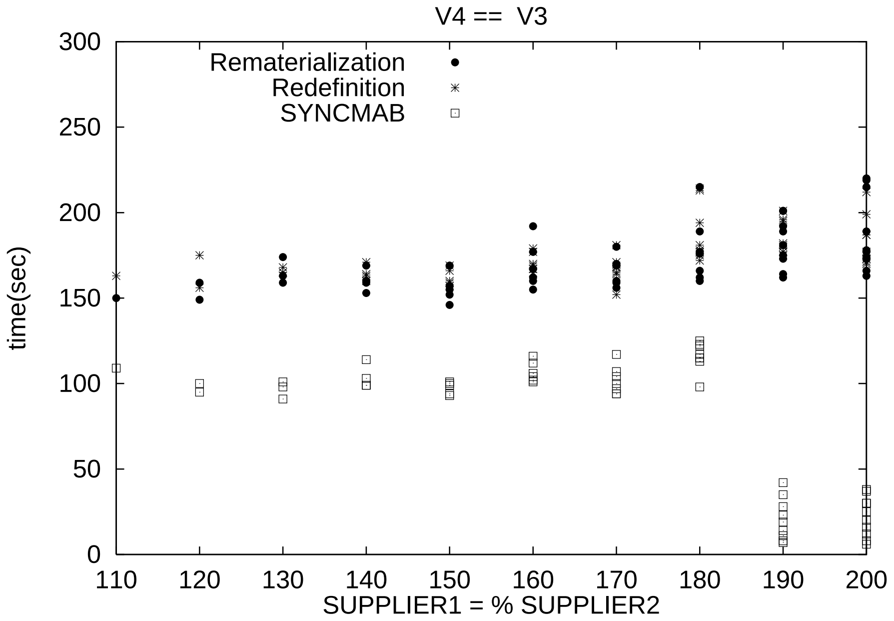


Figure 5.16: Case 7, for the AGGR-SPJ view $V4$, the legal rewriting $V3$ when $V3 \supseteq V4$ using $\mathcal{PC}$-constraint SUPPLIER1 $\supseteq$ SUPPLIER2: the experimental results for maintenance after synchronization with *insertion* by varying the size of the SUPPLIER1 when SUPPLIER1 $\supseteq$ SUPPLIER2.



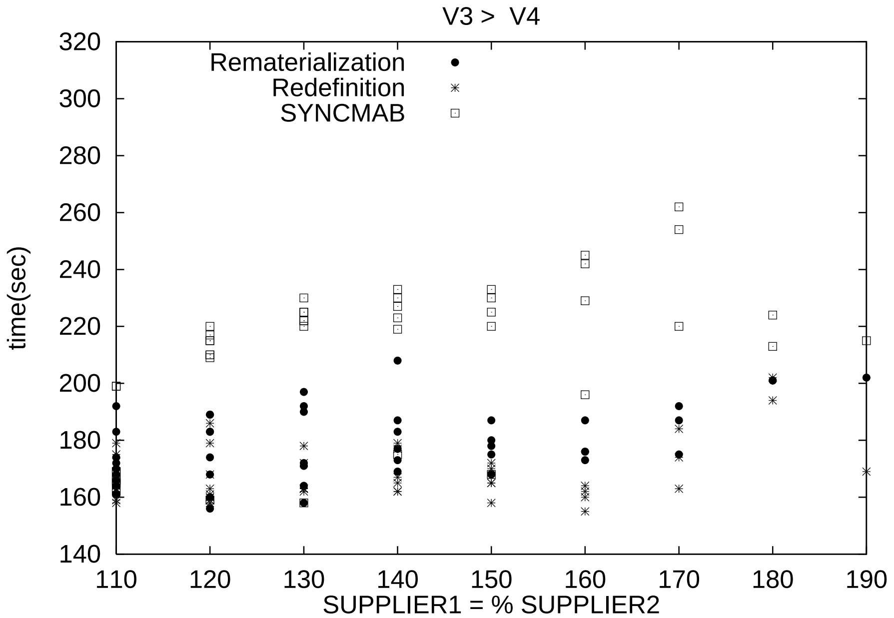


Figure 5.17: Case 7, for the AGGR-SPJ view $V4$, the legal rewriting $V3$ when $V4 \supseteq V3$ using $\mathcal{PC}$-constraint SUPPLIER1 $\supseteq$ SUPPLIER2: the experimental results for maintenance after synchronization with *insertion* by varying the difference between $V3$ and $V4$: Cases 7.1 (upper chart) and 7.2 (bottom chart).

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 particular cases, the view maintenance strategies proposed in this work are more efficient than recomputation or redefinition techniques.

- We perform a systematic experimental evaluation of our proposed algorithms and present the experimental results done using the TPC-D benchmark [64] for comparing our techniques with recomputation and redefinition strategies. For certain types of view synchronization, our view maintenance techniques perform four times better for both SPJ and aggregation views. Namely, this is true when the rewritten view extent is smaller or equal than the original view extent.

CHAPTER VI

Quality-Cost Model for View Synchronization Rewritings

In *EVE*, when a view is synchronized with a capability change, there are typically numerous possible new view *rewritings* that preserve the original view to some degree, depending on how the view was specified using E-SQL and what meta data about ISs is known (i.e., has been recorded using the MISD). Each of these new view redefinitions will in general preserve a different amount and different types of information, which we will term the *quality* of the view. Also, each rewriting will cause different *view maintenance costs*, since data will have to be collected from different ISs. We contend that, with these two dimensions, it is possible to *compare* candidate view redefinitions (i.e., *legal rewritings*) with each other, even if they are not equivalent (as commonly assumed for query rewritings in the context of query optimization [6, 16] or in more recent work on rewriting queries using views [36, 7]).

Since a large number of different legal rewritings may exist for a capability change and a given view, we need a model that allows us to compare these rewritings, establish some (numeric) ranking among them indicating the relative preference of one solution over alternate ones, and to possibly allow us to identify the *best* solution. This question becomes important with a growing number of ISs to query data from, especially if these ISs are independent from each other and are prone to undergo capability changes during the lifetime of a materialized view (as is the case on the Internet). A typical example would be a service that queries information about flights and hotel reservations from several travel agencies on the WWW. Here, it is likely that one of the participants in the system (e.g., an airline company or a hotel chain) changes the type of services (queries) it supports. This would cause our algorithms to generate a number of suggestions for a new view definition that preserve different amounts of information from the old view and which would have to be compared against each other.

To solve this problem, we present the *QC-Model* which is capable of comparing different view rewritings (generated as replacements for an original view that has become invalid). It measures the *efficiency* of a view rewriting in the two dimensions of *quality* and *cost*. Quality is a function of the view interface and the set of data being returned from a new view compared to the original view. Cost refers to *incremental view maintenance* costs and measures how expensive the new view would be for long-term maintenance of data in the data warehouse. The two dimensions, quality and cost, are composed of several relevant factors, such as interface and extent for the quality; transferred data, number of messages, and I/O cost for the cost. We have integrated these two measures into an efficiency model that lets us trade-off between the quality and cost of a view rewriting.

This QC-Model allows us to easily compare different candidate view rewritings based on some computable numeric efficiency score taking complex aspects of the solutions into account. Our model can thus be helpful to users by ranking all possible synchronization solutions in a linear order and allowing the user to make the final choice. Given that view definers in *EVE* can provide the system with preferences on view evolution (using E-SQL), the complete process of choosing among alternate solutions to view synchronization could even be made transparent to a view user.

Our experimental studies demonstrate the utility of the proposed efficiency model in terms of establishing a ranking among rewritings. Our studies succeed in identifying cases with trends and correlations among the different efficiency factors, as well as cases where the measures of cost and quality are independent from one another. The former findings then lead us to suggest possible heuristics for the optimization of the rewriting process itself. These heuristics may be utilized in the future to develop algorithms that find good legal rewritings without first having to compute values for a complete set of rewritten view definitions. This is however beyond the scope of this current work.

While we have initially developed this novel QC-Model in the context of our view synchronization problem of *EVE*, it is also likely to be useful for a number of different scenarios in the area of query reformulation such as query rewriting using views [36], where several different ways of rewriting a query are conceivable, or data warehouse maintenance [2, 70], for which our cost measurement approach and optimization insights gained by our performance studies can be valuable.

This work builds partly upon the *EVE* project and is complementary to our previous work on *finding* and *generating* view rewritings [31]. It goes beyond the work in traditional query optimization [23, 67, 2] since it is not concerned with finding the *best way* to execute a well-defined query, but with finding the *best (not necessarily equivalent) query* for a particular problem. It builds upon previous cost models of incremental view maintenance [6, 70, 72] but extends the QC-Model to now also incorporate a measure of divergence from the intended view specification.

In summary, the contributions of this chapter are fourfold: First, it identifies the new problem of trade-offs of quality against cost for query rewriting in general and for view synchronization in particular and the need for an efficiency model for assessing and evaluating these measures. Second, we explore how to overall find *good* replacements for materialized views that become invalid due to schema changes of underlying base tables. Third, we elaborate this by introducing a new measure of *quality* for such a replacement, and adopt existing cost models [70] for our purposes in order to establish an integrated efficiency model for view synchronization evaluation. The fourth contribution is an experimental evaluation of our findings that demonstrates the utility of our QC-Model.

The remainder of this chapter is organized as follows: Sections 6.2 and 6.3 present a detailed analytic model of quality and performance (cost) trade-offs, respectively. Section 6.4 summarizes experiments we have run to demonstrate the utility of our approach. Section 6.5 discusses our conclusions.

The work presented in this chapter is the result of the collaboration with other members of the EVE project and it has been published in several papers in [28, 29].

6.1 Our Approach for Ranking Legal Rewritings: An Integrated Analytic Model for Quality and Cost

In the *EVE*-system described in the previous section, several possible solutions for the rewriting of a view definition are to be expected for each schema change. Depending on the degree of redundancy in the information space, the view synchronizer may find a large and possibly exponentially (over the size of the information space) growing number of legal rewritings for an affected view. Each legal rewriting may preserve a different combination of attributes (of the original view) and may be specified on disparate base relations with different cardinalities at different sites and even computed differently, e.g., with different joins compared to the original view definition. Ideally, we would like to preserve the original view fully in terms of both view interface and view extent, and at the same time be able to maintain the materialized view in an economical way.

But since this goal cannot always be reached, we introduce measures of evaluating the efficiency of a view rewriting that is close to the "intent" of the initial view (although not necessarily equivalent) and also has low long-term view maintenance costs. We express the efficiency of a view rewriting in terms of its *quality* and *cost*. The first measure is the degree of divergence of quality (i.e., information preservation) of the new view with respect to the old view (See Section 6.2). The second measure takes the long term view maintenance cost associated with the legal rewriting into account (See Section 6.3). In this chapter, we will demonstrate that both quality and cost are very different for different rewritings, which makes it important to select a *good* rewriting out of all legal ones for a particular view and schema change. Our model focuses on how to define *efficiency*, i.e., an efficiency model for view rewritings (which we will refer to as the QC-Model), and how to compute this efficiency for a given view rewriting.

Note that all traditional view maintenance strategies conform to the notion that the rewritten query (or view) is *equivalent* to the original one. On the contrary, we relax the assumption that a query has to be replaced by an equivalent one and introduce instead the concept of *non-equivalent rewritings* with a notion of "legality" that is expressed by our preference model (E-SQL). Furthermore, the view results in *EVE* are materialized and used for an extensive period of time. This means more care has to be given to long-term maintenance cost as opposed to one-time view recomputation cost.

As an example, a legal rewriting may be able to preserve all the attributes in the old view, but only return a small subset of the original results (compared to the original view result). Another legal rewriting may not be able to preserve all the attributes, but may preserve the original results (when projecting both the new and the old views at the common subset of attributes). Similarly, a legal rewriting may preserve information to a higher degree (i.e., have a higher quality) but at the price of also a higher expected view maintenance cost. An ideal legal rewriting should preserve the original view interface and generate the same set of results without introducing any surplus tuples, and it should be efficiently maintainable in the long run. If these ideal goals cannot be reached, we try to (1) assess the divergence from this ideal and assign this divergence a normalized numerical value, (2) estimate (normalized) long-term view maintenance cost, and (3) establish a ranking among otherwise non-comparable rewritings by combining these two measures into one in order to compare different view rewritings.

Our proposed model gives a user of our *EVE*-system a tool to express the importance of these factors by assigning weights to them. These weights are called the *trade-off parameters* in our model. A legal rewriting ranks high and is chosen by our system, if it shows low divergence from the original view and has low maintenance cost. The *EVE*-system will recommend to replace the affected view by the legal rewriting which ranks the highest in this measurement system. Our system can also show the other legal rewritings to the user in the order of their numeric efficiency ranking. Alternatively, a user could also tune the legal rewriting selection result by setting the tradeoff parameters, if desired. A validation of the utility of our proposed QC-Model based on experimental studies can be found in Section 6.4.

6.2 Efficiency Model: Quality of a Legal Rewriting

6.2.1 Information Preservation in Rewritings

The information returned by a view is of great importance to the view end-users. Thus in this section, we evaluate the set of legal rewritings generated by the view synchronizer in terms of the amount of information preserved in the rewritings.

The information preserved in a view can be discussed in terms of two aspects, namely the attributes preserved in the view interface and the view extent returned by executing its SQL query. Ideally, we would like to replace an affected view V by a legal rewriting V_i that *fully preserves* the original view. That is, (1) V_i preserves the original view interface, although some information may be taken from other information sources (denoted by $\text{Attr}(V_i) = \text{Attr}(V)$), and (2) it returns the same set of tuples as the original view on the original information space (denoted by $\text{Ext}(V_i) = \text{Ext}(V)$). Otherwise, V_i is said to *diverge* from the original view V. A legal rewriting V_i is *less preferred* than another legal rewriting V_j in terms of the amount of information preservation (denoted by $V_i <_{IP} V_j$), if $\text{Attr}(V_i) \subseteq \text{Attr}(V_j)$ and $\pi_{Attr(V_i) \cap Attr(V_j)}(V_i) \subseteq \pi_{Attr(V_i) \cap Attr(V_j)}(V_j)$. That is to say, the "closer" a legal rewriting is to the original view in terms of information preservation, the more preferred it is. For example, all *base rewritings* are preserving more than a *derived rewritings* obtained from it (see Chapter II, Section 2.5.2 where the concepts of derived and base rewritings were defined).

6.2.2 Information Preservation on the View Interface

The attributes in a view interface can be classified into four categories according to their *(attribute-dispensable, attribute-replaceable)* parameter values (see Figure 6.1). Each row represents one type of category to which a preserved attribute may belong. Figure 6.1 has three columns: column one shows the values for the *(attribute-dispensable, attribute-replaceable)* parameters, column two the preservation requirement, and column three the weight. Since any legal rewriting must preserve all the indispensable attributes of the original view V, independent of whether the attributes can be taken from other information sources, all attributes in categories 3 and 4 must be preserved in the final view interface. Thus, we do not assign weights to attributes in categories 3 and 4, and we do not include these attributes in our discussion any further.

Four Categories for Preserved Attributes		
Category (dispensable, replaceable)	Preservation Requirement	Weight
C1: $(true, true)$	–	w_1
C2: $(true, false)$	–	w_2
C3: $(false, true)$	must stay	n/a
C4: $(false, false)$	must stay	n/a

Figure 6.1: Four Categories for Preserved Attributes.

However a view rewriting is still legal if it omits attributes in categories 1 or 2. In effect, different legal rewritings may preserve different combinations of attributes in these two categories. Therefore, we define weights w_1 and w_2 ($0 \leq w_1, w_2 \leq 1$) for a view definer to set as needed. We will discuss effects of changing weight parameters in the evaluation section (Section 6.4), while in this section we now assume one fixed setting of w_1 and w_2. When w_1 and w_2 are not explicitly specified by the user, EVE uses the default values set in the system ($(w_1, w_2) = (0.7, 0.3)$). The default settings have the property $w_1 > w_2$ [1]. This represents the fact that EVE is in favor of preserving the replaceable attributes (i.e., attributes in category 1). A view

[1] Note that the absolute values of w_1 and w_2 are not as important as their relative values, since the measure will be normalized later.

having replaceable attributes may be evolved further as more schema changes occur (as our experimental evaluation in Section 6.4 confirms, whereas having relatively many non-replaceable attributes (i.e., attributes in category 2) has a negative effect on the further evolvability of a view. In other words, it is harder to find good legal rewritings for a view if its view elements are non-replaceable.

6.2.3 Information Preservation on View Extent

Besides considering the attributes preserved in the legal rewritings, the *sets of tuples* returned by the queries will also have an impact on the user's satisfaction with a view rewriting V_i. When the view interfaces of a legal rewriting V_i and the original view V are not the same, the extent preservation evaluation is done by comparing tuples on the common subset of attributes only [2] (see Chapter II, Table 2.7 for the π-notations). If it is not possible to find a legal rewriting that is equivalent to the old view, we choose a legal rewriting that produces a view extent as *close* as possible to the original one. Some legal rewritings may have a larger number tuples in V preserved, but at the same time generate many extra tuples that were not in V. On the other hand, some legal rewritings may preserve less tuples in V, but also generate less surplus tuples. In the next section, we discuss how to generate a *good* rewriting according to the user's preference by making a choice which tries to maximize the overlapping extends of the old and new views, and minimize the differences.

6.2.4 Metric of Quality: Degree of Divergence ($\mathcal{DD}$)

We will now present the metric of quality, i.e., the *degree of divergence* $\mathcal{DD}$, that we define to appraise the quality of legal rewritings of a view. In this section, we first discuss information preservation in terms of the view interface, then we explain

[2] When the view interfaces of V_i and V are the same, the extent comparison is done as usual.

our findings on information preservation in terms of the view extent. Finally, we unify the discussions into one quality measure – the *Degree of Divergence* of a legal rewriting from the original view.

6.2.5 Degree of Divergence on the View Interface ($\mathcal{DD}_{attr}(V_i)$)

Let V be a view and V_i ($i \geq 1$) a legal rewriting of V under the capability change $\mathcal{CC}$. Let Attr(V_i) ($i \geq 0$) be the attributes specified in the SELECT clause of V_i. As mentioned earlier, all indispensable attributes in V must be preserved in every legal rewriting V_i for it to be considered legal. So the indispensable attributes do not have to be included in our discussion. An attribute A in the view interface of V_i, $A \in Attr(V_i)$, has two boolean properties *attribute-dispensable* $\mathcal{AD}(A)$ and *attribute-replaceable* $\mathcal{AR}(A)$ corresponding to their evolution parameters set in the E-SQL definition of the view V_i. Thus, the attributes in $Attr(V_i)$ are classified according to the following rules:

$$\begin{aligned} A_i^1 &= \{A \mid A \in Attr(V_i) \ \wedge \mathcal{AD}(A) = true \wedge \mathcal{AR}(A) = true\} \\ A_i^2 &= \{A \mid A \in Attr(V_i) \ \wedge \mathcal{AD}(A) = true \wedge \mathcal{AR}(A) = false\} \end{aligned}$$

We define the quality of the view interface of a view V_i as:

$$Q_{V_i} = \ |A_i^1| \cdot w_1 + |A_i^2| \cdot w_2, \ for \ i \geq \ 1 \tag{6.1}$$

where $|A_i^j|$ is the number of attributes of V_i that fall into the category j (i.e., the cardinality of the set A_i^j). The quality of the original view V is defined likewise and denoted by Q_V.

The (normalized) degree of divergence of V_i from V in terms of the view interface, denoted by $DD_{attr}(V_i)$, can then be defined as:

$$DD_{attr}(V_i) = \begin{cases} 0 & \text{if } Q_V = 0 \\ \frac{Q_V - Q_{V_i}}{Q_V} & \text{otherwise} \end{cases}$$

This is a measure of *distance* of the interface of a view rewriting from the original view interface. $Q_V = 0$ means that the attributes contained in the original view V are all indispensable (since indispensable attributes do not have weights and are not considered in this computation). In this case, if V is evolvable and a legal rewriting V_i is found, then V_i must preserve the indispensable attributes entirely. That is, $Q_{V_i} = Q_V$ (i.e., $DD_{attr}(V_i) = 0$). However, when there are one or more attributes in Attr(V) that are dispensable, then $Q_V > 0$ and $DD_{attr}(V_i)$ is computed as defined above. When V_i does not preserve any of the dispensable attributes, then $Q_{V_i} = 0$ and $DD_{attr}(V_i) = 1$. So in terms of the view interface, a legal rewriting V_i is preferred to the legal rewriting V_j if $DD_{attr}(V_i) < DD_{attr}(V_j)$.

6.2.6 Degree of Divergence on the View Extent ($\mathcal{DD}_{ext}(V_i)$)

The view extent of a legal rewriting V_i may diverge from the original view extent of V in two aspects:

D1. The (relative) number of tuples from the original view V that are not preserved in the new view V_i, denoted by

$$DD_{ext\_D1}(V_i) = 1 - \frac{|V_i \cap_\pi V|}{|V^{\pi(V_i)}|} = \frac{|V \setminus_\pi V_i|}{|V^{\pi(V_i)}|} \tag{6.2}$$

D2. The (relative) number of surplus tuples in the new view V_i that are not in the original view V, denoted by

$$DD_{ext\_D2}(V_i) = \frac{|V_i \setminus_\pi V|}{|V_i^{\pi(V)}|} = \frac{|V_i^{\pi(V)}| - |V \cap_\pi V_i|}{|V_i^{\pi(V)}|} \tag{6.3}$$

Note that, intuitively, the number of tuples that are *not preserved* (**D1**) has to be related to the size of the original view extent $|V^{\pi(V_i)}|$, whereas the number of extra tuples coming into the new view (**D2**) must be seen relative to the size of the *new* view extent $|V_i^{\pi(V)}|$. Figure 6.2 shows the four possible cases of the relationship between the original view V and a view rewriting V_i (none, either or both of **D1**, **D2** can be empty sets).

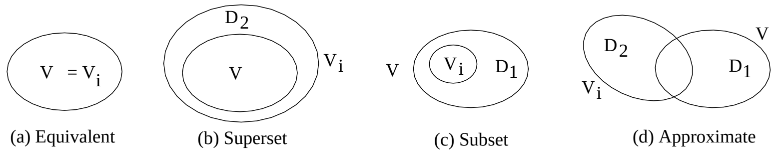


Figure 6.2: Divergence of New View Extent V_i from Original View Extent V.

The total extent divergence of V_i from V is the weighed sum of $DD_{ext\_D1}(V_i)$ and $DD_{ext\_D2}(V_i)$, denoted by $\mathcal{DD}_{ext}(V_i)$, and defined as follows:

$$\begin{aligned} DD_{ext}(V_i) &= \varrho_1 \cdot DD_{ext\_D1}(V_i) + \varrho_2 \cdot DD_{ext\_D2}(V_i) \\ &= \varrho_1 \cdot \frac{|V^{\pi(V_i)}| - |V_i \cap_\pi V|}{|V^{\pi(V_i)}|} + \varrho_2 \cdot \frac{|V_i^{\pi(V)}| - |V \cap_\pi V_i|}{|V_i^{\pi(V)}|} \\ &= 1 - \frac{(\varrho_1 |V_i^{\pi(V)}| + \varrho_2 |V^{\pi(V_i)}|) \cdot |V_i \cap_\pi V|}{|V^{\pi(V_i)}||V_i^{\pi(V)}|} \end{aligned} \tag{6.4}$$

where ϱ_1, ϱ_2 are the trade-off parameters between $DD_{ext\_D1}(V_i)$ and $DD_{ext\_D2}(V_i)$ (ϱ_1, $\varrho_2 \geq 0$ and $\varrho_1 + \varrho_2 = 1$). Remember that we compare the common subset of attributes in the view extents of V_i and V only, with duplicates removed first. Again, the view definer is given a chance to set the trade-off parameters. If the view definer does not set the parameters explicitly, then the default setting $(\varrho_1, \varrho_2) = (0.5, 0.5)$ is used.

When the view definer sets the view-extent parameter $\mathcal{VE}$ = '$\equiv$', no view extent divergence is allowed, i.e., $DD_{ext\_D1}(V_i)$ and $DD_{ext\_D1}(V_i)$ must be both zero. Therefore, we do not discuss this case further. When the view-extent parameter $\mathcal{VE}$ = '$\supseteq$', the first number ($DD_{ext\_D1}(V_i)$) has to be zero for a rewriting to be legal (since all objects of V must also appear in the new view V_i). Therefore, we only have to compute the second measure in order to know the efficiency of V_i in terms of view extent information preservation [3]. On the contrary, when the view-extent parameter $\mathcal{VE}$ = '$\subseteq$', the second measure returned is always zero. Therefore, we only need to compute the first measure [4]. Whenever the view-extent parameter $\mathcal{VE}$ ='$\approx$', both numbers have to be computed (and for meaningful results we should have $0 < \varrho_1, \varrho_2 < 1$). For $\mathcal{VE}$ = '$\supseteq$' and $\mathcal{VE}$ = '$\subseteq$', none of the expensive set intersection operations is required. For these cases, the degree of divergence can be computed by counting the numbers of tuples in the legal rewriting V_i and the original view V (since either $V \subset_\pi V_i$ or $V_i \subset_\pi V$, so the size of the intersection is equal to the size of the smaller relation).

When $\mathcal{VE}$ = '$\subseteq$', then $V_i \subseteq_\pi V$. The expression of $DD_{ext}(V_i)$ becomes:

$$
\begin{aligned}
DD_{ext}(V_i) &= \varrho_1 \cdot DD_{ext\_D1}(V_i) + \varrho_2 \cdot DD_{ext\_D2}(V_i) \\
&= \varrho_1 \cdot \frac{|V \setminus_\pi (V_i \cap_\pi V)|}{|V^{\pi(V_i)}|} + \varrho_2 \cdot \underbrace{\frac{|V_i \setminus_\pi (V \cap_\pi V_i)|}{|V_i^{\pi(V)}|}}_{0 \text{ because } V_i \subseteq_\pi V} \\
&= \varrho_1 \cdot \frac{|V^{\pi(V_i)}| - |V_i \cap_\pi V|}{|V^{\pi(V_i)}|} \\
&= \varrho_1 \cdot \frac{|V^{\pi(V_i)}| - |V_i^{\pi(V)}|}{|V^{\pi(V_i)}|} \qquad (6.5)
\end{aligned}
$$

When $\mathcal{VE}$ = '$\supseteq$', then $V_i \supseteq_\pi V$. The expression of $DD_{ext}(V_i)$ becomes:

$$DD_{ext}(V_i) = \varrho_1 \cdot DD_{ext\_D1}(V_i) + \varrho_2 \cdot DD_{ext\_D2}(V_i)$$

[3]Since the value of ϱ_1 is not relevant in this case, one could set $\varrho_1 = 0$ and $\varrho_2 = 1$ for all "superset"-views.

[4]Similarly, this case can be supported by having $\varrho_1 = 1$ and $\varrho_2 = 0$

$$
\begin{aligned}
&= \varrho_1 \cdot \underbrace{\frac{|V \setminus_\pi (V_i \cap_\pi V)|}{|V^{\pi(V_i)}|}}_{0 \text{ because } V_i \supseteq_\pi V} + \varrho_2 \cdot \frac{|V_i \setminus_\pi (V \cap_\pi V_i)|}{|V_i^{\pi(V)}|} \\
&= \varrho_2 \cdot \frac{|V_i \setminus_\pi (V \cap_\pi V_i)|}{|V_i^{\pi(V)}|} \\
&= \varrho_2 \cdot \frac{|V_i^{\pi(V)}| - |V^{\pi(V_i)}|}{|V_i^{\pi(V)}|} \qquad (6.6)
\end{aligned}
$$

6.2.7 Total Degree of Divergence

With the findings of this section, we now define the total degree of divergence of V_i from V as:

$$DD(V_i) = \varrho_{attr} \cdot DD_{attr}(V_i) + \varrho_{ext} \cdot DD_{ext}(V_i), \ where\ \varrho_{attr}, \varrho_{ext} \geq 0 \ and\ \varrho_{attr} + \varrho_{ext} = 1. \qquad (6.7)$$

'

6.2.8 Experimental Estimation of the Overlapping View Extents

The computation of the total degree of divergence cost $DD()$ given in Equation (6.7) for a legal rewriting depends on estimating the sizes of the candidate view rewritings *without* computing their extents. Using the testbed described in Section 5.4, Chapter V, we conduct experiments for estimating the size of the legal rewriting $V1$ obtained from the SPJ view $V2$. The formula we use for estimating the size of the legal rewriting is based only on the size of the to-be-replaced relation **SUPPLIER2**, the size of the difference between the relations **SUPPLIER1** (the replacement) and **SUPPLIER2**, and the size of the old view $V2$.

Theoretically, we expect the join selectivity of a replacement relation S in the legal rewriting V' to be the same as for the to-be-replaced relation R in the original view V. Let the original view V and its legal rewriting V' be defined by Equations

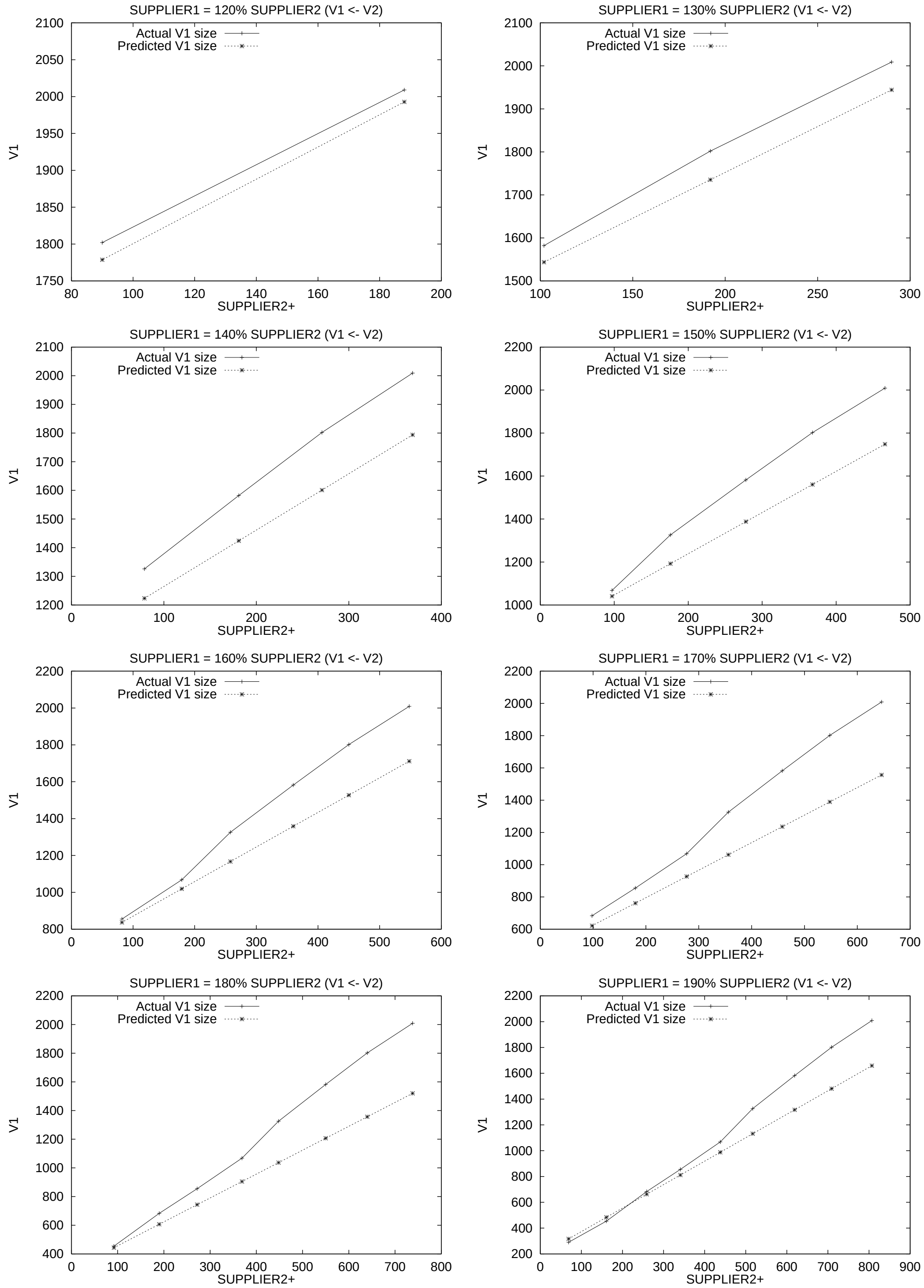


Figure 6.3: "*delete-relation* SUPPLIER2" Capability Change for SPJ View $V2$: Predicting the size of the $V1$.

(3.24) and (3.10), respectively. If we assume that the join selectivities are the same for the two views V and V' relative to the relation R and S, respectively, then we obtain the formula from Equation 6.8 ($|X|$ denotes the size of the relation X, ΔR^+ and ΔR^- are defined in Figure 5.3, Chapter V):

$$|V'| = |V| + \frac{|\Delta R^+|}{|\mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})|} - \frac{|\Delta R^-|}{|\mathcal{CV}(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})|} \tag{6.8}$$

The experimental results obtained applying the estimation formula given in Equation (6.8) are shown in Figure 6.3 for the synchronized view $V1$ after capability change "*delete-relation* SUPPLIER2" (see Section 5.4).

6.3 Efficiency Model: View Maintenance Cost of a Legal Rewriting

6.3.1 View Maintenance Basics

We assume that data content updates on the base relations, e.g., inserts or deletes of tuples to/from the base relations, take place more frequently than capability changes in the information space. Therefore, we choose to rank the legal rewritings by their *long term* view maintenance costs [5]. A legal rewriting is considered to be *preferred* if its expected view maintenance costs are low compared to other legal rewritings. We further assume that a conventional incremental view maintenance algorithm similar to the one specified in [70] is used to bring the view extent up-to-date right after the IS data is updated. We assume the IS data updates are sufficiently spaced from each other, so concurrent data updates are not considered in this work. Considering concurrent updates would significantly complicate this portion of our analytical model, but we expect that it would not have a large enough impact on

[5]The cost for recomputing the original view extent after a view re-definition is a one-time cost. Thus we do not judge the legal rewriting on this one-time view update cost.

our findings to justify this extra effort.

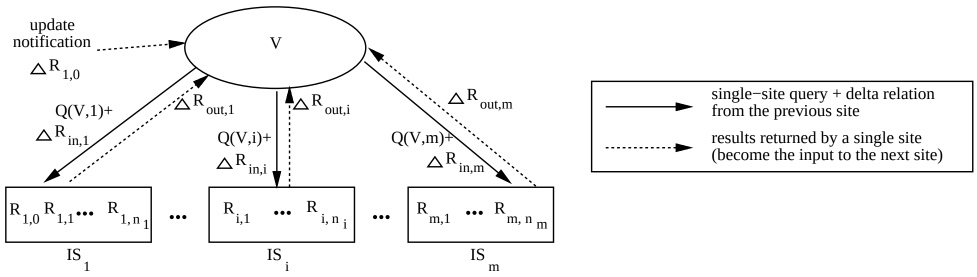


Figure 6.4: View Maintenance Process.

Now, we briefly introduce the view maintenance algorithm used for keeping the view extent up-to-date after one data content change. The cost for multiple updates can then be computed by summing over all individual costs. For the following, we assume a view V that references relations $R_{1,0}, R_{1,1}, \ldots,\ R_{1,n_1}, \ldots, R_{i,1}, \ldots,$ $R_{i,n_i}, \ldots, R_{m,1}, \ldots,\ R_{m,n_m}$ residing in m ISs (see Figure 6.4). Let $n = 1 + \sum_{i=1}^{m} n_i$ be the total number of relations in the information space that are referred to in the view (that is, including the relation where the data update originated). Further assumptions are:

- the ISs are fully cooperative and are able to join their local relations with incoming delta-relations.
- the data warehouse is not doing any joins.
- the partial results are sent along to the next IS (for all ISs).
- the maintenance is done in a non-parallel way.
- the relations referenced in the view are joined in an order that does not make

it necessary to again query data from an IS already visited[6].

- each source may have more relations that are not used by the view definition V, but these relations are not relevant to our computation and thus are not shown in Figure 6.4.

Without loss of generality, we assume $IS_1.R_{1,0}$ changes its data content, and there are n_1 other relations in the same source referenced in V. After the change is completed, IS_1 notifies EVE about the data change by sending the update information $\Delta R_{1,0}$ to the view site. Upon receiving this data update notification, EVE decides which views are affected by it. Then our view maintainer brings the view extents of these affected views up-to-date by executing the following view maintenance algorithm.

Algorithm 1 ***View-maintenance(V):***

```
begin
  1. Delta = data content update at source 1 by relation R(1,0)
  2. for (i = 1; i <= m; i++) {
  3.     view maintainer sends appropriate single site query Q(V,i) with Delta to source i;
  4.     source i sends the query results, a new Delta, back to view maintainer;
  5. }
  6. view maintainer updates the view extent of V
end
```

Note that the delta relation ($\Delta R_{out,i}$) sent back by the information source IS_i to the warehouse becomes the delta relation ($\Delta R_{in,i+1}$) sent along with the single site query Q(V,i+1) to the next information source IS_{i+1} [7]. In order to compute the join at information source IS_{i+1}, the tuples of the delta relation $\Delta R_{in,i+1}$ are created as a new relation at the IS which is then joined with the local relations.

For the estimation of the view maintenance costs of the legal rewritings, similar to prior work, we assume the following database statistics:

[6]If this is the case, we can treat such a physical IS as two or more logical ones and count it more than once in our computation, which would then generate the correct results for the cost estimates

[7]Note that if there is only a single relation at IS_1 referred in V, then the view maintainer does not need to send a query to IS_1.

1. The cardinality (number of tuples) of each relation R is known and denoted as $|R|$.

2. The size of each attribute $R.A$ is known as $s_{R.A}$ (and registered in the MKB). From this information, we can estimate the size of any set of attributes in a query sent to an IS and the size of the results returned by an IS.

3. The join selectivity (js) is the percentage of tuples in a relation that would join (with an equijoin condition[8]) with tuples in the other relation. For simplicity, we assume the join selectivity is a constant for any two relations across the information sources (and is also registered in the MKB).

4. We assume there is a local selection condition for each relation involved in a view definition. Similar to joins, we further assume all the operators of the local condition are equality-based (in order to have a constant local selectivity σ_{IS_i})[9].

5. $|R|$ and js do not change significantly as updates occur.

6. K = the number of bytes per physical block.

Similar to previous work [70], we now introduce three major cost factors (for a single data content update) for a particular legal rewriting: the number of messages exchanged, the number of bytes transferred, and the I/O cost at the local ISs. Then, we present our workload model that is used to compare the view maintenance costs of different legal rewritings.

[8]To simplify we assume all the relations are joined by some equijoin conditions.

[9]The reason for this assumption is to keep the current dicussion simple. The model can be extended to handle all comparison operators.

6.3.2 Cost Factor Based on Number of Messages Exchanged (CF_M)

The number of messages exchanged between the information space and the view site for a single base data update, denoted as CF_M, is in the range $[0, 2m]$ (with m denoting the number of information sources involved in the view). To be more specific:

$$CF_M = \begin{cases} 0 & \text{if } m = 1 \text{ and } n_1 = 0 \\ 2 & \text{if } m = 1 \text{ and } n_1 > 0 \\ 2 \cdot (m-1) & \text{if } m > 1 \text{ and } n_1 = 0 \\ 2 \cdot m & \text{otherwise} \end{cases}$$

n_1 is the number of relations in the update-generating IS *besides* the relation where the update occurred. The best case $CF_M = 0$ occurs when there is only one relation referred to in the view V (or when V is self-maintainable as discussed in [15]). Note that when there is only one relation in IS_1 referred to in V ($n_1 = 0$), then no query needs to be sent to IS_1.

6.3.3 Cost Factor Based on Bytes of Data Transferred (CF_T)

Considering an information space consisting of n relations $R_1, \ldots, R_n$ in m information sources $IS_1, \ldots, IS_m$, we can derive a general computation for the number of bytes transferred (assuming the algorithm described earlier in the Section 6.3.1). This computation assumes that one inserted/deleted tuple is sent from IS_1 to the view site, which is the initial delta relation (the first line in Equation 6.10). Then this delta relation is sent down to the information source IS_1 to join with other relations in IS_1 referred to in the view definition (the first term in the second line), and the resulting new delta relation (the second term in the second line) is sent back to the view site. The same process iterates through all the information sources referred to in the view to build up the delta relation that contains the tuples affected by the data update.

In summary, the number of bytes transferred can thus be approximated by:

$$\begin{aligned}
CF_T \;=\; & \underbrace{s_{R_{1,0}}}_{update\ notification} \\
+ \; & \underbrace{s_{R_{1,0}}}_{\Delta R_{in,IS_1}} + \underbrace{\sigma_{IS_1} \cdot J_{IS_1} \cdot s_{\Delta R_{out,IS_1}}}_{\Delta R_{out,IS_1}} \\
+ \; & \underbrace{\sigma_{IS_1} \cdot J_{IS_1} \cdot s_{\Delta R_{out,IS_1}}}_{\Delta R_{in,IS_2}} + \underbrace{\sigma_{IS_1}\sigma_{IS_2} \cdot J_{IS_1}J_{IS_2} \cdot s_{\Delta R_{out,IS_2}}}_{\Delta R_{out,IS_2}} \\
+ \; & \cdots \\
+ \; & \underbrace{(\sigma_{IS_1} \cdot \ldots \cdot \sigma_{IS_{m-1}})(J_{IS_1} \cdot \ldots \cdot J_{IS_{m-1}}) s_{\Delta R_{out,IS_{m-1}}}}_{\Delta R_{in,IS_m}} \qquad (6.9) \\
+ \; & \underbrace{(\sigma_{IS_1} \cdot \ldots \cdot \sigma_{IS_m})(J_{IS_1} \cdot \ldots \cdot J_{IS_m}) s_{\Delta R_{out,IS_m}}}_{\Delta R_{out,IS_m}}
\end{aligned} \qquad (6.10)$$

where s_R is the size (sum of the length of attributes in bytes) of the relation R or intermediate query result R, σ_{IS_i} is the average selectivity for the selection conditions used in the single-site query to source IS_i [10], and J_{IS_i} is the estimated size of the resulting join relation $R_{i,1} \bowtie R_{i,2} \bowtie \cdots \bowtie R_{i,n_i}$ returned by the source IS_i. That is, $J_{IS_i} \approx js^{n_i} \cdot |R_{i,1}| \cdot |R_{i,2}| \cdot \ldots \cdot |R_{i,n_i}|$, with js^{n_i} being the average join selectivity for this IS, raised to the power of the number of relations in this IS.

If all selectivities (σ), join selectivities (js), relation cardinalities ($|R|$), and tuple sizes (s) of the relations are assumed to be the same for all Rs, we can simplify the above summation as follows:

$$CF_T = 2s + 2\sum_{j=1}^{m-1} \left(\sigma^j \cdot (|R| \cdot js)^{n_R(j)} \cdot s(1 + n_R(j))\right) + \sigma^j \cdot (|R| \cdot js)^{n_R(j)} \cdot s(1 + n_R(m)) \qquad (6.11)$$

[10]Since we assume there is one local condition for each relation residing in IS_i, $\sigma_{IS_i} = \sigma_{R_{i,1}} \cdot \sigma_{R_{i,2}} \cdot \ldots \cdot \sigma_{R_{i,n_i}}$ for $1 \le i \le m$.

with $n_R(k) = \sum_{i=1}^{k} n_i$.

6.3.4 Cost Factor Based on I/O ($CF_{I/O}$)

In this section we use the total number of estimated input and output operations performed by local ISs in order to process incremental view maintenance for each legal rewriting (in the information space) as a criterion to rank the legal rewritings. Intuitively, a legal rewriting is *preferred* if it requires less I/O-operations (resources) from the overall information space to keep its view extent up-to-date in the long run.

Let $CF_{I/O,IS_i}$ be the number of estimated I/Os at the information source IS_i. $CF_{I/O,IS_i}$ is the sum of the I/Os of the relations that reside at source IS_i, i.e., incorporating the I/O-costs of all relations at IS_i. Then the total number of I/Os in the information space, denoted as $CF_{I/O}$, is the sum of the I/Os at all m sources, i.e.,

$$CF_{I/O} = \sum_{i=1}^{m} CF_{I/O,IS_i} \tag{6.12}$$

$CF_{I/O}$ is influenced by many parameters, such as the number of relations referred to in the view definition, the tuple sizes of the relations, the cardinalities of the relations, the join selectivity factors of the joined relations, the index structures available for each join attribute, the size of local buffers available to the information sources and the view site, and the join methods available to each of the ISs.

6.3.5 Total View Maintenance Cost for a Single Data Update

The total view maintenance cost of a view V with respect to a single data update can now be defined as:

$$Cost(V) = CF_M \cdot cost_M + CF_T \cdot cost_T + CF_{I/O} \cdot cost_{I/O} \tag{6.13}$$

where $cost_M$, $cost_T$, and $cost_{I/O}$ are the unit prices for sending a message, transferring a data block, and performing a disk I/O, respectively.

6.3.6 Workload Model

On the one hand, different legal rewritings of a view may make use of information from different information sources. And, on the other hand, a particular data update only affects the views that refer to this data. Therefore, it is not sufficient to compare various legal rewritings with respect to the same set of data updates. Instead, we have the following choices for a workload model for our system:

M1. We assume the number of updates of a relation is proportional to the number of tuples in the relation. This assumption is equivalent to data updates happening to p percent of a relation's tuples within a given time frame. So a view V is facing a total of $p \cdot \sum_{i=1}^{m}(|R_{i,1}| + |R_{i,2}| + \cdots + |R_{i,n_i}|)$ updates per time unit [11].

M2. We assume each relation R has a constant number of updates per time unit u, independent of the size or the location of R. If an information source IS_i has n_i of its relations used by V, then V is facing a total of $u \cdot \sum_{i=1}^{m} n_i$ data updates per time unit.

M3. We assume each information source is facing a constant number of data updates u. In this case, view V would face a total of $m \cdot u$ updates.

M4. We assume each legal rewriting is affected by a constant number u (but different sets) of data content updates. In this case, we would have to make further assumptions as to the distribution of these updates over all ISs, e.g., we could assume that data updates happen equally for each view element in the view.

[11] Remember that we have m ISs and each IS_i has n_i relations for $1 \leq i \leq m$.

Note that after selecting a specific workload model, we can compute the total view maintenance costs, COST(V_i), for the updates within a certain time unit. If we assume that there are k legal rewritings for an affected view, the total cost of legal rewriting V_i can be normalized as follows:

$$COST^*(V_i) = \frac{COST(V_i) - min_{1 \le j \le k}(COST(V_j))}{max_{1 \le j \le k}(COST(V_j)) - min_{1 \le j \le k}(COST(V_j))} \tag{6.14}$$

This gives us a view maintenance cost between 0 and 1 that we can trade off against the view quality (Section 6.2).

6.3.7 Overall Efficiency of a Legal Rewriting

The *overall efficiency* of a legal rewriting can now be computed as

$$QC(V_i) = 1 - (\varrho_{quality} \cdot \mathcal{DD}(V_i) + \varrho_{cost} \cdot COST^*(V_i)) \tag{6.15}$$

with $0 \le \varrho_{quality}, \varrho_{cost} \le 1$ and $\varrho_{quality} + \varrho_{cost} = 1$. With both quality and cost normalized, this number will be between 0 and 1. An efficiency of 0 means a legal rewriting that preserves no information (which renders it "useless"), An efficiency of 1 would identify a "perfect" legal rewriting preserving the complete view interface and all tuples at no cost. Since the incremental view maintenance cost will never be zero, an efficiency of 1 can only be reached if the costs are given no consideration, i.e., if $\varrho_{cost} = 0$.

6.4 Experimental Evaluation

We now set out to verify the validity of our proposed *QC-Model* and get a deep understanding of the interplay between quality and maintenance costs through a number of experiments. These experiments were conducted in the context of our *EVE* system, i.e., all rewritings of a view were being generated by our synchronization algorithm [30, 40]. However, as our system is not fully instrumented with our QC-Model yet, the reported measures are computed using our algebraic findings instead of measuring them directly from an implemented system. This means, that in this section, we evaluate the *QC-Model* rather than the complete *EVE* system.

6.4.1 Experiment 1: "Survival" of a View

The number of capability changes a view can "survive"[12] depends on its evolution parameter settings and the degree of data redundancy in the information space. In general, when the evolution parameters are set to *dispensable* and *replaceable* and when data is amply duplicated in the information space, then the view has a higher chance to survive in an evolving environment. As an example, we assume an attribute $R.A$ referred to in a view is replaceable. When $R.A$ is deleted from its site, *EVE* will be able to replace R with its replica from another information site. On the other hand, if $R.A$ is non-replaceable, then even if there is a replica of $R.A$ in the information space, *EVE* will not be able to replace $R.A$ in the future. Therefore, when salvaging an affected view definition, *EVE* gives a higher priority to view components with their evolution parameters set to *replaceable*. Employing this strategy, *EVE* has a higher chance to keep the view alive in the future. Theoretically, if there is a high number of data replicas (i.e., larger than the number of relevant *delete*–capability-changes) in the information space and the view components are replaceable, then a view could be kept alive indefinitely. This observation is supported by the following experiment in our *EVE* system. Let's assume a view is defined as follows:

$$
\begin{array}{ll}
\text{CREATE VIEW} & V_0\ (\mathcal{VE} = \text{“}\supseteq\text{”})\ \text{AS} \\
\text{SELECT} & R.A\ (\mathcal{AD} = true, \mathcal{AR} = true), \\
 & R.B\ (\mathcal{AD} = true) \\
\text{FROM} & R\ (\mathcal{RR} = true)
\end{array}
\qquad (6.16)
$$

[12]The view can be evolved by our algorithm and stay valid after these capability changes.

We now assume that $R.A$ is deleted by its information provider. Further, we assume two relations, $S(A, C)$ and $T(A, D)$, in other information sources are related to the relation R. We have two $\mathcal{PC}$ constraints defined in the MKB: $\mathcal{PC}_{R,S} = (\pi_A(R) \equiv \pi_A(S))$ and $\mathcal{PC}_{R,T} = (\pi_A(R) \subseteq \pi_A(T))$.

Then, there are three alternative ways to evolve V_0:

- One possible solution is to drop the attribute $R.B$, since it is nonreplaceable but dispensable, and then replace $R.A$ with an appropriate attribute from either relation S or T. Using this strategy, we get the two legal rewritings V_1 and V_2.

$$\begin{array}{ll} \text{CREATE VIEW} & V_1\ (\mathcal{VE} = \text{“}\supseteq\text{”})\ \text{AS} \\ \text{SELECT} & S.A\ (\mathcal{AD} = true, \mathcal{AR} = true) \\ \text{FROM} & S\ (\mathcal{RR} = true) \end{array} \tag{6.17}$$

$$\begin{array}{ll} \text{CREATE VIEW} & V_2\ (\mathcal{VE} = \text{“}\supseteq\text{”})\ \text{AS} \\ \text{SELECT} & T.A\ (\mathcal{AD} = true, \mathcal{AR} = true) \\ \text{FROM} & T\ (\mathcal{RR} = true) \end{array} \tag{6.18}$$

- Another possible solution is to simply drop the attribute $R.A$ from V_0, since it is dispensable. We get V_3 as a legal rewriting:

$$\begin{array}{ll} \text{CREATE VIEW} & V_3\ (\mathcal{VE} = \text{“}\supseteq\text{”})\ \text{AS} \\ \text{SELECT} & R.B\ (\mathcal{AD} = true) \\ \text{FROM} & R\ (\mathcal{RR} = true) \end{array} \tag{6.19}$$

Ignoring the view extent quality factor for the time being, if $w_1 > w_2$ (that is, if we give a larger weight to replaceable attributes than to non-replaceable attributes), then *EVE* would choose V_1 and V_2 over V_3 (attribute A is in category 1, and attribute B is in category 2; see Section 6.2.2 for a definition of these categories and weight factors). On the other hand, if $w_2 > w_1$, then *EVE* may choose V_3 over the other

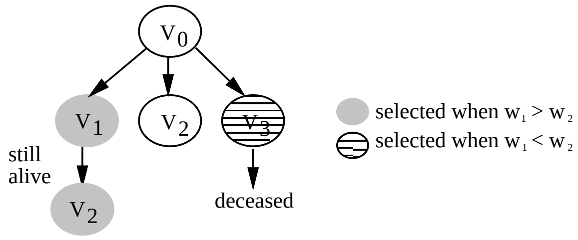


Figure 6.5: The Life Span of Legal Rewritings.

two legal rewritings (see Figure 6.5). Assume EVE chooses V_1 to rewrite V_0. If S gets deleted later, then the view can still be evolved by rewriting it into V_2. On the other hand, if V_3 were to be chosen for this first rewriting, then any further capability change in the information space will cause the view to become undefined. This supports the default setting of $w_1 > w_2$ for the QC-Model.

6.4.2 Experiment 2: Ratio between Relations and ISs

In this experiment, we study the relationships between the number of ISs involved in a view and the view maintenance cost attributed to the view. We conduct this experiment by varying the number of ISs involved in a view, while fixing all other parameter settings, such as the number of relations referred in a view, the selectivity and the join selectivity. The main purpose of this experiment is to investigate whether it is beneficial to retrieve all information from as few sites or as many sites as possible.

For this experiment, we assume that six relations are used in the view. That implies there are at most six information sources involved in the view (each relation resides in a different information source) and at least one information source (all six relations are in one site). The system parameters for our experiment are summarized in Table 6.1.

With this experimental setting, there are six possible scenarios corresponding to

Name	Meaning	Default Value
n	Total number of relations in the information space	6
m	Total number of information sites referred in a view	{1,6}
$\|R_i\|$	Cardinality of the relation R_i, for all i	400
s_{R_i}	Tuple size of the relation R_i, for all i	100
σ	Selectivity of a local condition	0.5
js	Join selectivity factor	0.005
bfr	Blocking factor	10

Table 6.1: List of System Parameters.

# Sites (m)	Relation Distribution (# of relations in each site)
1	(6)
2	(1,5), (2,4), (3,3), (4,2), (5,1)
3	(1,1,4), (1,2,3), (1,3,2), (1,4,1), (2,1,3), (2,2,2), (2,3,1), (3,1,2), (3,2,1), (4,1,1)
4	(1,1,1,3), (1,1,2,2), (1,1,3,1), (1,2,1,2), (1,2,2,1), (1,3,1,1), (2,1,1,2), (2,1,2,1), (2,2,1,1), (3,1,1,1)
5	(1,1,1,1,2), (1,1,1,2,1), (1,1,2,1,1), (1,2,1,1,1), (2,1,1,1,1)
6	(1,1,1,1,1,1)

Table 6.2: Relation Distribution across Information Sources.

the six rows of Table 6.2. Row 1 for example indicates that the data for the view is retrieved from a single IS, row 2 indicates it is from two ISs, and so on. Within each scenario, the relations may be distributed differently among the ISs. For example, when there are two information sources involved in a view, one relation can be retrieved from one IS and the other five relations from another IS – represented by (1,5) on the second row; two relations from one IS and the remaining four relations from another IS – represented by (2,4); and so on. Case (3,3) exhibits the most even distribution, since there is an equal number of relations in each IS. Cases (1,5) and (5,1) are the most skewed relation distributions. Note that (1,5) is different from the Case (5,1), because in this experiment we assume that data updates are initiated at the first IS. Therefore, (1,5) and (5,1) may incur different view maintenance costs.

We first compute the view maintenance cost for every relation distribution in each scenario, then we compute the average view maintenance cost for that scenario. The results are shown in Figure 6.6.

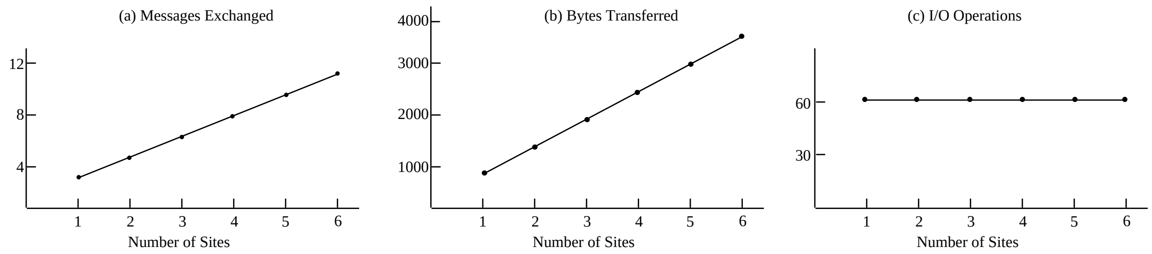


Figure 6.6: The Relationships between Numbers of ISs in the View and Three View Maintenance Cost Factors.

As shown in Figure 6.6, the number of messages exchanged and the number of bytes transferred between the view site and the information space both increase when the number of information sources involved in a view increases. That is, the view maintenance cost of a single data update increases when the number of information sources involved in a view goes up.

6.4.3 Experiment 3: Relation Distribution

Now we look at the previous experiment from another angle. Within each scenario (i.e., with a fixed number of information sources involved in a view) we study whether the relation distributions affect the view maintenance costs. Namely, we study whether the view maintenance costs are lower when the relations referred to in a view are evenly (or *uniformly*) distributed among the information sources or when they are unevenly distributed. The results are summarized in Figure 6.7. The graphs show the number of bytes transferred for three particular join selectivities. For each setting of js, we compare possible distributions of 6 relations in 2,3, and

4 information sources, respectively. The possibilities are listed in Table 6.2, in the chart we group the cases (i.e., $(1, 5) \equiv (5, 1)$).

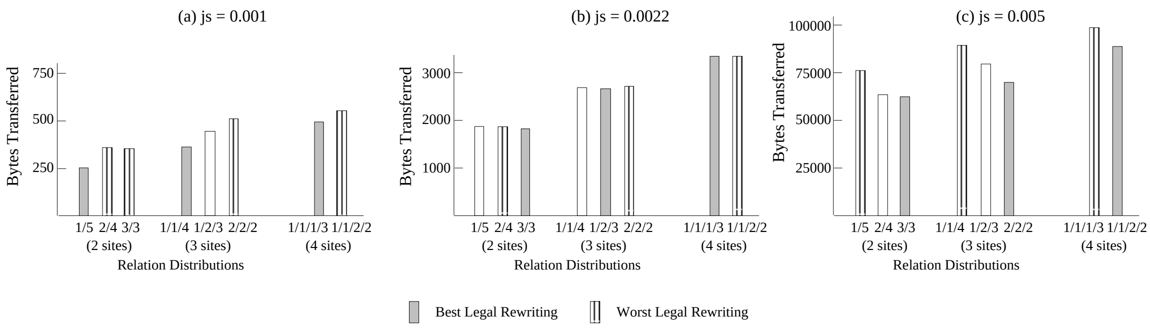


Figure 6.7: The Relationships between Evenness of Relation Distributions and View Maintenance Costs.

We find no apparent relationship between the view maintenance costs and the evenness of the relation distribution. Our experimental results show that when the number of tuples in the intermediate results during a view update grows fast (i.e., we have a relatively high average join selectivity js in our information space), it is beneficial to have a more evenly distributed relation allocation (see Figure 6.7(c) with $js = 0.005$). On the other hand, when the number of joined tuples of the delta relation does not grow as fast, it may be advantageous to have a more skewly distributed information space (see Figure 6.7(a) when $js = 0.001$). There are cases where the evenness of the relation distributions does not have a clear impact. However, as stated in Experiment 2, we have found a dependency of the view maintenance costs on the number of ISs involved. Therefore, minimizing the number of ISs involved in a view rewriting should have a higher priority over choosing a certain relation distribution among the ISs.

6.4.4 Experiment 4: Relation Cardinality

In this experiment, we study the relationship between the cardinalities of the substituted relations and the *overall efficiency* of the legal rewritings. We conduct this experiment by varying the cardinalities of the substituted relation while keeping all other parameter settings the same. Let us assume a view V is defined as follows:

$$\begin{array}{ll} \text{CREATE VIEW} & V\ (\mathcal{VE} = `\approx\text{'})\ \text{AS} \\ \text{SELECT} & \cdots,\ R_2.A\ (\mathcal{AR} = true), R_2.B\ (\mathcal{AR} = true), R_2.C\ (\mathcal{AR} = true) \\ \text{FROM} & R_1, R_2\ (\mathcal{RR} = true) \\ \text{WHERE} & \cdots \end{array} \tag{6.20}$$

Let us assume that relation R_2 is deleted by its information provider, and that there are five relations $S_1, \cdots, S_5$ in the information space that are identified by the view synchronizer to be appropriate substitutes for R_2. Five new views, $V_1 \ldots V_5$ can be defined that are formed by replacing relation R with the respective relation R_n. The cardinalities of R_2 and the substitute relations for our experiment are summarized in Table 6.3.

Site Name	Relation Name	Cardinality
IS_1	$R_2(A, B, C)$	4000
IS_2	$S_1(A, B, C)$	2000
IS_3	$S_2(A, B, C)$	3000
IS_4	$S_3(A, B, C)$	4000
IS_5	$S_4(A, B, C)$	5000
IS_6	$S_5(A, B, C)$	6000

Table 6.3: Cardinalities of R_2, $S_1, \cdots, S_5$.

We further assume that the following inter-relationships among these relations hold true: Relation S_1 is contained in relation S_2, denoted by a $\mathcal{PC}$ constraint: $\mathcal{PC}_{S_1,S_2} = (S_1 \subseteq S_2)$, S_2 in turn is contained in S_3, S_3 is equivalent to the deleted relation R_2,

S_3 is contained in S_4, and S_4 contained in S_5 (i.e., $S_1 \subseteq S_2 \subseteq S_3 = R_2 \subseteq S_4 \subseteq S_5$). Therefore, replacing R_2 with S_i, for $1 \leq i \leq 5$, we get five alternate yet legal rewritings with different view extents and view maintenance costs[13]. Setting the system parameters to $w_1 = 0.7$, $w_2 = 0.3$, $\varrho_{D1} = 0.5$, $\varrho_{D2} = 0.5$, $\varrho_{attr} = 0.7$, $\varrho_{ext} = 0.3$, $cost_M = 0.1$, $cost_T = 0.7$, $cost_{I/O} = 0.2$, $\varrho_{quality} = 0.9$, and $\varrho_{cost} = 0.1$, we get the metrics of quality and cost that are summarized in Table 6.4 (see also Case 1 in Figure 6.8). The other two cases in Table 6.4 and Figure 6.8 are obtained with ($\varrho_{quality} = 0.75$, $\varrho_{cost} = 0.25$) and ($\varrho_{quality} = 0.5$, $\varrho_{cost} = 0.5$), respectively.

Rewriting	DD_{attr}	DD_{ext}	DD	Cost (Normalized Cost)	$QC(V_i)$	Rating
V_1	0	0.25	0.075	842.3 (0)	0.9325	3
V_2	0	0.13	0.0375	1193.3 (0.25)	0.94125	2
V_3	0	0.00	0.00	1544.3 (0.5)	0.95	1
V_4	0	0.10	0.027	1895.3 (0.75)	0.898	4
V_5	0	0.17	0.045	2246.3 (1)	0.855	5

Table 6.4: Ranking of Legal Rewritings for Experiment 4. (Detailed Data for Case 1)

In Section 6.2 we postulated that the degree of divergence $DD(i)$ for a view rewriting V_i will be large for a relation whose size is very different from the size of the original relation, and vice versa. The cost of a legal rewriting will be larger, all other factors equal, with a growing size of the replaced relation(s). Trading off these two factors against each other will therefore lead to different results depending on how the trade-off parameters are set. Our experiment validates these findings.

For example, when the parameters are set to ($\varrho_{quality} = 0.9$, $\varrho_{cost} = 0.1$, Case 1), the QC-Model chose legal rewriting V_3 over the other four legal rewritings. Here, we give a high priority to the quality of the rewriting, which is best when the replacing relation comes as close as possible to the original relation, which is the case in legal

[13]Note that $\mathcal{VE}$ = '$\approx$' for this view as given in Equation 6.20

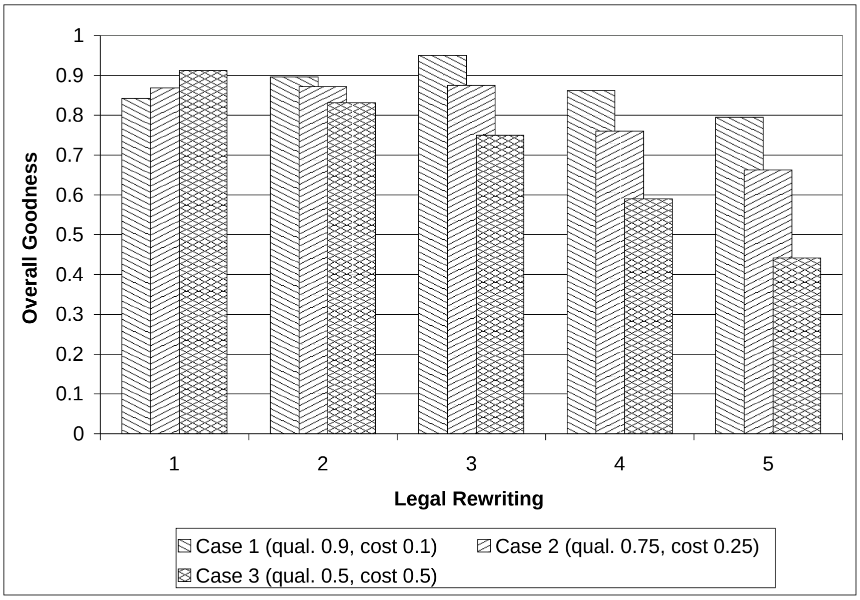


Figure 6.8: Results of Assessing Legal Rewritings for Experiment 4.

rewriting V_3. The graph depicted in Figure 6.8 shows that the overall efficiency increases from legal rewriting V_1 until V_3 (because the size of the replacing relation approaches the size of the original relation), then becomes worse as the difference between the relation sizes grows bigger.

However, in Case 3, with ($\varrho_{quality} = 0.5$, $\varrho_{cost} = 0.5$), the cost has a larger impact on the overall efficiency of the legal rewriting. Since the cost is continuously increasing as the replacing relations get bigger (i.e., from legal rewriting V_1 to V_5), the overall efficiency of the rewritings decreases, so rewriting V_1 (with the smallest replacing relation) is chosen by our view synchronizer. Even in Case 2, the influence of the cost on the total result is large enough for V_1 to be selected as best legal rewriting.

Two observations we made from Figure 6.8 are:

- If we focus our attention on the legal rewritings V_3, V_4, and V_5 (labeled 3, 4, and

5 in Figure 6.8, rows 3 to 5 in Table 6.4), we can see that these rewritings are obtained by substituting the deleted relation R_2 by a *superset* relation. Among these three legal rewritings, V_3 is always ranked highest among the three in various parameter settings. This is because the degrees of divergence (fourth column in Table 6.4, labeled DD) as well as the view maintenance costs (fifth column, labeled Cost) go up when the cardinalities of the replaced relations go up. For these cases, the trade-off parameters have no influence on what rewriting is selected to be best. A consequence is that if we have only superset replacements at our disposal, the replacement that is closest to the original in terms of the relation size is also the smallest replacement and will always rank best among legal rewritings.

- If we focus on the legal rewritings V_1, V_2, and V_3 (labeled 1, 2, and 3 in Figure 6.8, rows 1 to 3 in Table 6.4), these rewritings are obtained by replacing the deleted relation R_2 with a *subset* relation. The degrees of divergence of the rewritings go down as the sizes of the replacement relations go up (column four in the table), but the view maintenance cost of the legal rewritings increases with the cardinality of the substituted relations (column five). Therefore, the overall efficiency of these rewritings depends on the trade-off parameters. For Case 1, V_3 is the best among the three. For Cases 2 and 3, i.e, when the view maintenance costs are weighted heavier, then V_1 is ranked higher by the efficiency model.

6.4.5 Experiment 5: Workload Models

While our previous experiment computed the view maintenance cost for a single data update, Experiment 5 considers the effect of the four different workload models

M1, M2, M3 and M4 as defined in Section 6.3.6 for computing the view maintenance cost within a period of time.

Workload model M1 assumes a number of updates which is proportional to the size of a relation. In order to evaluate the influence of this workload model on our findings, we look again at Experiment 4, which compares five different legal rewritings with different relation sizes (cardinalities). With the cardinalities of the replacing relations growing from rewriting V_1 through V_5 (i.e., from left to right in Figure 6.8), the number of updates in this workload model grows proportionally. As an example, we assume a ratio of 1 update per 100 tuples. This gives us the situation in Table 6.5.

Rewriting	DD	Cost	# of updates	Normalized Cost	$QC(V_i)$	Rating
V_1	0.075	842.3	20	0	0.9325	3
V_2	0.0375	1193.3	30	0.25	0.94125	2
V_3	0.00	1544.3	40	0.5	0.95	1
V_4	0.027	1895.3	50	0.75	0.898	4
V_5	0.045	2246.3	60	1	0.855	5

Table 6.5: Ranking of Legal Rewritings for Experiment 5. (Workload Model M1)

We observe that a workload model only influences the *cost* (and not the *quality*) of a legal rewriting, and in the case of M1, the influence is proportional to the relation size. However, since our model normalizes the cost factor before combining it into the overall efficiency measure, both the normalized cost factors and hence the final efficiency values are unchanged.

The workload model M2 assumes a constant number of updates per relation, i.e., a total number of updates proportional to the number of relations in a view. As there is no direct correlation between the number of relations in a view and the quality of the view, a general evaluation would not be very meaningful. However, assuming we could find other rewritings with the same quality but with a different number of relations, then the workload model M2 would encourage us to pick a legal rewriting with the fewest relations possible as the cost factor would be most reduced.

The workload model M3 assumes that there are n updates per information source per time unit. That is, if a view is specified over m information sources, then there is a total of $m \cdot n$ updates for the view per time unit. Extending Experiment 2, the results obtained from this experiment are shown in Table 6.6 and in Figure 6.9.

Rewriting	# sites	# updates	CF_M	CF_T	$CF_{I/O}$
V_1	1	10	30	8000	310
V_2	2	20	92	27200	620
V_3	3	30	186	57600	930
V_4	4	40	312	99200	1240
V_5	5	50	470	152000	1550
V_6	6	60	660	216000	1860

Table 6.6: Assessments of Legal Rewritings for Experiment 5.

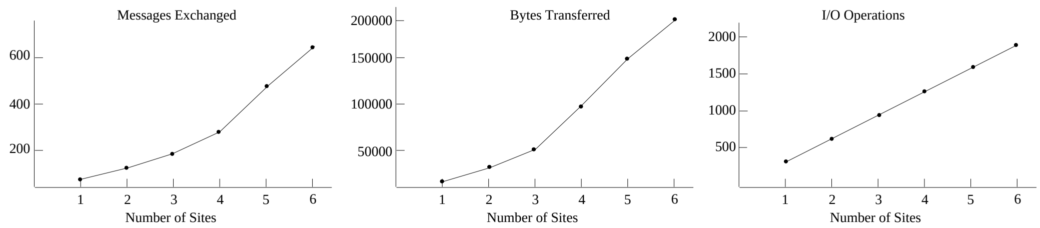


Figure 6.9: Results of Assessing Legal Rewritings for Experiment 5.

Under this experimental setup, our efficiency model would favor a legal rewriting that has the smallest possible number of information sources involved in the view definition to replace the affected view definition. Under this workload model, a small number of information sites has the advantage of a small number of updates, plus the overall efficiency is also better for rewritings with a smaller number of ISs.

Workload model M4 (i.e., having a fixed number of data updates for each legal rewriting) gives the same results as when considering a single data update, since a fixed number is multiplied with the single data update view maintenance cost for each scenario (as discussed in Experiment 2). Hence, it need not be considered any further here.

6.4.6 Heuristics

Based on the findings in Sections 6.2 and 6.3, we identify some heuristics that can help a view synchronizer to pick a good legal rewriting without having to compute all possible legal rewritings for a certain capability change.

As Equation (6.10) suggests and we now confirmed with Experiments 2 and 3, a view optimizer would prefer a legal rewriting with a smaller number of information sources and with relations with smaller cardinalities.

Experiment 4 supports the following heuristic: A view optimizer should choose a legal rewriting whose view component replacement is as close as possible to the original view component in terms of size. For example, when a relation R is deleted from its site, if two relations S and T are both legitimate replacements for R and we know $T \supseteq S \supseteq R$, then we would select relation S as the replacement since its size is closer to the size of the original relation.

Experiment 5 supports some findings on heuristics for different workload models (cf. also Section 6.3.6). For the workload model M1 (number of updates proportional to relation size), we would prefer a legal rewriting that refers to smaller relations, i.e., we would use the smallest relation that provides a satisfactory amount of information to the view user. For the models M1 and M2 (constant number of updates per relation), and M3 (constant number of updates per IS), we would aim to minimize the number of information sites referred in a view. Even in workload model M4 (globally constant number of updates for a legal rewriting) with the number of information sites fixed, we would minimize the number of joins or the number of primitive clauses in the WHERE clause.

Assuming that data updates are equally likely to occur at each relation (model M2), we can find an intuitive way to minimize the *number of messages* exchanged. All other parameters equal, the number of messages exchanged between the information space and the view site is minimized when the number of information sources referred in V is minimized, since no messages have to be sent between information sources when relations are located in the same IS. That is, a legal rewriting is chosen over other legal rewritings if the number of information sources involved in its view definition is smaller.

Lastly, we would choose a legal rewriting with a smaller number of relations referred to in the FROM clause, e.g., even if only one attribute is deleted from a relation R, we would replace R entirely if an appropriate relation can be found that already participates in the view definition (this is valid for all workload models).

6.5 Summary

In this chapter, we focussed on performance issues raised by view synchronization. Since view evolution under capability changes of underlying data sources will generate a large number of possible *rewritings* for an original view definition, it is necessary to compare these rewritings and identify the *best* solution to maintaining a view. A novel measure of *efficiency* is introduced that explores the two dimensions of *quality* and *cost* and leads to the definition of the QC-Model. This model can be used to establish a ranking among alternate legal view rewritings for an affected view definition. It turns out that a ranking is possible among seemingly incomparable solutions using the efficiency model we developed, and that it is feasible to introduce *parameters* to trade off quality against cost (and also sub-dimensions of either against each other).

Since the number of view rewritings is potentially large, we also discussed heuristics that can be used to prune the search space for the best legal rewriting for a given view, information space, and capability change. For instance, it is always preferable to select a legal rewriting that refers to a minimal number of information sources. Also, a rewriting that replaces a dropped relation with a relation similar in size is always preferred. The experiments that we ran using our system and varying different parameters of the information space (database size, number of information sources, "strictness" of view definition in E-SQL) support our findings.

CHAPTER VII

Related Work

We are the first to study the problem of view adaptation in the data warehousing approach caused by capability changes of participating information sources [60, 30, 40, 48, 59, 31, 39, 27, 46, 47, 49, 26].

In Chapter I [60], we establish a taxonomy of view adaptation problems that identifies alternate dimensions of the problem space, and hence serves as a framework for characterizing and distinguishing our view synchronization problem from other (previously studied) view adaptation problems. In Chapter II of this dissertation [30], we define the overall *EVE* framework and introduce the key concepts of the *EVE* approach: *Evolvable*-SQL (E-SQL) view definition language for SPJ and aggregation views; the *Model for Information Source Description* (MISD); the *semantics* of view synchronization. We present the research issues associated to the view synchronization problem in Chapter III by defining the dimensions that characterize the view synchronization problem space [60]. Then, in the same chapter, we give a complete set of solutions for SPJ and aggregation view synchronization when *only* containment constraints are specified among ISs [48, 59]. Chapter IV [40] attempts to extend the types of constraints used in the synchronization process by providing solutions for *complex replacement* strategies that find suitable legal rewritings in cases when containment-based techniques would fails. In Chapter V [47] we study the problem of view *maintenance after synchronization* and conduct extensive experimental evaluation of our proposed maintenance strategies. Since a large set of alternate view rewritings may be found by the proposed view synchronization process, QC-model for ranking candidate legal rewritings is introduced in Chapter VI. Issues of how to estimate the sizes of candidate legal rewritings based on the constrains used to obtained them, are discussed and experimental results are presented for our solution.

While no one has addressed the view synchronization problem as we do, there is research done in the general context of data warehousing that relates to *EVE* approach. We outline below, divided into topics areas, some of these related works.

Information Integration Systems.

EVE system can be seen as an information integration system using view technology to gather and customize data across heterogeneous information sources. On this venue, related work that addresses the problem of information integration are among others the SIMS [3] and SoftBot [9] projects. In the SIMS project, a unified schema is apriori defined and the user interaction with the system is via queries posed against the unified schema. Although addressing different issues, SIMS's process of translating a user query into subqueries targeting external relations raises some of the same problems as finding the right substitution for an affected view component in *EVE*. The SoftBot project has a very different approach to query processing as they assume that the system has to discover the "link" among data sources that are described by action schemas. While related to our view synchronization algorithms CVS, the SoftBot planning process also has to discover connections among information sources when very different source description languages are used. None of the two projects address the particular problem of evolution under capability changes of participating external information sources.

In the work of Levy et al. [35], a global information system is designed using the world-view approach where the external information sources are described relative to the unified world-view relations. The language used here to describe external relations relative to the world-view schema parallels our MKB description language, except the fact that we don't have an apriori defined schema. Further, we introduce the concept of a join constraint in our model that allows expressing default conditions among external relations that should be used by the system to attempt to integrate information instead of evaluating (blindly) all possible Cartesian combinations based on value matches (full disjunction) [44, 45]. The problem of view evolution as posed by our work, i.e., that the world view itself may evolve, is not discussed in [35].

Papakonstantinou et al. [52, 51] are pursuing the goal of information gathering across multiple sources. Their proposed language OEM assumes queries that explicitly list the source identifiers of the database from which the data is to be taken. Like our MISD model, their data model allows information sources to describe their capabilities, but they don't assume that these capabilities could be changed and thus they do not address the view synchronization problem.

In our previous work [45, 44], we have proposed the Dynamic Information Integration Model (DIIM) to allow information sources to dynamically participate in an information integration system. The DIIM query language allows loosely specified queries that the DIIM system refines into executable, well-defined queries based on the capability descriptions each information source exports when joining the DIIM system. For this, the notion of *connected relations* is introduced as a natural extension of the concept of full disjunction [11]. In the default case when only natural joins are defined in the IS descriptions in the MKB it then can be shown that the semantics of these two concepts (connected rules and full disjunction) are equivalent [44]. AI planning techniques are used in DIIM for query refinement. In *EVE*, instead, we now assume that precise (SQL) queries are used to define views (instead of loosely-specified ones), and thus query refinement in the sense of DIIM is not needed.

View Redefinition.

The problem of view redefinition is recently studied by Gupta et al. [17] and Mohania et al. [37]. They 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. Their algorithms could potentially be applied in the context of our overall framework, once *EVE* has determined an acceptable view redefinition. However, in both approaches [17, 37], the base relations are assumed to remain unchanged (i.e., the data in the base relations is still available during the maintenance process). In our system, a new challenge arises: after a capability change of an autonomous IS, the *EVE* system no longer has access to the old data in the changed relation. For example, after an attribute is dropped from a base relation and the view definition is being evolved accordingly, the *EVE* system can no longer access the the dropped attribute, when attempting to maintain the affected view. 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 local changes of different types. However, 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 local change could potentially be huge. Most importantly, in *EVE,* we can take advantage of semantic information constraints between the old and new view enabling us to treat the complex definition changes as one atomic change.

Query Rewriting using Views.

Research on query reformulation using materialized views Levy et al. [33, 36, 61] 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* (e.g., the new reformulated query could be a subset of the original query). This approach to query reformulation [36] has some similarities with our view synchronization process, but again it is set in a different environment and has different goals. Namely, we have extended the idea of query reformulation by using a well-defined query language E-SQL to specify constraints on query reformulation, thus, when in compliance with those constraints, we allow the view redefinitions to be for example a subset or a superset of the original view. And, if not possible to preserve all view attributes (from the SELECT clause), our view redefinition semantic allows to specify evolution preferences that add flexibility and in the same time let the view definer control the view evolution process.

Transparent Schema Evolution.

A solution for a related problem is given in transparent schema evolution (TSE) project [55, 56, 26], namely, to use view technology to handle schema changes transparently. However, this TSE work is all done in a centralized environment, assuming one single global database that is cooperating, i.e., that is maintaining all information possibly still used by any views defined on top of it. In the TSE framework, a user specifies schema changes against her special-tailored view schema defined over one common base schema. The TSE system is responsible for deriving an alternate view schema to simulate the effects of schema evolution while preserving the current view schemas. When the schema change is capacity-reducing/preserving, TSE derives the target view schema from the original view schema. When the schema change is capacity-augmenting, TSE first translates the schema change request specified on the view schema into an in-place schema change specified on the base schema, augments the base schema, derives the target view schema based on the augmented base schema, and finally restores the original base schema by applying an inverse schema change transformation on the augmented base schema. In both cases, In TSE, the existing view schemas are not affected by schema changes, because the original base schema upon which they all are defined is always preserved. Unlike the problem addressed in this dissertation, a delete operation specified against a view is not actually executed as a delete against the base schema rather simply desired data is hidden from that particular view. Thus the view evolution problem of *EVE* is not an issue in TSE. Furthermore, all view schemas are kept in the system. In *EVE* system, instead we consider schema (capability) changes coming from the information space, and we study the effects of such IS capability changes on existing view's definition. If all of the ISs were willing to preserve the relations that are referenced by the view definitions (i.e., no schema changes were to happen on base relations), then EVE behaves like TSE.

Query Optimization in Distributed Systems.

The problem of *query optimization* has been addressed in many database works, for instance by Jarke et al. [23], van den Berg et al. [67], or Agrawal et al. [2]. These works are concerned with optimizing a given query in order to execute it in an efficient way. View synchronization in the *EVE* environment encounters a different problem, namely we have to select a good (but not necessarily *equivalent*) query among several possible ones. One component of the desirability of a new view is of course the cost of view maintenance after IS data updates. For this, we hence are able to adopt and adapt a measure similar to traditional view maintenance costs [70], measuring different parameters in an incremental update environment. But for our purposes, we also have to look at the quality of different view definitions in comparison to the original query in terms of their degrees of preservation of interfaces and extents, a problem that does not occur in traditional query optimization.

View Maintenance after Data Updates in Distributed Systems.

View maintenance after data updates was extensively studied in the literature [69, 16, 6, 2]. 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 in one of the base relations used in the view. However, in the case of maintenance after synchronization there are three potential changes: (1) a schema-level change in one of the base relations; (2) this generally implies data changes as well in this base relation, and (3) schema changes in the view definition itself (the view definition is synchronized). This distinction makes the two problems complementary and 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.

For the problem of *incremental view maintenance*, a concept which we use in our performance studies, earlier work has been done by several other projects in the literature [8, 18]. Blakeley et al. [6] are concerned with a centralized environment only.

Also, they have looked at incremental view maintenance assuming non-concurrent updates (updates are sufficiently spaced to not interfere with each other, each update reaches the data warehouse before the next update is executed at any of the base relations). Lately, work on concurrent updates has been done. Based on the concept of updates interfering with each other due to long transmission times between base relations and the data warehouse, these works attack increasingly complex scenarios of handling concurrent updates by collecting update information in queues and handling them in batches. Zhuge et al. [70, 72] introduce the ECA algorithm for incremental view maintenance and report on findings on the cost of their algorithm, but in a different environment from ours (a single information source is assumed). They also give a taxonomy for different *levels of correctness* for view maintenance algorithms. A second paper by the same authors ("Strobe", [71]) extends their findings towards multi-source information spaces, but does not incorporate any performance model or cost studies. Agrawal et al. [2] propose the SWEEP-algorithm, which can ensure consistency of the data warehouse in a larger number of cases compared to the Strobe family of algorithms. Finally, [70] contains a performance study similar to ours. However, their work is limited to a comparison between traditional view recomputation and incremental view maintenance algorithms, and does not compare quality and cost between different rewritings for a query, while the latter is the topic of our work.

To determine the *quality* of a view rewriting, we need to estimate sizes of overlapping view extents in order to determine how much information is retained by a new query and how much meaningless new data is introduced. This in some way parallels the concept of *precision and recall* used in the field of information retrieval [57], although it is set in an entirely different context. Information retrieval generally does not deal with selecting subsets of tuples from a typed relation nor with combining such relation fragments via joins into larger result tuples. Rather, the work on precision and recall establishes measures of how well boolean queries perform on textual documents in terms of term similarities and counts.

CHAPTER VIII

Conclusions

8.1 Dissertation Contributions

Our work [60, 30, 40, 59, 48, 28, 50] is the first to study the new view adaptation problem, which we coin *view synchronization* problem, in dynamic information integration systems. This dissertation studies solution approaches for evolution process of both view definition and view extent triggered by capability changes of information sources participating in such environments. The work presented is the first to study in great detail the synchronization based on containment constraints, complex view substitution techniques and maintenance of the view extent after synchronization. The *Evolvable View Environment* framework is proposed as a generic approach to solve issues related to view evolution under capability changes for both view definition adaptation and view extent maintenance after synchronization. In order to find alternative replacements for components of a view affected by IS capability changes, *EVE* system maintains descriptions of the capabilities of ISs as well as interrelationships between ISs in a meta-knowledge base. Equipped with *Evolvable-SQL* and the model for information source descriptions, we propose a complete set of solutions for select-project-join and aggregation view synchronization in the presence of alternative constraints specified among participating ISs. Maintenance strategies for view extent adaptation after view synchronization are designed based on different system parameters such as availability of base relations and types of evolution parameters.

To summarize, the dissertation makes the following contributions:

- *The Design of a Model for Information Description Language* that captures content and capability properties of heterogeneous information sources.
- *The Development of a View Definition Language* that allows view users to express preferences about the view evolution process.
- *The Introduction of the Concept of synch-equivalence* between E-SQL specifications to express the real evolution flexibility of an E-SQL view definition.
- *The Definition of the View Evolution Semantics* that lay the foundation of what is a legal rewriting the synchronization process could produce, for both SPJ and aggregation views.
- *The Design of Algorithms for View Synchronization* that achieve the proposed view evolution semantics in the presence of different types of constraints specified for underlying information sources, for SPJ and aggregation E-SQL views as well.
- *The Proposal of a Cost Model* based on which view redefinitions found by the synchronization process could be analytically ranked based on the user/application requirements.
- *The Development of Materialized View Maintenance after View Synchronization* techniques that incrementally adapt view extents after redefinition.

- *The Implementation of a TPC-D Benchmark Testbed* for performance evaluation of the view maintenance algorithms and comparison with alternative techniques for view maintenance.
- *The Design and Implementation of a Working System* for information integration with support for view evolution under capability changes in distributed environments.

This dissertation has opened up a new direction of research by identifying view synchronization as an important and so far unexplored problem of current view technology in dynamic large-scale environments such as the WWW. This work has laid a solid foundation for addressing the new problem of how to maintain views in data warehousing system, and is thus likely to be beneficial for many diverse applications such as web-based information services and electronic catalog providers.

8.2 Future Work

View adaptation issues are important in most of the information systems that are based on customized integration of data across dynamic, highly heterogeneous large information spaces. Such applications could be found in very diverse domains ranging from business (e.g., trade-market analysis, etc), and libraries (e.g., museums, multimedia on-line resources) to science (e.g., decision support applications, integration of scientific results, etc.).

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 dissertation. We list some of these problems below:

- Extend the current *EVE* approach to E-SQL view other than SPJ and aggregation views which are treated in this dissertation. For example, an interesting type of views for which synchronization could potentially be applied is views using data-cube operator [13, 38, 21]. Applications using the N-dimensional generalization of group-by and aggregation operators are mostly defined in data warehousing context, and hence must define mechanisms to evolve the view definitions and their extents under capability changes.

- Apply *EVE* approach to other data models, for example, object-oriented, hierarchical, object-relational, as common models for information source descriptions. The characteristics of these models such as inheritance, may enhance the expressiveness of of the middle layer.

- Explore other semantic constraints that could be used in the process of view synchronization, for example functional dependencies, or in a hierarchical model, class-subclass relationships.

- Extend the model for information source description (MISD) to accommodate query capabilities (i.e., query templates) of a larger class of information sources. We assume in this dissertation that all participating ISs (via their wrappers) support SQL queries which for many types of ISs (such as unstructured data sources) may not be possible.

- Incorporate the use of the QC-model defined in this work within the synchronization process for reduction of the search space of the synchronization algorithms. In the current *EVE* approach, we assume that *all* legal rewritings are found and *then* the cost model is applied to choose the "best" candidate.

- Combine 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 [70, 72]. We believe that more complex strategies must be designed to deal with simultaneous capability and data changes in underlying information sources.

APPENDICES

APPENDIX A

The Proofs for POC Theorems

A.1 Proof for Theorem 5, Section 3.4.

Proof.

V' defined in Equations (3.10) and (3.11) has the properties P1 and P2 from Definition 1: i.e., V' is no longer affected by the capability change "*delete*−*relation* R" (it no longer uses R). Property P4 imposes that individual evolution parameters for each component be satisfied in V'. Indeed, in V' the components replaced are relation R in the FROM clause, all attributes in $Attr(V(R))_{\mathsf{SELECT}}(f,t)$ and $Attr(V(R))_{\mathsf{WHERE}}(f,t)$; and, some of the attributes in $Attr(V(R))_{\mathsf{SELECT}}(t,t)$ and $Attr(V(R))_{\mathsf{WHERE}}(t,t)$. All these substitutions satisfy the evolution parameters of the replaced components. The attributes in $Attr(V(R))_{\mathsf{SELECT}}(t,f)$ and the attributes not replaced by $S(Attr(V(R))_{\mathsf{SELECT}}(t,t))$ in the SELECT clause of the view V were dropped in V' which is in agreement with their evolution parameters. The conditions in the WHERE clause of the view V referring to the attributes in $Attr(V(R))_{\mathsf{WHERE}}(t,f)$ and $Attr(V(R))_{\mathsf{WHERE}}(t,t)$ not replaced by $S(Attr(V(R))_{\mathsf{WHERE}}(t,t)$ were dropped as well in V' making their evolution parameters satisfied.

We are left then to prove that the property P3 from Definition 1 is satisfied as well. Namely, we have to prove that $V' \ \delta_\pi \ V$. We consider in the following cases corresponding to each of the possible values for the view-extent evolution parameter $\mathcal{VE} = \delta$.

Condition 3, Case 1: $\mathcal{VE} = \delta = \subseteq$.

We have to prove that for $\phi \in \{\equiv, \subseteq\}$ in Equation (3.5), Condition 1, V' is a π-subset of V, i.e., $V' \subseteq_\pi V$ ($\subseteq_\pi$ was defined in Figure 2.7). For this case, Condition 3, Case 1, imposes that all attributes of R that appear in the WHERE clause are replaced by attributes of S. That is

(0) $\mathcal{CV}'(S(Attr(V(R))_{\mathsf{WHERE}}(f,t)), S(Attr(V(R))_{\mathsf{WHERE}}(t,t)), \bar{W})$ has the same set of primitive clauses as

$\mathcal{CV}(Attr(V(R))_{\mathsf{WHERE}}(f,t), Attr(V(R))_{\mathsf{WHERE}}(t,t), \underbrace{Attr(V(R))_{\mathsf{WHERE}}(t,f)}_{\emptyset}, \bar{W})$[1].

Or, put it differently, that $|Attr(V(R))_{\mathsf{WHERE}}(t,t)| = |S(Attr(V(R))_{\mathsf{WHERE}}(t,t))|$ and $Attr(V(R))_{\mathsf{WHERE}}(t,f) = \emptyset$ (by Condition 3, Case 1).

Let t' be a tuple in the view V', $t' \in V'$. Then there exists some tuples in S, $R_1, \ldots, R_n$ that derive the tuple t' in V'. I.e., the following properties hold[2]:

(1) $\exists t_S \in S$, such that $t'[S(Attr(V(R))_{\mathsf{SELECT}})] = t_S[S(Attr(V(R))_{\mathsf{SELECT}})]$,

(2) for all $1 \leq i \leq n$, $\exists t_i \in R_i$, such that $t'[R_i.\bar{D}_i] = t_i[R_i.\bar{D}_i]$,

(3) $t_S, t_1, \ldots t_n$ derive t' in V',

(4) $\mathcal{CV}'(t_S[S(Attr(V(R))_{\mathsf{WHERE}})], t_1, \ldots t_n)$ is satisfied.

From Condition 1, Equation (3.5), and Conditions 2 and 3, we have that the attributes of S used in the new view definition are among the ones used in the Equation (3.5). I.e.,

(5) $(S(Attr(V(R))_{\mathsf{SELECT}}(f,t) \cup S(Attr(V(R))_{\mathsf{SELECT}}(t,t)),\ S(Attr(V(R))_{\mathsf{WHERE}}(f,t)) \cup$

[1]If some of the clauses in the WHERE clause are dropped in V' then in general we cannot prove $V' \subseteq_\pi V$.

[2]$S(Attr(V(R))_{\mathsf{SELECT}})$ denotes replacements for the attributes $Attr(V(R))_{\mathsf{SELECT}}(f,t)$ and $Attr(V(R))_{\mathsf{SELECT}}(t,t)$. $S(Attr(V(R))_{\mathsf{WHERE}})$ denotes replacements for attributes $Attr(V(R))_{\mathsf{WHERE}}(f,t)$ and $Attr(V(R))_{\mathsf{WHERE}}(t,t)$.

$S(Attr(V(R))_{\text{WHERE}}(t,t))\ ) \subseteq S.\bar{B}$.

From Equation (3.9), ϕ must be in $\{\equiv, \subseteq\}$ when $\mathcal{VE} = \subseteq$. Then from Equation (3.5) we have that there exists a tuple $t_R \in R$ such that t_S is π-equal to t_R:

(6) $t_S[S.\bar{B}] = t_R[R.\bar{A}]$.

We want to show that $t_R \in R$, $t_1 \in R_1, \ldots, t_n \in R_n$ derive a tuple t in V such that t is π-equal to t', $t =_\pi t'$.

From (0), (5) and (6) we have that

(7) $t_S[S(Attr(V(R))_{\text{WHERE}})] = t_R[Attr(V(R))_{\text{WHERE}}]$[3] and $t_S[S(Attr(V(R))_{\text{SELECT}})] =_\pi t_R[Attr(V(R))_{\text{SELECT}}]$ where $S(Attr(V(R))_{\text{SELECT}}) \cup S(Attr(V(R))_{\text{WHERE}})$ are all attributes in $S.\bar{B}$ that replace attributes of R (they must include at least the indispensable and replaceable attributes of R as enforced by Condition 2).

From (0),(4) and (7) we have

(8) $\mathcal{CV}(t_R[Attr(V(R))_{\text{WHERE}}], t_1, \ldots t_n)$ is satisfied.

Then from (8) we can deduce that

(9) the tuples $t_R \in R$, $t_1 \in R_1, \ldots, t_n \in R_n$ derive a tuple t in V.

Now let's prove that $t =_\pi t'$. From (2) and (9) we have that

(10) for all $1 \le i \le n$, $(t_i \in R_i)$, $t'[R_i.\bar{D}_i] \stackrel{(2)}{=} t_i[R_i.\bar{D}_i] \stackrel{(9)}{=} t[R_i.\bar{D}_i]$.

From (1) and (7) we have that

(11) $t'[S(Attr(V(R))_{\text{SELECT}})] \stackrel{(1)}{=} t_S[S(Attr(V(R))_{\text{SELECT}})] \stackrel{(7)}{=_\pi} t_R[Attr(V(R))_{\text{SELECT}}] \stackrel{(9)}{=} t[Attr(V(R))_{\text{SELECT}}]$.

In (10) and (11) we have proven that $t =_\pi t'$. Since t' was an arbitrary chosen tuple of V', we have proven that $V' \subseteq_\pi V$.

<u>Condition 3, Case 2: $\mathcal{VE} = \delta = \supseteq$.</u>

We have to prove that for $\phi \in \{\equiv, \supseteq\}$ in Condition 1, Equation (3.5), V is a

[3] From (0) we have that $|S(Attr(V(R))_{\text{WHERE}})| = |Attr(V(R))_{\text{WHERE}}|$.

subset of V'. I.e., $V' \supseteq_\pi V$.

Let t be a tuple in V, $t \in V$. Then there exist some tuples in R, $R_1, \ldots, R_n$ that derive t in V. Thus the following properties are true:

(1) $\exists t_R \in R$, such that $t[Attr(V(R))_{\mathsf{SELECT}}] = t_R[Attr(V(R))_{\mathsf{SELECT}}]$,

(2) for all $1 \leq i \leq n$, $\exists t_i \in R_i$, such that $t[R_i.\bar{D}_i] = t_i[R_i.\bar{D}_i]$,

(3) $t_R, t_1, \ldots t_n$ derive t in V,

(4) $\mathcal{CV}(t_R, t_1, \ldots t_n)$ is satisfied.

From Condition 1 (Equation (3.5)) (with $\phi \in \{ \equiv , \supseteq \}$), there exists $t_S \in S$ such that

(5) $t_R[R.\bar{A}] = t_S[S.\bar{B}]$.

Property (5) implies

(6) $t_S[S(Attr(V(R))_{\mathsf{WHERE}})] =_\pi t_R[Attr(V(R))_{\mathsf{WHERE}}]$,

(7) $t_S[S(Attr(V(R))_{\mathsf{SELECT}})] =_\pi t_R[Attr(V(R))_{\mathsf{SELECT}}]$.

(In (6) and (7) we use π-equality sign ($=_\pi$) as not all attributes from $Attr(V(R))_{\mathsf{SELECT}}$and $Attr(V(R))_{\mathsf{WHERE}}$have replacements in $S(Attr(V(R))_{\mathsf{SELECT}})$ and $S(Attr(V(R))_{\mathsf{WHERE}})$, respectively.)

We want to prove that the tuples $t_S, t_1, \ldots, t_n$ derive a tuple t' in V', and this tuple is equal to t, i.e., $t' =_\pi t$.

Property (6) implies that

(8) $\mathcal{CV}'(t_S[S(Attr(V(R))_{\mathsf{WHERE}})], t_1, \ldots, t_n)$[4] is satisfied, because t_S replaces exactly t_R for which WHERE clause is satisfied (Property (4)).

Property (8) states that the tuples $t_S, t_1, \ldots, t_n$ satisfy the condition from the WHERE clause of the view V'. Thus this set of tuples derives a tuple t' in V'. From (1), (2), (3), (6) and (7) we have that t' is equal to t. More precisely,

[4]Note that this case can be proven in general when $\mathcal{CV}'$ is obtained from $\mathcal{CV}$ by dropping some of the conditions and replacing the attributes of R.

(9) $t'[S(Attr(V(R))_{\text{SELECT}})] = t_S[S(Attr(V(R))_{\text{SELECT}})] \overset{(7)}{=}_\pi t_R[Attr(V(R))_{\text{SELECT}}]$ $\overset{(1)}{=} t[Attr(V(R))_{\text{SELECT}}]$,

(10) for all $1 \leq i \leq n$, $t'[R_i.\bar{D}_i] = t_i[R_i.\bar{D}_i] \overset{(2)}{=} t[R_i.\bar{D}_i]$.

Hence, we can conclude that $t' =_\pi t$. Since t was chosen arbitrary from V, we have proven that $V' \supseteq_\pi V$.

<u>Condition 3, Case 3: $\mathcal{VE} = \delta = \equiv$.</u>

We want to show that $V' \equiv_\pi V$ when $\phi = \equiv$ in Condition 1, Equation (3.5). Hence, we have to prove two inclusions:

(I) $V' \subseteq_\pi V$ and

(II) $V' \supseteq_\pi V$.

The inclusion (I) is implied by Case 1 proven above when $\phi = \equiv$ with the restriction imposed in (0). Similarly, the inclusion (II) is implied by Case 2 with $\phi = \equiv$. Then, we conclude that $V \equiv_\pi V'$.

Note that for the aggregation view V' defined in Equation (3.11), we can compare the aggregation attributes defined by aggregation functions that are **duplicate-insensitive**, such as MIN, MAX, and $COUNT(DISTINCT)$. These relationships are shown in Table (3.2) for the three aggregation functions MIN, MAX, and $COUNT(DISTINCT)$. When $V'\ supseteq_\pi \mid\ \equiv_\pi\ V$, all old groups from V are still in the new view V', with possible new tuples added to each of them. This is a consequence of the fact that the relation S is a superset of or equal to the to-be-deleted relation R, and hence it brings more tuples in the view V'; and, to the fact that by dropping some of the attributes from the SELECT list, two or more groups of V can become a new group in V' (and hence, each group of V' has more tuples than the corresponding groups of V). This property of more tuples in each group of the new view V', has immediate consequence on the aggregation functions applied to that group. For example, if we compute $COUNT(DISTINCT)$ in one of the groups of V', its result is bigger than or equal to the result of $COUNT(DISTINCT)$ applied to the corresponding group of the view V. The same can be said for the functions MAX and MIN.

Q.E.D.

A.2 Proof for Theorem 6, Section 3.4.

Proof.

V' defined in Equations (3.22), (3.23) and (3.20), (3.21) has properties P1 and P2 from Definition 1: i.e., V' is no longer affected by the capability change "*delete-attribute* $R.A$". Property P4 imposes that individual evolution parameters for each component be satisfied in V'. Indeed, in V' the only components replaced are the attribute $R.A$ in the SELECT clause (if the attribute is in the SELECT clause and it is replaceable), and the replaceable conditions from the WHERE clause referring to the attribute $R.A$. In both cases the replacement was the attribute $S(R.A)$. Thus, only replaceable components were replaced which proves that the replaceable evolution parameters are satisfied.

For the dispensable evolution parameter satisfaction, we have two types of dropped components: (1) in the case the view extent evolution parameter is $\subseteq$ or $\equiv$, the view V' defined in Equation (3.22) was obtained from V by dropping the attributes from $Attr(V(R))_{\mathsf{SELECT}}$which are not in $R.\bar{A}$. According to Equation (3.18) (Condition 3, Case 1), these must be disposable attributes, thus their evolution parameters are satisfied. (2) for the view extent evolution parameters $\approx$ and $\supseteq$, the only components dropped in the view V' (defined in Equation (3.23)) are nonreplaceable conditions from the WHERE clause referring to the attribute $R.A$. Given that the view V is evolvable under the capability change "*delete-attribute* $R.A$", these conditions must be disposable (conforming to Theorem 1). Thus, in this case, the dispensable evolution parameters are satisfied as well.

Finally, we have to prove that the property P3 from Definition 1 is satisfied as well. Namely, we have to prove that $V' \ \delta_\pi \ V$, where the view-extent evolution parameter is $\mathcal{VE} = \delta$. We consider in the following cases corresponding to each of the possible values for this parameter, i.e., $\delta \in \{\subseteq, \supseteq, \equiv\}$.

<u>Condition 3, Case 1: $\mathcal{VE} \ = \ \delta \ = \ \subseteq$.</u>

In this case, ϕ from Equation (3.5), Condition 1, must be in $\{ \equiv , \subseteq \}$, i.e.,

(0) $\pi_{\bar{B}}(S) \ \subseteq | \equiv \ \pi_{\bar{A}}(R)$.

We have to prove that V' is a π-subset of V, i.e., $V' \subseteq_\pi V$.

Let $t' \in V'$ be a tuple in the view V' defined in Equation (3.22). Then there must exist some tuples in the relation R', S, $R_1, \ldots R_n$ such that:

(1)$\exists t_{R'} \in R'$, $\exists t_S \in S$, $\exists t_1 \in R_1$, ..., $\exists t_n \in R_n$ such that t' is derived from $T_{R'}, t_S, t_1, \ldots, t_n$.

(2) If the attribute $R.A$ was replaced in the SELECT clause then $t'[S.B] = t_S[S.B]$.

(3) For all $i = \overline{1, n}$, $t'[R_i.\bar{D}_i] = t_i[R_i.\bar{D}_i]$.

(4) $\mathcal{CV}(t_S[S.B], t_{R'}[Attr(V(R))_{\text{WHERE}} \setminus \{R.A\}], t_1[\bar{W}], \ldots, t_n[\bar{W}])$ is satisfied.

(5) $t_{R'}[R'.\bar{A}' \cap Attr(V(R))] = t_S[S.\bar{B}' \cap S(Attr(V(R)))]$ is satisfied.

From (0) and (1) we have that

(6) $\exists t_R \in R$ such that $t_S[S.\bar{B}] = t_R[R.\bar{A}]$.

We want to show that (I) the tuples $t_R \in R$, $t_1 \in R_1$, ..., $t_n \in R_n$ derive a tuple t in the original view V such that (II) $t =_\pi t'$. For (I), it is enough to show that the condition $\mathcal{CV}$ from the WHERE clause of the view V (defined in Equation (3.49)) is satisfied by the tuples $t_R, t_1, \ldots, t_n$.

In Condition 3, Case 1, Equation (3.16) we give the condition that all attributes of relation R used in the WHERE clause are among $R.\bar{A}$. Then from (5) and (6) we have:

(7) $t_R[R.A] \stackrel{(6)}{=} t_S[S.B]$ and

(8) $t_R[Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.A\}] \stackrel{(6)}{=} t_S[S(Attr(V(R))_{\mathsf{WHERE}}) \setminus \{S.B\}]$
$\stackrel{(5)}{=} t_{R'}[Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.A\}]$.

Hence (4), (7) and (8) imply (I), i.e.,

(9) $\mathcal{CV}(t_R[R.A], t_R[Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.A\}], t_1[\bar{W}], \ldots, t_n[\bar{W}])$ is satisfied.

Let t be the tuples derived by $t_R, t_1, \ldots, t_n$. Let's now prove (II) from above, i.e., $t =_\pi t'$.

If the attribute $R.A$ is replaceable in the SELECT clause of the view V, then from (2), (5) and (6) we have:

(10) $t[R.A] = t_R[R.A] \stackrel{(6)}{=} t_S[S.B] \stackrel{(2)}{=} t'[S.B]$ and

(11) $t[Attr(V(R))_{\mathsf{SELECT}} \cap R.\bar{A}] \stackrel{(6)}{=} t_S[S(Attr(V(R))_{\mathsf{SELECT}}) \cap S.\bar{B}] \stackrel{(5)}{=} t_{R'}[Attr(V(R))_{\mathsf{SELECT}} \cap R.\bar{A}]$.

The equalities (10) and (11) proves that t and t' have the same values for the common attributes of the views V and V', thus we can conclude $t =_\pi t'$.

Condition 3, Case 2: $\mathcal{VE} = \delta = \supseteq$.

In this case, ϕ from Equation (3.5), Condition 1, must be in $\{ \equiv , \supseteq \}$, i.e.,

(0) $\pi_{\bar{B}}(S) \ \supseteq | \equiv \ \pi_{\bar{A}}(R)$.

We have to prove that V' (defined in Equation (3.23)) is a π-superset of V, i.e., $V' \supseteq_\pi V$.

Let $t \in V$ be a tuple in the view V defined in Equation (3.49). Then there must exist some tuples in the relations R, $R_1, \ldots R_n$ such that:

(1)$\exists t_R \in R$, $\exists t_1 \in R_1$, $\ldots$, $\exists t_n \in R_n$ such that t is derived from $t_R, t_1, \ldots, t_n$.

(2) If the attribute $R.A$ is in the SELECT clause of V then $t[R.A] = t_R[R.A]$ and $t[Attr(V(R))_{\mathsf{SELECT}} \setminus \{R.A\}] = t_R[Attr(V(R))_{\mathsf{SELECT}} \setminus \{R.A\}]$.

(3) For all $i = \overline{1,n}$, $t[R_i.\bar{D}_i] = t_i[R_i.\bar{D}_i]$.

(4) $\mathcal{CV}(t_R[R.A], t_R[Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.A\}], t_1[\bar{W}], \ldots, t_n[\bar{W}])$ is satisfied.

For the tuple $t_R \in R$ defined above, and from (0), it is implied that

(5) $\exists t_S \in S$ such that $t_S[S.\bar{B}] = t_R[R.\bar{A}]$.

Let $t_{R'} \in R'$ be the tuple obtained from t_R when the attribute $R.A$ was dropped. We want to prove that the tuples $t_{R'} \in R'$, $t_S \in S$, $t_1 \in R_1, \ldots t_n \in R_n$ derive a tuple t' in the view V' and this tuple is π-equal to the tuple t from the original view V.

Note again that, as in the case of the capability change "*delete-relation* R", for the aggregation view V' defined in Equations (3.20) and (3.21) we can compare the aggregation attributes defined by aggregation functions that are **duplicate-insensitive**, such as MIN, MAX, and $COUNT(DISTINCT)$. These relationships are shown in Tables 3.6 and 3.7. The same arguments we give in the proof of Theorem 5 in Appendix A.1 are valid for the capability change "*delete-attribute* $R.A$".

Q.E.D.

APPENDIX B

The Proofs for SPOC Theorems

B.1 Proof for Theorem 7, Section 3.5.4.

Proof.

V' defined in Equations (3.45) and (3.46) has properties P1 and P2 from Definition 1: i.e., V' is no longer affected by the capability change "*delete−relation R*" (it no longer uses R). Property P4 imposes that individual evolution parameters for each component be satisfied in V'. In V' the components replaced are relation R in the FROM clause, all attributes in $Attr(V(R))_{\mathsf{SELECT}}(f,t)$ and $Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(f,t)$; and, some of the attributes in $Attr(V(R))_{\mathsf{SELECT}}(t,t)$ and $Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(t,t)$. All these substitutions satisfy the evolution parameters of the replaced components. The attributes in $Attr(V(R))_{\mathsf{SELECT}}(t,f)$ and the attributes not replaced by $S(Attr(V(R))_{\mathsf{SELECT}}(t,t))$ in the SELECT clause of the view V were dropped in V' as they are dispensable. The conditions in the WHERE clause of the view V referring to the attributes in $Attr(V(R))_{\mathsf{WHERE}}(t,f)$ and $Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(t,t)$ not replaced by $S(Attr(V(R))^{\mathrm{JOIN}}_{\mathsf{WHERE}}(t,t))$ were dropped as well in V' making their evolution parameters satisfied.

It remains to show: (I) select conditions on R in V have their evolution parameters satisfied in V' (e.g., replaceable conditions were replaced by conditions in $\mathcal{C}_V(S.\bar{B}_2)$ or

$\mathcal{C}_S(S.\bar{B}_1)$, Equation (3.45)) and (II) the view-extent evolution parameter is satisfied for V'.

<u>Condition 3, Case 1: $\mathcal{VE} = \subseteq$.</u>

From Condition 3, Equation (3.40), ϕ in the $\mathcal{PC}$-constraint (Equation (3.27)) must be $\subseteq$ or $\equiv$. Below, we prove that in this case the above (I) and (II) are true.

(I) First, let's show that any select condition on R in V (i.e., the ones using attributes in $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}$) have their evolution parameters satisfied. Condition 3 imposes in Equation (3.35) that no select conditions have evolution parameters (t, f). The fact the view V is evolvable (Theorem 1) under "*delete−relation R*" capability change implies that no conditions in the WHERE clause have evolution parameters (f, f). For select conditions with evolution parameters (f, t) and (t, t), Condition 3 , Case 1, Equation (3.37) imposes that any attribute $R.C$ referred in one of such conditions $CV(R.C)$ is either (RA) replaced by an attribute in $S.\bar{B}$ or (RC) the condition using $R.C$ is implied by a condition in the $\mathcal{PC}$-constraint. For case (RA), Step 2 of the SPOC algorithm augments the $\mathcal{PC}$-constraint with the condition $CV(S(R.C))$ (among others), condition which is part of the expression $\mathcal{PC}''(S)$ (Equation (3.42)). Thus, the condition $CV(S(R.C))$ replaces the condition $CV(R.C)$ in V' which is consistent to the evolution parameters of this condition. In case (RC), the condition $CV(R.C)$ is replaced by conjunction of the conditions in $\mathcal{C}_S(S.\bar{B}_1)$ in V' (Equation (3.45)), again consistent to this condition's evolution parameters.

(II) We have to show that view-extent evolution parameter $\mathcal{VE} = \subseteq$ is satisfied. That is $V' \subseteq_\pi V$. For this, we first show that $\mathcal{PC}''(S) \subseteq_\pi V(R)$ ($\mathcal{PC}''(S)$ is defined in Equation (3.42), $V(R)$ is defined in Equation (3.32)).

Let t_S a tuple in $\mathcal{PC}''(S)$. From Equation (3.42) (with $\phi \in \{\subseteq, \equiv\}$), there exists a tuple t_R in R, $t_R \in \mathcal{PC}''(R)$, such that:

(0) $t_R[R.\bar{A} \cap (Attr(V(R)) \setminus Attr(V(R))(t,f))] = t_S[S.\bar{B}']$;

(1) t_R satisfies $\mathcal{C}_R(\bar{A}_1)$ and $\mathcal{C}_V((Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}} \setminus Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(t,f)) \cap R.\bar{A})$ (Equation (3.42)): i.e., $\mathcal{C}_R(t_R[\bar{A}_1])$ and

$\mathcal{C}_V(t_R[(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}} \setminus Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(t,f)) \cap R.\bar{A}])$ are true.

From Equation (3.35), we have that $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(t,f)$ is empty, thus (1) could be written as:

(2)$\mathcal{C}_R(t_R[R.\bar{A}_1])$ and $\mathcal{C}_V(t_R[Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}} \cap R.\bar{A}])$ are true.

We want to show that $\mathcal{C}_V(t_R[Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}])$ (Equation (3.32)) is also true. Note that not all conditions in the conjunction $\mathcal{C}_V(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}})$ (Equation (3.32)) are among the ones in $\mathcal{C}_V(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}} \cap R.\bar{A})$ (i.e., not all local attributes of R have replacements in $S.\bar{B}$).

Let a condition $CV(R.C)$ be a local condition on R in $\mathcal{C}_V$ referring the attribute $R.C$. From Condition 3, Case 1, Equation (3.37), $R.C$ is either in $R.\bar{A}$ (case (RA)), thus the condition $CV(R.C)$ is among the ones in $\mathcal{C}_V(Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}} \cap R.\bar{A})$:

(3) $\mathcal{C}_V(t_R[R.C])$ is true.

Or, there exists a condition $C_R(R.C)$ in $\mathcal{C}_R(R.\bar{A}_1)$ that implies the condition $CV(R.C)$ (case (RC) in Equation (3.37)). From (2), $C_R(t_R[R.C])$ is true, then:

(4) $CV(t_R[R.C])$ is true (implied by $C_R(t_R[R.C])$).

From (3) and (4), we conclude that any condition $CV(R.C)$ in $V(R)$ is satisfied by the tuple t_R, thus:

(5) $t_R[R.\bar{A}] \in \pi_{R.\bar{A}}(V(R))$ $(t_R \in_\pi V(R))$.

Properties (0) and (5) imply that the tuple t_S projected on the common attributes is also in $V(R)$:

(6) $t_S[S.\bar{B}'] \in \pi_{R.\bar{A} \cap (Attr(V(R)) \setminus Attr(V(R))(t,f))}(V(R))$ $(t_S \in_\pi V(R))$.

Thus we proved above that $\mathcal{PC}''(S) \subseteq_\pi V(R)$ where common attributes for relation R are all attributes in the WHERE clause, all attributes in $Attr(V(R))_{\text{SELECT}}(f,t)$ and some of the attributes in $Attr(V(R))_{\text{SELECT}}(t,t)$. From this inclusion it is now easy to see that $V' \subseteq_\pi V$: V' (Equation (3.44)) is defined by replacing $V(R)$ in V (Equation (3.32)) with $\mathcal{PC}''(S)$. We showed $\mathcal{PC}''(S)$ is a subset of $V(R)$ (see (6) above) and that it preserves all the attributes used in join conditions between $V(R)$ and R_1, R_2, $\ldots$, R_n. Thus, V' is a π-subset of V where common attributes are all attributes in $Attr(V(R))_{\text{SELECT}}(f,t)$ and some of the attributes in $Attr(V(R))_{\text{SELECT}}(t,t)$.

<u>Condition 3, Case 2: $\mathcal{VE} = \supseteq$.</u>

In Condition 3, Equation (3.40), ϕ in the $\mathcal{PC}$-constraint (Equation (3.27)) must be $\supseteq$ or $\equiv$ when $\mathcal{VE}$ is $\supseteq$. As above, we have to prove that in this case (I) and (II) are true.

(I) Select conditions on R in V (i.e., the ones using attributes in $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}$) have their evolution parameters satisfied in V': (1) conditions with the evolution parameters (t,f) are dropped; (2) some of the conditions with the evolution parameters (t,t) are dropped if there are no replacements for the attributes referred in them; some of the conditions with the evolution parameters (t,t) are replaced (Step 2, SPOC Algorithm) if their attributes have replacements in $S.\bar{B}$; (3) conditions with evolution parameters (f,t) are all replaced: Condition 3 , Case 2, Equation (3.38) imposes that any attribute $R.C$ referred in one of such conditions $CV(R.C)$ is either (RA) replaced by an attribute in $S.\bar{B}$ or (RC) the condition using $R.C$ implies a condition in the $\mathcal{PC}$-constraint. In case (RA), Step 2 of the SPOC Algorithm augments the $\mathcal{PC}$-constraint with the condition $CV(S(R.C))$ (among others), condition which is part of the expression $\mathcal{PC}''(S)$ (Equation (3.42)), hence the condition $CV(S(R.C))$ replaces the condition $CV(R.C)$ in V' which is consistent to the evolution parameters of this condition. In case (RC), the condition $CV(R.C)$ is replaced by conjunction of the conditions in $\mathcal{C}_S(S.\bar{B}_1)$ in V' (Equation (3.45)), again consistent to this condition's evolution parameters.

(II) We have to show that view-extent evolution parameter $\mathcal{VE} = \supseteq$ is satisfied. That is $V' \supseteq_\pi V$. We first show $\mathcal{PC}''(S) \supseteq_\pi V(R)$ ($\mathcal{PC}''(S)$ is defined in Equation (3.42), $V(R)$ is defined in Equation (3.32)).

Let t_R be a tuple in $V(R)$, defined in Equation (3.32), hence:

(0) $\mathcal{C}_V(Attr(V(R))^{\text{LOCAL}}_{\mathsf{WHERE}})$ is satisfied by t_R, i.e., $\mathcal{C}_V(t_R[Attr(V(R))^{\text{LOCAL}}_{\mathsf{WHERE}}])$ is true.

From Condition 3, Case 2, Equation (3.39), $\mathcal{C}_R(R.\bar{A}_1)$ must be satisfied as well by the tuple t_R, i.e.:

(1) $\mathcal{C}_R(t_R[R.\bar{A}_1])$ is true.

From (0) and (1), we imply that t_R satisfies both select conditions from the expression $\mathcal{PC}''(R)$ defined in Equation (3.42), thus:

(2) $t_R[R.\bar{A} \cap (Attr(V(R)) \setminus Attr(V(R))(t,f))] \in \mathcal{PC}''(R)$.

But in Equation (3.42), ϕ is $\supseteq$ or $\equiv$, which implies that it must exists a tuple t_S in S that is π-equal to t_R:

(3) $t_S[S.\bar{B}'] \in \mathcal{PC}''(S)$, and

(4) $t_S[S.\bar{B}'] = t_R[R.\bar{A} \cap (Attr(V(R)) \setminus Attr(V(R))(t,f))]$.

Then we can conclude that $V(R)$ is π-subset of $\mathcal{PC}''(S)$ where the common attributes for relation R are all attributes in the WHERE clause in $Attr(V(R))^{\text{JOIN}}_{\mathsf{WHERE}}(f,t)$ (from Equation (3.34)), all attributes from select clause in $Attr(V(R))_{\mathsf{SELECT}}(f,t)$ (from Equation (3.34)), and some of the rest of the attributes in $Attr(V(R))_{\mathsf{WHERE}}$and $Attr(V(R))_{\mathsf{SELECT}}$. From this inclusion it is now easy to see that $V' \supseteq_\pi V$: V' (Equation (3.44)) is defined by replacing $V(R)$ in V (Equation (3.32)) with $\mathcal{PC}''(S)$. We showed $\mathcal{PC}''(S)$ is a superset of $V(R)$ (see (3) and (4) above). The expression $\mathcal{PC}''(S)$

preserves all the attributes used in join conditions with the evolution parameters (f, t) and only some of the attributes referred in conditions with evolution parameters (t, t). In V' the rest of join conditions on R are dropped which makes V' a superset of V, property consistent to the extent-view evolution parameter $\mathcal{VE} = \supseteq$. Hence, V' is a π-superset of V where common attributes are all attributes in $Attr(V(R))_{\mathsf{SELECT}}(f, t)$ and some of the attributes in $Attr(V(R))_{\mathsf{SELECT}}(t, t)$.

<u>Condition 3, Case 3: $\mathcal{VE} = \equiv$.</u>

We want to show that $V' \equiv_\pi V$ when $\phi = \equiv$ as imposes by Condition 3, Equation (3.40).

Hence, we have to prove two inclusions:

(I) $V' \subseteq_\pi V$ and

(II) $V' \supseteq_\pi V$.

Note that in this case imposes both conditions from Case 1 and Case 2 combined.

The inclusion (I) is implied by Condition 3, Case 1 proven above when $\phi = \equiv$.

Similarly, the inclusion (II) is implied by Case 2 with $\phi = \equiv$ Then, we conclude that $V \equiv_\pi V'$.

Note that, as in the case of the capability change "*delete-relation R*" for POC algorithm, for the aggregation view V' defined in Equation (3.46) we can compare the aggregation attributes defined by aggregation functions that are **duplicate-insensitive**, such as MIN, MAX, and $COUNT(DISTINCT)$. These relationships are shown in the Tables (3.10) and (3.11). The same arguments we give in the proof of Theorem 5 in Appendix A.1 are valid for the SPOC algorithm for the capability change "*delete-relation R.A*".

Q.E.D.

B.2 Proof for Theorem 8, Section 3.5.6.

Proof. We consider a candidate $\mathcal{PC}$-constraint of the type (I) as defined in Section 3.5.6. Namely, the to-be-deleted attribute $R.X$ is not among the attributes of the relation R projected out in the $\mathcal{PC}$-constraint. We have to prove that the view V' defined in Equations (3.66) and (3.67) is a legal rewriting of the view V defined in Equation (3.49) under the capability change "*delete-attribute* $R.X$". Table 3.12 summarizes the conditions under which the synchronization of the view V can be done.

The view V' has properties P1 and P2 from Definition 1: i.e., V' is no longer affected by the capability change "*delete-attribute* $R.X$" (it no longer uses the attribute $R.X$).

Property P4 imposes that individual evolution parameters for each component be satisfied in V'. In V' the only components *replaced* are the local WHERE conjuncts of the view V using the attribute $R.X$ (they are replaced by the conjunction $\mathcal{C}_S(S.\bar{B}_1)$). But from Condition 2, (D2) and Condition 3, (E1) we have that all these local WHERE conjuncts are replaceable as $R.X$ can only appear in the set $Attr(V(R))^{\text{LOCAL}}_{\text{WHERE}}(\_, t)$ and hence their evolution parameters are satisfied. The components of the view V that are *dropped* can be: (A) the expression from the SELECT clause of the view V that refer to $R.X$; in Cases 1 and 3, all SELECT attributes of the relation R that are not in $R.\bar{A}$ (the SELECT clause in these cases have only $(Attr(V(R))_{\text{SELECT}} \cap R.\bar{A}) \setminus \{R.X\}$); (B) the conjuncts from the WHERE clause that refer to $R.X$. For the components of type (A), all such expressions are dispensable in all Cases 1, 2 and 3, as imposed by Condition 2, D1 and Condition 3, (E0). For the components of type (B), we distinguish between Cases 1 and 3, on one side, and Case 2, on the other side.

In Case 2, Condition 2, (D2) states that the attribute $R.X$ appears either in local replaceable WHERE conjuncts or in dispensable WHERE conjuncts. $\mathcal{CV}'$ from Equation (3.67) is obtained by replacing the local replaceable conjuncts with the expression $\mathcal{C}_S(S.\bar{B}_1$. Condition 3, (E4) states that *all* such local conditions are replaced by one of the conjuncts from $\mathcal{C}_S(S.\bar{B}_1$. The rest of dispensable conjuncts referring to $R.X$ are dropped in $\mathcal{CV}'$. Hence, their evolution parameters are satisfied in this case. For Cases 1 and 3, no WHERE conjunct is dropped in V' as the only conjuncts referring to $R.X$ are local WHERE conjuncts that are replaced by $\mathcal{C}_S(S.\bar{B}_1$.

It remains to show that the property P3 is true, i.e., the view-extent evolution parameter is satisfied by V'.

<u>Condition 3, Case 1: $\mathcal{VE} =\subseteq$.</u> We have to prove that the view-extent evolution parameter $\mathcal{VE} =\subseteq$ is satisfied. That is, $V' \;\; subseteq_\pi \; V$ where the common set of attributes are $(Attr(V(R))_{\text{SELECT}} \;\cap\; R.\bar{A}) \;\setminus\; \{R.X\}$.

Let $t' \in V'$ be a tuple in the view V' (Eq. (3.67)). Then from the definition of the view V' we have:

(0) there exists the tuples $s \in S$, $r' \in R'$, $t_1 \in R_1$, ... , $t_n \in R_n$ such that $s, r', t_1, ..., t_n$ generate the tuple t' in V'. For convenience we will use the notation $t_{1,...,n}$ to denote the concatenation of the tuples $t_1, t_2, ..., t_n$.

From (0) we have:

(1) $t'[(Attr(V(R))_{\text{SELECT}} \;\cap\; R.\bar{A}) \;\setminus\; \{R.X\}] = r'[(Attr(V(R))_{\text{SELECT}} \;\cap\; R.\bar{A}) \;\setminus\; \{R.X\}]$.

(2) $t'[\bar{T} \cap \text{Attr}(R_i)] = t_i[\bar{T} \cap \text{Attr}(R_i)]$, for each $i = 1, \ldots, n$. I.e.,$t'[\bar{T}] = t_{1,...,n}[\bar{T}]$.

(3) $r'[R'.\bar{A} \cap Attr(V(R))] = s[S.\bar{B} \cap S(Attr(V(R)))]$.

(4) $\mathcal{C}_S(s[S.\bar{B}_1])$ is true.

(5) $\mathcal{CV}(r'[Attr(V(R))_{\text{WHERE}} \setminus \{R.X\}], t_{1,...,n}[\bar{W}])$ is true.

From (4) we can conclude that the tuple s must be in the relation $\mathcal{PC}(S)$, i.e.,

(6) $s[S.\bar{B}] \in \underbrace{\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)}S)}_{\mathcal{PC}(S)}$.

The $\mathcal{PC}$-constraints from Condition 1 and (6) imply that there exists a tuple $r \in R$ such that:

(7) r satisfies the condition $\mathcal{C}_R(R.\bar{A}_1)$, i.e., $\mathcal{C}_R(s[R.\bar{A}_1])$ is true.

(8) $r[R.\bar{A}] \in \underbrace{\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)}R)}_{\mathcal{PC}(R)}$.

(9) $r[R.\bar{A}] = s[S.\bar{B}]$.

The relation R' was obtained from R by dropping the attribute $R.X$. Since, $R.X$ is not among the attribute $R.\bar{A}$ we have that:

(10) $R'.\bar{A} \cap Attr(V(R)) = b\ R.\bar{A} \cap Attr(V(R))$.

From the above (1), (3), (9) and (10) we have:

(11) $r[R.\bar{A} \cap Attr(V(R))] = s[S.\bar{B} \cap S(Attr(V(R)))] = r'[R'.\bar{A} \cap Attr(V(R))]$

From (5) and (11) we finally conclude that the tuple r and the tuples $t_1, ..., t_n$ satisfied the condition $\mathcal{CV}()$ from (5) (since from (11), r' and r have the same values on the attributes needed in this expression):

(12) $\mathcal{CV}(r[Attr(V(R))_{\mathsf{WHERE}} \setminus \{R.X\}], t_{1,...,n}[\bar{W}])$ is true.

We want to prove that the tuples $r \in R$ and $t_1 \in R_1, ..., t_n \in R_n$ generate a tuple t in the view V and this tuple T is has the same values as t' on the common set of attributes.

From (7) and Condition 3, (E2) we can conclude that all local WHERE conjuncts on $R.X$ in the original view V are satisfied by the tuple r. That is, from (7) we know $\mathcal{C}_R(R.\bar{A}_1)$ is true for the tuple R and from (E2) we imply that all local conditions in $\mathcal{CV}$ referring to $R.X$ are also satisfied by the tuple r: (13) $\mathcal{CV}(r[R.X])$ is true.

If (12) and (13) are true than all WHERE conjuncts of the original view V are satisfied by the tuples r, $t_1, ..., t_n$. Thus, these tuples generate a tuple t in V such that:

(14) $t[Attr(V(R))_{\text{SELECT}}] = r[Attr(V(R))_{\text{SELECT}}]$ and

(15) $t[\bar{T}] = t_{1,\ldots,n}[\bar{T}]$.

From (1), (11) and (14), we conclude that the tuples t' and t have same values on the common attributes from R':

(16) $t'[(Attr(V(R))_{\text{SELECT}} \cap R.\bar{A}) \setminus \{R.X\}] \overset{(1)}{=} r'[(Attr(V(R))_{\text{SELECT}} \cap R.\bar{A}) \setminus \{R.X\}]$ $\overset{(11)}{=} r[(Attr(V(R))_{\text{SELECT}} \cap R.\bar{A}) \setminus \{R.X\}] \overset{(14)}{=} t[(Attr(V(R))_{\text{SELECT}} \cap R.\bar{A}) \setminus \{R.X\}]$.

It is left to show that for the rest of the common attributes, i.e., $\bar{T}$, t and t' have the same values. The expression $\bar{T}$ refers only to attributes from R_1, .., R_n. The same set of tuples $t_1, ..., t_n$ are used to generate the two tuples t and t' and hence they must have the same values on $\bar{T}$ (following from (2) and (15)).

Condition 3, Case 2: $\mathcal{VE} = \supseteq$. We have to show that the view-extent evolution parameter $\mathcal{VE} = \supseteq$ is satisfied. That is, $V' \supseteq_\pi V$.

Let the tuple $t \in V$ be a tuple in the original view V defined in Eq. (3.49). Then from the definition of the view V we have:

(0) there exists the tuples $r \in R$, $t_1 \in R_1$, ... , $t_n \in R_n$ such that $r, t_1, ..., t_n$ generate the tuple t in V.

From (0) we have:

(1) $t[Attr(V(R))_{\text{SELECT}}] = r[Attr(V(R))_{\text{SELECT}}]$.

(2) $t[\bar{T} \cap \text{Attr}(R_i)] = t_i[\bar{T} \cap \text{Attr}(R_i)]$, for each $i = 1, \ldots, n$. I.e.,$t[\bar{T}] = t_{1,\ldots,n}[\bar{T}]$.

(3) $\mathcal{CV}(r[Attr(V(R))_{\text{WHERE}}], t_{1,\ldots,n}[\bar{W}])$ is true.

From (3) and Condition 3, (E3) we can conclude that the tuple r must satisfies as well $\mathcal{C}_R(R.\bar{A}_1)$:

(4) $\mathcal{C}_R(r[R.\bar{A}_1])$ is true.

Thus r is a tuple in the relation $\mathcal{PC}(R)$, i.e.,

(5) $r[R.\bar{A}] \in \underbrace{\pi_{R.\bar{A}}(\sigma_{\mathcal{C}_R(R.\bar{A}_1)} R)}_{\mathcal{PC}(R)}$.

The $\mathcal{PC}$-constraints from Condition 1 and (6) imply that there exists a tuple $s \in R$ such that:

(6) s satisfies the condition $\mathcal{C}_S(S.\bar{B}_1)$, i.e., $\mathcal{C}_R(s[R.\bar{A}_1])$ is true.

(7) $s[S.\bar{B}] \in \underbrace{\pi_{S.\bar{B}}(\sigma_{\mathcal{C}_S(S.\bar{B}_1)} S)}_{\mathcal{PC}(S)}$.

(8) $s[S.\bar{B}] = r[R.\bar{A}]$

We will show below that the tuples $s \in S$, $r[R'] \in R'$, $t_1 \in R_1$, ... , $t_n \in R_n$ generate a tuple t' in V' (defined in Equation (3.67)) and, moreover, $t' \in V'$ and $t \in V$ are π-equivalent.

From the above (8) we have:

(11) $r[R.\bar{A} \cap Attr(V(R))] = s[S.\bar{B} \cap S(Attr(V(R)))]$.

The expression $\mathcal{CV}'$ is obtained from $\mathcal{CV}$ by dropping all conditions referring to $R.X$ that don't imply a conjunct in $\mathcal{C}_R(R.\bar{A}_1)$. From (3) we conclude that $\mathcal{CV}'$ must be satisfied by the tuples $r[R'], t_1, ..., t_n$: (12) $\mathcal{CV}'(r[Attr(V(R))_{\textsf{WHERE}} \setminus \{R.X\}], t_{1,...,n}[\bar{W}])$ is true.

From (6), (11) and (12) we can conclude that all local WHERE conjuncts in the view V' are satisfied by the tuples $s, r[R'], t_1, ..., t_n$. Thus, these tuples generate a tuple t' in V' such that:

(13) $t'[Attr(V(R))_{\textsf{SELECT}} \setminus \{R.X\}] = r[Attr(V(R))_{\textsf{SELECT}} \setminus \{R.X\}]$ and

(14) $t'[\bar{T}] = t_{1,...,n}[\bar{T}]$.

From (1) and (13), we conclude that the tuples t' and t have same values on the common attributes from R':

(15) $t'[Attr(V(R))_{\textsf{SELECT}} \setminus \{R.X\}] \stackrel{(13)}{=} r[Attr(V(R))_{\textsf{SELECT}} \setminus \{R.X\}] \stackrel{(1)}{=} t[Attr(V(R))_{\textsf{SELECT}} \setminus \{R.X\}]$.

It is left to show that for the rest of the common attributes, i.e., $\bar{T}$, t and t' have the same values. The expression $\bar{T}$ refers only to attributes from R_1, .., R_n. The same set of tuples $t_1, ..., t_n$ are used to generate the two tuples t and t' and hence they must have the same values on $\bar{T}$ (following directly from (2) and (14)).

Condition 3, Case 3: $\mathcal{VE} = \equiv$.

We want to show that $V' \equiv_\pi V$ when $\phi = $ "$\equiv$". Hence, we have to prove two inclusions:

(I) $V' \subseteq_\pi V$ and

(II) $V' \supseteq_\pi V$.

Note that in this case Conditions 2 and 3 impose all conditions from Case 1 and Case 2 combined.

The inclusion (I) is implied by Condition 3, Case 1 proven above given that Conditions 2 and 3 from that Case 2 are satisfied (i.e., D1, D2, E0, E1 and E2 are fulfilled).

Similarly, the inclusion (II) can be proven in the same way as for Case 2 knowing that (D1), (D2), (E3) and (E4) are fulfilled.

Both inclusion (I) and (II) being true, we can conclude that $V \equiv_\pi V'$.

Note that, as in the case of the capability change "*delete-attribute R*", for the aggregation view V' defined in Equation (3.50), we can compare the aggregation attributes defined by aggregation functions that are **duplicate-insensitive**, such as MIN, MAX, and $COUNT(DISTINCT)$. These relationships are shown in the Tables 3.16, 3.17, 3.18 and 3.19. The same arguments we give in the proof of Theorem 5 in Appendix A.1 are valid for the capability change "*delete-attribute R.A*" for the SPOC algorithm.

Q.E.D.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley Pub., 1995.

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

[3] Y. Arens, C. A. Knoblock, and W.-M. Shen. Query Reformulation for Dynamic Information Integration. *Journal of Intelligent Information Systems*, 6 (2/3):99–130, 1996.

[4] E. A.Rundensteiner, A. Koeller, X. Zhang, A. Lee, and A. Nica. Evolvable View Environment. In *Demo Session Proceedings of SIGMOD'99*, 1999. to appear.

[5] D.E. Atkins, W.P. Birmingham, E.H. Durfee, E. Glover, T. Mullen, E.A. Rundensteiner, E. Soloway, J. Vidal, R. Wallace, and M. Wellman. Toward Inquiry-Based Education Through Interacting Software Agents. *IEEE Computer*, page 69, May 1996.

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

[7] S. Chaudhuri, R. Krishnamurthy, and S. Potamianos. Optimizing Query with Materialized Views. In *Proceedings of IEEE International Conference on Data Engineering*, 1995.

[8] L.S. Colby, T.Griffin, L.Libkin, I.S.Mumick, and H.Trickey. Algorithms for Deferred View Maintenance. In *Proceedings of SIGMOD*, pages 469–480, 1996.

[9] O. Etzioni and D. Weld. A Softbot-Based Interface to the Internet. *Communication of ACM*, 1994.

[10] EVE Project. Online EVE Demo and Tutorial: `http://davis.wpi.edu/dsrg/EVE`, 1998.

[11] C. Galindo-Legaria. Outerjoins as disjunctions . *Proceedings of SIGMOD*, pages 348–358, 1994.

[12] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Integrating and Accessing Heterogeneous Information Sources in TSIMMIS . *Proc. of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments, Stanford, California*, March 1995.

[13] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. In *Proceedings of IEEE International Conference on Data Engineering*, pages 152–159, 1996.

[14] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-Query Processing in Data Warehousing Environments. In *International Conference on Very Large Data Bases*, June 1995.

[15] A. Gupta, H.V. Jagadish, and I.S. Mumick. Data Integration using Self-Maintainable Views. In *Proceedings of International Conference on Extending Database Technology (EDBT)*, pages 140–144, 1996.

[16] 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.

[17] 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.

[18] A. Gupta, I.S. Mumick, and V.S. Subrahmanian. Maintaining Views Incrementally. In *Proceedings of SIGMOD*, pages 157–166, 1993.

[19] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing Queries Across Diverse Data Sources. In *International Conference on Very Large Data Bases*, pages 276–285, 1997.

[20] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, and V. Vassalos. Template-Based Wrappers in the TSIMMIS System. In *Proceedings of SIGMOD*, pages 532–535, 1997.

[21] V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing Data Cubes Efficiently. In *Proceedings of SIGMOD*, June 1996.

[22] N. Huyn. Efficient View Self-Maintenance. In *Proceedings of the Workshop on Materialized Views: Techniques and Applications*, pages 17–25, June 1996.

[23] M. Jarke and J. Koch. Query Optimization in Database Systems. *ACM Computing Surveys*, pages 111–152, 1984.

[24] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. *Proceedings of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments, Stanford, California*, March 1995.

[25] A. Koeller, E. A. Rundensteiner, and N. Hachem. Integrating the Rewriting and Ranking Phases of View Synchronization. In *Proceedings of the ACM First International Workshop on Data Warehousing and OLAP (DOLAP'98)*, November 1998.

[26] A. J. Lee. *View-Based Solutions for Schema Evolution Problems: Application Interoperability and View Synchronization.* PhD thesis, June 1998.

[27] 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.

[28] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Data Warehouse Evolution: Trade-offs between Quality and Cost of Query Rewritings. In *Proceedings of IEEE International Conference on Data Engineering*, 1999.

[29] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Non-Equivalent Query Rewritings. In *International Database Conference*, Hong Kong, July 1999.

[30] 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 (CASCON'97), Best Paper Award*, pages 1–14, November 1997.

[31] 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.

[32] A. J. Lee and E. A. Rundensteiner. Data Warehouse Evolution: Consistent Metadata Management. In *IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC), Session: Data Mining and Data Warehousing*, San Diego, Cal., Oct. 10-14 1998.

[33] 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.

[34] A. Y. Levy, Y. Sagiv, and D. Srivastava. Towards Efficient Information Gathering Agents. *Proc. of the AAAI Spring Symposium on Software Agents, Stanford, California*, March 1994.

[35] A. Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *Journal of Intelligent Information Systems. Special Issue on Networked Information Discovery and Retrieval*, 1995.

[36] 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.

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

[38] I. Mumick, D. Quass, and B. Mumick. Maintenance of Data Cubes and Summary Tables in a Warehouse. In *Proceedings of SIGMOD*, May 1997.

[39] A. Nica, A. J . Lee, and E. A. Rundensteiner. The Complex Substitution Algorithm for View Synchronization. Technical Report WPI-CS-TR-97-8, Worcester Polytechnic Institute, Dept. of Computer Science, 1997.

[40] 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.

[41] A. Nica and E. A. Rundensteiner. A Constraint-based Object Model for Structured Document Management. Technical Report CSE-TR-237-95, University of Michigan, Ann Arbor, EECS Dept. CSE Division, 1995.

[42] A. Nica and E. A. Rundensteiner. Uniform Structured Document Handling using a Constraint-based Object Approach. In *Book chapter in ADL'95, Springer Verlag, Advances in Digital Libraries (ADL'95), A Forum on Research and Technology Advances in Digital Libraries*, May 1995.

[43] A. Nica and E. A. Rundensteiner. The Dynamic Information Integration Model. Technical Report CSE-TR-311-96, University of Michigan, Ann Arbor, EECS Dept. CSE Division, 1996.

[44] A. Nica and E. A. Rundensteiner. Loosely-Specified Query Processing in Large-Scale Information Systems. *International Journal of Cooperative Information Systems*, 1997.

[45] A. Nica and E. A. Rundensteiner. On Translating Loosely-Specified Queries into Executable Plans in Large-Scale Information Systems. In *Proceedings of Second IFCIS International Conference on Cooperative Information Systems CoopIS'97*, pages 213–222, June 1997.

[46] 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.

[47] 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 Science, 1998.

[48] 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.

[49] A. Nica and E. A. Rundensteiner. View Maintenance after View Synchronization. Technical Report WPI-CS-TR-98-24, Worcester Polytechnic Institute, Dept. of Computer Science, 1998.

[50] A. Nica and E. A. Rundensteiner. View Maintenance after View Synchronization. In *International Database Engineering and Applications Symposium (IDEAS'99)*, August, Montreal, Canada 1999.

[51] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A Mediation System Based on Declarative Specifications. In *Proceedings of IEEE International Conference on Data Engineering*, 1996.

[52] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *Proceedings of IEEE International Conference on Data Engineering*, pages 251–260, March 1995.

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

[54] D. Quass and J. Widom. On-Line Warehouse View Maintenance. In *Proceedings of SIGMOD*, pages 393–400, 1997.

[55] Y. G. Ra and E. A. Rundensteiner. A Transparent Object-Oriented Schema Change Approach Using View Schema Evolution. In *Proceedings of IEEE International Conference on Data Engineering*, pages 165–172, March 1995.

[56] Y. G. Ra and E. A. Rundensteiner. A transparent schema-evolution system based on object-oriented view technology. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, August 1998.

[57] V. V. Raghavan, G. S. Jung, and P. Bollmann. A Critical Investigation of Recall and Precision as Measures of Retrieval System Performance. *ACM Transactions on Office Information Systems*, pages 205–229, July 1989.

[58] M. T. Roth and P. M. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *International Conference on Very Large Data Bases*, pages 266–275, 1997.

[59] 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.

[60] 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.

[61] 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.

[62] T.J. Teorey. *Database Modeling and Design, 2nd Edition.* Morgan Kaufmann Pub., 1994.

[63] A. Tomasic, L. Raschid, and P. Valduriez. Scaling Heterogeneous Databases and the Design of DISCO. Technical report, Technical Report, 1995, INRIA.

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

[65] J.D. Ullman. *Principle of Database and Knowledge-Base Systems.* Computer Science Press, 1989.

[66] J.D. Ullman and J. Widom. *A First Course in Database Systems, 1st Edition.* Prentice Hall Engineering, Science and Math., 1997.

[67] C. A. van den Berg and M.L. Kersten. An Analysis of a Dynamic Query Optimization Schema for Different Data Distributions. In J. C. Freytag, D. Maier, and G. Vossen, editors, *Query Processing for Advanced Database Systems*, chapter 15, pages 449–473. Morgan Kaufmann Pub., 1994.

[68] V. Vassalos and Y. Papakonstantinou. Describing and Using Query Capabilities of Heterogeneous Sources. In *International Conference on Very Large Data Bases*, pages 256–265, 1997.

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

[70] 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.

[71] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *International Conference on Parallel and Distributed Information Systems*, December 1996.

[72] 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.

ABSTRACT

VIEW EVOLUTION SUPPORT FOR INFORMATION INTEGRATION SYSTEMS OVER DYNAMIC DISTRIBUTED INFORMATION SPACES

by

Anisoara Nica

Chair: Elke A. Rundensteiner

Challenging issues for creating and maintaining tailored information gathering systems over large-scale information spaces (e.g., Digital Libraries, the World Wide Web) include the diversity of the information sources (ISs) in terms of their structures, query interfaces and search engines, as well as the dynamics of sources continuously being added, removed or upgraded. Current information integration systems are often based on static apriori defined views that gather information from heterogeneous information sources and provide the user with a uniform view of the information space for browsing and querying. This dissertation addresses one of the largely unexplored issues that such information integration systems raise, namely, the evolution and maintenance of data warehouses when the underlying information sources change their capabilities, i.e., schema level changes.

The overall solution approach that this dissertation puts forth consists of defining the problem of view evolution triggered by capability changes of ISs and designing evolution algorithms that achieve synchronization of the affected views in the presence of these types of changes. The contributions made by this dissertation include (1) an extension of the SQL view definition language that allows the user to apriori specify evolution preferences, e.g., whether dropping or changing a view component is acceptable; (2) a formal definition of what is a legal view rewriting under capability changes, i.e., semantics of view evolution; (3) algorithms for view synchronization that find a modified view definition in response to a capability change of an IS; (4) algorithms for the maintenance of materialized views after the view synchronization process; (5) experimental evaluations comparing the maintenance strategies after view synchronization with alternative maintenance techniques; and (6) the development of an working system incorporating some of the proposed evolution algorithms.