Reducing the Size of Auxiliary Data Needed to Support Materialized View Maintenance in a Data Warehouse Environment

Lubomir Stanchev, Indiana University – Purdue University Fort Wayne, 2101 E. Coliseum Blvd., Fort Wayne, IN 46805, USA; E-mail: stanchel@ipfw.edu

ABSTRACT

A data warehouse consists of a set of materialized views that contain derived data from several data sources. Materialized views are beneficial because they allow efficient retrieval of summary data. However, materialized views need to be refreshed periodically in order to avoid staleness. During a materialized view refresh only changes to the base tables are transmitted from the data sources to the data warehouse, where the data warehouse should contain the data from the base tables that is relevant to the refresh. In this paper we explore how this additional data, which is commonly referred to as auxiliary views, can be reduced in size. Novel algorithms that exploit non-trivial integrity constraints and that can handle materialized views defined over queries with grouping and aggregation are presented.

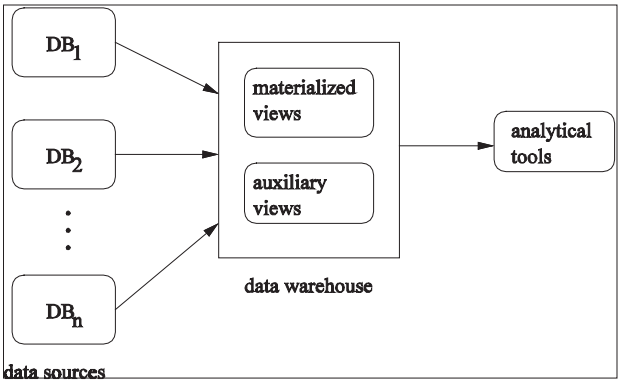
1. INTRODUCTION

A data warehouse contains aggregated data derived from a number of data sources and is usually used by OnLine Analytical Processing (OLAP) tools and data mining tools for the purpose of decision making (see Figure 1 and [GM95]).

The data sources consist of several databases, which usually contain huge amounts of data (e.g., the day-to-day transactions of a store chain). Conversely, *materialized views* (MVs) contain summary data compiled from several data sources. The main challenge in implementing the data warehouse architecture is keeping the materialized views up-to-date.

We will refer to the one time synchronization of a MV with the content of its underlying tables as a *view refresh* and to the continuous process of synchronization as *view maintenance*. In order to demonstrate how a MV can be refreshed, consider a MV V that is defined as the inner join of the base tables T_1 and T_2. If the changes ΔT_1 are applied to the table T_1, then the changes that need to be applied to V can be expressed as $\Delta T_1 \bowtie T_2$ [1]. However, in general it is impossible to calculate the value of T_2 knowing only the old value of V. Therefore, all the tuples in T_2 that can potentially join with ΔT_1 need to be stored in an *auxiliary view* on the data warehouse site.

Figure 1. The data warehouse model



In this paper we extend previous research on the subject (e.g., [QGMW96]) and explain how integrity constraints can be exploited to reduce the size of the created auxiliary views. In particular, our contributions include considering novel types of integrity constraints that can further reduce the size of the auxiliary views and algorithms for handling MVs defined over queries with grouping and aggregation.

2. RELATED RESEARCH

The problem of MV maintenance has been studied for over twenty years (see [BLT86]). The papers [GJM96] and [H96a] are excellent references on the problem of making MVs self-maintainable. MV maintenance over object-relational database schemas, similar to the one used in this paper, is presented in [ZM98], while [AHRVW98] describes how to maintain MVs over semi-structured data.

The paper [QGMW96] is an excellent source on exploiting integrity constraints to reduce the size of auxiliary views. However, it covers only candidate and foreign key integrity constraints and considers only conjunctive queries without grouping and aggregation. The paper [H96b] presents an algorithm for testing the self-maintainability of a MV in the presence of arbitrary functional dependencies.

3. PROBLEM DESCRIPTION

Our database schema consists of base tables and MVs, where only base tables can be updated by the users of the system. Each base table has the system attribute `ID`, which is a unique tuple identifier (and therefore a key for each table). The other attributes of a table are either *standard*, that is, from one of the predefined types (e.g., integer, string, etc.), or *reference* and contain the `ID` value of a tuple that is in the database (In other words, we require that all reference attribute define a referential non-null foreign key constraint). In addition, we impose the acyclicity requirement that there cannot exist reference attributes A_1, ..., A_n on tables T_1, ..., T_n, respectively, such that attribute A_i references table T_{i+1} for i = 1 to n-1 and attribute A_n references table T_1.

Given a MV V stored on the data warehouse, a database schema, and the type of changes that are allowed to the view's underlying tables, our goal is to find the smallest set of auxiliary views for V, where the precise definition of an auxiliary view follows. Note that we require that both the old and new values of updated tuples to be sent to the data warehouse.

Definition 1 (self-maintainable set of materialized views) The set of MVs $\overline{V}$ is self-maintainable iff every MV in the set can be refreshed using only the old values of $\overline{V}$ and the changes to the underlying base tables.

Definition 2 (auxiliary materialized views) The set of MVs $\overline{V}_1$ is an auxiliary set of MVs for the MV V iff $V \cup \overline{V}_1$ is a self-maintainable set of materialized views.

We will refer to the query that defines a MV as the *underlying query* for the view. In this paper we only consider MVs with underlying queries that are select-project-join queries (no self-joins allowed) with possible grouping and aggregation. We require that the selection condition of the underlying query is a conjunction of atomic predicates of the form "$T_1.Pf_1?P_1$" or "$T_1.Pf_1?T_2.Pf_2$", where *Pf* is used to denote a *path function* (precise definition follows), T - a base table, P - an atomic value, and "?" - an element of the set $\{>,\geq,=,<,\leq\}$.

Definition 3 (path function) A path function *Pf* has the general syntax $A_1^{a_1}.\ldots.A_n^{a_n}.A_{n+1}$, where $\{A_i\}_{i=1}^n$ are derived attributes and $\{a_i\}_{i=1}^n$ are elements of the set {1,-1}. Given a tuple t, we define $t.Pf$ to be equal to $((\ldots(t \Rightarrow A_1^{a_1}) \Rightarrow \ldots) \Rightarrow A_n^{a_n}).A_{n+1}$. Note that $t \Rightarrow A$ is used to denote the set of tuples with `ID` $t.A$ for $t \in t$. Similarly, $t.A^{-1}$ is used to denote all tuples t' for which $t' \Rightarrow A$ is in the set t. (We have used $t \Rightarrow A$ as a shorthand for $\{t\} \Rightarrow A$ and A as a shorthand for A^1.) The expression $t.Pf$ is *well defined* when it represents a set that contains a single value, where we will use $t.Pf$ to denote this value.

In addition to the key constraints defined by the ID attributes and the referential constraints defined by the reference attributes, our algorithms can take advantage of the following two integrity constraints:

(def $T.Pf$) $\Leftrightarrow$ ($\forall t \in T$) ($t.Pf$ is well defined) and

($T.Pf_1$=$T.Pf_2$) $\Leftrightarrow$ (def $T.Pf_1$) $\wedge$ (def $T.Pf_2$) $\wedge$ ($\forall t \in T$)($t.Pf_1 = t.Pf_2$).

The first constraint denotes that a path function is well defined and the second constraint states that we will reach the same value if we follow either of the two paths.

Our running example is based on the database schema shown in Figure 2. We have used ellipses around base table names and round rectangles around primitive types. Also, we have used dashed lines to denote standard attributes, solid lines to denote reference attributes, and the ID attributes of the tables are not shown. We assume that the following integrity constraints hold for the schema (in addition to the described key and foreign key constraints): ($SECT.dep = SECT.class.dep$) and (def $PROF.prof^{-1}.dep.group$).

Example 1 Suppose that only additions and deletions that are consistent (i.e., do not violate the integrity constraints) and primitive (i.e., single tuple) are allowed to the base tables of our example schema and consider a MV V defined using the following underlying query: $\pi^d_{S.number,C.code,P.name,D.name}\sigma_{D.group='ARTS' \text{and} C.number>300 \text{and} P.age>30}(S \bowtie C \bowtie D \bowtie P)$, where π^d is used to denote duplicate preserving projection, the first letters are used to denote the corresponding base tables, and the join conditions are on the respective reference attributes. The auxiliary views that are created by our algorithm are shown in Table 1.

Figure 2. The example database schema

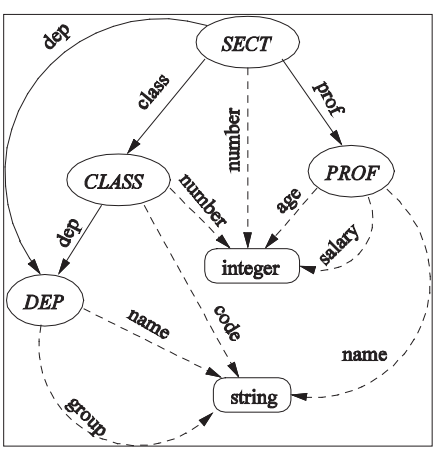


Table 1. Auxiliary views for motivating example

auxiliary view	underlying query
V_P	$\pi^d_{\text{ID, name}}(\sigma_{age>30 \wedge prof^{-1}.dep.group='ARTS'}(P))$
V_D	$\pi^d_{\text{ID, name}}(\sigma_{group='ARTS'}(D))$
V_C	$\pi^d_{\text{ID, code, dep}}(\sigma_{dep.group='ARTS' \wedge number>300}(C))$

In the paper we will show that V can be incrementally refreshed using the following formula:

$$V^{new}=V^{old}+\pi^d_{S.number,\ C.code,\ P.name,\ D.name}(V_P^{new} \bowtie V_D^{new} \bowtie V_C^{new} \bowtie \Delta S),$$

where "+" is used to denote the application of changes (bag version) and the superscripts *old* and *new* are used to denote the content of the table before and after an update, respectively.

We next demonstrate the potential benefit of our algorithm. Consider the four example base tables and suppose they contain the number of tuples shown in Table 2. Suppose that 2 of the departments are in the 'ARTS' group, 10% of the classes pass the predicate "*number*>300", and 80% of the professors pass the predicate "age>30". Also, suppose that 5% of the classes for which "*number*>300" are in a department that is part of the 'ARTS' group and 2% of the profs teach classes in a departments that is part of the 'ARTS' group.

The third column in Table 2 shows the sizes of the auxiliary views if only predicates from the underlying query of the MV are applied to the auxiliary views (i.e., the algorithm from [HZ96] is applied). The forth column shows the sizes of the auxiliary views if the algorithm from [QGMW96] is applied. It extends [HZ96] by removing the auxiliary view for the *SECT* table and storing only classes that are in a department that belongs to the 'ARTS' group. The last column shows the sizes of the auxiliary views when our algorithm is applied. It improves on the previous algorithm by storing only professors who teach courses in a department from the 'ARTS' group.

4. PROPOSED SOLUTION

Consider a MV V with the following underlying query: $\pi^d_{A_1,\ldots,A_n}(\sigma_{E_1 \wedge \ldots \wedge E_e}(T_1 \text{ x } \ldots \text{x } T_t))$ (x is used to denote a cross product), a database schema Σ, and suppose that only consistent primitive insertions and deletions to the tables $\{T_i\}_{i=1}^t$ are allowed. Then the following algorithm produces a set of auxiliary views for V.

Algorithm 1

Step 1. Create an undirected graph with vertices corresponding to the elements of the set $\{T_i\}_{i=1}^t$. For each condition in the set $\{E_i\}_{i=1}^e$, draw an edge between the tables involved in the condition. (In particular, if only a single table is involved in the condition, then draw a loop edge around it.) Next, delete

Table 2. Comparison on the number of tuples for our example

Base Relation	Tuples in Base Relation	Tuples in Auxiliary views ([HZ96])	Tuples in Auxiliary Views ([QGMW96])	Tuples in Auxiliary Views (our algorithm)
SECT	100 000	100 000	0	0
CLASS	50 000	5 000	250	250
DEP	30	2	2	2
PROF	2 000	1600	1600	32
	152 030	61 602	1852	284

all vertices that have no edges connected to them and no attributes in the set $\{A_i\}_{i=1}^{a}$. Then examine the subgraph induced by the edges labeled with equality predicates. If there is a vertex in this subgraph with the properties: (1) all its edges are in the subgraph, (2) removing the vertex will not change the number of connected components in the subgraph, and (3) the vertex's table does not contain attributes in the set $\{A_i\}_{i=1}^{a}$, then remove the vertex and repeat the procedure until possible. Finally, rewrite the underlying query Q of the MV V by deleting the tables that correspond to deleted vertices. (This also involves deleting from Q any predicates on the deleted tables.)

Step 2. For each table T_i in Q (i=1 to t), create an auxiliary view V_i that contains all the tuples of T_i. We will use Q_i to refer to the underlying query for V_i.

Step 3. Consider a table T_i in Q and the corresponds auxiliary view V_i created in the previous step. If the selection condition of Q contains one or more atomic predicates on the table T_i, then add these predicates to the selection condition of Q_i via conjunction. Similarly, add to Q_i a duplicate preserving projection on the attributes of T_i that are projected out in Q union the attributes of T_i that appear in an atomic selection predicate of Q that involves attributes from other tables. The described procedure is applied for i=1 to t.

Step 4. If there is a table T_i in Q that has the property that every table in Q can be reached starting from the table T_i and following reference attributes, then remove V_i from the set of auxiliary views.

Step 5. If there exist a table T_i and a path function $Pf = A_1^{a_1}.\ldots.A_n^{a_n}.A_{n+1}$ such that: (1) Step 4 was not applied to T_i, (2) (`def` $T_i.Pf$) and (3) The table reached by following the path $A_1^{a_1}.\ldots.A_n^{a_n}$ from the table T_i contains an atomic predicate $p(A_{n+1})$ in Q, then add $p(Pf)$, via conjunction, to the selection condition of Q_i.

Going back to Example 1, Step 1 was not applied. Step 2 was applied to create the auxiliary views: V_P, V_D, V_C, and V_S, which initially contain the respective base tables. Step 3 was applied to add the predicate "*age*>30" to V_P, the predicate "*group*='ARTS' " to V_D, and the predicate "number>300" to V_C. The step also applies the projections shown in Table 1. For example, the `ID` attributes are projected for all four tables because they appear in the join conditions. Step 4 was applied to remove the auxiliary view V_S. Finally, Step 5 added the predicate "$prof^{-1}.dep.group$='ARTS'" and "$dep.group$='ARTS' " to V_P and V_C, respectively.

The following theorem addresses the correctness of Algorithm 1.

Theorem 1: Algorithm 1 produces a set of auxiliary views that make V self-maintainable relative to the defined assumptions.

Proof: Step 1 uses the available integrity constraints to rewrite Q into an equivalent query that references fewer tables and therefore does not affect the correctness of the algorithm.

The created auxiliary views in Step 2 make V self-maintainable. In particular, since $\pi^{d}_{A_1,\ldots,A_n}\sigma_{g}(T+\Delta T) = \pi^{d}_{A_1,\ldots,A_n}\sigma_{g}(T)+\pi^{d}_{A_1,\ldots,A_n}\sigma_{g}(\Delta T)$, the changes to each auxiliary view can be calculated by applying the selection condition of its underlying query followed by the duplicate preserving projection operation of its underlying query to the changes of its underlying table. Then the new value of V can be calculated as $Q(V_1,\ldots,V_t)$ (we use $Q(R_1,\ldots,R_t)$ to denote the result of Q when the table T_i is substituted with table R_i for i=1 to t).

We will next examine two cases: when Step 4 was not applied and when it was applied.

Case 1 (*Step 4 was not applied*) We will use $V_{i,r}$ to denote the auxiliary view for V_i after Step r. Note that $V^{new}=Q(T_1^{new},\ldots,T_t^{new})=\pi^{d}\sigma(T_1^{new}\bowtie\ldots\bowtie T_t^{new})$. We will show that $\pi^{d}\sigma(T_1^{new}\bowtie\ldots\bowtie T_t^{new})=\pi^{d}\sigma(V_{1,j}^{new}\bowtie\ldots\bowtie T_{t,j}^{new})$ for j={2,3,5}, which proves that $V^{new}=\pi^{d}\sigma(V_{1,5}^{new}\bowtie\ldots\bowtie T_{t,5}^{new})=Q(V_1^{new},\ldots,V_t^{new})$ and therefore the selected auxiliary views make V self-maintainable.

First, note that $\pi^{d}\sigma(T_1^{new}\bowtie\ldots\bowtie T_t^{new})=\pi^{d}\sigma(V_{1,3}^{new}\bowtie\ldots\bowtie V_{t,3}^{new})$. The reason is that $\pi^{d}\sigma(T_1^{new}\bowtie\ldots\bowtie T_t^{new})=\pi^{d}\sigma(V_{1,2}^{new}\bowtie\ldots\bowtie V_{t,2}^{new})$ (direct consequence of Step 2) and applying Step 3 to $V_{i,2}$ for i = 1 to t does not change the value of the expression $\pi^{d}\sigma(V_{1,2}^{new}\bowtie\ldots\bowtie V_{t,2}^{new})$. In particular, Step 3 first applies to $V_{i,2}$ the single table predicates of Q. This will not change the above expression because $\sigma_E(R_1\bowtie R_2)=\sigma_E(\sigma_E(R_1)\bowtie R_2)$ if E is a predicate only on the attributes of R_1. Next, Step 3 removes from $V_{i,2}$ attributes that do not participate in the join condition and that are not projected in Q. This rule will not affect the expression because $\pi_{\bar{A}}(R_1\bowtie R_2)=\pi_{\bar{A}}(\pi_{\bar{B}}(R_1)\bowtie R_2)$ when $\bar{A}\subseteq\bar{B}$ and $\bar{B}$ are attributes of R_1 that do not participate in the join condition.

Next, consider what happens when step 5 is applied to the auxiliary views of the expression: $\pi^{d}\sigma(V_{1,3}^{new}\bowtie\ldots\bowtie V_{t,3}^{new})$. In particular, this step substitutes auxiliary views with more restrictive auxiliary views that contain only tuples that can join with the other auxiliary views. Therefore, since $R_1\bowtie R_2=\sigma_E(R_1)\bowtie R_2$ when E is a predicate that selects tuples of R_1 that join with R_2, our expression will not change after the application of Step 5 to its auxiliary views.

Case 2 (*Step 4 was applied to table* T_1). Note that

$$\begin{aligned}
&V^{new}=Q(T_1^{new},\ldots,T_t^{new})\\
&=\pi^{d}\sigma((T_1^{old}+\Delta T_1)\bowtie T_2^{new}\bowtie\ldots\bowtie T_t^{new})\\
&=\pi^{d}\sigma(T_1^{old}\bowtie(T_2^{old}+\Delta T_2)\bowtie\ldots\bowtie(T_t^{old}+\Delta T_t))+\pi^{d}\sigma(\Delta T_1^{old}\bowtie T_2^{new}\bowtie\ldots\bowtie T_t^{new})\\
&=V^{old}+\pi^{d}\sigma(T_1^{old}\bowtie\ldots\bowtie T_{n-1}^{old}\bowtie\Delta T_n+\ldots+T_1^{old}\bowtie\Delta T_2\bowtie\ldots\bowtie\Delta T_t)+Q(\Delta T_1, T_2^{new},\ldots,T_t^{new})
\end{aligned}$$

We will next show that the second expression in the above formula empty and therefore $V^{new}=V^{old}+Q(\Delta T_1, T_2^{new},\ldots,T_t^{new})$. Indeed, consider the join of T_1^{old} with an insertion or deletion to the table T_2. The fact that T_2 can be reached from T_1 following reference attributes guarantees that the result of this join will be empty.

It remains to show that $\pi^{d}\sigma(\Delta T_1, T_2^{new},\ldots,T_t^{new})=\pi^{d}\sigma(\Delta T_1, V_2^{new},\ldots, V_t^{new})$, which will prove the theorem. However, this can be proven the same way we proved that applying Steps 2, 3, and 5 to the auxiliary views in Case 1 do not change the value of the expression.

Note that the above theorem only shows that the selected by Algorithm 1 auxiliary views make the input MV V self maintainable, but does not explain how V can be incrementally refreshed. However, when Step 4 was applied, $V^{new}=V^{old}+Q(\Delta T_1, V_2^{new},\ldots,V_t^{new})$ and therefore $\Delta V = Q(\Delta T_1, V_2^{new},\ldots,V_t^{new})$. Of course, before ΔV can be calculated, the auxiliary views need to be refreshed.

When Step 4 was not applied, the formula for calculating ΔV is:

$$V^{new}=Q(V_1^{new}, V_2^{new},\ldots,V_t^{new})=V^{old}+\pi^{d}\sigma(V_1^{old}\bowtie\ldots\bowtie V_{t-1}^{old}\bowtie\Delta V_t+\ldots+\Delta V_1\bowtie\ldots\bowtie\Delta V_t),$$

where the brackets contain 2^n-1 expressions covering the cases where V_i is represented as V_i^{old} and as ΔV_i.

Before describing our algorithm for selecting auxiliary views for a MV with aggregation, we present an example.

Example 2 Consider the MV V with the underlying query:

$$_{D.name}\mathcal{F}_{\text{count(S.ID) as } sect\_count}(D\bowtie C\bowtie S)$$

and suppose that only consistent insertions and deletions are allowed to the underlying tables. We will first rewrite the query as the equivalent query:

$$_{D.name}\mathcal{F}_{\text{count(S.ID) as } sect\_count}(D\bowtie S).$$

Then we will create the auxiliary view $V_D=\pi^{d}_{name,\mathtt{ID}}(D)$. If a section is inserted/deleted, then we will use V_D to find the department's section and then add/subtract 1 to the value of the attribute *sect_count* of the corresponding tuple in V. If such a tuple does not exist in V, then one should be created with *sect_count*=1 (a tuple should exist when deletion is performed). Of course, if the *sect_count* of a tuple in V becomes 0, then the tuple should be deleted from the MV. If a department is inserted or deleted, then only V_D needs to be updated because a new or deleted department can not join with an existing section.

Next, consider a MV V defined with the following underlying query:

$$_{B_1,...,B_b}\mathcal{F}_{agg_1(C_1),\ ...,\ agg_s(C_e)}\ \pi^{d}_{A_1,...,A_a}\ (\sigma_{E_1\wedge...\wedge E_e}\ (T_1 \times ... \times T_t)),$$

where $agg \in \{\texttt{min}, \texttt{max}, \texttt{sum}, \texttt{count}, \texttt{avg}\}$. Suppose the MV is defined over a schema allowing only consistent insertions and deletions to the tables $\{T_i\}_{i=1}^{t}$, where we add the requirement that at most one operation can be performed on each tuple to disallow tuple updates. Then the following algorithm can be used to create the set of auxiliary views for V.

Algorithm 2

Step 1. Suppose that the MV V is defined using the query $Q(T_1,...,T_t)$ and let Q_C by the conjunctive query formed from Q by stripping its grouping and aggregation. Apply Step 1 from Algorithm 1 to rewrite Q_C and then rewrite Q accordingly.

Step 2. Modify Q and add a $\texttt{count}(A)$ aggregation (if one does not already exists) if there exists the aggregation $\texttt{sum}(A)$ or $\texttt{avg}(A)$ for some attribute A and Q does not contain a *min* or *max* aggregation. (This step adopts the mechanism of managing views with aggregation from [MQM97].)

Step 3. If Q contains a *min* or *max* aggregation, then apply Steps 2, 3, and 5 from Algorithm 1 to Q_C to create the set of auxiliary views for V. Otherwise, apply to Q_C Steps 2, 3, 4, and 5 from Algorithm 1 to create the set of auxiliary views for V.

Going back to Example 2, Step 1 was applied to rewrite the query and Step 4 from Algorithm 1 was applied to remove the auxiliary view for the *SECT* table. Step 2 of Algorithm 2 was not applied.

Theorem 2. Algorithm 2 produces a set of auxiliary views that make V self maintainable relative to the defined assumptions.

Proof(Sketch): Note that Step 1 rewrites the original query. Step 2 just adds additional attributes to V. Therefore, we only need to show that the created in Step 3 auxiliary views make V self-maintainable.

First, consider the case when Step 4 from Algorithm 1 was not applied and let us use V^C to denote the MV with underlying query $Q_C(T_1,...,T_t)$. Then Theorem 1 implies that $V^{C\ new} = Q_C(V_1^{new},...,V_t^{new})$. The new value for V can be computed by applying the grouping and aggregation from Q to $Q_C(V_1^{new},...,V_t^{new})$ and therefore the selected set of auxiliary views makes V self-maintainable.

Next, consider the case when Step 4 from Algorithm 1 was applied. Then

$$V^{new} = V^{old} \oplus Q\ (\Delta T_1, V_2^{new},...,V_t^{new}),$$

where $\oplus$ is a new operation that calculates the correct value for the count, sum, and avg attributes. In particular, an addition/deletion of a tuple from $Q(\Delta T_1, V_2^{new},...,V_t^{new})$ causes the value of the count attribute in the matching tuple in V^{old} to be incremented/decremented by 1. Similarly, it causes the $\texttt{sum}$ attribute in this tuple to be incremented/decremented by the value of the attribute on which the summation is performed in the tuple that is added/deleted from $Q\ (\Delta T_1, V_2^{new},...,V_t^{new})$. Note that tuples that have a 0 for the count attribute should be removed from the query result for V^{new}. Finally, the value of an $\texttt{avg}$ attribute is calculated as the result of dividing the value of the $\texttt{sum}$ attribute by the value of the count attribute.

Note that Algorithms 1 and 2 will have to be modified if updates are allowed. In particular, attributes can be classified as *protected* and *exposed* (see [QGMW96]). Protected attributes are projected in the underlying query of the MV, but no predicates are defined on them. Conversely, exposed attributes are the ones on which selection or join predicates are defined. Updating a protected attribute will not affect the two algorithms. However, in the presence of updates on exposed updates Step 4 of Algorithm 1 can not be applied. Similarly, Step 3 of Algorithm 1 cannot be applied to add predicates on exposed attributes. Finally, Steps 5 of Algorithm 1 cannot be applied if the path function *Pf* passes through tables that contain exposed attributes.

5. CONCLUSION

The paper presents novel algorithms for creating auxiliary views in the context of a data warehouse environment. The algorithm for MVs defined over queries without grouping and aggregation creates smaller auxiliary views than existing algorithms by exploring a richer set of integrity constraints. The algorithm for minimizing the size of auxiliary views for MVs defined over queries with aggregation solves a novel problem.

One topic for future research is focusing on the problem of completeness, that is, showing that the two algorithms produce a minimal set of auxiliary views relative to the explored types of integrity constraints.

REFERENCES

[AHRVW98] S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, J. Wiener, Incremental Maintenance for Materialized Views over Semi structured Data, *VLDB*, 1998

[BLT86] J. Blakely, P. Larson, and F. Tompa, Efficiently Updating Materialized Views, *SIGMOD*, 1986

[GJM96] A. Gupta, H. Jagadish, I. Mumick, Data Integragtion using Self-Maintainable Views, *ICDT*, pp. 140-144, 1996

[GM95] A. Gupta and I. Mumick, Maintenance of Materialized Views: Problems, Techniques, and Applications, *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, June, 1995

[H96a] N. Huyn, Efficient View Self-Maintenance, *Stanford University technical report*, http://www-db.stanford.edu/pub/papers/fdvsm.ps, 1996

[H96b] N. Huyn, Efficient View Self-Maintenance, *Proceedings of ACM Workshop on Materialized Views: Techniques and Applications*, 1996

[HZ96] R. Hull and G. Zhou, A Framework for Supporting Data Integration using the Materialized and Virtual Approaches, *SIGMOD*, 1996

[MQM97] I. Mumick, D. Quass, B.Mumick, Maintenance of Data Cubes and Summary Tables in a Data Warehouse, *SIGMOD*, 1997

[QGMW96] D. Quass, A. Gupta, I. Mumick, and J. Widom, Making Views Self-Maintainable for Data Warehousing, *PDIS*, pp. 158-169, 1996

[ZM98] Y. Zhuge, H. Garcia-Molina, Graph Structures Views and their Incremental Maintenance, *ICDE*, 1998

ENDNOTE

[1] Note that in order for ΔT_1 to be a relational table, each tuple in it needs to be tagged as "to be inserted" or "to be deleted" and the relational algebra operations need to be redefined to handle marked tuples - for details see [BLT86].