Reducing the Size of Auxiliary Data Needed to Maintain Materialized Views
by Exploiting Integrity Constraints

by Lubomir Stanchev[*]


## Abstract

A *data warehouse* consists of *materialized views*, which contain derived data from several *data sources*. The advantage of using materialized views is that they contain the results of standard queries and therefore when such queries are posed, the data sources those queries are based on, which are usually costly to access because of their size and remoteness, don't have to be accessed. However in order for the materialized views to contain up-to-date data they have to be updated periodically. Such synchronization of the materialized views with the data sources could be slow if the later have to be queried for a correct update to be done - i.e. if just the data for the changes done to the data sources is insufficient. A way querying the data sources during materialized view update can be avoided is by storing any data from the data sources, which could be relevant to an update of the materialized views, on the data warehouse site. Such auxiliary data is stored in *auxiliary views* and has the characteristic that it makes the data stored at the data warehouse *self-maintainable*, i.e. it can be updated correctly from the log of changes done to the data sources.

In this paper we propose unique algorithms for keeping the size of such auxiliary views relatively small. This is achieved by using the additional information about the integrity constraints that hold on the data sources. More specifically we use an object-oriented model to describe a database schema and a subset of OQL as our query language. The proposed in the paper algorithms produce auxiliary views that make a materialized view *self-maintainable* relative to the three possible operations on a database: addition, deletion and object update.


## 1. Introduction

Generally, a *data warehouse* contains aggregated data derived from a number of data sources and is usually used by *On-Line Analytical Processing* (OLAP) tools and data mining tools for the purpose of decision making (see figure 1).

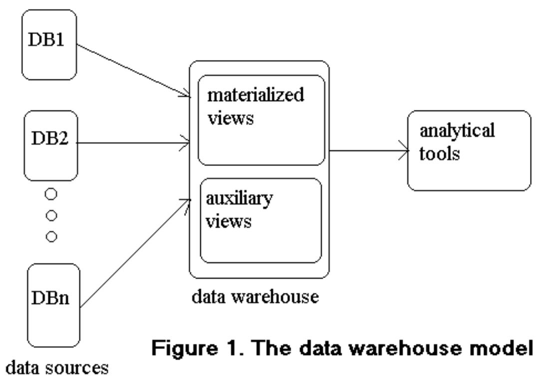


**Figure 1. The data warehouse model**

[*] e-mail: lp2stanc@uwaterloo.ca
web-page: www.math.uwaterloo.ca/~lp2stanc

The *data sources* consist of several databases, usually containing huge amount of data. The day-to-day transactions of a store-chain, for some period of time, is a typical example of the kind of data stored in the data sources. The *materialized views* on the other hand contain summary data, compiled from several data sources. For example the result of standard queries may be cashed onto the materialized views in order to provide for faster response time.

The main challenge in implementing the so presented architecture of a data warehouse is maintaining the materialized views up to date with respect to changes made to the data sources. For example suppose that the materialized view $V$ is calculated from the tables $T_1$ and $T_2$ in the data sources via the following relational algebra query: $V=T_1 \bowtie T_2$, where $\bowtie$ stands for natural join. Then suppose that $T_1$ is modified to $T_1'$. Then $V$ has to be modified to $T_1' \bowtie T_2$, but in general it is impossible to calculate this new value of V, knowing only the change made to the table $T_1$ and the old value for $V$. In other words to be able to calculate the new value of $V$ we have to either query the data sources to get the updated state of the table $T_1$ or have some information about the integrity constraints that hold on the data sources.

The method we propose for updating a materialized view is by storing *auxiliary views* in the data warehouse. To define what data those auxiliary views should contain we need some formal definitions. Suppose that we have a database schema $\Sigma$ and a data query $Q$ over that schema. Let's assume that at time 0 the data sources form the database $D_0$, which is an instance of the database schema $\Sigma$, and that $V_0=Q(D_0)$ is the materialized view stored at the data warehouse. Now at time 1, $D_0$ will change to $D_1$ (i.e. $D_1=D_0+\mu$, where $\mu$ is the set of changes made on $D_1$). Then the set of views $V_a=\{R_1 ,R_2 ,...,R_n\}$ ($n\in\mathbb{N}$) form a set of auxiliary views relative to the schema $\Sigma$, the database $D_0$, the query $Q$ and the set of allowed changes to the database $D_0$ if there exists a query $Q'$ such that $Q(D_0+\mu)=Q'(V_0, V_a, \mu)$, for any $\mu$ from the allowed set of changes. For example if $V_a=D_0$ then we can define $Q'(V_0,V_a,\mu)=Q(V_a+\mu)$ and therefore we will have $Q'(V_0,V_a,\mu)=Q(D_0+\mu)$ which shows that $D_0$ contains the set of auxiliary views needed to update $V_0$. However it turns out that it is possible to find a valid set of auxiliary views $V_a$ much smaller than $D_0$.

In this paper we present a way for calculating small auxiliary views by forming the queries $Q_a=\{Q_{a,1} ,Q_{a,2},...,Q_{a,n}\}$ (called the *auxiliary queries*) from $\Sigma$ and $Q$, such that $R_i=Q_{a,i}(D_0)$, ($i$=1 to $n$) form a valid set of auxiliary views. As well we give an algorithm for recalculating the auxiliary views $V_a$ for $D_0$, in order to make them auxiliary for the updated database instance $D_0+\mu$. Figure 2 summarized the model presented in this introduction. Solid lines are used to show how views/databases are derived. The dotted line is used to show a possible derivation of $V_1$ (the value of the materialized view $V$ at time 1), which is not used because the value of $D_1$ is unavailable or costly to access from the data warehouse. As well the figure shows that the auxiliary views themselves have to updated. Note that throughout this paper we use several variations of the presented in the figure model. Our decision on which model to use depends on the structure of the query $Q$, relative to which the auxiliary views are calculated (for example queries containing aggregation are processed differently than conjunctive queries).

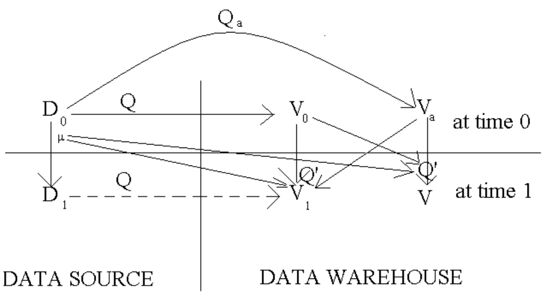


**Figure 2. Shows how a materialized view and the corresponding auxiliary views can be updated**

In what follows in section 2 we will give some definitions related to database model we will be using in this paper. In section 3 a more detailed specification of the problem which we are solving is presented. In section 4 a methodology of how small auxiliary views can be produced is presented. Section 5 contains examples of how the rules from the previous section are applied. In section 6 a short summary of related research is presented and finally in section 7 some possible areas for future research on the problem of maintaining materialized views are covered.

## 2. Definitions regarding the used data model

In this section we explore the object-oriented data model we will use through out the paper. We will use a dialect of description logic (a modification of CLASSIC [2]) for our model. This will allow for simple expression of integrity constraints and for deductive capabilities over them. In the presented model a database schema is presented as a set of subsumptions of the form $C_i<D_i$ where $C_i$ are concrete concepts (classes) and $D_i$ are descriptions. A description D for our purposes can have the form:

| | |
|---|---|
| $D$:== C | (primitive concept) |
| \| **ALL** | (all elements) |
| \| (**all** $Pd\ D$) | (typing constraint) |
| \| (**rc** $Pd$) | (referential constraint) |
| \| (**and** $D_1\ D_2$) | (conjunction) |
| \| (**same** $Pd_1\ Pd_2$) | (equational constraint). |

Note that each primitive concept can have a number of attributes and roles (in description logic set-value attributes are referred to as roles). A path description *Pd* denotes a composition of roles and attributes forming a path. The above set of constructs is only a subset of the CLASSIC description logic but is sufficient for the presented in the paper algorithms (for a full set see [3] and for the extension CLASSIC/FD which adds functional dependency constraints see [4]). A description is semantically interpreted relative to a domain $\Delta$ and an interpretation $I=<\Delta,^{I}>$. The interpretation $I$ assigns a subset of $\Delta$ to each primitive concept, a total function over $\Delta$ to each attribute and a predicate to each role. The interpretation of a description is governed by the following rules:

$\mathbf{ALL} = \{x\in\Delta\}$

$(\mathbf{all}\ Pd\ D) = \{x\in\Delta \mid \forall y\in\Delta, Pd^{I}(x,y) \rightarrow y\in D^{I}\}$

$(\mathbf{rc}\ Pd) = \{x\in\Delta \mid (\exists_1\ y\in\Delta, \text{ s.t. } Pd^{I}(x,y))\}$

$(\mathbf{and}\ D_1\ D_2) = D_1^{I} \cap D_2^{I}$

$(\mathbf{same}\ Pd_1\ Pd_2) = \{x \mid \forall y\in\Delta,\ Pd_1^{I}(x,y) \leftrightarrow Pd_2^{I}(x,y)\}$

Here the interpretation of a path description *Pd* is defined recursively by composing the interpretations of the involved in the path description roles and attributes. As well note that the symbol $\exists_1$ is used to represent there exists exactly 1. A subsumption constraint $C<D$ can be viewed as a logical formula which is true if $C^{I}$ is a subset of $D^{I}$. For a mapping of description logic to first order predicate logic see [3]. An example schema over the so proposed dialect of description logic is:

CLASS < (**all** c_dep DEP)
CLASS < (**all** code STRING)
CLASS < (**all** number INT)
SECT < (**all** s_class CLASS)
SECT < (all s_prof PROF)
SECT < (**all** number INT)
SECT < (**all** s_dep DEP)
SECT < (**same** s_class.c_dep s_dep)
PROF < (**all** name STRING)
PROF < (**all** *teaches* SECT)
PROF < (all age INT)
PROF < (all salary INT)
PROF < (rc *teaches*.s_class.c_dep.group)
PROF < (**same** teaches.s_prof ID)
DEP < (**all** name STRING)
DEP < (**all** group STRING)

The presented example database schema describes the courses offered at a university, where each course has several sections, each section is taught by a professor and each course is taught within a department. Note that we have used italic font to distinguish between attributes and roles (i.e. *teaches* is a role).

As well each class has a system attribute **ID** which is unique for the whole schema (this is the case in most object-oriented database models). The interpretation of this attribute is the identity function. As well note that standard data types (i.e. integers, strings, etc.) are considered to be primitive concepts in out model.

An axiomatization for the presented dialect follows. Note that from now on we will use $C<(\mathbf{ra}\ Pd\ D)$ as a short for $C<(\mathbf{and}\ (all\ Pd\ D)\ (\mathbf{rc}\ Pd))$.

**1.** $C < C$
**2.** if $C_1 < C_2$ and $C_2 < D$ then $C_1 < D$
**3.** if $C < D$ then $C < (\mathbf{all\ ID}\ D)$
**4.** if $C_1 < (\mathbf{all}\ Pd_1\ C_2)$ and $C_2 < (\mathbf{all}\ Pd_2\ D)$ then $C_1 < (\mathbf{all}\ Pd_1 \circ Pd_2\ D)$
**5.** $\mathbf{ALL} < (\mathbf{same}\ Pd\ Pd)$
**6.** if $D < (\mathbf{same}\ Pd_1\ Pd_2)$ then $D < (\mathbf{same}\ Pd_2\ Pd_1)$
**7.** if $D < (\mathbf{same}\ Pd_1\ Pd_2)$ and $D < (\mathbf{same}\ Pd_2\ Pd_3)$ then $D < (\mathbf{same}\ Pd_1\ Pd_3)$
**8.** if $D < (\mathbf{same}\ Pd_1\ Pd_2)$ then $D < (\mathbf{same}\ Pd_1 \circ Pd\ Pd_2 \circ Pd)$ for any path description *Pd*
**9.** if $C_1<(\mathbf{ra}\ Pd\ C_2)$ and $C_2<(\mathbf{same}\ Pd_1\ Pd_2)$ then $C_1<(\mathbf{same}\ Pd \circ Pd_1\ Pd \circ Pd_2)$
**10.** if $C_1<(\mathbf{all}\ Pd\ C_2)$ and $C_2<(\mathbf{same}\ Pd_1\ Pd_2)$ and $C_1<(\mathbf{all}\ Pd \circ Pd_1\ D)$ then $C_1<(\mathbf{all}\ Pd \circ Pd_2\ D)$
**11.** if A is an attribute or the *ID* function then $ANY < (rc\ A)$
**12.** if $C_1 < (\mathbf{ra}\ Pd_1\ C_2)$ and $C_2 < (\mathbf{rc}\ Pd_2)$ then $C_1<(\mathbf{rc}\ Pd_1 \circ Pd_2)$
**13.** if $C_1<(\mathbf{ra}\ Pd\ C_2)$ and $C_2<(\mathbf{same}\ Pd_1\ Pd_2)$ and $C_1<(\mathbf{rc}\ Pd \circ Pd_1\ D)$ then $C_1<(\mathbf{rc}\ Pd \circ Pd_2\ D)$
**14.** If $C<(\mathbf{ra}\ Pd_1\ C_1)$ and $C<(\mathbf{rc}\ Pd_2)$ and $C<(\mathbf{ra}\ Pd_3\ D)$ and $C<(\mathbf{same}\ Pd_2 \circ Pd_4\ Pd_1)$ and $C<(\mathbf{same}\ Pd_2 \circ Pd_5\ Pd_3)$ then for $\forall Pd$ such that $C_1<(\mathbf{ra\ Pd}\ D)$ it will be true that $C<(\mathbf{same}\ Pd_1.Pd\ Pd_3)$
**15.** if $C<(\mathbf{ra}\ Pd_1\ D)$ and $C<(\mathbf{ra}\ Pd_2\ C_2)$ and $C<(\mathbf{same}\ Pd_2\ Pd_3\ Pd_1)$ then $C_2<(\mathbf{ra}\ Pd_3\ D)$
**16.** if $D< (\mathbf{all}\ D_1\ D_2)$ then $D<D_1$ and $D<D_2$

We haven't prove that the above axiomatization is full, but we believe such full axiomatization exists. However the general problem of determining whether the subsumtion $C<D$ is a consequence of a set of subsumtion constraints is not decidable in our case, the reason being that some of the above axioms may be applied infinitely many times for some schemas. For an example of a description logic dialect, which is decidable, see [10].

So far in the paper we have shown some of the theoretical background behind the database model we will be using. In what follows in the next section a precise specification of the problem of creating small auxiliary views will be given. After that we will continue by presenting a methodology for tackling the problem.

## 3. Problem Specification

We are now ready to give a formal definition of the problem we will be solving in this paper. Given an object-oriented database schema $\Sigma$, we will denote by $Models(\Sigma)$ the set of all valid instances of $\Sigma$. As well let *changes* be a predicate that holds true only for the set of changes allowed on an instance of $\Sigma$. Given a query $Q$ over $\Sigma$ we will formally define the predicate that determines if the set of queries $Q_a$ forms a set of auxiliary queries as:

(auxiliary_queries $Q_a(\Sigma, Q, changes)$) $\equiv$ ($\exists$ query Q' ) s.t. ($\forall$D $\in$Models($\Sigma$)) and ($\forall\mu\in$changes($\Sigma$)) [if (D+$\mu$)$\in$Models($\Sigma$) then Q(D+$\mu$) = Q'($\mu$,Q(D),$Q_a$(D))]

The definition of a valid set of auxiliary views $R_i$, $i$=1 to $n$, relative to a database schema $\Sigma$, an instance $D\in models(\Sigma)$, a query $Q$ and a predicate *changes* is:

(auxiliary_views $\{R_i\}_{i=1}^{n}$,$\Sigma$,D,Q,changes) $\equiv$ ($\exists$ query Q' ) s.t. ($\forall\mu\in$changes($\Sigma$)) [if (D+$\mu$)$\in$Models($\Sigma$) then Q(D+$\mu$)=Q'($\mu$,Q(D), $\{R_i\}_{i=1}^{n}$)]

We would like the auxiliary views $R_i = Q_{a,i}\ _i(D_0)$ to be constructed in such a way that *(auxiliary_views* $\{R_i\}_{i=1}^{n}$*,$\Sigma$,D,Q,changes)* to hold. As well we would like the auxiliary views to be recalculated at time 1, in such a way that if $R_{i,new}$ is their value at time 1, *(auxiliary_views* $\{R_{i,new}\}_{i=1}^{n}$*,$\Sigma$,D+$\mu$,Q,changes)* to hold.

The queries we will work with are restricted OQL queries of the form (the main reason for this restriction is to simplify our reasoning):

**select distinct** [AGR] $X_1.A_1$, ..., $X_q.A_r$
**from** $C_1$ *as* $X_1$,...,$C_k$ *as* $X_n$
**where** $E_1$ *and* $E_2$ *and ...* $E_s$
**group by** $X_{m_1}.A_{p_1}, X_{m_2}.A_{p_2},..., X_{m_k}.A_{p_k}$

where $C_1, C_2,..C_k$ are the classes over which the query is executed, $X_1,X_2,..,X_n$ are aliases for those classes, $n>=k$ (i.e. more than one copy of one class can participate in the query), $A_1,A_2,...A_r$ are attributes of the classes with aliases $X_1,..X_q$ ($q<=n$, $r>=q$) and with [] are bracketed optional components, where AGR can be one of SUM,AVG,MIN,MAX and COUNT. As well $E_1,..E_s$ are conditions of the form $X_i.Pd_1$ *={>,>=,<,<=} atomic value* or $X_i.Pd_1 = X_j.Pd_2$, where $Pd_1$ and $Pd_2$ are path descriptions that start at the classes $X_i'$ and $X_j'$ with aliases $X_i$ and $X_j$ respectively (for some i,j$\in$[1..n]) and visit only classes or subclasses in the set $\{C_1,C_2,..C_k\}$. We will also require that $X_i'$<(**rc** $X_i$ $Pd_1$) and $X_j'$<(**rc** $X_j$ $Pd_2$) hold for the database schema over which the query is executed. As well we would like $X_{m_1}.A_{p_1}, X_{m_2}.A_{p_2},..., X_{m_k}.A_{p_k}$ to be a subset of $X_1.A_1, X_2.A_2,..., X_q.A_r$. An example of a query over our example database schema is:

**select distinct** S.number, C.code, P.name, D.name
**from** SECT as S, CLASS as C, DEP as D, PROF as P
**where** S.s_class=C.ID and C.c_dep=D.ID and S.s_prof = P.ID and D.group = "ARTS" and C.number>300 and P.age >=30

This query will return all sections that are taught by a professor, who is 30 years of age or older, and are part of a course, with course number greater than 300, which is taught at a department which is part of the *ARTS* group of departments.

Another important decision for our future discussion is how will the database be stored physically. There are several ways to implement the physical storage of an object oriented database. For the purpose of our discussion we will assume that we store a set of structures (may be organized in some way: for example sorted, hashed, or indexed by some of the attributes or roles) for all objects of a particular class (primitive concept) except for the classes corresponding to standard data type (i.e. integers, strings, etc.). For objects having a reference to the later, we could store the value for the type rather than an object reference. Each structure for an object consists of the object's **ID** together with the values of the different attributes/roles the object has. For example an object of the class PROF in our example database schema may be stored as: {**ID**=1, *age* = 43, *teaches* = {&2,&3}}. This corresponds to an object with **ID**=1 from the class PROF, which has value 43 for the attribute *age* and the value for the attribute *teaches* is the set of objects with IDs 2 and 3 (i.e. the professor teaches two sections that are represented by objects having object IDs 2 and 3).

Before ending this section we have to define the type of operations relative to which we will make the data in the warehouse self-maintainable. Those operations can be expressed by the following self-explainable primitives:

**addobject** *object_id* (*class*,$property_1$={...},...$property_n$={...}}
**delobject** *object_id* (*class*,$property_1$={...},...$property_n$={...}}
**updateobject** *object_id*(*class*,$property_{1,old}$={...},... $property_{n,old}$={...}, $property_{1,new}$= {...},...$property_{n,new}$={...}),

where italic font is used to represent the parameters of the operations. Note that when deleting an object we only need the object's ID to perform the operation. However, having additional data for the object being deleted (or updated) is useful for updating materialized views.

We will assume that the updates are done in groups, called transactions (each transaction contains a set of operations). We will demand of each group of operations combined into a transaction to have the property that when applied to a database instance over a schema with certain integrity constraints, that those integrity constraints continue to hold on the database instance after applying all the operations from the transaction.

To summarize, in this section we have defined auxiliary views, relative to a database schema, the set of queries that are allowed on the schema and the allowed changes on an instance of the database schema. As well we gave a formal definitions of the set of queries our algorithm will support and the type of changes relative to which we will keep the materialized views self-maintainable. Having done this we are ready to present a methodology for creating small auxiliary views.

## 4. The Algorithms

Imagine we have the conjunctive query $Q$:

**select distinct** $X_1.A_1, \ldots, X_q.A_r$
**from** $C_1$ *as* $X_1$,...,$C_k$ *as* $X_n$
**where** $E_1$ *and* $E_2$ *and* ... $E_s$ ,

a database schema $\Sigma$, and the predicate *changes* defined to allow only additions and deletions of objects, but without allowing more than one operation on the same object in one transaction. The algorithm for calculating the auxiliary views relative to this scenario is applying rules 1 to 6 consecutively. The rules are:

**1. Draw an undirected graph with vertices corresponding to the class aliases in the *from* condition and draw edges between vertices corresponding to class aliases involved in the same *where* condition $E_i$ and label them with $E_i$ (in particular if only one alias is involved in the condition draw a loop around the corresponding vertex). Next delete all vertices which have no edges connected to them and whose corresponding class aliases do not appear in the select clause of the query. After that examine the subgraph consisting only of vertices whose proper (non loop) edges are labeled with equality restrictions. If there is a vertex in the so created graph, which doesn't have loops and removing it will not change the number of connected components of the graph and whose corresponding class alias does not appear in the select statement of the query then remove the vertex and repeat the procedure. At the end rewrite the query $Q$ by removing from its *where* condition all class aliases corresponding to deleted vertices.** The proposed procedure gives an algorithm for removing useless classes from the *from* condition of the query (for an illustration of applying this rule see example 2).

**2. Start with one auxiliary view for each class alias $X_i$ that is part of the *from* condition of the query formed by applying rule 1 (we may have more than one auxiliary view for a class)**. Clearly the set of auxiliary views calculated in this way is a valid set of auxiliary views relative to the query $Q$, because it contains all objects from the database $D$, which are relevant to the query.

**3. Apply projection of attributes and roles to each auxiliary view, leaving only those attributes and roles that participate in the query $Q$. Then apply every selection in the**

***where*** **condition of *Q* of the form *res*(*X.Pd*), where *res* is a predicate, *X* a class alias and *Pd* a path description.** This is a generalization of a rule in [7], which covers only SQL queries. Informally if some objects don't pass the *where* condition of the query, then since we don't allow for objects to be modified, they will not influence the value of the materialized view *V* after a transaction is applied and therefore we don't need to store them in the warehouse.

**4. If there is a class *X* with exactly one alias *X'*, such that its key appears in the view *V*, and for every class $Y \in \{X_i\}_{i=1 \text{ to } n}$, there exists a path description *Pd* such that X<ra(*Pd*,*Y*) then we can remove the auxiliary view for the class *X* from the set of auxiliary views. Since we demand that this rule is applied at most once, we will assume w.l.o.g. that this rule, if applied, is applied to the class $C_k$ and its only alias $X_n$.** This rule is a generalization of a similar rule proposed and proven in [12]. Informally the reason the auxiliary view for $X_n$ can be removed from the set of auxiliary views is that for every object of the class $C_k$ we can have at most one entry in the view *V*. Moreover every tuple in *V* represents exactly one object of the class $C_k$. Therefore to add/remove a tuple from *V* we will need an object to be added/removed from $C_k$. Therefore we can join the objects that are added/removed from $C_k$ and pass the *where* conditions of the query *Q* with the updated values for the other auxiliary views (because ∀Y ∃*Pd*, s.t. X < **ra**(*Pd*,*Y*)) to produce the changes made to the materialized view *V*.

**5. Let *Pd* be a path description such that the $C_i$<(ra *Pd* $C_j$) hold on the database schema *Σ* for some classes $C_i$ and $C_j$, and there exist aliases $X_i'$ and $X_j'$, such that neither rule 4, nor this rule was applied on them. Then if *V*<(same $X_i'$.*Pd* $X_j'$) can be deduced to be true, we can reduce the auxiliary view for $X_i'$, by removing all objects *o* from it for which *o.Pd* is not part of the query result (see example 1).** Again this generalizes a rule proposed and proven in [12]. Informally, the reason why the auxiliary view for $X_i'$ can be reduced is that we know that every object in $C_i$ is related (relative to the path description *Pd*) to exactly one object in $C_j$, and only those objects in the auxiliary view for $X_i'$ that are related to objects in $X_i'$ which pass the query condition could influence the value of the materialized view *V*.

**6. If we have more than one auxiliary view for a class, combine all the auxiliary views for that class into one, and eliminate all duplicate tuples (remember that we are examining only simple *select distinct* queries).**

Examples of how these rules can be applied in practice are shown in the next section. The theorem that shows that the above rules produce a valid set of auxiliary views, how those views can be used to update the materialized views *V* and how the auxiliary views themselves can be updated follows.

**Theorem 1.**

Let *Σ* be a database schema, $D_0 \in$Models(*Σ*) be an instance over that schema and μ∈*changes*, where the predicate *changes* allows for addition/deletion of objects, but not more than one operation on the same object in the same transaction. Let as well $D_1=(D_0+\mu)\in$Models(*Σ*) and *Q* be a conjunctive query of the form:

$$\begin{array}{ll} \textit{select distinct} & X_1.A_1, \ldots, X_q.A_r \\ \textit{from} & C_1 \textit{ as } X_1, \ldots, C_k \textit{ as } X_n \\ \textit{where} & E_1 \textit{ and } E_2 \textit{ and } \ldots E_s . \end{array}$$

Let as well $R_i=Q_{a,i}(C_{i,0})$, i=1 to *n*, where $Q_{a,i}$ is the query produced by applying rules 1 to 5 consecutively and $C_{i,0}$ is used to denote the set of all objects in $D_0$ belonging to the class corresponding to the class alias $X_i$. Let $R_{i,new}=Q_{a,i}(R_i+\mu_{C_i}^{+-})$ and

$$V_1 = \begin{cases} Q(\{R_{i,new}\}_{i=1}^{n}) \textit{ if rule 4 is not used} \\ Q(D_0) + Q(\{R_{i,new}\}_{i=1}^{n-1}, \mu_{C_n}^{+-}) \textit{ otherwise} \end{cases}$$

Then

**(i)** $R_{i,new}=Q_{a,i}(C_{i,0}+\mu_{C_i}^{+-})$ for i=1 to n'

**(ii)** $V_1=Q(D_0+\mu)$

**(iii)** (*auxiliary_views* $\{R_i\}_{i=1}^{n'}$,*Σ*,*D*,*Q*,*changes*)

**(iv)** (*auxiliary_views* $\{R_{i,new}\}_{i=1}^{n'}$,*Σ*,*D*+*μ*,*Q*, *changes*),

where *n'*=*n*-1 if rule 4 was applied in calculating $Q_{a,i}$ and *n'*=*n* otherwise.

*Note 1*. The notation *Q*(*set_of_classes*) is used to represent the result of the query *Q* when the classes in the *where* statement of the query *Q* are replaced by the class: *set_of_classes.* As well we will use the notation *Q*(*set_of_objects*) to represent the result of executing the query *Q* over a database containing the set of objects: *set_of_objects*.

*Note 2.* The notation $\mu_{C_i}^{+-}$ is used to represent the set of additions and deletions of objects of the class with alias $X_i$ that occurred in $\mu$. By $\mu_{C_i}^{+-}$ *+set_of_objects* we denote the result of applying the changes $\mu_{C_i}^{+-}$ to the set of objects: *set_of_objects*.

The proof of the theorem follows. In the theorem $R_{i,new}$ are the new values of the auxiliary views and the formula for $V_1$ shows how the materialized view can be updated. Points (iii) and (iv) of the theorem show that the sets $\{R_i\}_{i=1}^{k}$ and $\{R_{i,new}\}_{i=1}^{k}$ are valid sets of auxiliary views at time 0 and 1. Rule 6 doesn't participate in the theorem because it simply says that after we have applied rules 1 to 5 we can eliminate any tuple duplicates, the reason being we examine only *select distinct* queries in this paper. Informally the theorem is true when rule 4 was not applied in calculating $Q_{a,i}$ because the formula for $V_1$ is based on the updated auxiliary views $R_{i,new}$, which have all the necessary data about the class aliases they represent relative to the query Q. As well in the case when rule 4 was applied in calculating $Q_{a,i}$, we will have exactly one update to $V$, for each tuple in $\mu_{C_n}^{+-}$ that passes the conditions in the *where* statement of the query $Q$ and therefore we can join $\mu_{C_n}^{+-}$ with the new values of the auxiliary views to produce $\Delta V=V_1-V_0$.

***<u>Proof of Theorem 1:</u>***

We first prove the following lemma:

***Lema 1*:** Let $Q$ be a conjunctive query of the form shown in theorem 1, but over a single class $C$, and $\mu$ be a set of changes on the objects of $C$, such that $\mu\in$ *changes* (*changes* is described in the theorem 1 definition). Then $Q(C_{objects}+\mu)=Q(C_{objects})+Q(\mu)$, where we use $C_{objects}$ to represent all the objects belonging to class $C$.

*Note 1.* By $Q(C_{objects})+Q(\mu)$ we denote the set of objects that are in the union of $Q(\mu^+)$ and $Q(C_{objects})$, but are not in $Q(\mu^-)$, where $\mu^+$ denotes the set of objects that are added and $\mu^-$ the set of objects that are deleted in the set of changes $\mu$.

*<u>Proof:</u>* The query $Q$ has the form $\pi_{\text{projection}}(\sigma_{\text{selection}}(\text{objects}))$. Since in the equation $Q(C_{objects}+\mu)=Q(C_{objects})+Q(\mu)$ both sides have the same projection of attributes, we can ignore the projections from our analysis, i.e. we can assume w.l.o.g. that $Q$ has the form $\sigma_{\text{selection}}(\text{objects})$. Now $o\in Q(C_{objects}+\mu)$ is equivalent to ($o\notin\mu^-$ and $o\in C_{objects}\cup\mu^+$ and $\sigma_{\text{selection}}(o)=o$) (i.e. o pass the where condition of the query Q). But this is true if and only if it is true that $\sigma_{selection}(o)=o$ and $o=\sigma_{selection}(o)\notin\sigma_{selection}(\mu^-)$, and $o=\sigma_{selection}(o)\in\sigma_{selection}(C_{objects})$ and $o=\sigma_{selection}(o)\in\sigma_{selection}(\mu^+)$ which is equivalent to $o\in Q(C_{objects})+Q(\mu)$. ▨

**(i)** $R_{i,new}= Q_{a,i}(R_i+\mu_{C_i}^{+-})$ = because of lemma 1 $Q_{a,i}(R_i)+Q_{a,i}(\mu_{C_i}^{+-})$ = by the definition of *Ri* $Q_{a,i}(Q_{a,i}(C_{i,0}))+Q_{a,i}(\mu_{C_i}^{+-})$ = because $Qa,i$ is a conjunctive query over a single class $Q_{a,i}(C_{i,0})+Q_{a,i}(\mu_{C_i}^{+-})$ = because of lemma 1 $= Q_{a,i}(C_{i,0}+\mu_{C_i}^{+-})$. ▨

**(ii) case 1) rule 4 is not applied in forming $Q_{a,i}$**

Let $C_{i,1} = C_{i,0} + \mu_{C_i}^{+-}$. Then $V_1 = Q(\{R_{i,new}\}_{i=1}^{n}) =^{(i)} Q(\{Q_{a,i}(C_{i,0}+\mu_{C_i}^{+-})\}_{i=1}^{n})$ = by the definition of Ci,1 $Q(\{Q_{a,i}(C_{i,1})\}_{i=1}^{n})$ and $Q(D_0+\mu)=Q(\{C_{i,0}+\mu_{C_i}^{+-}\}_{i=1}^{n})$ = by the definition of Ci,1 $Q(\{C_{i,1}\}_{i=1}^{n})$. Therefore we have to prove that $Q(\{Q_{a,i}(C_{i,1})\}_{i=1}^{n})= Q(\{C_{i,1}\}_{i=1}^{n})$ which will be true if we can prove that $Q(\{Q_{a,i,j}(C_{i,1})\}_{i=1}^{n}) = Q(\{C_{i,1}\}_{i=1}^{n})$ (*) for $j$=2,3,5 and 6, where $Q_{a,i,j}$ is the query that is created by applying rules 1 to $j$ to the auxiliary view corresponding to $X_i$. Rule 1 only rewrites the query $Q$ to an equivalent query with fewer classes if this is possible and therefore doesn't affect our reasoning, i.e. we can assume w.l.o.g that it is not applied. Let's denote by $R_{i,j}$ the auxiliary view produced after applying rule $j$ to $C_{i,1}$. The equation * holds for $j$=2 because rule 2 sets $R_{i,2}=C_{i,1}$, and therefore $Q_{a,i,2}$ is the identity.

By executing rule 3 to $R_{i,2}$, $i$=1..$n$, we apply all the projections and some of the selections from $Q$ on the objects of $R_{i,2}$, i.e. $Q_{a,i,3}$ contains the same projections and a subset of the selections of $Q$ and therefore * holds for $j$=3. Now observe that the tuples in $Q_{a,i}'(C_{i,1})$ are a subset of $C_{i,1}$ for $i$=1 to $n$, where $Q_{a,i}'$ is the query $Q_{a,i}$, but without the projections (this follows from the definition of $Q_{a,i}$). It follows that $Q(\{Q_{a,i}(C_{i,1})\}_{i=1}^{n})$ is a subset of $Q(\{C_{i,1}\}_{i=1}^{n})$ because $Q$ is a conjunctive query.

To prove that * holds for $j$=5 we need to prove that $Q(\{C_{i,1}\}_{i=1}^{n})$ is a subset of $Q(\{Q_{a,i,5}(C_{i,1})\}_{i=1}^{n})$. Indeed let's assume that $t \in Q(\{C_{i,1}\}_{i=1}^{n}) = Q(\{Q_{a,i,3}(C_{i,1})\}_{i=1}^{n})$ and $t \notin Q(\{Q_{a,i,5}(C_{i,1})\}_{i=1}^{n})$. Then if $t = Q(\{t_i\}_{i=1}^{n})$, where $t_i \in C_{i,1}$, there must exist $g \in [1,n]$, such that $t_g \in Q_{a,g,3}(C_{g,1})$ and $t_g \notin Q_{a,g,5}(C_{g,1})$, i.e. rule 5 was used to delete $t_g$ from $R_{g,5}$. Now if $class(t_g)=C_i$ since rule 5 was applied on $t_g$, it must be true that $V_1$<(same $X_i'.Pd'$ $X_j'$) and $C_i$<(ra Pd $C_j$) for some path description $Pd$, where $X_i'$ is an alias for the class $X_i$ and $X_j'$ is an alias for the class $C_j$. Therefore there exists $h \in [1,n]$ such that $class(t_h)=C_j$, $t_h \in Q_{a,h,3}(C_{h,1})$ and $t_g.Pd=t_h$ and $t_h \in Q_{a,h,3}(C_{h,1})$ and therefore rule 5 could not have been applied on $t_g$ and the path description $Pd$ - contradiction.

**case 2) rule 4 is applied in forming $Q_{a,i}$**

Let $V_0=Q(D_0)$. Then we have to prove that $V_1 = V_0 + Q(\{R_{i,new}\}_{i=1}^{n-1}, \mu_{C_n}^{+-})$ holds in this case. Since rule 4 was applied, as it says in the rule w.l.o.g. we can assume it was applied to the class $C_k$ and its only alias $X_n$. If we apply case 1) to the database $D_0$ without the class $C_k$ we will get that $Q(\{R_{i,new}\}_{i=1}^{n-1}) = Q(\{C_{i,1}\}_{i=1}^{n-1})$. Now note that since for every class $C_i$, $i<n$ there exists a path description $Pd$ such that $C_k$<***ra***$(Pd,C_i)$ it follows that for each tuple of $C_k$ added during the update, exactly one corresponding (relative to the path description $Pd$) tuple will be added from the other classes. As well for each deleted from $C_k$ tuple exactly one corresponding tuple will be deleted from the other classes. As well since the key of the class $C_k$ appears in $V$ there will be a one to one correspondence between the tuples in $C_k$ which pass the query condition and the tuples in V. Therefore $V_1$-$V_0$= $Q(\{C_{i,1}\}_{i=1}^{n-1}, \mu_{C_n}^{+-}) = Q(\{R_{i,new}\}_{i=1}^{n-1}, \mu_{C_n}^{+-})$, which is what we wanted to prove. ▨

**(iii)** We have to prove that (*auxiliary_views* $\{R_i\}_{i=1}^{n'}$,$\Sigma$,$D_0$,$Q$,*changes*), i.e. that *( (∃ query Q' ) s. t. ( ∀μ∈changes(∑)) [if (D+μ)∈Models(Σ) then Q(D+μ)=Q'(μ,Q(D₀),* $\{R_i\}_{i=1}^{n}$ *)].* But this is true because in (ii) we have proven that Q(D+μ)=

$$V_1 = \begin{cases} Q(\{R_{i,new}\}_{i=1}^{n}) \text{ if rule 4 is not used} \\ Q(D_0) + Q(\{R_{i,new}\}_{i=1}^{n-1}, \mu_{C_n}^{+-}) \text{ otherwise} \end{cases}$$

where $R_{i,new}=Q_{a,i}(R_i+\mu_{C_i}^{+-})$, i.e. a query $Q'$ with the required property exists and it is the right side of the formula for $V_1$. ▨

**(iv)** We have to prove that (*auxiliary_views* $\{R_{i,new}\}_{i=1}^{n'}$,$\Sigma$,$D+\mu$,$Q$,*changes*) (***). But since $R_{i,new}=Q_{a,i}(C_{i,0}+\mu_{C_i}^{+-})$, where $C_{i,0}+\mu_{C_i}^{+-}$ is the set of all objects corresponding to the alias $X_i$ in $D_0+\mu$ we can apply (iii) relative to the database schema $D+\mu$ and the set of auxiliary view $R_{i,new}$ to prove that (***) holds. ▨ ■

If the query Q has aggregates, i.e. it is of the form:

*select distinct* [AGR] $X_1.A_1$,..., [AGR] $X_q.A_r$
*from* $C_1$ *as* $X_1$,...,$C_k$ *as* $X_n$
*where* $E_1$ *and* $E_2$ *and ...* $E_s$
*group by* $X_{m_1}.A_{p_1}, X_{m_2}.A_{p_2},..., X_{m_k}.A_{p_k}$.

then our algorithm has to be modified. Theorem 2 summarizes how rules 1 to 6 can be used when we have aggregates.

**Theorem 2.**

Let $\Sigma$ be a database schema, $D \in Models(\Sigma)$ an instance of that schema and $Q$ be a query of the form:

*select distinct* [AGR] $X_1.A_1$,..., [AGR] $X_q.A_r$
*from* $C_1$ *as* $X_1$,...,$C_k$ *as* $X_n$
*where* $E_1$ *and* $E_2$ *and ...* $E_s$
*group by* $X_{m_1}.A_{p_1}, X_{m_2}.A_{p_2},..., X_{m_k}.A_{p_k}$.

Let $\mu$ be a set of changes on $D_0$, such that $\mu \in changes$, where the predicate *changes* allows for addition/deletion of objects, but not more than one operation on the same object. As well let $C_{i,0}$ be the set of all objects in $D_0$ belonging to the class corresponding to the alias $X_i$ of the query $Q$. Also let $Q_c$ be the conjunctive query corresponding to $Q$, i.e. the query $Q$, without the *group by* line and without the aggregations. As well let's define $R_i=Q_{a,i}(C_{i,0})$, for $i$=1 to $n$, $R_{i,new}=Q_{a,i}(R_i+\mu_{C_i}^{+-})$, and define $V_1 = Q_{input}(\{R_{i,new}\}_{i=1}^{n})$ if rule 4 was not applied in calculating $Q_{a,i}$ and

$V_1=Q_{input}(D_0)\oplus Q_{input}(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-})$ otherwise. The definitions of $Q_{a,i}$, $Q_{input}$ and $Q_{result}$ go as follows:

**case 1)** If $Q$ contains *min* or *max* aggregates we define $Q_{a,i}$, $i$=1 to $n'$, as the result of applying rules 1 to 5 consecutively except for rule 4, relative to the database schema $\Sigma$ and the query $Q_c$. In this case $Q_{input}= Q$ and $Q_{result}$ is the identity query.

**case 2)** if Q doesn't contain *min* or *max* aggregates we define $Q_{a,i}$, $i$=1 to $n'$, as the result of applying rules 1 to 5 consecutively, relative to the database schema Σ and the query $Q_c$. As well we define $Q_{input}$ as the query Q, where each *sum* or *avg* aggregate is substituted by a pair of *sum-count* aggregates (for example we substitute the query *select ....avg salary as avg_sal...* with *select .... sum salary as sum_sal, count salary as count_sal ...*). Finally $Q_{result}$ is defined from the equation $Q=Q_{input}\circ Q_{result}$.

Then if $(D_0+\mu)\in Models(\Sigma)$ the following conditions will hold:

**(i)** $R_{i,new}=Q_{a,i}(C_{i,0}+\mu_{C_i}^{+-})$ for $i$=1 to $n'$

**(ii)** $V_1=Q_{input}(D_0+\mu)$ and $Q_{result}(V_1)=Q(D_0+\mu)$

**(iii)** (*auxiliary_views* $\{R_i\}_{i=1}^{n'}$,$\Sigma$,$D$,Q,*changes*)

**(iv)** (*auxiliary_views* $\{R_{i,new}\}_{i=1}^{n'}$,$\Sigma$,$D+\mu$,Q,*changes*),

where $n'=n$-1 if rule 4 is applied in forming $Q_{a,i}$ and $n'=n$ otherwise.

*Note 1:* The operator ⊕ for the expression $V_1=Q_{input}(D_0)\oplus Q_{input}(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-})$ is defined as all the tuples in $Q_{input}(D_0)$ "plus" the corresponding additions from $Q_{input}(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+})$ "minus" the corresponding deletions from $Q_{input}(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{-})$ ($\mu_{C_n}^{+}$ denotes the additions to the class with alias $X_n$ in $\mu$ and $\mu_{C_n}^{-}$ - the deletions). By "plus"/"minus" we mean that when we calculate the result of "adding" to $t_1\in Q_{input}(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+})$ the tuple $t_2\in Q_{input}(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+})$ and then "subtracting" the tuple $t_3\in Q_{input}(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{-})$, such that $t_1$,$t_2$ and $t_3$ have the same values for the non-aggregate attributes the values for the aggregates attributes is calculated as shown in table 1. In the table *value*, *value.c* and *value.s* are the result values when updating a single *count* attribute and a couple of *count* and *sum* attributes respectively (couples of *count-sum* attributes are created by the application of the query $Q_{input}$). As well in table 3 $t_i.c$ and $t_i.s$ ($i$=1,3) are used to represent the respective values of the couple of *count-sum* attributes being processed.

| | |
|---|---|
| single *count* attribute | if $t_1.c+t_2.c-t_3.c>0$, value= $t_1.c+t_2.c-t_3.c$, otherwise don't add a tuple to $V_1$ |
| a couple of *count - sum* attributes | if $t_1.c+t_2.c-t_3.c>0$, value.c=$t_1.c+t_2.c-t_3.c$, value.s=$t_1.s+t_2.s+t_3.s$, otherwise don't add a tuple to $V_1$ |

*Table 1. Shows how the ⊕ operator adds aggregated attribute values*

The proof of theorem 2 follows. The theorem adapts part of the mechanisms for managing aggregate views presented in [11]. For a demonstration of how the theorem is applied in practice see example 3 in the next section. Several explanations about the theorem are in order. First, if we have the aggregates *min* or *max* we apply all rules but 4 relative to $Q_c$, which yields one auxiliary view for every set of objects in $D_0$ belonging to the same class alias. Since those views are auxiliary, relative to the query $Q_c$ and the formula $V_1=V_0+\Delta V$ is not used (rule 4 is not applied), the materialized view together with those auxiliary views will form a self-maintainable set of views relative to the query $Q$.

Now in the case, when we don't have the aggregates *min* or *max* in the query $Q$, we shall keep in the data warehouse the materialized view for $Q_{input}(D)$ rather than for $Q(D)$. The reason for this is that when rule 4 is applied, we have the formula $V_1=V_0+\Delta V$ and we cannot directly update query results containing *sum* and *avg* aggregate attributes. However if we store a pair of *sum-count* aggregated attributes for each *sum* and for each *avg* aggregate attribute, then we can update $V_1$ and the auxiliary views successfully. In the case when we don't have *sum* or *avg* aggregated attributes, $Q_{input}$ is defined as $Q$, i.e. we store the materialized view for $Q(D_1)$. The disadvantage of our approach is that when we want to calculate the result of $Q(D_1)$ from the materialized view we need to calculate the query $Q_{result}(V_1)$. However this can be done very fast because the query $Q_{result}$ can be executed quickly and as shown in table 2.

| $Q_{result}$ value<br>input attribute(s) | value of *count* attribute | value for $Q_{result}$ |
|---|---|---|
| a pair of *sum-count* attributes corresponding to an *avg* attribute | > 0 | value of sum attribute / value of *count* value |
| a pair of *sum-count* attributes corresponding to a *sum* attribute | > 0 | value of *sum* |
| a pair of *sum-count* attributes corresponding to a *sum* or *avg* attribute or a single *count* attribute | =0 | the tuple doesn't appear in $Q_{result}$ |
| other attribute | N/A | value of the attribute |

*Table 2. Shows how $Q_{result}$ works*

It can be easily seen that the condition in the theorem $Q_{input} \circ Q_{result} = Q$ holds. A characteristic of the query $Q_{result}$ is that it scans it's single input table only once and returns the result tuples continuously - i.e. it takes little time for the *first* and *next* tuple of the result to be calculated.

***Proof of Theorem 2:***

Let's define $Q_{agr}$, such that $Q_{input}=Q_c \circ Q_{agr}$, i.e. $Q_{agr}$ is the query with aggregates that can be applied after $Q_c$ is applied to reach the same result as if $Q_{input}$ is applied in the first place.

**case 1)** Applying theorem 1 to the query $Q_c$ we get that *1.i)* $R_{i,new}=Q_{a,i}(C_{i,0}+\mu_{C_i}^{+-})$ and *1.ii)* $Q_c(D_0+\mu)=Q_c(\{R_{i,new}\}_{i=1}^{n})$. But then *1.ii)* is equivalent to $Q_{agr}(Q_c(D_0+\mu))=Q_{agr}(Q_c(\{R_{i,new}\}_{i=1}^{n}))$ or $Q_{input}(D_0+\mu)=Q_{input}(\{R_{i,new}\}_{i=1}^{n})=V_1$ and $Q_{result}(V_1)=V_1=Q_{input}(D_0+\mu)=Q(D_0+\mu)$ which proves ii). Now (*auxiliary_views* $\{R_i\}_{i=1}^{n}$,$\Sigma$,$D$,Q,*changes*) holds because $V_1=Q(\{R_{i,new}\}_{i=1}^{n})$ is a query for calculating $Q(D+\mu)$ based on $\mu$, $Q(D_0)$ and $\{R_i\}_{i=1}^{n}$. Finally to prove that (*auxiliary_views* $\{R_{i,new}\}_{i=1}^{n'}$,$\Sigma$,$D+\mu$,Q,*changes*) holds we can apply the same reasoning as in theorem 1 for proving *(iv)*.

**case 2)** We have two sub-cases

<u>*case 2.1) rule 4 was not applied in forming $Q_{a,i}$*</u>

Applying theorem 1 to the query $Q_c$ we get that *1.i)* $R_{i,new}=Q_{a,i}(C_{i,0}+\mu_{C_i}^{+-})$ and *1.ii)* $Q_c(D_0+\mu)=Q_c(\{R_{i,new}\}_{i=1}^{n})$. But then *1.ii)* is equivalent to $Q_{agr}(Q_c(D_0+\mu))=Q_{agr}(Q_c(\{R_{i,new}\}_{i=1}^{n}))$ or $Q_{input}(D_0+\mu)=Q_{input}(\{R_{i,new}\}_{i=1}^{n})=V_1$ and $Q_{result}(V_1)=Q_{result}(Q_{input}(D_0+\mu))=Q(D_0+\mu)$ which proves ii). Now (*auxiliary_views* $\{R_i\}_{i=1}^{n}$,$\Sigma$,$D$,Q,*changes*) holds because $Q_{result}(Q_{input}(\{R_{i,new}\}_{i=1}^{n}))$ is a query for calculating $Q(D_0+\mu)$ based on $\mu$, $Q(D_0)$ and $\{R_i\}_{i=1}^{n}$. Finally to prove that (*auxiliary_views* $\{R_{i,new}\}_{i=1}^{n'}$,$\Sigma$,$D+\mu$,Q,*changes*) holds we can apply the same reasoning as in theorem 1 for proving *(iv)*.

<u>*case 2.2) rule 4 was applied in forming $Q_{a,i}$*</u>

***Lemma 2:***

$Q_{agr}(Q_c(D_0)+Q_c(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-}))=Q_{agr}(Q_c(D_0))\oplus Q_{agr}(Q_c(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-}))$

<u>*Proof:*</u>

Note that $t \in Q_{agr}(Q_c(D_0)+Q_c(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-}))$, if and only if the value of the aggregate attributes of $t$ were calculated by adding the changes from $Q_c(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-})$ to $Q_c(D_0)$ and then applying $Q_{agr}$. Since $Q_{agr}$ was applied, all the *count* aggregated attributes for $t$ were greater than 0, because otherwise $Q_{agr}$ would not have returned the tuple $t$ as a result. But the last conditions are equivalent to $t \in Q_{agr}(Q_c(D_0)) \oplus Q_{agr}(Q_c(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-}))$ because all $\oplus$ does is to add the aggregate attributes and eliminate the tuples having count aggregates of 0, which proves the lemma. ▨

Applying theorem 1 to the query $Q_c$ we get that *1.i)* $R_{i,new}=Q_{a,i}(C_{i,0}+\mu_{C_i}^{+-})$ and *1.ii)* $Q_c(D_0+\mu)=Q_c(D_0)+Q_c(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-})$. But then if we apply $Q_{agr}$ to both sides of *1.ii* we get $Q_{input}(D_0+\mu) = Q_{agr}(Q_c(D_0+\mu)) = Q_{agr}(Q_c(D_0)+Q_c(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-})) \overset{\text{lemma2}}{=} Q_{agr}(Q_c(D_0))\oplus Q_{agr}(Q_c(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-}))=Q_{input}(D_0)\oplus Q_{input}(\{R_{i,new}\}_{i=1}^{n-1},\mu_{C_n}^{+-})=V_1$. As well $Q(D_0+\mu)=Q_{result}(Q_{input}(D_0+\mu))=Q_{result}(V_1)$, which completes the prove of (ii). Now (*auxiliary_views* $\{R_i\}_{i=1}^n$,$\Sigma$,$D$,Q,*changes*) holds because $Q_{result}(Q_{input}(D_0) \oplus Q_{input}(\{R_{i,new}\}_{i=1}^{n-1}, \mu_{C_n}^{+-}))$ is a query for calculating $Q(D_0+\mu)$ based on $\mu$, $Q(D_0)$ and $\{R_i\}_{i=1}^n$. Finally to prove that (*auxiliary_views* $\{R_{i,new}\}_{i=1}^{n'}$,$\Sigma$,$D+\mu$,Q,*changes*) holds we can apply the same reasoning as in theorem 1 for proving *(iv)*. ■

So far we have only looked at the case where object updates are not allowed. If we allow for object updates, the above algorithms have to be modified. One way this can be done is to avoid applying rules involving a path description that passes through classes which roles or attributes (we will call them properties from here on) are allowed to be modified. We don't claim that this is an optimal approach, but it clearly works.

More specifically we classify the properties that are part of the query $Q$ (see [12]). *Protected* properties are properties that appear in the *select* clause of the QQL query, and therefore are atomic attributes, but don't appear in the *where* clause. *Exposed* properties on the other hand are properties which appear in the *where* clause of the query $Q$. Updates taking place in the same transaction will be considered as a deletion of an object followed by the addition of an object with the same object ID, but different parameters. Updating protected attributes will not influence the above algorithms. The reason being that since none of the properties involved in the *where* condition are updated, the set of auxiliary views produced by the algorithms will be valid. The only modification is that since an object may appear more than once in $\mu$, when applying the formulas $V_1=Q(D_0) + Q(\{R_{i,new}\}_{i=1}^{n-1}, \mu_{C_n}^{+-})$ or $V_1 = Q_{input}(D_0) \oplus Q_{input}(\{R_{i,new}\}_{i=1}^{n-1}, \mu_{C_n}^{+-})$ the objects in the first part of the right expressions may have to be updated more than once.

As far as updating exposed properties goes, rule 4 can not be applied altogether because it relies on the fact that there are no object modifications from any class participating in the query. As well rule 3 can not be used to do selections on conditions containing exposed properties. The reason is that if we allow for properties of the *where* clause of the query to be updated, then we don't know which objects will pass the query condition after a transaction is finished. Similarly rules 5 and 6 can not be applied if the path description *Pd*, that is part of their preconditions, passes through classes which contain exposed properties. The reason is that such exposed property allow for an object in $C_i$ to be related to different object in $C_j$ after the end of a transaction (see the definition of rules 5 and 6).

No formal proofs regarding object updates is presented in this paper because we believe that the last 3 paragraphs give enough justification why the presented in them methodology works. As well there is extensive literature on the problem of object updates when protected and exposed properties are involved (for example see [12]). For an example of how exposed properties can prevent us from applying some of the rules see Example 3.

In this section algorithms for calculating the necessary auxiliary views relative to a schema $\Sigma$, query $Q$ and predicate *changes* were presented. Algorithms for maintaining those auxiliary views and the corresponding materialized view were described as well. A demonstration of how the rules shown in this chapter can be applied in practice follows.

## 5. Examples

In this section we will show several examples of how the methodology presented in the previous section can be applied.

***Example 1***

Suppose we have our example database schema ∑ from section 2, the predicate *changes* defined to include only addition and deletion of objects, but without allowing two or more operations on the same object in one transaction, and our example query from section 3 rewritten as:

**select distinct** S.number, C.code, P.name, D.name
**from** SECT as S, CLASS as C, DEP as D, PROF as P, "ARTS" as G, 300 as N, 30 AS A
**where** S.s_class=C.ID and C.c_dep=D.ID and S.s_prof = P.ID and D.group = G.value and C.number>N.value and P.age >=A.value

What we have done is that we have presented the standard types as classes to comply with the database model we are using. As well we assume that the classes for the standard types have a default attribute *value* containing the value of the type. Let's assume as well that we are given that *number* is a key for the class SECT. Then by applying theorem 1, we have that $R_{PROF}$ $R_{DEP}$ and

$R_{CLASS}$ (see Table 3) is a valid set of auxiliary views relative to ∑, $Q$ and the predicate *changes* at time 0 and that $R^1_{PROF}$, $R^1_{DEP}$, $R^1_{CLASS}$ produce a valid set of auxiliary views at time 1 (we have excluded the auxiliary views for the standard types because they are irrelevant).

| Time<br><br>Auxiliary View | 0 | 1 |
|---|---|---|
| $R_{PROF}$ | select distinct P.Id, P.name<br>from PROF as P<br>where P.age >=30 and P.teaches.s_class.c_dep.group="ARTS" | select distinct P.Id, P.name<br>from ($R_{prof}+\mu^{+-}_{PROF}$) as P<br>where P.age >=30 and P.teaches.s_class.c_dep.group="ARTS" |
| $R_{DEP}$ | select distinct D.*<br>from Department as D<br>where D.group ="ARTS" | select distinct D.*<br>from ($R^1_{DEP}+\mu^{+-}_{DEP}$) as D<br>where D.group ="ARTS" |
| $R_{CLASS}$ | select distinct C.ID, C.code, C.c_dep<br>from CLASS as C<br>where C.c_dep in $R_{DEP}$ and C.number >300 | select distinct C.ID, C.code, C.c_dep<br>from ($R^1_{CLASS}+\mu^{+-}_{CLASS}$) AS C<br>where C.c_dep in $R^1_{DEP}$ and C.number >300 |

*Table 3. Shows the auxiliary tables at time 0 and at time 1 for example 1.*

The query Q' for calculating $V_1$ is:

Q' = $V_0$ +
**select distinct** S.number, C.code, P.name,D.name
**from** $R^1_{,prof}$ AS P, $R^1_{,dep}$ AS D, $R^1_{,class}$ AS C, $\mu^{+-}_{sect}$ AS S
**where** S.s_class=C.ID and C.c_dep=D.ID and S.s_prof = P.ID,

where $V_0$ is the materialized view at time 0. In this example rule 1 wasn't applicable, we applied rule 2 to start with one auxiliary view for each class, then rule 3 was applied to discard the irrelevant, relative to the query $Q$, properties (like the property *salary* of the class *PROF*), and we did all the necessary selections (like *P.age >=30*). We applied rule 4, because we can use the axioms from section 1 to derive that *SECT*<(**ra** s_class *CLASS*), *SECT*<(**ra** *s_class.c_dep* **DEP**) and *SECT*<(**ra** s_prof *PROF*) and as well we know that the attribute *number*, which appears in the view *V* is a key for the class *SECT*. Rule 5 can also be applied. Note that *CLASS* < (**ra** *c_dep DEP*), and *V*<(**same** *CLASS.c_dep DEP)* are given to hold. Therefore we need to store in the auxiliary view for the class *CLASS*, only objects that relate to objects in the auxiliary view for the class *DEP*, i.e. only classes that belong to a department in the *ARTS* group. Similarly, since we can deduce that *PROF* < (**ra** *teaches.s_class.c_dep.g_group G*) and V<(**same** *PROF.teaches.s_class.c_dep.g_group G*) we need only those professors, that are teaching sections belonging to a class, part of a department from the "ARTS" group of departments.

***Example 2***
Now to demonstrate how rule 1 can be used and how theorem 2 can be applied, suppose we have the query:

Q= **select distinct** count(S) as sect_count, D.name
**from** DEP as D, COURSE as C, SECTION as S
**where** D=C.c_dep and C=S.s_class
**group by** D

on the same schema and allowing only protected updates (i.e. updates on the attribute *D.name*). Since we have only the aggregate *count* we have case 2 of theorem 2 and $Q_{input}$=Q, $Q_{result}$=identity. Now using the integrity constraints we know we can apply the algorithm from rule 1 to rewrite the query as:

Q= *select distinct* count(S) as sect_count, D.name
*from* SECT as S, DEP as D,
*where* S.s_dep=D.ID
*group by* D

We have gained the advantage that we have less base classes in the query and therefore we will end up with fewer auxiliary views. The corresponding auxiliary views at time 0 are: $R_{dep}$= *select distinct D.name, D.ID from DEP as D* and at time 1 - $R^1_{dep}$= *select distinct D.name, D.ID from* ($R_{,DEP}+\mu^{+-}_{DEP}$) *as D* (again we don't show the auxiliary views for the classes corresponding to standard types). To calculate $V_1=Q_{input}(D_0)\oplus Q_{input}(R^{`}_{DEP},\mu^{+-}_{SECT})$ we can calculate consecutively $R^{+'}$, $R^{-'}$ and $R'_{temp}$ and R' as shown in table 4. We need the aggregations for $R^{+'}$ and $R^{-'}$ because we have allowed for more than one update on an object in this case.

| | |
|---|---|
| $R^{+'}$ | *Select* D.name, count($S^+$.s_dep) as plus_count<br>*from* $\mu^+_{sect}$ as $S^+$, $R^1_{dep}$ as D<br>*where* $S^+$.s_dep=D<br>*group by* D |
| $R^{-'}$ | *select distinct* D.name, count($S^-$.s_dep) as minus_count<br>*from* $\mu^-_{sect}$ as $S^-$, $R^1_{dep}$ as D<br>*where* $S^-$.s_dep=D<br>*group by* D |
| $R'_{temp}$ | *select distinct* V.name, V.sect_count+P.plus_count-M.minus_count as sect_count |

| | |
|---|---|
| | *from* $R^{+'}$ as P, $R^{-'}$ as M, $V_0$ as V<br>*where* V.name=P.name and V.name=D.name |
| *R'* | *select distinct* name, sect_count<br>*from* $R'_{temp}$<br>where sect_count>0 |

*Table 4. Shows* $V_1=Q_{input}(D_0)\square Q_{input}(R^{`}_{DEP},\mu^{+-}_{SECT})$ *is calculated for example 2*

***Example 3***

In this example we will demonstrate how we can deal with *exposed updates* (i.e. the case where update of exposed properties is allowed). Suppose we have the scenario from example 1, but the property *c_dep* of the class *CLASS* is exposed to updates. Then we cannot use rule 4 and we will have an auxiliary view for the class *SECT*. As well we can not apply the selection *C.c_dep in* $R_{DEP}$ in calculating the auxiliary for $R_{CLASS}$ (because *c_dep* is an exposed property) and the same applies for the constraint on the class *PROF*. In this case we will have the auxiliary views shown in table 5 (compare them to those in table 3).

| Time / Auxiliary view | 0 | 1 |
|---|---|---|
| $R_{PROF}$ | select distinct P.ID, P.name<br>from PROF as P<br>where P.age >=30 | select distinct P.Id, P.name<br>from ($R^1_{PROF}+\mu^{+-}_{PROF}$) as P<br>where P.age >=30 |
| $R_{DEP}$ | select distinct D.*<br>from Department as D<br>where D.group ="ARTS" | select distinct D.*<br>from ($R^1_{DEP}+\mu^{+-}_{DEP}$) as D<br>where D.group ="ARTS" |
| $R_{CLASS}$ | select distinct C.ID, C.code, C.c_dep<br>from CLASS as C<br>where C.number >300 | select distinct C.ID, C.code, C.c_dep<br>from ($R^1_{CLASS}+\mu^{+-}_{CLASS}$) AS C<br>where C.number >300 |
| $R_{SECT}$ | select distinct S.*<br>from SECT as S<br>where S.s_prof in $R_{PROF}$ | select distinct S.*<br>from $R^1_{SECT}+\mu^{+-}_{SECT}$ as S<br>where S.s_prof in $R^1_{PROF}$ |

*Table 5 Shows the auxiliary views for example 3*

Note that we partially compensated the fact that we couldn't apply the above rules by applying rule 5 to the class *SECT*.

This concludes our examples. We didn't give examples of all possible cases that can arise from the algorithms in section 4, but hopefully the presented examples were sufficient to demonstrate how the rules from section 4 can be applied. In what follows in section 6 we will touch on some of the research done in the area of materialized view maintenance and in section 7 we will mention areas of possible further research.

# 6. Related research

The problem of materialized view maintenance is relatively new (most of the works on this topic are written in the past 10 years). An example of a good survey paper on the problem is [5]. As far as the problem of producing self-maintainable views [6] and [8] give good algorithms. Regarding the problem of self-maintainable views when working with object-oriented databases, as was the case in this paper, [13] is a good reference.

A special class of object databases are the semi-structured databases, where the objects the database is composed of form a graph, but there is no database schema. An excellent paper on the problem of self-maintainable views based on semi-structured data is [1].

As far as exploiting integrity constraints in order to create small auxiliary views [12] is an excellent source of reference and it is the base for this paper. However it covers only how key and referential integrity constraints can be used in a relational database model and explores only conjunctive queries. As far as using functional dependencies when working with self-maintainable views goes, [9] is a unique reference. The paper presents algorithms for determining, whether a materialized view is self-maintainable, with respect to a conjunctive query, the view's content and the functional dependencies that hold on the database schema. That is, the paper only gives algorithms for testing whether a particular materialized view is self-maintainable without presenting an algorithm of how the view can be updated efficiently if the self-maintainability test fails.

# 7. Conclusion

In this paper unique algorithms for creating small auxiliary views that solve the problem of materialized view maintenance are presented. The uniqueness of the algorithms comes from the fact that they use information about the integrity constraints that hold on the data to reduce the size of the auxiliary views.

One weakness of this paper is that it doesn't explore the problem of completeness. That is, algorithms are presented and their correctness is proven, but it is not shown if those algorithms could be modified to produce even smaller auxiliary views, for example by further exploring the integrity constraints that can be deduced to hold. In fact we have shown (outside this paper) that the presented algorithm for producing auxiliary views when queries with aggregations are involved can be optimized to produce even smaller auxiliary views, i.e. the presented methodology doesn't calculate the smallest possible auxiliary views in all cases. Those findings, of how the presented algorithms can be modified to produce even smaller auxiliary views, could be the base for a future paper.

Another weakness of the paper is that no cost model is presented - i.e. the extent to which the proposed algorithms reduce the size of the auxiliary views in practice is not shown. One way a cost model can be obtained is by implementing a prototype software system, and experimenting on it to evaluate the effectiveness of the proposed algorithms in producing auxiliary views.

Finally, it is worth mentioning that using auxiliary views that make the data warehouse self-maintainable is only one possibility when solving the problem of materialized view maintenance. Another alternative is to relax the constraint of self-maintainability, but still use auxiliary and allow as well for data to be fetched on demand from the data sources. A characteristic of such a hybrid model is that it also allows integrity constraints to be exploited when deciding what auxiliary data to be stored at the data warehouse site and what data should be fetched on demand from the data sources.


## Acknowledgements:

I would like to express my sincere gratitude to my advisor Prof. Grant Weddell and his colleague Prof. Ken Salem (both form the University of Waterloo) for their valuable input during the paper writing and reviewing.


## About the author:

Lubomir Stanchev has a M.Sc. degree in computer science from the *University of Sofia*, Bulgaria and is currently a Ph.D. student at the *University of Waterloo*. His fields of interest include distributed, object-oriented and semi-structured databases.

## References


[1] S. Abiteboul , J. McHugh , M. Rys , V. Vassalos , J. Wiener, *Incremental Maintenance for Materialized Views over Semistructured Data*, 24th International Conference on Very Large Databases (VLDB), August, 1998.

[2] Alexander Borgida and Peter F. Patel-Schneider, A Semantics and Complete Algorithm for Subsumption in CLASSIC Description Logic. J. of AI Research, 1:277-308, 1994

[3] Alex Borgid, *On the relative expressiveness of description logics and predicate logics*, *Artificial Intelligence* v.82, pp. 353-367, 1996

[4] Alexander Borgida and Grand E. Weddell, *Adding uniqueness constraints to description logics*, *International Conference on Deductive and Object-Oriented Databases*, pp. 85-102, 1997

[5] A. Gupta, and I. S. Mumick*, Maintenance of Materialized Views: Problems, Techniques, and Applications*, Data Engineering Bulletin, 18(2), pp. 3-18, June 1995.

[6] Ashish Gupta, H. V. Jagadish, Inderpal Singh Mumick, *Data Integration using Self-Maintainable Views*. 5th International Conference on Extending Database Technology (IDBT),pp. 140-144,March 1996.

[7] R. Hull and G. Zhou, *A Framework for supporting data integration using the materilized and virtual approaches*, SIGMOD 1996.

[8] N. Huyn, *Exploiting Dependencies to Enhance View Self-Maintainability*, unpublished, http://www-db.stanford.edu/pub/papers/fdvsm.ps, 1996.

[9] N. Huyn, *Efficient View Self-Maintenance*, Proc. of ACM Workshop on Materialized Views: Techniques and Applications, Montreal, Canada, June 1996.

[10], Viatly L. Khizder, David Toman and Grand Weddell, *Reasoning about Duplicate Elimination with Description Logic*, *DOOD* 2000, to be published

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

[12] D. Quass, A. Gupta, I.S. Mumick, and J. Widom. *Making Views Self-Maintainable for Data Warehousing*, Proc. of Fourth Intl. Conf. on Parallel and Distributed Information Systems (PDIS '96), December 1996.

[13] Y. Zhuge , H. Garcia-Molina, *Graph Structured Views and Their Incremental*

*Maintenance*, Proc. of the Fourteenth International Conference on Data Engineering, February, 1998.

ERROR: undefined
OFFENDING COMMAND: --------------B700940112F5BD7D116F836D--

STACK:

0
0
0
0
0
0
0
0
0
0