Data Currency Quality Factors in Data Warehouse Design

Dimitri Theodoratos *
NTUA
Dept. of Electrical and Computer Eng.,
Athens, GR-15773
dth@dblab.ece.ntua.gr

Mokrane Bouzeghoub*
INRIA,
Domaine de Voluceau - B.P. 105,
Le Chesnay Cedex, FR-78153
Mokrane.Bouzeghoub@prism.uvsq.fr

Abstract

A Data Warehouse (DW) is a large collection of data integrated from multiple distributed autonomous databases and other information sources. A DW can be seen as a set of materialized views defined over the remote source data. Until now research work on DW design is restricted to quantitatively selecting view sets for materialization. However, quality issues in the DW design are neglected.

In this paper we suggest a novel statement of the DW design problem that takes into account quality factors. We design a DW system architecture that supports performance and data consistency quality goals. In this framework we present a high level approach that allows to check whether a view selection guaranteeing a data completeness quality goal also satisfies a data currency quality goal. This approach is based on an AND/OR dag representation for multiple queries and views. It also allows determining the minimal change propagation frequencies that satisfy the data currency quality goal along with the the optimal query evaluation and change propagation plans. Our results can be directly used for a quality driven design of a DW.

Research supported by the European Commission under the ESPRIT Program LTR project "DWQ: Foundations of Data Warehouse Quality"

Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, 14. - 15.6. 1999

(S. Gatziu, M. Jeusfeld, M. Staudt, Y. Vassiliou, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-19/

1 Introduction

A Data Warehouse (DW) is a large collection of data used by companies for On-Line Analytical Processing (OLAP) and Decisison Support System (DSS) applications [4]. Because of the enormous quantity of information available to companies nowadays, DWs often grow to be very large. Data warehousing is also an approach for integrating data from multiple, possibly very large, distributed, autonomous, heterogeneous databases and other information sources [44]: selected information from each source is extracted in advance, cleaned, translated and filtered as needed, merged with relevant information from other sources and stored in a repository. OLAP in the DW is decoupled as much as possible from On-Line Transaction Processing (OLTP) supported by the remote source databases. A DW can be abstractly seen as a set of materialized views defined over source relations. We provide below a brief overview of issues related to the design of a DW.

Query evaluation. OLAP and DSS applications make heavy use of complex queries (usualy with grouping/aggregation). These data intensive queries often require sequential scans. Ensuring high query performance is one of the most significant challenges when implementing a DW. To this end, the queries addressed to the DW are evaluated locally (using exclusively the materialized views) without accessing the original information sources. Therefore a complete rewriting [24] of the queries over the views materialized at the DW must be possible [41, 40].

View maintenance. When the source relations change, the materialized views need to be brought up-to-date. Typically, the DW is maintained separately from the operational source databases. Recent applications of DWs require data that are more current. In order to bring the materialized views up-to-date, different update scenarios can be envisaged. They can be classified according to the *class of queries and views* considered, the *type of changes*, the *maintenance strategy*, the *maintenance timing policy*, the

type of environment, and for a distributed environment the *type of information sources* [13, 44, 48].

Maintenance strategies. A maintenance strategy can be an *incremental* one, or a *recomputation* of the views from scratch. In an incremental strategy, the changes to the views are computed using the changes to the source relations [2, 27, 12, 9, 28, 38, 6]. Incremental maintenance can be significantly cheaper and more space efficient than recomputing the view from scratch, especially if the size of the view is much larger than the size of the changes [9, 49].

Maintenance timing policy. The maintenance timing policy can be either *immediate* or *deferred*. In an immediate timing policy [2, 3], a materialized view is maintained within or immediately after the transaction that changes the source relations. In a deferred timing policy [5], the maintenance of the view is delayed. It can be done *periodically* [34], at *query time* [16], or *on-demand* by the user. It can also be *triggered by events* at the sources (e.g. when the net change to a source database exceeds a certain threshold). Sometimes supporting multiple maintenance timing policies may be appropriate [6, 33].

Type of environment. Most of the research work on incremental view maintenance assumes a centralized database environment where a single system has control of the materialized views, the source relations, and the changes [27, 12, 9, 28]. Therefore, changes to the source relations and to the materialized views can be combined within the same transaction. DWs are typically distributed database environments. Some approaches to incremental view maintenance in distributed environments are based on timestamping the changes to the source relations [34, 33]. However, the source databases can be autonomous, a global clock cannot be assumed and therefore inconsistencies may appear [49]. Approaches that keep materialized view data loosely consistent with the remote sources are more appropriate for DW environments [49, 50, 18, 51].

Types of sources. Concerning the detection and handling of the changes of the source data, the sources can be of the following types [44]: *cooperative sources* (they provide active database features with triggering [26], and allow the detection, filtering, storage, processing, and propagation of changes to the DW to be programmed and occur automatically), *logged sources* (they maintain a log and allow changes of interest to be extracted by inspecting the log), *queryable sources* (they can be queried periodically in order to detect changes of interest), and *snapshot sources* (they only allow snapshots of data to be taken periodically and changes are extracted by comparing successive snapshots [23]).

Maintenance queries. When changes are reported by a data source, it may be necessary to issue queries to the same or other data sources in order to maintain the affected materialized views. In the case of an incremental maintenance strategy the queries involve also differentials (source relation changes). We call theses queries *maintenance queries*. Maintaining materialized views incurs a cost for computing maintenance queries, a cost for transmitting data from the sources to the DW and inversely (in a distributed DW environment), and a cost for applying the computed changes to the materialized views [45].

Multiquery optimization on maintenance queries. The changes taken into account for maintaining the materialized views at the DW may affect more than one view. Then multiple maintenance queries are issued against the source relations for evaluation. These maintenance queries may contain subexpressions that are identical, equivalent, or more generally subexpressions such that one can be computed from the other. We describe these subexpressions by the generic term *common subexpressions* [19]. The techniques of *multiple query optimization* [35, 36] allow these queries to be computed together by detecting common subexpressions between maintenance queries: non-optimal local query evaluation plans are combined into an optimal global plan which is more efficient to execute than executing separately the optimal local evaluation plan of each maintenance query.

Using auxiliary views to reduce the view maintenance cost. A global evaluation plan for maintenance queries can be executed more efficiently if some intermediate subqueries are kept materialized in the DW, or can be computed from views that are kept materialized in the DW [30, 42]. These materialized subqueries (views) are called auxiliary views. It is worth noting that an optimal global evaluation plan without auxiliary views can be completely different than the optimal global evaluation plan when materialized views are used. The existence of auxiliary views can greatly reduce the cost of evaluating maintenance queries. Indeed, the computation of the corresponding subqueries is avoided or simplified. Further, since DWs are typically distributed systems, access of the data sources and expensive data transmissions are reduced. Obviously, there is a cost associated with the process of maintaining the auxiliary materialized views. But, if this cost is less than the reduction to the maintenance cost of the initially materialized views, it is worth keeping the auxiliary views in the DW.

Self-maintenability. By appropriately selecting auxiliary views to materialize in the DW, it is possible to maintain the initial materialized views and the auxiliary views altogether, for any source relation change, without issuing queries against the source relations. Such a set of views is called *self-maintainable* [11, 18, 29].

DW design. When designing a DW, a number of choices are made first by the DW designer (e.g. the maintenance timing policy, the maintenance strategy, or the DW system architecture) dictated by physical parameters (e.g. the type

and the availability of data sources, the type of changes, the data transmission rates over the network, the hardware computational power etc.) and the requirements of the knowledge workers that will use the DW.

The next step in the DW design process concerns the selection of views to materialize in the DW according to use the DW is intended to (i.e. the queries the DW has to answer). This is the *DW view selection problem*. The view selection problem takes as input a set of queries or views and aims at selecting a set of views to materialize in the DW that minimizes the overall query evaluation cost, or the overall view maintenance cost, or a combination of both. A number of constraints may be additionally provided as input for satisfaction (e.g. the materialized views should fit in the space allocated for materialization or the view maintenance cost should not exceed a certain limit) [32, 17, 41, 15]. The DW design problem is complex. One of the reasons of its complexity relies on the fact that common subexpressions between the input queries need to be detected and exploited.

Quality factors in DW design. Until now research work on DW design is restricted to quantitatively selecting view sets for materialization. Quality issues in the DW design are neglected. However, the design of a DW at the logical and physical level is subject to a number of quality factors. These quality factors determine *quality goals* that have to be achieved through the design process [20, 22]. The present research work is done in the framework of the European Foundations of Data Warehouse Quality (DWQ) project. The goal of the DWQ project is to develop semantic foundations that will allow the designers of DWs to link their choice of deeper models, rich data structures and rigorous implementation techniques to quality factors in a systematic manner, thus improving the design, the operation and the evolution of DWs [21]. In this paper, we deal mainly with the quality factors of *data currency*, *data consistency*, *data completeness*, and *query and view maintenance performance* in the design of a DW.

1.1 The problem

When designing a DW, alternative view selections for materialization are examined in order to find one that satisfies the DW design goals. View selection algorithms for Data Warehousing proceed in a similar manner [17, 41, 15, 47].

The framework. We consider that the approach adopted for designing a DW aims at selecting a set of views for materialization that satisfies the following quality goals: (a) *data completeness*, (b) *data currency*, (c) *query evaluation and view maintenance performance*, and (d) *data consistency*. We suppose also that the sources are subject to a *source availability constraint*. We explain these notions below.

The source availability constraint states that the change propagation frequency from each source is restricted not to exceed a maximal frequency. These maximal frequencies are set by the administrators of the source databases and express the availability of the data sources and the degree of decoupling of OLTP at the operational data sources from DW activities.

The data completeness quality goal guarantees that the data necessary for answering the input queries are present at the DW. Therefore, it requires a complete rewriting of the input queries over the materialized views

The data currency quality goal upper bounds the time elapsed between the time point the answer to a query is returned to the user and the time point the most recent changes to a source relation that are taken into account in the computation of this answer are read (this time reflects the currency of answer data). The data currency quality goal is expressed by *currency constraints* associated with every source relation in the definition of every input query. The upper bound in a currency constraint (minimal currency required) is set by the knowledge workers according to their needs.

The query evaluation and view maintenance performance quality goal requires the minimization of a combination of these costs.

The data consistency quality goal ensures that at every moment the state of the DW reflects a certain state of the source relations. It is expressed by a number of properties that the DW data must satisfy. These properties are formally presented in Section 3.

This formalization of the problem of designing a DW using quality goals is novel. Further, it allows:

(a) stating currency constraints at the query level and not at the materialized view level as is the case in other approaches [33, 18]. Therefore, currency constraints can be exploited by DW view selection algorithms where the queries are the input, while the materialized views are the output (and therefore are not available).

(b) stating different currency constraints for different relations in the same query. This flexibility is necessary. For instance, a query that combines share prices and companies introduced in the stock market requires different currency constraints for the data derived from the source relation providing information on share prices and for the data derived from the source relation providing information on companies.

Problem addressed. In the framework set up above, we address the problem of checking whether for a view selection that satisfies the data completeness quality goal, there are change propagation frequencies that satisfy the given data currency and source availability constraints. Additionally, it is required:

(a) In case of a positive answer,
  (a1) the change propagation frequencies that guarantee the satisfaction of the currency and source availability constraints, while minimizing the

view maintenance cost, and

(a2) the optimal way the queries are evaluated from the materialized views (query evaluation plans), and the optimal way the changes to the source relations are propagated and applied to the affected materialized views (change propagation plans).

(b) In case of a negative answer, the set of source relations that cause the violation of a currency constraint.

We call this problem *currency constraint satisfiability* problem.

1.2 Contribution and outline

The main contributions in this paper can be summarized as follows.

- We provide a novel statement for the DW view selection problem based on quality goals and source availability constraints. In particular, the data currency quality goal is expressed by a set of currency constraints flexibly specified, on a per relation basis, at the query level.
- We describe a DW system architecture over remote autonomous sources that (a) supports the query evaluation and view maintenance performance quality goal (by evaluating queries exclusively from the materialized views, and by exploiting self-maintainability and auxiliary views), and (b) achieves the data completeness quality goal (by appropriately materializing views over the source relations at the DW), and the data consistency quality goal (by the appropriate selection of an update propagation process).
- We formally define the currency constraint satisfiability problem using an AND/OR dag representation for multiple queries and views. The multiquery AND/OR dag representation allows to take into account common subexpressions between queries and views and to formally express query evaluation and change propagation plans.
- In this framework, we present a high level approach for solving the currency constraint satisfiability problem. The approach proceeds by "pushing down" currency constraints from the queries to the materialized views along optimal query evaluation plans and by "pushing up" source availability constraints from the source relations to the materialized views along optimal change propagation plans.
- Our approach allows also to determine the minimal change propagation frequencies that satisfy the constraints along with the optimal query evaluation and change propagation plans.
- When the satisfaction of the data currency and source availability constraints is not possible, we provide the set of source relations that cause the violation of source availability and currency constraints. This information can guide the view selection algorithms in providing alternative view selections that satisfy the constraints, or can help the DW designer in finding a solution to the view selection problem by negotiating the relaxing of some currency constraints and/or source availability constraints.

The rest of the paper is organized as follows. Next section reviews related work. Section 3 presents the architecture of the data warehousing system, outlines change propagation and defines data consistency. In Section 4 we introduce multiquery AND/OR graphs, and provide initial definitions. We also state formally the currency constraint satisfiability problem. The different steps of our approach are presented in Section 4. In Section 5 we discuss improving a selected view set in order to satisfy the constraints. Finally, Section 6 contains concluding remarks and future research directions.

2 Related work

Answering queries using views has been studied in many papers, e.g. [24]. In particular, this issue, in connection to grouping/aggregation queries and views, has been studied in [10] for set semantics, and in [7] for multiset semantics.

Materialized view maintenance has been addressed in recent years by a plethora of researchers. A number of papers dealing with different aspects of materialized view maintenance are cited in the introduction and in subsequent sections. Different levels of data consistency of materialized view maintenance processes in distributed environments are discussed in [49, 50, 18, 51]. Data consistency in this paper is defined similarly to that in [18].

View selection problems for Data Warehousing usually follow the following pattern: select a set of views to materialize in order to optimize the query evaluation cost, or the view maintenance cost or a combination of both, possibly in the presence of some constraints. In [32] views are seen as sets of pointer arrays and the goal is to optimize the combined cost under a space constraint. [17] aims at minimizing the query evaluation cost in the context of aggregations and multidimensional analysis under a space constraint. Given a materialized SQL view, [30] presents an exhaustive approach as well as heuristics for selecting auxiliary views that minimize the total view maintenance cost. Given an SPJ view, [29] derives, using key and referential integrity constraints, a set of auxiliary views, other than the base relations, that eliminate the need to access the base relations when maintaining both the initial and the auxiliary views (i.e. that makes the views altogether self-maintainable). In [14] greedy algorithms are provided for selecting views to materialize that minimize the query evaluation cost under a space constraint. A solution for selecting views that minimize the combined cost is given in [47].

None of the previous approaches requires the queries to be answerable exclusively from the materialized views in a non-trivial manner (that is without considering that the base relations and the materialized views reside in the same site, and without replicating all the base relation at the DW). This requirement is taken into account in [41] where the problem of configuring a DW without space restrictions is addressed for a class of select-join queries. This work is extended in [42] in order to take into account space restrictions, multiquery optimization over the maintenance queries, and the use of auxiliary views when maintaining other views. Another extension of [41] deals with the same problem for a class of PSJ queries under space restrictions [40]. The approach adopted in the last three papers is tailored for the static DW design problem. An incremental version of the DW design problem (dynamic DW design) is addressed in [43].

A variation of the DW design problem endeavoring to select a set of views that minimizes the query evaluation cost under a total maintenance cost constraint is adopted in [15]. However, restricting the total view maintenance cost does not provide any guaranty whatsoever for the currency of the query answer data. Quite the contrary, reducing the view maintenance cost may result in lower update propagation frequencies, and therefore in stale query answer data.

An analytical study of optimal refresh policies based on parametrization of the average query response time and the average cost for materialized view maintenance is described in [37]. Yet, this analysis concern a single view defined over relations of the same source in a centralized environment without communications costs. Further, no currency constraints are introduced.

A periodical or at query time (hybrid) timing policy is adopted in [33] where a materialized view V is updated either at the end of a time period t or when a query Q is issued against V and the currency of V is unsatisfactory with respect to Q. An algorithm is provided that aims at minimizing the average updating cost of V per query by selecting a materialized view from which V is to be updated, and the time period t. Currency constraints are associated with the queries but they are different than those introduced here since they refer to the currency of views from which the query is answered, they do not characterize each relation in the query, and they are used to trigger the updating of the view V. Further, a centralized environment based on timestamping is assumed, while all the views are defined over a single source relation.

[18] characterizes the "freshness" of materialized views in a mediator using time upper bounds required for the different steps of the view maintenance process (e.g. communication delay, maintenance query processing delay etc.). That mediator system architecture is different than ours since queries can be answered also from the source relations, an immediate maintenance timing policy is adopted, and source relation changes are buffered at the mediator. Further, no currency constraints are considered, nor examining alternative change propagation plans in order to guarantee the "freshness" of the materialized views.

3 DW system architecture and operation

We describe in this section the architecture of the DW system on which our analysis is based. Among the goals of this architecture is high query performance and low view maintenance cost. Figure 1 illustrates the DW system architecture. On the bottom of the diagram are shown the remote source relations. The materialized views are kept at the DW component. Some of these views may be defined using other views. Knowledge workers and analysts, depicted on the top of the diagram, address their queries to the DW.

Query evaluation. The queries of the analysts are evaluated locally without accessing the source databases. Thus, this DW architecture satisfies the data completeness quality goal which requires a complete rewriting of the input queries over the materialized views. We consider that when changes are applied to a materialized view V, the previous state of V is available for evaluating queries using V. Therefore, the evaluation process is not delayed by the application of the changes to the materialized views.

Simple and auxiliary materialized views. The materialized views that are used in the optimal query evaluation plans of the queries are called *simple views*. The rest of the materialized views are used for reducing the view maintenance cost of the materialized views, and are called *auxiliary views*. Simple views too can be used in the same manner, yet they have to appear in the optimal query evaluation plan of a query.

Self-maintainability. For each source relation R_i there is at the DW a materialized view V_i obtained by applying selections and projections on R_i as much as possible such that all the views in the DW can be completely rewritten over the V_i's. These views are called *source relation images*. Select-project views are self-maintainable [11]. Other views defined over these views may also be materialized at the DW. The role of the source relation images is to guarantee the self-maintainability of all the materialized views. A source relation image can be either simple or auxiliary view.

Type of sources and changes. We assume that the source database systems are autonomous and can cooperate with the DW in the following sense: each source database system is able to keep in a buffer the tuples inserted to a source relation and the tuples deleted from it (modifications are modeled by deletions followed by insertions). A source is aware of the view definition of the corresponding source relation image. It can also filter the changes, compute the net changes to be applied to the source relation image, and

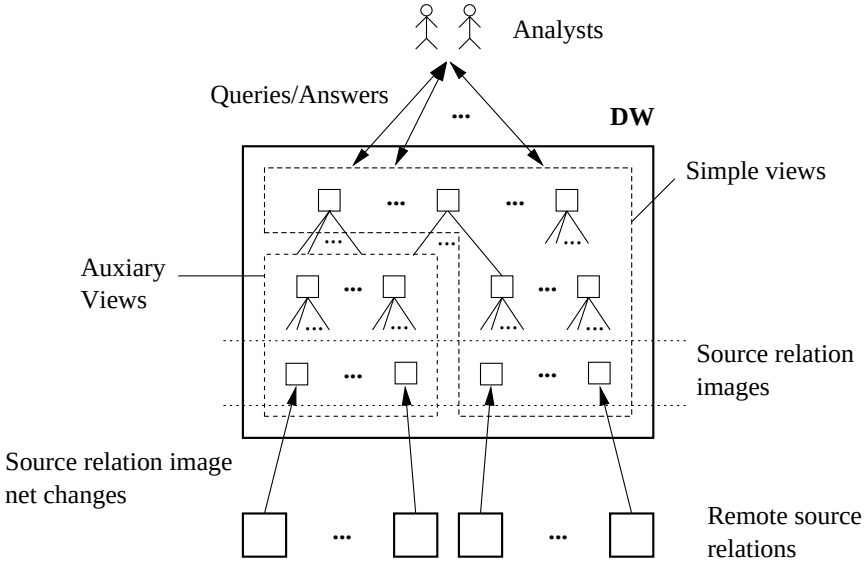

Figure 1: A DW system architecture

periodically transmit the changes to the DW.

Change filtering. Using the view definition of the corresponding source relation image, a source filters out changes that are irrelevant to the source relation images and projects only relevant attributes. A source relation change is *irrelevant* if it has no effect on the state of the source relation image, independently on the source relation state [1, 25]. Since the source relation images are select-project views, selection conditions involve only attributes of the source relation. Therefore, detecting irrelevant changes can be performed efficiently by the sources. Alternatively, source relation changes can be transmitted to and gathered in buffers at the DW. This alternative does not significantly change our approach, yet changes cannot be filtered by the data sources. Therefore, the communication cost is increased by the transmission of irrelevant changes and of useless attribute values of tuples from the remote sources to the DW.

Net changes. In order to avoid wasteful insertions and deletions (and data transmissions) when incrementally maintaining a materialized view from other materialized views in the DW (and a source relation image from the source relations), we consider the changes actually inserted to or deleted from each view (source relation). These changes are called *net changes*. For example, if a tuple is inserted and then deleted, it is not represented at all in the net changes. In the following 'changes' refer to 'net changes'.

Change propagation. Changes are propagated to the DW views periodically. At the end of the change propagation period for a source relation, the source database system performs three tasks: (a) it reads the changes in the corresponding buffer and flashes the content of the buffer, (b) it computes the changes, to be applied to the source relation image, using the source relation image view definition and the changes to the source relation, and (c) it transmits the source relation image changes to the DW.

There are two advantages when the computation of the source relation images is performed by the sources: (a) processing at the DW is saved, and (b) the transmission cost is reduced. Determining the change propagation frequency for each source relation is an objective of this paper. Each source relation frequency is upper bounded by a given maximal frequency for that source as indicated by the corresponding source availability constraint. The maximal frequencies are determined by the time the source data base can devote to supporting the maintenance of the DW.

When the net changes of a source relation image V arrive at the DW, they are propagated to the materialized views that are affected by these changes (that is the views that include the source relation in their view definition), according to a change propagation plan. Net changes from different sources are transmitted to the DW asynchronously. We assume that different changes from the same source relation are received by the DW in the order they are transmitted. Further, different changes are propagated to the affected views in the order they are received by the DW. The affected materialized views are maintained incrementally. During the propagation of the changes from a source relation to the materialized views, changes are not applied to a materialized view V until all the changes for all the other materialized views that are directly defined using V have been computed. Parts of the update transactions that propagate changes from the source relation images to the materialized views can be executed concurrently. Note that employing recomputation of the materialized views from scratch instead of an incremental view maintenance

strategy does not affect our approach.

Data consistency. The DW system we presented above is consistent. We define a DW system to be consistent if its data satisfy the following properties [18, 51].

(a) If a number of changes is applied to the source relations, and this activity has ceased, the state of the materialized views at the DW eventually reflects the final state of the source relations.

(b) The state of each materialized view at the DW at time t reflects some anterior state of the source relations (not necessarily the state of the source relations at a single time since the changes from multiple consecutive update transactions to the same source relation are transmitted together to the DW, and change transmissions from different sources is asynchronous).

(c) The states of a materialized view, defined using some source relation R, at times t_1 and t_2, $t_1 \leq t_2$, reflects the state of R at times t'_1 and t'_2, where $t'_1 \leq t'_2$.

4 Multiquery AND/OR dags and formal problem statement

We introduce in this section multiquery AND/OR dags. We then use this notion to formally describe change propagation to multiple views, to introduce time cost functions, and finally, to state the currency constraint satisfability problem.

4.1 Class of queries

We assume relational queries and views possibly with grouping/aggregation operations that have multiset (bag) semantics. A multiset algebra allows incrementally maintaining views that have the SQL multiset semantics [9, 28]. Duplicate retention (or at least a replication count) is essential if select-project views are to be self-maintainable [2, 11]. We think that it is necessary to include grouping/aggregation queries since they are extensively used in Data Warehousing applications.

4.2 Multiquery AND/OR dags

Alternative ways for evaluating a relational expression can be compactly represented by an *AND/OR dag* [32, 14]. A particular representation of AND/OR dags distinguishes between AND nodes and OR nodes [30] and has been developed initially for performing cost-based query optimization [8]. We use here this representation for multiple queries, extended with marked nodes to account for views materialized at the DW [43].

We start by defining expression and multiexpression AND/OR dags.

Definition 4.1 An *expression AND/OR dag* for an expression e defined over a set of views $\mathbf{V}$ is a rooted bipartite dag $\mathcal{G}_e$ defined as follows. The nodes of $\mathcal{G}_e$ are partitioned in AND nodes and OR nodes. AND nodes are called *operation nodes* and are labeled by operations while OR nodes are called *view nodes* and are labeled by views. In the following we may identify nodes with their labels. An operation node has one or two outgoing edges to view nodes and one incoming edge from a view node. Its meaning as an AND node is that its parent view can be obtained by applying the labeling operation to *all* its child views. A view node has one or more outgoing edges (if any) to operation nodes and one or more incoming edges (if any) from an operation node. Its meaning as an OR node is that the labeling view can be computed by applying *any* of the child operations to their respective child views. The root node and sink nodes of $\mathcal{G}_e$ are view nodes. The root node is labeled by e and represents alternative ways of evaluating e (alternative equivalent rewritings of e over $\mathbf{V}$), while the sink nodes are labeled by the views in $\mathbf{V}$.

Given a set of expressions $\mathbf{E}$ defined over a set of views $\mathbf{V}$, a *multiexpression AND/OR dag* $\mathcal{G}$ for $\mathbf{E}$ is an AND/OR dag resulting by merging the expression AND/OR dags for the expressions in $\mathbf{E}$. $\mathcal{G}$ is not necessarily a rooted dag (that is it does not necessarily have a single root). All the root nodes of $\mathcal{G}$ are view nodes labeled by expressions in $\mathbf{E}$ (but not all the expressions in $\mathbf{E}$ label necessarily root nodes). The sink nodes in $\mathcal{G}$ are labeled exactly by the views in $\mathbf{V}$. In addition, view nodes in a (multi)expression AND/OR dag *can be marked*. Marked nodes represent views materialized at the DW. □

Additonal auxiliary definitions are provided below.

Definition 4.2 A *(multi)expression dag* is a (multi)expression AND dag (that is is a (multi)expression AND/OR dag such that no view node has more than one outgoing edges).

A (multi)expression AND/OR dag $\mathcal{G}'$ is a *subdag* of a (multi)expression AND/OR dag $\mathcal{G}$ if and only if:

(a) dag $\mathcal{G}'$ is a subdag of dag $\mathcal{G}$,

(b) if an operation node of $\mathcal{G}$ is in $\mathcal{G}'$, all its child view nodes in $\mathcal{G}$ are in $\mathcal{G}'$, and

(c) all and only the marked nodes in $\mathcal{G}$ that are present in $\mathcal{G}'$ are marked nodes in $\mathcal{G}'$. □

We can now define query and multiquery AND/OR dags.

Definition 4.3 A *query AND/OR dag* for a query Q defined over a set of source relations $\mathbf{R}$ is an expression AND/OR dag for Q. The sink nodes of the dag are labeled by source relations. A query dag for Q is essentially a query evaluation plan of Q from the source relations. A *multiquery AND/OR dag* for a set of queries $\mathbf{Q}$ defined over $\mathbf{R}$ is an expression AND/OR dag for $\mathbf{Q}$. View nodes representing (and labeled by) the queries in $\mathbf{Q}$ are called *query nodes*. □

In the following we consider only multiquery AND/OR dags $\mathcal{G}$ for the set of queries satisfied by a DW, where the

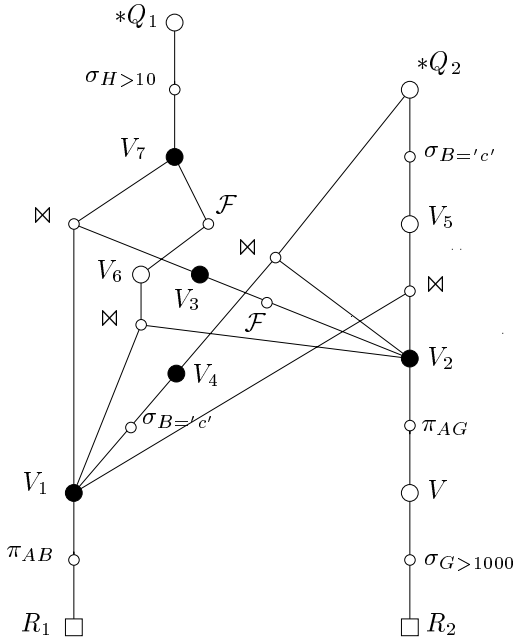

Figure 2: A multiquery AND/OR dag for $\mathbf{Q} = \{Q_1, Q_2\}$

marked nodes are exactly the materialized views of the DW. In these multiquery AND/OR dags, all the paths from a query node to a source relation contain a source relation image. Note that a source relation node R may coincide with its image node (for instance if there are no selections or projections over R in a query represented in $\mathcal{G}$). In this case R is replicated at the DW.

Example 4.1 Consider the source relations $R_1(\underline{A}, B, C)$ and $R_2(\underline{D}, E, G, A)$. Underlined attributes denote the key of the corresponding relation. Let Q_1 be the SQL query:

SELECT $R1.A,\ R1.B,$ COUNT$(R1.G)$ AS H
FROM $R1,\ R2$
WHERE $R1.A = R2.A$ AND $R2.G > 1000$
GROUP BY $R1.A,\ R1.B$
HAVING COUNT$(R1.G) > 10$

and Q_2 be the SQL query:

SELECT $R1.A,\ R1.B,\ R2.G$
FROM $R1,\ R2$
WHERE $R1.A = R2.A$ AND $R2.G > 1000$ AND $R1.B =' c'$

Figure 2 shows a multiquery AND/OR dag for $\mathbf{Q} = \{Q_1, Q_2\}$ over $\mathbf{R} = \{R_1, R_2\}$. Operation nodes are depicted by small circles while view nodes are depicted by bigger ones. $\bowtie$ denotes the natural join operation. Symbol $\mathcal{F}$ stands as a shorthand for the expression $< A > \mathcal{F} < count(G)$ as $H >$ denoting a grouping/aggregation operation: the prefix $< A >$ indicates the grouping attribute, while the suffix $< count(G)$ as $H >$ indicates the aggregate function, the aggregated attribute, and the attribute renaming. Query nodes are emphasized by preceding their names by a *. Two query dags for each query are represented. Note that as shown in [46] in this case the grouping/aggregation operator can be pushed past the join. Marked nodes (materialized views) are depicted by filled black circles. Source relation R_1 and R_2 are depicted by rectangles. Their images are the views V_1 and V_2 respectively. Remark that all the paths from query node Q_1 or Q_2 to source relation node R_1 or R_2 contain a source relation image. □

Construction of multiquery AND/OR dags. A multiquery AND/OR dag for a query Q can be constructed by applying transformation rules to an initial query dag. The initial query dag represents the expression defining Q. The transformation rules add new operation and view nodes and link these nodes with existing nodes by adding new edges. The dag resulting by the application of the transformation rules is a query AND/OR dag for Q [8]. A multiquery AND/OR dag for a set of queries $\mathbf{Q}$ can be constructed by merging equivalent view nodes of the query AND/OR dags for the queries in $\mathbf{Q}$. [31] presents transformation rules for multiquery AND/OR dags using a different representation scheme.

Expression AND/OR dags is a general formalism that captures many of the notions mentioned previously. In particular they can represent views and materialized views (source relation images, simple and auxiliary views), complete rewritings of the queries over the materialized views and/or the source relations, and common subexpressions between (maintenance) queries, between views, and between (maintenance) queries and views. Different types of expression AND/OR dags will be used below to represent query evaluation plans and change propagation plans.

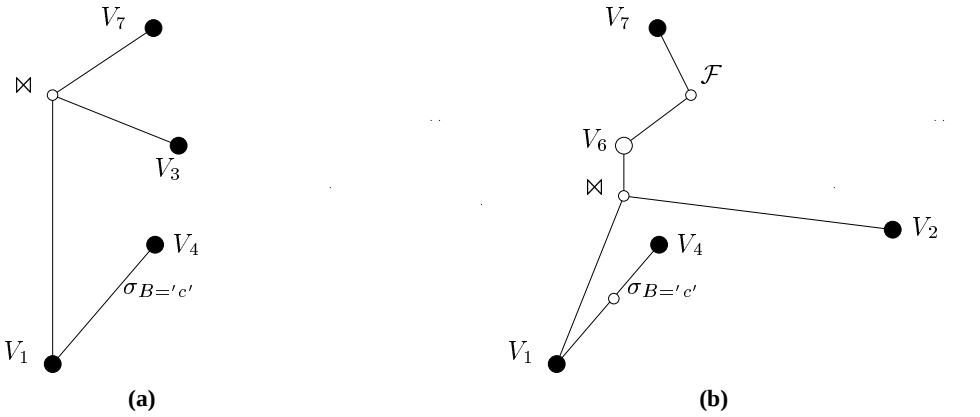

Figure 3: Change propagation dags for V_1

4.3 Query evaluation and change propagation dags

Consider a multiquery AND/OR dag $\mathcal{G}$ for a set of queries $\mathbf{Q}$. A query evaluation plan over materialized views is represented by a query evaluation dag defined as follows.

Definition 4.4 A *query evaluation dag for a query* $Q \in \mathbf{Q}$ is an expression AND subdag E of $\mathcal{G}$ such that:
(a) E is rooted at Q.
(b) All and only the sink nodes of E are marked nodes. □

Example 4.2 Consider the multiquery AND/OR of example 4.1. Figure 4 shows two query evaluation dags for the queries Q_1 and Q_2 respectively. □

A change propagation plan is represented by a change propagation dag defined below.

Definition 4.5 A *change propagation dag for a source relation image* V is a multiexpression AND subdag U of $\mathcal{G}$ such that:
(a) All the marked view nodes that are ancestor nodes of V in $\mathcal{G}$ (that is the marked view nodes that occur in a path from a root node to V in $\mathcal{G}$) are present in U, and the root nodes of U are among them.
(b) The sink nodes of U are marked nodes of $\mathcal{G}$, and V is one of them.
(c) The non-sink marked nodes in U are ancestor nodes of V. □

Clearly a change propagation dag is a connected graph.

Example 4.3 Consider the multiquery AND/OR dag of example 4.1. Figure 3 shows two different change propagation dags for the source relation image V_1. Figure 5 shows two change propagation dags for the source relation image V_2. As this example makes clear, there can be more than one change propagation dags for a source relation image in a given multiquery AND/OR dag. □

A change propagation dag for the image V of a source relation R indicates the way the changes to V are propagated to the views that are affected by these changes. Recall that the type of multiquery AND/OR dags we consider here implies that the materialized views that are affected by the changes to R are also affected by the changes to V. We describe this change propagation process below.

4.4 Incremental view maintenance using change propagation dags

The changes to the materialized views that are affected by the changes to a source relation image can be computed using the maintenance expressions provided in [9] for a multiset algebra and in [28] for grouping/aggregation operators under multiset semantics. We call these expressions maintenance queries. Maintenance queries involve in general the pre-update state of the source relation images (that is the state prior to the application of the changes), the pre-update state of the materialized view, and the changes to the source relation image.

Example 4.4 Consider the query $Q_2 = \sigma_{B='c'}(V_1 \bowtie V_2)$ of example 4.1 rewritten over the materialized views V_1 and V_2, and suppose that it is a view V materialized at the DW. Let ΔV_1 denote the tuples inserted to V_1, and ∇V_1 denote the tuples deleted from V_1. Recall that we consider net changes. Therefore, any tuple in ∇V_1 appears in V_1 at least as many times as in ∇V_1, and ΔV_1 and ∇V_1 do not have any tuple in common. The net changes to V are computed by the following maintenance queries. $\sigma_{B='c'}(\Delta V_1 \bowtie V_2)$ evaluates to the tuples to be inserted into V_6 and $\sigma_{B='c'}(\nabla V_1 \bowtie V_2)$ to the tuples to be deleted from V_6. V_1 and V_2 in the maintenance queries denote the pre-update states of views V_1 and V_2. □

The changes to the source relation images (which are select-project views) are computed using maintenance queries, exclusively from the changes to the source relations (select-project views are self-maintainable when multiset semantics are adopted).

If the maintenance queries for different materialized views that are affected by the changes to a source relation

image have a common subexpression, we can use the techniques of multiquery optimization to compute this subexpresion only once. Further if a subexpression (that does not include changes) of a maintenance query is a view already materialized in the DW (other than a source relation image), we can use this view as an auxiliary view in the computation of the maintenance query. Therefore, the computation of this subexpression from the source relation images is avoided.

We maintain the views affected by the changes to the image of a source relation R by considering a change propagation dag U for the image of R, and by proceeding in a bottom-up fashion. This way, we can take advantage of the views materialized in the DW that are not affected by the changes, while common subexpressions between maintenance queries are computed only once.

The changes to a (marked or non-marked) view node V that is affected by the changes to the source relation image are computed from the changes to its child view node(s) V_i. (A view node is affected by the changes to the source relation image if it is an ancestor of the source relation image sink node.) In general, the pre-update state of each V_i, and the pre-update state of V are also needed for this computation (V's pre-update state can of course be computed from the pre-update state of V_is). In particular cases, the pre-update state of V and/or the pre-update state of V_is are not needed [9, 28]. For instance, if V is a self-maintainable view (with respect to the changes to V_is) the pre-update state of V_is is not needed. As another example, if V is obtained by a selection or a projection or an additive union on V_is, neither the pre-update state of any V_is nor V's pre-update state are needed: V's changes can be computed exclusively from the changes to V_is. If V is marked (materialized), the computed changes are applied to it. However, these changes are not applied until the changes and the pre-update state (if needed) of the parent view nodes of V are computed. Thus the pre-update state of V remains available where needed.

The (pre-update state) of a non-marked view node V is computed from the (pre-update state of its) child view node(s) V_i. This computation is not necessary if V is not needed neither for the computation of the changes to V nor by any of its parent view nodes. A parent view node of V may need the pre-update state of V for the computation of its own changes or (in case it is a non-marked view) because its own pre-update state is needed by one of its parents. By a top-down scan of the change propagation dag, prior to the propagation of the changes, the non-marked view nodes that need not be computed can be detected [39].

Example 4.5 Consider the change propagation dag U for V_1 of Figure 3(b). There in only one non-marked view node V_6 in U. The only parent view node V_7 of V_6 is a marked node and does not need the pre-update state of V_6 for the computation of its own changes [28]. V_6 itself is obtained by joining its child nodes and thus its pre-update state is not needed for the computation of its own changes [9]. Therefore the computation of the pre-update state of V_6 is not needed in U. □

4.5 Time cost functions

We now introduce time cost functions that we use in determining the time needed to evaluate queries and to propagate changes to the materialized views.

Consider a multiquery AND/OR dag $\mathcal{G}$ for a set of queries $\mathbf{Q}$ defined over a set of source relations $\mathbf{R}$. Let U denote an update propagation plan in $\mathcal{G}$ for the image of source relation $R \in \mathbf{R}$.

With every operation node O in $\mathcal{G}$ a cost t_O is associated. Cost t_O denotes the time needed to compute the parent view node of O from the its child node(s), assuming that the later are already computed. $t_O = 0$ if the parent view node is marked.

With every operation node O in U, a cost t_O^U is associated. Cost t_O reflects the time needed to compute the parent view node V of O from its child node(s), if V is not marked and is needed in U (refer to the previous subsection). Therefore, $t_O^U = t_O$ if V is needed in U, and $t_O^U = 0$ otherwise.

With every operation node O in $\mathcal{G}$, a cost t_O^R is associated. t_O^R reflects the time needed to compute the changes to the parent view node V of O assuming that the pre-update state of V, and the pre-update state and the changes of the child view node(s) of O are available. $t_O^R = 0$ if O is not an ancestor of R in $\mathcal{G}$.

With every view node V in $\mathcal{G}$, a cost t_V^R is associated. t_V^R denotes the time needed to apply the changes to V when the changes to source relation R are propagated to the materialized views. $t_V^R = 0$ if V is not a materialized view or if V is not an ancestor of R in $\mathcal{G}$ (in the last case V it not affected by the changes to R). Clearly, t_O^R and t_V^R are independent of the change propagation dag for the image of R used to propagate the changes of V.

t_R^R denotes the time needed by the source holding R to compute the changes to the image of R from the changes stored in the buffer. The time needed to transmit the changes to the image of R from the source of R to the DW is denoted by t_R^{tr}.

We assume that the time is measured in units representing the lowest granularity of interest. The time costs depend on hardware features, the physical storage model and the availability of indexes which must also be updated. They also depend on the sizes of source relations and their changes which we assume that they are relatively stable over a long period of time. Table 1 summarizes the symbols for the cost functions introduced above and other symbols introduced below.

Given a query evaluation dag E for a query Q, the time t_Q^E needed to compute Q according to E is given by the

Symbol	Meaning
R	source relation
O	operation node
V	view node
U	update propagation dag for the image of R
Q	query defined using R
t_O	time needed to compute the parent view node of O
t_O^U	t_O if the parent view node of O is needed in U and 0 otherwise
t_O^R	time needed to compute the changes to the parent view node of O due to changes to R
t_V^R	time needed to apply the changes to V due to changes to R
t_R^R	time needed to compute the changes to the image of R due to changes to R
t_R^{tr}	time needed to transmit the changes to the image of R
t_R^Q	currency for the data from R in the answer of Q
T_R^Q	minimal currency required for the data from R in the answer of Q
f_Q	frequency of issuing Q
f_R	frequency of propagating the changes to the image of R
F_R	maximal frequency allowed for propagating the changes to the image of R

Table 1: Summary of symbols used and their meaning

formula:

$$t_Q^E = \sum_{O \in E} t_O$$

Different query evaluation dags for the same query yield different time costs. The query evaluation dag for Q yielding the minimal time cost is called *optimal query evaluation dag for* Q. The minimal time t_Q needed to compute Q is:

$$t_Q = \min_E \{t_Q^E\}$$

Let $\mathbf{Q} = \{Q_1, \ldots, Q_m\}$. For a query $Q_i \in \mathbf{Q}$, f_{Q_i} denotes the frequency with which query Q_i is issued against the DW. Then the DW query evaluation cost P is provided by the formula:

$$P = \sum_{i \in [1,m]} f_{Q_i} t_{Q_i} \qquad (1)$$

Given a change propagation dag U for the image V of a source relation R, the time t_R^U needed to propagate the changes to V to all the materialized views in U that are affected by these changes is given by the formula

$$t_R^U = \sum_{O \in U} t_O^U + \sum_{O \in U} t_O^R + \sum_{V \in U} t_V^R$$

Example 4.6 Consider the change propagation dag U for V_1 of Figure 3(b). As shown in example 4.5, the pre-update state of the non-marked view node V_6 (parent of the operation node $\bowtie$) need not be computed in U. All the other view nodes, parents of operation nodes in U, are materialized and thus their pre-update state is available. Therefore, $t_\sigma^U = t_\bowtie^U = t_\mathcal{F}^U = 0$. $t_{V_6}^{R_1} = t_{V_2}^{R_1} = 0$ since V_6 is not a marked view node, and V_2 is not an ancestor of V_1 in $\mathcal{G}$. Then, $t_{R_1}^U = (t_\sigma^{R_1} + t_\bowtie^{R_1} + t_\mathcal{F}^{R_1}) + (t_{V_1}^{R_1} + t_{V_4}^{R_1} + t_{V_7}^{R_1})$. □

Different change propagation dags for the same source relation image yield different time costs. The change propagation dag for V yielding the minimal time cost is called *optimal change propagation dag for* V. The minimal time t_R needed to propagate the changes to V to all the materialized views in $\mathcal{G}$ that are affected by the changes to V is:

$$t_R = \min_U \{t_R^U\}$$

Let $\mathbf{R} = \{R_1, \ldots, R_n\}$. For a source relation $R_i \in \mathbf{R}$, f_{R_i} denotes the frequency with which the changes to V are propagated from the corresponding source to the DW. Then the DW view maintenance cost M is given by the following formula:

$$M = \sum_{i \in [1,n]} f_{R_i} t_{R_i} \qquad (2)$$

We can now define source availability constraints

Definition 4.6 Let F_R be a frequency value expressing the maximal frequency allowed for the propagation of the changes from the source holding source relation R. A *source availability constraint* for R is an inequality of the form $f_R \leq F_R$. We also define $T_R = 1/F_R$. □

Before defining currency constraints, we provide a definition on answer data currency.

Definition 4.7 Let t_1 be the time point (commit point of the corresponding transaction) when the answer of a query Q is returned to the user. Suppose that the most recent changes to a source relation R that are propagated and applied to the affected views and are taken in consideration in the evaluation of Q are read at the source at time t_2. We define $t_R^Q = t_1 - t_2$. t_R^Q expresses the *currency of the data from R in the answer of Q*. □

Note that a higher value for t_R^Q means that the data from R in the answer of Q are "older" (less current). Currency constraints are defined as follows.

Definition 4.8 Let T_R^Q be a time value expressing the minimal currency required for the data from R in the answer of Q. A *currency constraint* is an inequality of the form $t_R^Q \leq T_R^Q$. □

The time costs associated with the view and operation nodes of $\mathcal{G}$ can be computed given statistics about the source relations and information about the hardware and the network. Our approach is independent of the cost model used but of course the solution to the problem depends on it. Query AND/OR dags used in real optimizers [8] incorporate also complex mappings of consecutive operations to a single operation node (e.g. a join followed by a projection) as well as physical properties (e.g sort order). These features affect the computed time costs, but we do not go to that depth of detail for simplicity.

4.6 The currency constraint satisfiability problem

We can now state formally the currency constraint satisfiability problem.

Input:

- A multiquery AND/OR dag $\mathcal{G}$ for a set of queries $\mathbf{Q}$ defined over a set of source relations $\mathbf{R}$.
- For every $R \in \mathbf{R}$, a source availability constraint $f_R \leq F_R$.
- For every $Q \in \mathbf{Q}$ and every R in the definition of Q, a currency constraint $t_R^Q \leq T_R^Q$.
- Time cost functions t_O, t_O^U, t_O^R, t_V^R, t_R^R, and t_R^{tr}.

Output:

- A decision on whether the constraints can be satisfied.
- If the constraints can be satisfied:
  - The minimal change propagation frequencies that guarantee the satisfaction of the constraints
  - The corresponding minimal view maintenance cost M, and the optimal change propagation dags for the images of the source relations in $\mathbf{R}$.
  - The minimal query evaluation cost P, and the optimal query evaluation dags for the queries in $\mathbf{Q}$.
- If the constraints can not be satisfied:
  - The source relations that cause the violation of the constraints.

5 A solution to the problem

We now present a solution to the currency constraint satisfiability problem. Our approach proceeds in three steps. In the first step the optimal query evaluation dags and the minimal query evaluation cost is determined, and the currency constraints are "pushed down" from the queries to the simple views. In the second step, the optimal change propagation dags are determined and the source availability constraints are "pushed up" from the source relations to the simple views. The third step checks constraint satisfability, and computes the minimal change propagation frequencies and the minimal view maintenance cost.

5.1 Pushing currency constraints down to the simple views

In this step we first detect the alternative query evaluation dags in $\mathcal{G}$ for every query in $\mathbf{Q}$. Then, the optimal one for each query is chosen among them. This procedure allows: (a) determining the simple views in $\mathbf{G}$ (that is the materialized views that occur in the optimal dags), and (b) computing the minimal query evaluation cost E using formula (1). Recall that, by definition, the only marked nodes occurring in a query evaluation dag are sink nodes.

Example 5.1 Consider the multiquery AND/OR dag $\mathcal{G}$ of example 4.1. One can easily see that there is only one query evaluation dag for Q_1, and two query evaluation dags for Q_2 in $\mathcal{G}$. Assuming, for the needs of this example, that there are not indexes on the materialized views, performing a natural join of V_1 and V_2 and then a selection is more time consuming than performing a single natural join of V_4 and V_2. Therefore, the optimal query evaluation dags for Q_1 and Q_2 are those depicted in Figure 4. The marked view nodes V_7, V_3 and V_4 are the simple views in $\mathcal{G}$. □

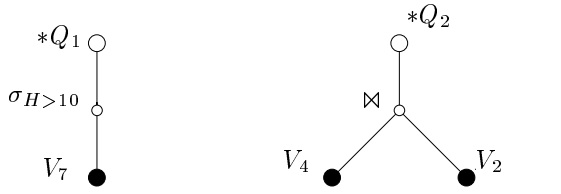

Figure 4: Optimal query evaluation dags for Q_1 and Q_2

Then, the currency constraints are "pushed" from the query nodes down to the simple views along optimal query evaluation dags as follows:
Suppose that a simple view V occurs in the optimal query evaluation dags for the queries $Q_1, \ldots, Q_k$. Let R be a source relation in the view definition of V. We define

$$H_V^R = \min_{i \in [1,k]} \{T_R^{Q_i} - t_{Q_i}\}$$

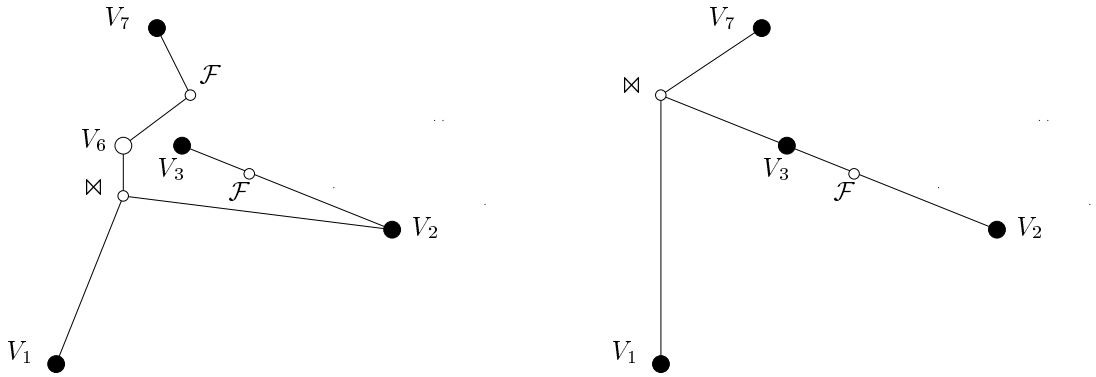

Figure 5: Change propagation dags for V_2

H_V^R represents the minimal currency required for the data from source relation R in the simple view V as set by the currency constraints associated with Q_i and R, $i = 1, \ldots k$.

5.2 Pushing source availability constraints up to the simple views

In this step we first detect the alternative change propagation dags for each source relation image in $\mathcal{G}$. Then, the optimal one for each source relation image is chosen among them.

Example 5.2 Consider the multiquery AND/OR dag $\mathcal{G}_V$ shown in Figure 2. $\mathcal{G}_V$ represents two change propagation dags for each of the source relation images V_1 (shown in Figure 3) and V_2 (shown in Figure 5). Assuming that there are no indexes, it is reasonable to consider that computing the changes to view node V_7 from the changes to V_1 and the pre-update state of V_3 is less time consuming than computing the changes to V_6 from the changes to V_1 and the pre-update state of V_2 and then the changes to V_7 from the changes to V_6 and the pre-update state of V_7. Note that V_3 has no more tuples than V_2, and the computation of V_6 is not needed as shown in example 4.5. Therefore the change propagation dag of Figure 3(a) is the optimal change propagation dag for V_1. By a similar argument, the change propagation dag of Figure 5(b) is the optimal change propagation dag for V_2. □

The source availability constraints are pushed from the source relations up to the simple views along optimal change propagation dags as follows:
Consider a simple view V defined using source relation R. We define

$$L_V^R = T_R + t_R^R + t_R^{tr} + t_R$$

L_V^R represents the maximal currency allowed for the data from source relation R in the simple view V as set by the source availability constraint associated with R. Note that we do not fix a specific order for applying the changes to the simple views in a change propagation dag. Therefore, we are sure that the new state of each of these views is available for querying only when the propagation of the changes using this propagation dag has finished.

5.3 Checking constraint satisfaction

This step starts by checking the satisfaction of the constraints. The constraints are satisfied if and only if for every simple view V in $\mathcal{G}$ and for every source relation R in the view definition of V, $L_V^R \leq H_V^R$.

A source relation R violates the constraints if and only if there is a view V defined using R such that $L_V^R \not\leq H_V^R$. The information on the violating source relations can be exploited by the DW designer in providing better view selections for materialization (see next section).

Suppose now that the constraints are satisfiable. We show how the minimal change propagation frequencies that guarantee the satisfaction of the constraints can be determined. Consider a source relation R and let $V_1, \ldots, V_l$ be the simple views that are ancestors of R in $\mathcal{G}$. We define

$$H_R = \min_{i \in [1,l]} \{H_R^{V_i}\}$$

Let $T_R^{min} = H_R - (t_R^R + t_R^{tr} + t_R)$. Clearly, since the constraints are satisfied, $H_R - (t_R^R + t_R^{tr} + t_R) \geq T_R$. Then, the minimal change propagation frequency for R is $f_R^{min} = 1/T_R^{min}$.

The corresponding minimal view maintenance cost M^{min} is computed using formula (2):

$$M^{min} = \sum_{i \in [1,n]} f_{R_i}^{min} t_{R_i}$$

6 Discussion

When the constraints cannot be satisfied, our approach detects the source relations that violate the constraints. The DW designer can use this information in order to improve the selected view set. Suppose for instance that the view

$V = \sigma_{C_1}(R_1) \bowtie V'$, where V' is a complex view, is materialized at the DW. Further, suppose that there are no materialized views involving the source relations used in the definition of V' (besides source relation images). Then, by adding the materialized view V' as an auxiliary view the time cost of the optimal change propagation plan for R_1 is reduced. This addition may be sufficient for satisfying the constraints that were previously violated. Note, however, that this change may cause the violation of other constraints involving source relations in the definition of V'.

The selected view set can also be improved in order to satisfy the constraints by removing redundant auxiliary views. Consider for instance the case where in each optimal change propagation dag, a specific auxiliary view is either a root node, or does not appear at all in it. Such a view is redundant: it is not used for answering the queries (since it is not a simple view), and is not useful for reducing the view maintenance cost of other materialized views (since it is a root node in the optimal plans where it appears). Removing such a view V from the DW reduces the cost of propagating changes along the optimal change propagation dags that contain V. This reduction may be sufficient for satisfying a constraints that was previously violated. In [39] we provide an approach for detecting redundant views in a DW. Even when the constraints are satisfied, by removing redundant views, the change propagation frequencies (and therefore the view maintenance cost) can be further reduced.

If the time cost of the change propagation dags cannot be further reduced, the DW designer can opt for hardware improvements (e.g. faster links between the sources that violate the constraints and the DW). As a final solution, he can negotiate with the source administrators and the analysts the relaxing of the source availability and/or the currency constraints.

Our approach can be combined with a view selection algorithm. In [41, 42, 40] such algorithms are provided that generate alternative view sets satisfying the data completeness quality goal and choose the one that minimizes a combination of the query evaluation and view maintenance cost. Using our results, the generated view sets can be checked, instead, for satisfaction of the quality goals. This suggests for a quality driven DW design. Note finally that the detection of violated constraints and of useless auxiliary views can guide the devise of heuristics. The later are necessary for pruning the search space of alternative view selections which can be very large.

7 Conclusion and possible extensions

Up to now research work on DW design issues is restricted to quantitatively selecting view sets for materialization in a DW. However the design of a DW is subject to a number of quality factors. In this paper we have presented a novel statement for the DW view selection problem based on the satisfaction of performance, data consistency, data completeness, and data currency quality goals. The data currency quality goal is specified by a number of detailed currency constraints at the query level while availability constraints at the data source level are also taken into account. We have described a DW system architecture that supports the performance quality goal and satisfies the data consistency and completeness quality goals. In this framework we have addressed the problem of checking whether a view selection satisfies the given constraints. We have presented an approach for solving this problem that uses an AND/OR dag representation for multiple queries and views. Our approach allows also determining the change propagation frequencies that minimizes the view maintenance time cost and computes the optimal change propagation and query evaluation plans. More importantly, it can help the DW designer in the improvement of the selected view set, and can cooperate with DW design algorithms to generate a view set that satisfies the quality goals.

Future work includes the integration of our approach with DW view selection algorithms and the experimental validation of an automatic quality-oriented DW design process.

References

[1] J. A. Blakeley, N. Coburn, and P. A. Larson. Updating derived relations: detecting irrelevant and autonomously computable updates. *ACM Transactions on Database Systems*, 14(3):369–400, 1989.

[2] J. A. Blakeley, P. Larson, and F. W. Tompa. Efficiently updating materialized views. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 61–71, 1986.

[3] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *Proc. of the 20th Intl. Conf. on Very Large Data Bases*, pages 577–589, 1991.

[4] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, 1997.

[5] L. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey. Algorithms for Deferred View Maintenance. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 469–480, 1996.

[6] L. Colby, A. Kawagushi, D. Lieuwen, I. S. Mumick, and K. Ross. Supporting Multiple View Maintenance Policies. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 1997.

[7] S. Dar, H. V. Jagadish, A. Y. Levy, and D. Srivastava. Answering SQL Queries with Aggregation using Views. In *Proc. of the 22nd Intl. Conf. on Very Large Data Bases*, pages 318–329, 1996.

[8] G. Graefe and W. J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proc. of the 9th Intl. Conf. on Data Engineering*, pages 209–217, 1993.

[9] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 328–339, 1995.

[10] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-Query Processing in Data Warehousing Environments. In *Proc. of the 21st Intl. Conf. on Very Large Data Bases*, pages 358–369, 1995.

[11] A. Gupta, H. Jagadish, and I. S. Mumick. Data Integration using Self-Maintainable Views. In *Proc of the 5th EDBT Conf.*, pages 140–144, 1996.

[12] A. Gupta, I. Mumick, and V. Subrahmanian. Maintaining views incrementally. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 157–166, 1993.

[13] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques and applications. *Data Engineering*, 18(2):3–18, 1995.

[14] H. Gupta. Selection of Views to Materialize in a Data Warehouse. In *Proc. of the 6th Intl. Conf. on Database Theory*, pages 98–112, 1997.

[15] H. Gupta and I. S. Mumick. Selection of Views to Materialize Under a Maintenance Cost Constraint. In *Proc. of the 7th Intl. Conf. on Database Theory*, pages 453–470, 1999.

[16] E. Hanson. A Performance Analysis of View Materialization Strategy. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 440–453, 1987.

[17] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing Data Cubes Efficiently. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 1996.

[18] R. Hull and G. Zhou. A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 481–492, 1996.

[19] M. Jarke. Common Subexpression Isolation in Multiple Query Optimization. In Kim, Reiner, and Batory, editors, *Query Processing in DB Systems*, pages 191–205. Springer-Verlag, 1984.

[20] M. Jarke, M. Jeusfeld, C. Quix, and P. Vassiliades. Architecture and Quality in Data Warehouses. In *Proc. of the 10th Intl. Conf. on Advanced Information Systems Engineering*, pages 93–113, 1997.

[21] M. Jarke and Y. Vassiliou. Data Warehouse Quality: A Review of the DWQ Project. In *Proc. of the 2nd Intl. Conf. on Information Quality* Cambridge, Mass., pages 98–112, 1997.

[22] M. Jeusfeld, C. Quix, and M. Jarke. Design and Analysis of Quality Information for Data Warehouses. In *Proc. of the 17th Intl. Conf. on Conceptual Modeling, Springer LNCS 1507*, 1998.

[23] W. J. Labio and H. Garcia-Molina. Efficient Snapshot Differential Algorithms for Data Warehousing. In *Proc. of the 22nd Intl. Conf. on Very Large Data Bases*, pages 63–74, 1996.

[24] A. Levy, A. O. Mendelson, Y. Sagiv, and D. Srivastava. Answering Queries using Views. In *Proc. of the ACM Symp. on Principles of Database Systems*, pages 95–104, 1995.

[25] A. Y. Levy and Y. Sagiv. Queries Independent of Updates. In *Proc. of the 19th Intl. Conf. on Very Large Data Bases*, pages 171–181, 1993.

[26] F. Llirbat, E. Simon, and D. Tombroff. Using Versions in Update Transactions. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 96–105, 1997.

[27] X. Qian and G. Wiederhold. Incremental Recomputation of Active Relational Expressions. *IEEE Transactions on Knowledge and Data Engineering*, 3(3):439–450, 1991.

[28] D. Quass. Maintenance Expressions for Views with Aggregation. In *Workshop on Materialized Views: Techniques and Applications*, pages 110–118, 1996.

[29] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making Views Self Maintainable for Data Warehousing. In *Proc. of the 4th Intl. Conf. on Parallel and Distributed Information Systems*, 1996.

[30] K. A. Ross, D. Srivastava, and S. Sudarshan. Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 447–458, 1996.

[31] N. Roussopoulos. The Logical Access Path Schema of a Database. *IEEE Transactions on Software Engineering*, 8(2):563–573, 1982.

[32] N. Roussopoulos. View Indexing in Relational Databases. *ACM Transactions on Database Systems*, 7(2):258–290, 1982.

[33] A. Segev and W. Fang. Currency-based updates to distributed materialized views. In *Proc. of the 6th Intl. Conf. on Data Engineering*, pages 512–520, 1990.

[34] A. Segev and J. Park. Updating distributed materialized views. *IEEE Transactions on Knowledge and Data Engineering*, 1(2):173–184, 1989.

[35] T. K. Sellis. Multiple Query Optimization. *ACM Transactions on Database Systems*, 13(1):23–52, 1988.

[36] K. Shim, T. K. Sellis, and D. Nau. Improvements on a Heuristic Algorithm for Multiple-Query Optimization. *Data & Knowledge Engineering*, 12:197–222, 1994.

[37] J. Srivastava and D. Rotem. Analytical Modeling of Materialized View Maintenance. In *Proc. of the 5th ACM Symp. on Principles of Database Systems*, pages 126–134, 1988.

[38] M. Staudt and M. Jarke. Incremental Maintenance of Externally Materialized Views. In *Proc. of the 22nd Intl. Conf. on Very Large Data Bases*, pages 75–86, 1996.

[39] D. Theodoratos. Detecting Redundnacy in Data Warehouse Design. *Technical Report, Knowledge and Data Base Systems Laboratory, Electrical and Computer Engineering Dept., National Technical University of Athens*, pages 1–20, 1999.

[40] D. Theodoratos, S. Ligoudistianos, and T. Sellis. Designing the Global Data Warehouse with SPJ Views. To appear in *Proc. of the 11th Intl. Conf. on Advanced Information Systems Engineering*, 1999.

[41] D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 126–135, 1997.

[42] D. Theodoratos and T. Sellis. Data Warehouse Schema and Instance Design. In *Proc. of the 17th Intl. Conf. on Conceptual Modeling, Springer LNCS 1507*, pages 363–376, 1998.

[43] D. Theodoratos and T. Sellis. Dynamic Data Warehouse Design. To appear in *Proc. of the 1st Intl. Conf. on Data Warehousing and Knowledge Discovery, Springer-Verlag, LNCS*, 1999.

[44] J. Widom. Research Problems in Data Warehousing. In *Proc. of the 4th Intl. Conf. on Information and Knowledge Management*, pages 25–30, Nov. 1995.

[45] J. Wiener, H. Gupta, W. Labio, Y. Zhuge, H. Garcia-Mollina, and J. Widom. A System Prototype for Warehouse View Maintenance. In *Workshop on Materialized Views: Techniques and Applications*, 1996.

[46] W. Yan and P.-Å. Larson. Performing Group-By before Join. In *Proc. of the 10th Intl. Conf. on Data Engineering*, pages 89–100, 1994.

[47] J. Yang, K. Karlapalem, and Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 136–145, 1997.

[48] G. Zhou, R. Hull, R. King, and J.-C. Franchitti. Supporting Data Integration and Warehousing Using H20. *Data Engineering*, 18(2):29–40, 1995.

[49] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 316–327, 1995.

[50] Y. Zhuge, H. Garcia-Molina, and J. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *Proc. of the 4th Intl. Conf. on Parallel and Distributed Information Systems*, 1996.

[51] Y. Zhuge, J. Wiener, and H. Garcia-Molina. Multiple View Consistency for Data Warehousing. In *Proc. of the 13th Intl. Conf. on Data Engineering*, pages 289–300, 1997.