# On the Content of Materialized Aggregate Views


Stéphane Grumbach
INRIA
Rocquencourt BP 105
78153 Le Chesnay, France
Stephane.Grumbach@inria.fr

Leonardo Tininini*
CNR-IASI
viale Manzoni 30
00185 Roma, Italy
tininini@iasi.rm.cnr.it



## ABSTRACT

We consider the problem of answering queries using *only* materialized views. We first show that if the views subsume the query from the point of view of the information content, then the query can be answered using only the views, but the resulting query might be extremely inefficient. We then focus on aggregate views and queries over a single relation, which are fundamental in many applications such as data warehousing. We show that in this case, it is possible to guarantee that as soon as the views subsume the query, it can be completely rewritten in terms of the views in a simple query language. Our main contribution is the conception of various rewriting algorithms which run in polynomial time, and the proof of their completeness which relies on combinatorial arguments. Finally, we discuss the choice of materializing or not ratio views such as average and percentage, important for the design of materialized views. We show that it has an impact on the information content, which can be used to protect data, as well as on the maintenance of views.


## 1. INTRODUCTION

The manipulation of aggregate data is now a fundamental issue in many areas such as data warehousing [28; 14; 22], OLAP systems [17; 4; 16], etc. In such applications, queries involve aggregation over evolving data of very large size. The use of materialized aggregate views, which can be maintained, might strongly increase the efficiency of query processing.

Beyond the optimization problem, materialized views constitute sometimes the only data available when the initial data cannot be accessed anymore. The practical importance of this problem is increasing with applications involving mobile computing [29; 6], which often requires to deal with the locally available data, as well as statistical databases [23; 25; 26], where the initial data often do not exist per say, or are made unavailable for reasons of privacy [21; 20]. The complexity of answering queries using materialized views depends upon the syntax of the views and the queries as shown in [11; 2; 3].

In this paper we focus on the problem of answering queries using *only* materialized views. This problem has been widely addressed in the literature for various classes of queries and views, such as conjunctive queries [18; 8], aggregate queries [27; 9; 13; 9], etc. Algorithms have been designed to rewrite the initial queries against the database in queries against the views, and the complexity of these algorithms has been investigated.

To the best of our knowledge though, the problem of deciding if views contain enough information to answer a query has not been addressed. In other words, the failure of the previously developed algorithm to rewrite a query, does not imply that the views do not contain the desired information. Deciding if the views allow to answer a query is a fundamental question both positively for answering the query if no other data are available, and negatively, for protecting data with the guarantee that the views do not permit to derive sensitive information.

We characterize the information content of collections of views or queries. We say that a collection of queries, $\mathcal{Q}$, subsumes another one, $\mathcal{Q}'$, if $\mathcal{Q}$ has at most the distinguishing power of $\mathcal{Q}'$, or, more formally, if for each pair of instances $I_1, I_2$, if they are undistinguishable by $\mathcal{Q}$ (same answers for all queries on the two instances), they are undistinguishable by $\mathcal{Q}'$.

Our first result shows that a query can be answered using a collection of views iff the views subsume the query. The resulting query though might be extremely inefficient. We also consider rewritings restricted to fixed query languages, as is usually done. It is easy to see though that in general a restricted rewriting results in incomplete methods, that fail to answer in presence of enough information.

We then focus on a class of queries and views defined by one aggregation over one relation. Although this restriction might seem drastic, these queries are fundamental in many applications in OLAP and data warehousing [12; 17; 15], where data are first extracted according to specified criteria and then aggregated along several "dimensions" to analyze.


*This work was partly supported by the Italian Istituto Nazionale di Statistica - ISTAT

*Iceberg* queries and cubes are also based on a similar assumption [10; 7].

We first consider the case of flat aggregations such as *count*, and iterated *sum* and *multiply*. We propose an algorithm **AggRew1**, for views and queries of this form, which perform a rewriting of the queries in terms of the views in linear time. The main result of the paper is to show that this algorithm is complete, in the sense that it produces a rewriting iff the views subsume the query. We believe that this is the first result of this type.

We then consider the case of ratio aggregations such as *average*, and *percentage*. In this case, we distinguish between two strategies: (i) materialization of flat views only, and (ii) materialization of ratio views. We see that in the first case the previous algorithm can be extended, leading again to a complete method running in linear time. In the second case, we have shown that, even if some manipulations may produce an exponential burst of the computation, a polynomial time algorithm can be obtained. We have also demonstrated its completeness in a weaker sense with respect to a set of elementary operations.

These results provide insight in database schema design. Indeed, the choice of the views to be materialized will determine what queries can be answered and can be used to make data accessible, or on the contrary to protect sensitive data. It should be emphasized that we consider the problem of answering queries using views in a framework independent from the data. It is clear that in some cases a query can be answered from the views over a *given* instance, although there is no general method that allows to answer the query from the views for *all* instances. We do not consider the data dependent case in the present work.

The paper is organized as follows. In the next section we present two practical examples with aggregate views, and analize the possibility of answering queries using the views. In Section 3, we introduce the concept of subsumption between collections of queries. The algorithm for rewriting flat queries from flat views is presented in Section 4. It is shown to be complete and to run in linear time. Finally, in Section 5, we compare flat and ratio materialized views. We present a complete linear-time algorithm for the first case and a polynomial time algorithm for the second.

## 2. MOTIVATIONS

We consider the problem of rewriting queries by using views expressed as simple aggregations over the same relational expression. The following examples, expressed in SQL-syntax, illustrate the technical problem.

EXAMPLE 1. *The data warehouse of a company contains some precomputed views extracted from its sale database. Among them, we consider the main relation $R$ in the database of schema:*

`(Product,Color,State,Date,Qty)`

*which contains an already aggregated version of the database. Consider the following query.*

**select** *Product, Color,* **sum***(Qty)*
**from** *R*
**group by** *Product, Color*

*This query can be easily answered by using only the following view, without accessing the original relation.*

**create view** *V1* **as**
**select** *Product, Color, Date,* **sum***(Qty)* **as** *Q*
**from** *R*
**group by** *Product, Color, Date*

*the rewritten (equivalent) query being:*

**select** *Product, Color,* **sum***(Q)*
**from** *V1*
**group by** *Product, Color*

*It is less obvious however whether the query can be answered using the following views:*

**create view** *V2* **as**
**select** *Product, State, Date,* **sum***(Qty)* **as** *Q*
**from** *R*
**group by** *Product, State, Date*

**create view** *V3* **as**
**select** *Color, State, Date,* **sum***(Qty)* **as** *Q*
**from** *R*
**group by** *Color, State, Date*

*The fact that the two views have a high level of detail and the presence in the views of the attributes required by the query may suggest that such extraction is possible. However, we will prove that it is not the case.* □

In the foregoing example, $Q$ has been obtained by "removing" the attribute Date from $V1$. Such an operation is usually called *summarization*, and also, in the OLAP terminology, *roll-up* [26]. In the following we will generally refer to it by the former term and say for instance that $Q$ can be obtained by summarizing the attribute Date of $V1$. We next consider an example of percentage query rewriting.

EXAMPLE 2. *A mobile phone company maintains a data warehouse containing the details on the traffic among its customers and particularly a table (relation)* `InterCalls` *on the schema*

`(Call-id,Cust,Day,Source,Dest,Plan,Dur)`

*where Call-id is the identifier of the call, Cust is the calling customer, Day is the day of the call, Source is the antenna to which the customer is connected when the call starts, Dest, the antenna to which the called customer is connected when the call starts, Plan is the calling plan of the customer, and Dur is the duration of the call.*

*Some aggregations are applied on this table, to compare the use of the antennas, guarantee an optimal workload, and propose new calling plans, according to the previous aspects. In particular the following view, has been precomputed:*

```
create view V1 as
select   Day, Source, Dest, Plan, N/D
from     ((select    Day, Source, Dest, Plan,
                         count(*) as N
           from      Intercalls
           group by  Day, Source, Dest, Plan)
           natural join
           (select   Day, Source, count(*) as D
           from      Intercalls
           group by  Day, Source))
```

*V1 represents the percentage of phone calls by day, calling plan, source and destination antennas, w.r.t. the total number of calls for the same day and source antenna. We suppose that further studies are required and that the following queries are posed to the DW system.*

*The problem we address is whether the queries can be answered without accessing the table* `InterCalls`.

*Query Q1:*

```
select   Day, Source, Plan, N/D
from     ((select    Day, Source, Plan,
                         count(*) as N
           from      Intercalls
           group by  Day, Source, Plan)
           natural join
           (select   Day, Source, count(*) as D
           from      Intercalls
           group by  Day, Source))
```

*Query Q2:*

```
select   Day, Source, Dest, Plan, N/D
from     ((select    Day, Source, Dest, Plan,
                         count(*) as N
           from      Intercalls
           group by  Day, Source, Dest, Plan)
           natural join
           (select   Day, count(*) as D
           from      Intercalls
           group by  Day))
```

*Query Q3:*

```
select   Day, Source, Dest, Plan, N/D
from     ((select    Day, Source, Dest, Plan,
                         count(*) as N
           from      Intercalls
           group by  Day, Source, Dest, Plan)
           natural join
           (select   Day, Source, Plan,
                         count(*) as D
           from      Intercalls
           group by  Day, Source, Plan))
```

*We show in particular that the queries $Q1$ and $Q3$ can be computed by using the view $V1$ and propose an algorithm to rewrite them, while we show that the information content of $V1$ is not sufficient by itself to answer the query $Q2$ and hence no rewriting can be obtained. Finally, let us consider a new view $V2$:*

```
create view V2 as
select   Day, Source, Dur, N/D
from     ((select    Day, Source, Dur,
                         count(*) as N
           from      Intercalls
           group by  Day, Source, Dur)
           natural join
           (select   Day, count(*) as D
           from      Intercalls
           group by  Day))
```

*$V2$ seems to have no influence w.r.t. the answerability of the queries above. However, we show that $V1$ and $V2$ together allow to compute the query $Q2$.* □

# 3. THE INFORMATION CONTENT

In this section, we formally consider the following question: What information is contained in a view? and the related problem: Given a collection of views, can a query be answered directly using the views? We address the question from the point of view of the information content. Note that we consider the problem independently of the data, that is we are interested in queries which can be answered directly using the views for *all* instances. For that purpose, we will introduce a concept of *subsumption* between collections of queries.

We first briefly recall the concepts of databases and queries. A *(database) schema* $s$ is a finite set of relation symbols together with their arities. We consider as universe the set of rational numbers $\mathbb{Q}$ (in practice, an additional non numerical universe can be assumed too). An *instance* $I$ over $s$ is an interpretation of the relation symbols of $s$ by finite relations of the corresponding arity over $\mathbb{Q}$. A *query* (or *view*) over $s$ is a total recursive mapping from instances over $s$ to relations of some fixed arity. We do not make any assumption of genericity on the queries [5], since we consider for simplicity only a numerical domain. Clearly the following results can be extended to a framework with a non-numerical domain and queries generic with respect to the non-numerical universe. We will denote both the output relation, and the mapping with the same symbol.

Let $\mathcal{Q} = (Q_1, \cdots, Q_\ell)$ be a collection of queries over some schema $s$, and let $I$ be an instance over $s$. We denote by $\mathcal{Q}(I)$ the collection of relations $(Q_1(I), \cdots, Q_\ell(I))$. We can now define the *subsumption* between two collections of queries.

Definition 1. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be two collections of queries over some schema $s$. We say that $\mathcal{Q}$ subsumes $\mathcal{Q}'$, denoted $\mathcal{Q} \models \mathcal{Q}'$, iff for each pair, $I_1, I_2$, of instances over $s$, $\mathcal{Q}(I_1) = \mathcal{Q}(I_2)$ implies $\mathcal{Q}'(I_1) = \mathcal{Q}'(I_2)$.*

The relation of subsumption defines a semilattice over collections of queries. Let $\mathbf{Q}_s$ be the set of all queries over schema $s$, and $\mathcal{P}(\mathbf{Q}_s)$, the set of all subsets of $\mathbf{Q}_s$. The subsumption relation, defines a preorder over $\mathcal{P}(\mathbf{Q}_s)$. We introduce the corresponding equivalence relation $\approx$ defined by $\models \wedge \dashv$.

We consider the structure $(\mathcal{P}(\mathbf{Q}_s)/\approx, \dashv, \cup)$, defined as follows. Consider two elements $S$, and $S'$ of $\mathcal{P}(\mathbf{Q}_s)$, and let us denote by $\bar{S}$, and $\bar{S}'$ the corresponding elements in $\mathcal{P}(\mathbf{Q}_s)/\approx$. Then $\bar{S} \models \bar{S}'$ iff $S \models S'$, and $\bar{S} \cup \bar{S}' = \overline{S \cup S'}$. It is easy to see that the union operation defines the *least upper bound* of two sets of queries for the subsumed-by relation. The greatest element is $\mathbf{Q}_s$, while $\emptyset$ is the smallest one.

PROPOSITION 1. *$(\mathcal{P}(\mathbf{Q}_s)/\approx, \dashv, \cup)$ is a join semilattice.*

**Proof**: Consider two elements $S$, and $S'$ of $\mathcal{P}(\mathbf{Q}_s)$. Clearly, $S \cup S' \models S$, and $S \cup S' \models S'$. Let $S''$ be such that $S'' \models S$, and $S'' \models S'$. We want to prove that $S'' \models S \cup S'$. That is for all instances $I_1, I_2$ over $s$, $S''(I_1) = S''(I_2) \Rightarrow (S \cup S')(I_1) = (S \cup S')(I_2)$.

Assume that $I_1, I_2$ are such that $S''(I_1) = S''(I_2)$. It follows that $S(I_1) = S(I_2)$, and $S'(I_1) = S'(I_2)$. Therefore for each query $q$ in $\bar{S}$, and each query $q$ in $\bar{S}'$, $q(I_1) = q(I_2)$. Since $\overline{S \cup S'} = \bar{S} \cup \bar{S}'$, it follows that for all queries $q$ in $\overline{S \cup S'}$, $q(I_1) = q(I_2)$, and therefore, $(S \cup S')(I_1) = (S \cup S')(I_2)$. $\square$

Proposition 1 is a fundamental tool for the proofs of completeness of the rewriting algorithms, as it greatly reduces the number of views to be considered. It is used in particular in the proof of the lemmas for Theorem 4.

We now consider the related view problem. Given a collection of views $\mathcal{V}$ over some schema $s$, and an instance $I$ over $s$, we denote by $I^{\mathcal{V}}$ the extension of $I$ to the views in $\mathcal{V}$, with the views interpreted by their results on $I$. Two queries $q$ and $q'$ over relation symbols in $s \cup \mathcal{V}$ are *equivalent modulo* $\mathcal{V}$ [9] if for any instance $I$ over $s$, $q(I^{\mathcal{V}}) = q'(I^{\mathcal{V}})$.

DEFINITION 2. *Given a collection of views $\mathcal{V}$ and a query $q$ both over schema $s$, a query $q'$ over $\mathcal{V}$ is a rewriting of $q$ using $\mathcal{V}$ if $q$ and $q'$ are equivalent modulo $\mathcal{V}$.*

Note that the rewriting is at this stage not related to any target query language.

We can now define the notion of answering queries using views. Let $q$ be a query, and $\mathcal{V}$ a collection of views over some schema $s$. We say that $q$ can be *answered directly using* $\mathcal{V}$ if there is a rewriting of $q$ using $\mathcal{V}$.

The next theorem relates the problem of rewriting using views to the information content of the views.

THEOREM 2. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be two collections of queries over some schema $s$. Then (i) for all $q \in \mathcal{Q}'$, $q$ can be answered directly using $\mathcal{Q}$, iff (ii) $\mathcal{Q} \models \mathcal{Q}'$.*

**Proof**: (sketch) $(i) \Rightarrow (ii)$: if $q \in \mathcal{Q}'$ can be answered directly using $\mathcal{Q}$, then there is a rewriting $q'$ over $\mathcal{Q}$ which is equivalent to $q$ modulo $\mathcal{Q}$. Consider two instances $I_1, I_2$ over $s$. If $\mathcal{Q}(I_1) = \mathcal{Q}(I_2)$, then $q'(I_1) = q'(I_2)$, and hence $q(I_1) = q(I_2)$. If this holds for all $q \in \mathcal{Q}'$, it follows that $\mathcal{Q} \models \mathcal{Q}'$.

$(ii) \Rightarrow (i)$: Conversely, if $\mathcal{Q} \models \mathcal{Q}'$, let $q \in \mathcal{Q}'$. We define a query $q'$ over $\mathcal{Q}$ which for each instance $I$, maps $\mathcal{Q}(I)$ to $q(I)$ as follows. It enumerates all instances $J$ over $s$, and checks if $\mathcal{Q}(I) = \mathcal{Q}(J)$. For such an instance $J$, it computes $q(J)$. $\square$

The previous theorem shows that if there is enough information in the views, there is a rewriting. Consider two collections of queries $\mathcal{Q}$, and $\mathcal{Q}'$. The fact that $\mathcal{Q}$ does not subsume $\mathcal{Q}'$, implies that there are instances $I_1, I_2$ which can be distinguished by $\mathcal{Q}'$, but are undistinguishable by $\mathcal{Q}$. It follows that (at least some) queries in $\mathcal{Q}'$ cannot be answered directly using $\mathcal{Q}$, since $\mathcal{Q}'(I_1)$ and $\mathcal{Q}'(I_2)$ are distinct, while $\mathcal{Q}(I_1)$ and $\mathcal{Q}(I_2)$ are identical. On the other hand, if $\mathcal{Q}$ subsumes $\mathcal{Q}'$, $\mathcal{Q}$ has a distinguishing power at least equal to that of $\mathcal{Q}'$, and it suffices to answer the query.

A *rewriting algorithm* is an algorithm which, given a collection of views $\mathcal{V}$ and a query $q$ over schema $s$, produces a rewriting $q'$ of $q$ using $\mathcal{V}$. For a given algorithm, we will write $\mathcal{V} \vdash q$ whenever there is a rewriting of $q$ using $\mathcal{V}$. A rewriting algorithm is *sound* if for each collection of views $\mathcal{V}$ and each query $q$, $\mathcal{V} \vdash q \Rightarrow \mathcal{V} \models q$. It is *complete* if for each collection of views $\mathcal{V}$ and each query $q$, $\mathcal{V} \models q \Rightarrow \mathcal{V} \vdash q$. (Note that this notion of completeness should be distinguished from the fact that the rewriting produces a query involving only the views and no relations of the initial schema as in [18].)

Theorem 2 is only of theoretical interest, and in practice, it is desirable that the rewriting produces queries of a restricted form. The objective is that (i) the rewritten query can be obtained efficiently, and (ii) it can be evaluated efficiently as well. We consider these aspects in the sequel, by studying the complexity of *rewriting algorithms*, restricted to queries rewritten in fixed query languages.

Note that in general if the output query language is restricted, there might be no effective method producing a complete rewriting. For instance, for first-order queries and views, there is no effective rewriting since the equivalence of queries is undecidable. More generally, the decidability of the problems of containment and equivalence of queries is a prerequisite to the existence of an effective complete rewriting method. In the next section, we present examples of query languages for which there are complete rewriting algorithms.

# 4. COMPLETE REWRITING

We now turn to a restricted class of queries, that include the queries of Examples 1 and 2, and can be expressed with the following aggregate operators [13], defined over multisets which *count, sum*, and *multiply* the elements. The operators sum and multiply are defined over the rationals. Let $\psi$ be a relational expression (e.g. a relation). We assume that variables correspond to attributes of the relational expression independently of their written position. The correspondence is made explicit only when necessary.

- **count** $C^{\bar{z}}\psi(\bar{x}, \bar{z})$ is a partial function mapping a tuple $\bar{a}$ such that $\exists \bar{z}\psi(\bar{a}, \bar{z})$, to the cardinality $c_a$ of the set $\{\bar{z} | \psi(\bar{a}, \bar{z})\}$.
- **sum** $\Sigma_y^{\bar{z}}\psi(y, \bar{x}, \bar{z})$ is a partial function mapping a tuple $\bar{a}$ such that $\exists y\bar{z}\psi(y, \bar{a}, \bar{z})$, to the sum $s_a$ of the elements of the multiset[1] $\{\!\{y | R(y, \bar{a}, \bar{z})\}\!\}$.

[1]The multiplicity is related to the number of $\bar{z}$ values. More formally, $s_y$ is the sum of the elements of the multiset $\{\!\{y \times C^{\bar{z}}\psi(y, \bar{a}, \bar{z}) | \exists \bar{z}\psi(y, \bar{a}, \bar{z})\}\!\}$, and similarly for multiply.

- **multiply** $\Pi_y^{\bar{z}}\psi(y,\bar{x},\bar{z})$ is a partial function mapping a tuple $\bar{a}$ such that $\exists y\bar{z}\psi(y,\bar{a},\bar{z})$, to the product $p_a$ of the elements of the multiset[1] $\{\!\{y|\psi(y,\bar{a},\bar{z})\}\!\}$.

We introduce the following abbreviations of operators which can be obtained as fractions of the initial ones, *percentage*, $Perc$, and *average*, $Avg$, obtained as a fraction of two counts and/or sums.

The variable $y$ is called the *aggregated variable*, $\bar{x}$ the *argument* of the function, and $\bar{z}$ the *multiset parameter*. The expressions $C^{\bar{z}}\psi(\bar{x},\bar{z})(\bar{x})$, $\Sigma_y^{\bar{z}}\psi(y,\bar{x},\bar{z})(\bar{x})$, and $\Pi_y^{\bar{z}}\psi(y,\bar{x},\bar{z})(\bar{x})$ are (aggregate) terms. The free variables of an aggregate term are the variables of $\psi$ which are neither aggregated nor used as multiset parameter. The *arguments* of the function correspond to the free variables of the aggregate term. For readability, we use the overlined notation $\overline{C^{\bar{z}}\psi(\bar{x},\bar{z})}(\bar{x})$ to distinguish between the variables of $\psi$ and the arguments of the function.

*Terms* and *relational expressions* are mutually recursively defined as follows. Variables, constants, and aggregate terms (see above) are terms. Moreover, if $t, t'$ are terms, then $t \times t'$, $t^{t'}$, and $\frac{t}{t'}$ are terms. A relation from the schema is a relational expression. Moreover, if $x$ is a variable, and $t$ an aggregate term, then $x = t$ is a relational expression. Relational expressions are closed under the application of aggregate operators.

The previous definition allows to express complex functions with nested aggregation and arithmetic operations. In the sequel though, we consider only *queries* of a very simple form, expressible by one aggregation over a single relation $R$, while more complex functions are necessary for intermediate steps of the rewriting algorithms. We have seen that such queries are relevant for a wide variety of practical situations (e.g. Data Warehousing, Data Cubes, Statistical DBs, etc.) and have been extensively studied [19; 12; 17; 1; 10].

We introduce a convenient shorthand notation for aggregate formulae. We denote by $Y$, $A$, $B$, $D$, $N$, respectively the set containing the aggregated variable $y$, and the sets containing the variables in $\bar{x}^A$, $\bar{x}^B$, $\bar{x}^D$, $\bar{x}^N$, all corresponding implicitly to the attributes $V$ of $R$.

$$C(A) \text{ for } \overline{C^{\bar{x}^N}(R)}(\bar{x}^A) \qquad (A \cup N = V)$$

$$S_Y(A) \text{ for } \overline{\Sigma_y^{\bar{x}^N}(R)}(\bar{x}^A) \qquad (A \cup Y \cup N = V)$$

$$M_Y(A) \text{ for } \overline{\Pi_y^{\bar{x}^N}(R)}(\bar{x}^A) \qquad (A \cup Y \cup N = V)$$

$$Perc_{A,B} = \frac{C(A,B)}{C(A)} \text{ for } \frac{\overline{C^{\bar{x}^N}(R)}(\bar{x}^A,\bar{x}^B)}{\overline{C^{\bar{x}^D}(R)}(\bar{x}^A)} \qquad (N \cup A \cup B = D \cup A = V)$$

$$Avg_Y(A) = \frac{S_Y(A)}{C(A)} \text{ for } \frac{\overline{\Sigma_y^{\bar{x}^N}(R)}(\bar{x}^A)}{\overline{C^{y\bar{x}^N}R}(\bar{x}^A)} \qquad (A \cup Y \cup N = V)$$

We also use this notation to manipulate functions.

$$\overline{\Sigma^N(F(N,A))}(A) \text{ for } \overline{\Sigma_s^{\bar{x}^N}(s = F(\bar{x}^N,\bar{x}^A))}(\bar{x}^A)$$

$$\overline{\Sigma^{YN}(Y * F(Y,N,A))}(A) \text{ for } \overline{\Sigma_s^{\bar{y}\bar{x}^N}(s = y * F(y,\bar{x}^N,\bar{x}^A))}(\bar{x}^A)$$

In the sequel, we denote queries and views by the corresponding aggregate function, e.g. we say that the query $S_Y(A)$ can be answered by using the view $C(Y,A,B)$.

The query and views of Example 1 are defined by

Q1: $S_{Qty}(Product, Color)$
V1: $S_{Qty}(Product, Color, Date)$
V2: $S_{Qty}(Product, State, Date)$
V3: $S_{Qty}(Color, State, Date)$

while the queries and views of Example 2 are defined by:

V1: $\frac{C(Day,Source,Dest,Plan)}{C(Day,Source)}$
Q1: $\frac{C(Day,Source,Plan)}{C(Day,Source)}$
Q2: $\frac{C(Day,Source,Dest,Plan)}{C(Day)}$
Q3: $\frac{C(Day,Source,Dest,Plan)}{C(Day,Source,Plan)}$
V2: $\frac{C(Day,Source,Dur)}{C(Day)}$

We can now present a complete algorithm, **AggRew1**, to rewrite queries $F(X)$ of the form $C(X)$, $S_Y(X)$ or $M_Y(X)$, by using views $f_i(W_i)$ of the same type.

**The algorithm AggRew1**

**Input:** a query $F(X)$ to rewrite ($F = C$, $S_Y$, $M_Y$) and a collection of views $f_i(W_i)$ ($f_i = C$, $S_{U_i}$, $M_{U_i}$)
**Output:** a rewritten query $F'(X)$ using only the views or a failure message
**switch**($F$)
**case** $C$:
  **for each** count view $C(W_i)$
    **if** $X \subseteq W_i$
    **then return** $F'(X) = \overline{\Sigma^{W_i - X}(C(W_i))}(X)$
**case** $S_Y$:
  **for each** count view $C(W_i)$
    **if** $(X \cup Y) \subseteq W_i$
    **then return** $F'(X) = \overline{\Sigma^{W_i - X}(Y * C(W_i))}(X)$
  **for each** sum view $S_{U_i}(W_i)$
    **if** $Y = U_i$ and $X \subseteq W_i$
    **then return** $F'(X) = \overline{\Sigma^{W_i - X}(S_{U_i}(W_i))}(X)$
**case** $M_Y$:
  **for each** count view $C(W_i)$
    **if** $(X \cup Y) \subseteq W_i$
    **then return** $F'(X) = \overline{\Pi^{W_i - X}(Y^{C(W_i)})}(X)$
  **for each** multiply view $M_{U_i}(W_i)$
    **if** $Y = U_i$ and $X \subseteq W_i$
    **then return** $F'(X) = \overline{\Pi^{W_i - X}(M_{U_i}(W_i))}(X)$
**endswitch**
/*all attempts to rewrite have failed*/
**return** a failure message

It is easily shown that **AggRew1** correctly computes the query of Example 1 by using the view $V1$ and that it does not compute any rewriting using (only) the views $V2$ and $V3$.

The complexity of the algorithm **AggRew1** is rather low as shown by the next result.

PROPOSITION 3. *The algorithm* **AggRew1** *terminates in time linear in the number of views and their arities.*

**Proof**: Each view function is examined at most once by the algorithm, and the variables of each function are scanned once to check set containment. □

The main result of this section is the proof that the algorithm **AggRew1** is complete.

THEOREM 4. *The algorithm* **AggRew1** *is complete.*

The proof of Theorem 4 relies on the following lemmas. The first lemma is devoted to the rewriting of count queries using count, sum and multiply views. It shows that sum and multiply views are useless and that summarization is the only operation that allows to obtain a count query from a collection of count views. The second and third lemmas show that only summarization and a particular from of rewriting based on count views can be used to rewrite sum and multiply queries.

LEMMA 5. *A collection of count, sum and multiply views $f_i(W_i)$ subsumes a count query $C(X)$ iff at least one of the $f_i$ is of the form $C(X, Z')$ ($Z'$ possibly empty).*

LEMMA 6. *A collection of count, sum and multiply views $f_i(W_i)$ subsumes a sum query $S_Y(X)$ iff at least one of the $f_i$ is of the form $C(Y, X, Z')$ or of the form $S_Y(X, Z')$ ($Z'$ possibly empty).*

LEMMA 7. *A collection of count, sum and multiply views $f_i(W_i)$ subsumes a multiply query $M_Y(X)$ iff at least one of the $f_i$ is of the form $C(Y, X, Z')$ or of the form $M_Y(X, Z')$ ($Z'$ possibly empty).*

The proof of these lemmas involve combinatorial techniques. The main difficulty relies in the construction of two instances such that the query returns distinct values on the two instances, while all views that are not of the form described in the lemma return identical values on both instances, thus showing that they do not subsume the query. We give a sketch of the proof of Lemma 6, which is the most involved.

**Proof of Lemma 6**: (sketch) The $\Leftarrow$ part is easy. For the $\Rightarrow$ part it is shown that any view (and hence, by Proposition 1 any collection of views) that does not satisfy the conditions of the lemma is subsumed by the following collection of views: $C(X_{-i}, Y, Z)$, $C(X, Z)$, $S_{X_i}(X_{-i}, Y, Z)$, $S_Y(X_{-i}, Z)$, $S_{Z_j}(X, Y, Z_{-j})$, $M_{X_i}(X_{-i}, Y, Z)$, $M_Y(X_{-i}, Z)$, $M_{Z_j}(X, Y, Z_{-j})$ (where $i = 1, \ldots, |X| = N$, $j = 1, \ldots, |Z| = M$ and by $X_{-i}$ we denote the set $X$ without the $i$-th variable). Consequently, if these functions are unable to distinguish two instances, then any other collection of views not satisfying the condition of the lemma is unable to distinguish them.

The instances are based on a *chess-board* that is illustrated in the simple case where $N = M = 1$ (see Figure 1: the white little cubes represent the tuples of $I_1$, the gray ones the tuples of $I_2$).

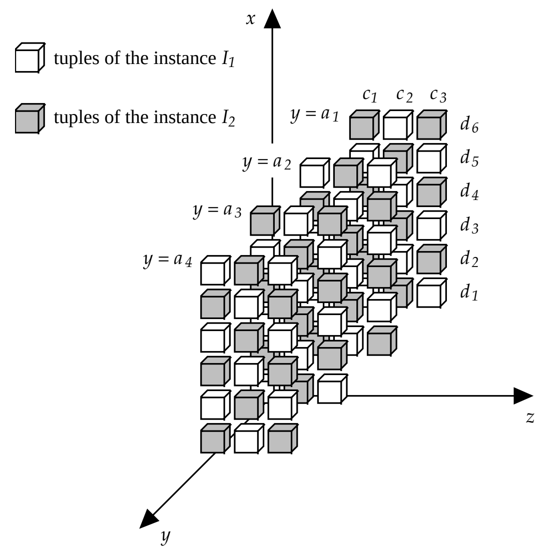


**Figure 1: The pair of chessboard instances**

For each variable in $X$ we consider a domain of 6 values $d_1, d_2, d_3, d_4, d_5, d_6$ such that the equalities $d_1 + d_3 + d_5 = d_2 + d_4 + d_6$ and $d_1 * d_3 * d_5 = d_2 * d_4 * d_6$ hold (e.g. $d_1 = 16$, $d_2 = 10$, $d_3 = 15$, $d_4 = 20$, $d_5 = 5$ and $d_6 = 6$). For each variable in $Z$ we similarly consider a domain of 3 values $c_1, c_2, c_3$ such that the equalities $c_1 + c_3 = c_2$ and $c_1 * c_3 = c_2$ hold (e.g. $c_1 = 1.5$, $c_2 = 4.5$ and $c_3 = 3$). Finally, we consider a domain of four values $\{a_1, a_2, a_3, a_4\}$ for the variable in $Y$, satisfying the condition $a_1 * a_3 = a_2 * a_4$ (e.g. $a_1 = 1$, $a_2 = 2$, $a_3 = 6$ and $a_4 = 3$).

It is easily shown that the query function $S_Y(X)$ returns distinct values on the two instances, e.g. $S_Y(d_1) = S_Y(16) = 2a_1 + a_2 + 2a_3 + a_4 = 19$ on $I_1$ and $S_Y(d_1) = S_Y(16) = a_1 + 2a_2 + a_3 + 2a_4 = 17$ on $I_2$. On the other hand all functions not satisfying the conditions of the lemma are unable to distinguish the two instances and produce the same values on both instances. The chess-board instantiation is then generalized to arbitrary values of $N$ and $M$. □

# 5. MATERIALIZED VIEWS

In this section, we consider the problem of designing materialized aggregate views. We consider two distinct strategies: (i) only materializing views with flat aggregation (e.g. count, sum and multiply), and to compute complex aggregation (e.g. average, percentage) from the former, and (ii) materializing any aggregate views. While the choice might not be visible to the user who might see logical views, materialized or computed from the materialized ones, it has a serious impact on the queries which can be answered, the complexity of query rewriting, and the incremental maintenance of views after updates.

## 5.1 Flat aggregate views

We first consider the case where only flat views are materialized. In this case, we design an algorithm to rewrite count, sum, multiply, average, percentage queries from materialized count, sum, multiply views. The algorithm **AggRew2** extends algorithm **AggRew1** by adding two cases in the case analysis as follows.

**The algorithm AggRew2**

**Input:** a query $F(X)$ to rewrite ($F = C$, $S_Y$, $M_Y$, $Perc_{X,X'}$, $Avg_Y$) and a collection of views $f_i(W_i)$ ($f_i = C, S_{U_i}, M_{U_i}$)
**Output:** a rewritten query $F'(X)$ using only the views or a failure message

**switch**($F$)
**case** $C$, $S_Y$, $M_Y$:
  ... /* as in **AggRew1** */
**case** $Perc_{A,B}$
  /* $F(X) = \frac{C(A,B)}{C(A)}$ ($X = A \cup B$) */
  **for each** count view $C(W_i)$
    **if** $(A \cup B) \subseteq W_i$
    **then return** $F'(X) = \frac{\overline{\Sigma^{W_i-(A\cup B)}(C(W_i))}(A,B)}{\overline{\Sigma^{W_i-A}(C(W_i))}(A)}$
**case** $Avg_Y$:
  **for each** count view $C(W_i)$
    **if** $(X \cup Y) \subseteq W_i$
    **then return** $F'(X) = \frac{\overline{\Sigma^{W_i-X}(Y * C(W_i))}(X)}{\overline{\Sigma^{W_i-X}(C(W_i))}(X)}$
  **for each** sum view $S_{U_j}(W_j)$
    **if** $Y = U_j$ and $X \subseteq W_j$
    **then**
      **for each** count view $C(W_i)$
        **if** $X \subseteq W_i$
        **then return**
          $F'(X) = \frac{\overline{\Sigma^{W_j-X}(S_{U_j}(W_j))}(X)}{\overline{\Sigma^{W_i-X}(C(W_i))}(X)}$
      **return** a failure message
**endswitch**
/*all attempts to rewrite have failed*/
**return** a failure message

The complexity of the algorithm **AggRew2** is rather low as shown by the next result.

COROLLARY 8. *The algorithm* **AggRew2** *terminates in time linear in the number of views and their arity.*

The proof follows directly from Proposition 3. Note that the inner loop that scans the count views does not produce (together with the outer loop that scans the sum views) a quadratic time: the count views are scanned only once after a "good" sum view has been detected. If the scan does not detect also a "good" count view, the algorithm exits with a failure message.

COROLLARY 9. *The algorithm* **AggRew2** *is complete.*

The proof follows directly from Theorem 4.

It is easily shown that the algorithm **AggRew2** computes the queries $Q1$, $Q2$ and $Q3$ of Example 2, by using the view $V1$ stored in flat form (in fact $C(Day, Source, Dest, Plan)$ is stored in practice).

## 5.2 Ratio aggregate views

In the sequel, we consider an algorithm for rewriting queries and views with ratio aggregation. Note that if views are materialized as ratios in the database, the information content of the views decreases, as shown by the following theorems.

THEOREM 10. *A percentage view* $v : \frac{C(N_v)}{C(D_v)}$ *subsumes a percentage query* $q : \frac{C(N_q)}{C(D_q)}$ *iff* $D_v \subseteq D_q$ *and* $N_q \subseteq N_v$.

**Proof**: The $\Leftarrow$ part is straightforward and is demonstrated later as a consequence of the soundness of the E-operators (Proposition 12). For the $\Rightarrow$ part we demonstrate that if $D_v \not\subseteq D_q$ or $N_q \not\subseteq N_v$ then the view cannot subsume the query. Particularly, if $N_q \not\subseteq N_v$, then the view could not subsume the query even if stored in flat form, by Corollary 9. Hence, assume that $D_v \not\subseteq D_q$, but $N_q \subseteq N_v$. It follows that there exists at least one variable $\eta$ which is in $D_v$ (and consequently also in $N_v$) but not in $D_q$. Let us consider also one variable $\xi$ which is not in $N_v$ (such a variable certainly exists because the count removes at least one variable). For simplicity, let us assume that $\xi$ and $\eta$ correspond to the last two attributes of $R$ and let us consider a tuple value $\bar{c}$ for the other attributes, two values $\hat{\xi}_1$ and $\hat{\xi}_2$ for $\xi$, and three values $\hat{\eta}_1$, $\hat{\eta}_2$, $\hat{\eta}_3$ for the variable $\eta$. Finally, let us consider the two instances

$I_1 = \{\langle \bar{c}, \hat{\xi}_1, \hat{\eta}_1 \rangle, \langle \bar{c}, \hat{\xi}_2, \hat{\eta}_2 \rangle\}$, and

$I_2 = \{\langle \bar{c}, \hat{\xi}_2, \hat{\eta}_1 \rangle, \langle \bar{c}, \hat{\xi}_1, \hat{\eta}_2 \rangle, \langle \bar{c}, \hat{\xi}_2, \hat{\eta}_3 \rangle\}$.

It is easily verified that $v$ has uniformly value 1 both for $I_1$ and $I_2$, while $q$ has uniformly value 1/2 for $I_1$ and value 1/3 for $I_2$. Consequently $v \not\models q$. □

By the previous theorem, the query $Q2$ of Example 2 cannot be computed by using only the view $V1$.

THEOREM 11. *An average view* $v : \frac{S_Y(A)}{C(A)}$ *subsumes an average query* $q : \frac{S_Y(B)}{C(B)}$ *iff* $A = B$.

The proof is analogous to that of Theorem 10.

The algorithm proposed in this section extends the previous one to the case of views stored as ratios. In particular the new algorithm allows the rewriting of the queries $Q1$ and $Q3$ of Example 2, by having $V1$ stored as a ratio.

The result of completeness obtained in the previous section together with Theorems 10 and 11 restrict the operations definable on percentage and average queries and views. We have therefore introduced the following elementary (E-) operations based on summarizations and reductions of ratios.

They are applied to (possibly couples of) count, sum, percentage and average functions and produce new functions of the same type.

**E-operations over ratio views**

**E1:** $C(A, B) \Rightarrow C(A)$
**E2:** $S_Y(A, B) \Rightarrow S_Y(A)$
**E3:** $C(Y, A) \Rightarrow S_Y(A)$
**E4:** $C(A, B) \Rightarrow \frac{C(A,B)}{C(A)}$
**E5:** $S_Y(A), C(A) \Rightarrow \frac{S_Y(A)}{C(A)}$
**E6:** $\frac{C(A,B)}{C(A)}, C(A) \Rightarrow C(A, B)$
**E7:** $\frac{S_Y(A)}{C(A)}, C(A) \Rightarrow S_Y(A)$
**E8:** $\frac{S_Y(A)}{C(A)}, S_Y(A) \Rightarrow C(A)$
**E9:** $\frac{C(Y,A)}{C(A)} \Rightarrow \frac{S_Y(A)}{C(A)}$
**E10:** $\frac{S_Y(A,B)}{C(A,B)}, \frac{C(A,B)}{C(A)} \Rightarrow \frac{S_Y(A)}{C(A)}$
**E11:** $\frac{C(A,B,D)}{C(A,B)}, \frac{C(A,B)}{C(A)} \Rightarrow \frac{C(A,B,D)}{C(A)}$
**E12:** $\frac{C(A,B,D)}{C(A)} \Rightarrow \frac{C(A,B)}{C(A)}$
**E13:** $\frac{C(A,B,D)}{C(A)} \Rightarrow \frac{C(A,B,D)}{C(A,B)}$

PROPOSITION 12. *The E-operations are sound.*

In this case the strategy to answer a query using a collection of views is not straightforward as in the case of count, sum and multiply views. In particular, there are operations like E11 that may require an opportune "preparation" of the two operands (by applying the operations E12 and E13) to be applied and the criteria for the application are not obvious. A simple algorithm can be designed, based on the non deterministic choices of the views and of the E-operations to apply and it can be shown that the algorithm works in non deterministic polynomial time.

However, we show in the following that by considering a different strategy, based on a distinct (but equivalent) set of operations, the problem can be solved in polynomial time and we provide an algorithm based on such a strategy. The algorithm is based on the idea of maintaining a minimal number of "active" functions during the computation, delaying the application of operations that produce an exponential growth, and instead using such operators to reduce the number of active functions.

We distinguish K-operations that produce "really new" functions and S-operations that are used both to reduce the set of active functions (on which K-operations are applied) and to produce the required function at the end of the computation. In practice, if one function can be obtained from one or more of the other functions by applying an S-operation, then it is removed from the set $\mathcal{F}$ of the active functions. A "local" S-reduction is included in the definition itself of the K-operations, hence the functions on the left side of the arrow are the functions that are active before the application of the operation, the functions on the right side are the functions that are active after the application of the operation and local reduction. Note that some of the S-operations coincide with E-operations, particularly those which produce an exponential growth of the number of functions.

**S-operations over ratio views**

**S1:** $C(A, B) \Rightarrow C(A)$
**S2:** $S_Y(A, B) \Rightarrow S_Y(A)$
**S3:** $C(Y, A, B) \Rightarrow S_Y(A)$
**S4:** $C(A, B, D) \Rightarrow \frac{C(A,B)}{C(A)}$
**S5:** $C(Y, A, B) \Rightarrow \frac{S_Y(A)}{C(A)}$
**S6:** $S_Y(A, B), C(A, D) \Rightarrow \frac{S_Y(A)}{C(A)}$
**S7:** $\frac{C(Y,A,D)}{C(A')}$ (where $A' \subseteq A$) $\Rightarrow \frac{S_Y(A)}{C(A)}$
**S8:** $\frac{S_Y(A,B)}{C(A,B)}, \frac{C(A,B,D)}{C(A')}$ (where $A' \subseteq A$) $\Rightarrow \frac{S_Y(A)}{C(A)}$
**S9:** $\frac{C(A,B,D)}{C(A')}$ (where $A' \subseteq A$) $\Rightarrow \frac{C(A,B)}{C(A)}$

**K-operations over ratio views**

**K1:** $\frac{S_Y(A)}{C(A)}, S_Y(A, D) \Rightarrow C(A), S_Y(A, D)$
**K2:** $\frac{S_Y(A)}{C(A)}, C(A, D) \Rightarrow S_Y(A), C(A, D)$
**K3:** $\frac{C(A,B)}{C(A)}, C(A, D) \Rightarrow C(A, B), C(A, D)$
**K4:** $\frac{C(Y,A,B)}{C(A)}, S_Y(A, D) \Rightarrow C(Y, A, B), S_Y(A, D)$
**K5:** $\frac{C(A,A',A'',B,B')}{C(A,A')}, \frac{C(A,A',A'',B,B'')}{C(A,A'')} \Rightarrow$
$\frac{C(A,A',A'',B,B')}{C(A,A'')}, \frac{C(A,A',A'',B,B')}{C(A,A')}, \frac{C(A,A',A'',B,B'')}{C(A,A'')}$
**K6:** $\frac{C(A,B,B')}{C(A)}, \frac{S_Y(A,B)}{C(A,B)}, S_Y(A, D) \Rightarrow$
$C(A, B, B'), S_Y(A, B), S_Y(A, D)$
**K7:** $\frac{C(A,B,B')}{C(A)}, \frac{S_Y(A,B)}{C(A,B)}, C(Y, A, D) \Rightarrow$
$C(A, B, B'), S_Y(A, B), C(Y, A, D)$

Note that K-operations do not introduce any new count or sum function. Indeed, it is evident that all resulting functions are constituted by extracting/rearranging the count and sum functions in (the numerators and denominators of) the operands.

PROPOSITION 13. *The number of functions generated by the iterated application of the K-operations (with local minimization) is at most quadratic in the number of view functions.*

**Proof**: First of all note that, apart from K5, the application of the K-operations does not increase the global number of functions. Let us consider now the increasing effect of K5. Since the only K-operation that produces new percentage functions is K5, from the previous remark, it follows that the number of percentage functions that can be generated by the iterated application of K5 is given at most by all possible combinations of numerators of percentage functions with denominators of percentage functions, i.e. by the square of the number of percentage functions in the database. □

EXAMPLE 3. *Let us consider a database containing m percentage views ($m = 2p$) and let us consider the percentage views* $\frac{C(x_1,\dots,x_p,x_{p+1})}{C(x_1)}$, $\frac{C(x_1,\dots,x_p,x_{p+2})}{C(x_2)}$, $\dots$, $\frac{C(x_1,\dots,x_p,x_m)}{C(x_p)}$. *It is easily shown that the operation K5 produces a quadratic number of percentage functions (given by all possible numerator/denominator combinations) and that such number can not be reduced by exploiting the dependencies expressed by the S-operations.* □

We can now introduce the algorithm **AggRew3**. Note that the set $\mathcal{F}$ contains the currently active functions, while $\mathcal{N}$ the newly generated ones (for simplicity the sets $\mathcal{F}$ and $\mathcal{N}$ are assumed to be globally visible to the algorithm and the procedures). Note also that the actual computation of the K-closure is not required. Only the description of the obtained functions needs to be maintained and the actual computation can be delayed at the end of the rewriting process, when the really necessary functions are computed. Also, the rewriting could be attempted after each K-iteration, without waiting for the completion of the K-closure.

**The algorithm AggRew3**

**Input:** a query $F(X)$ to rewrite ($F = C$, $S$, $M$, $Perc$, $Avg$) and a collection of views $f_i(W_i)$ ($f_i = C$, $S$, $M$, $Perc$, $Avg$)

**Output:** a rewritten query $F'(X)$ using only the views or a failure message

**Procedure S-Reduce** scans the functions in $\mathcal{F} \cup \mathcal{N}$ and eliminates all functions that can be obtained by other views in $\mathcal{F} \cup \mathcal{N}$ applying one of the S-operations.

**Procedure K-Iterate** scans the functions in $\mathcal{F}$ trying to obtain new functions by using the K-operations. New functions are inserted into $\mathcal{N}$.

**Procedure S-Rewrite** tries to rewrite the given query at the end of the K-iterations, by applying one S-operation to the functions in $\mathcal{F}$.

```
let N = ∅
S-Reduce(F ∪ N)
repeat
    F = F ∪ N
    let N = ∅
    K-iterate(F)
    S-Reduce(F ∪ N)
until N = ∅
S-Rewrite(F, F)
```

Before studying the main properties of the algorithm, let us show that the problems of view usability introduced in Section 2 are solved by the proposed algorithm.

EXAMPLE 4. *Consider the following queries and view of Example 2:*

$$V1 : \frac{C(Day, Source, Dest, Plan)}{C(Day, Source)}$$

$$Q1 : \frac{C(Day, Source, Plan)}{C(Day, Source)}$$

$$Q2 : \frac{C(Day, Source, Dest, Plan)}{C(Day)}$$

$$Q3 : \frac{C(Day, Source, Dest, Plan)}{C(Day, Source, Plan)}$$

*$Q1$ and $Q3$ are rewritten by using the view $V1$ and applying the operation S9. Obviously the query $Q2$ is not computed, since the information content is not sufficient (see Theorem 10). Note that the introduction of one additional view*

$$V2 : \frac{C(Day, Source, Dur)}{C(Day)}$$

*makes the rewriting possible and indeed the function is computed by the algorithm by applying the K5 operation.* □

We now state the main properties of the proposed algorithm.

THEOREM 14. *The procedure of S-reduction is Church-Rosser and works in polynomial time.*

THEOREM 15. *The algorithm* **AggRew3** *is complete with respect to the E-operations.*

THEOREM 16. *The complexity of algorithm* **AggRew3** *is polynomial in the number of views and their arity.*

# 6. CONCLUSION

We have considered the possibility of using materialized views from a database to answer queries over the database. Our purpose was to study the following question: *Is there always a way to answer the query by using only the views, whenever the views contain enough information?* Although we have shown that the answer is yes, the resulting query is of no practical use in general.

We have then considered this question in a restricted framework with views and queries with one aggregation over one relation. These restrictions although drastic correspond to a class of queries of practical importance in data warehousing that has already been extensively studied [19; 12; 17; 1; 10]. We have provided tractable and complete (in the sense of information content) methods for various cases of aggregate views and queries with or without ratios. To the best of our knowledge this is the first result in this direction.

Our results shed lights to the problem of designing materialized views. At first it seems that it is preferable to *materialize* only flat views, and to offer derived *logical* ratio views such as average . This leads to a linear time complete algorithm for rewriting queries. This approach has been pursued by many authors [12; 24; 27; 1].

Nevertheless this approach is not always possible. First of all, there are many examples of pulled data that are in ratio format, and they need to be managed. Second, the decision to give access to ratios might be motivated by privacy reasons.

For example, the company of Example 2 would certainly not make available the view containing the detail of the number of calls by day and state, but would probably exchange some data with other companies on the percentage of calls directed to mobile and wired networks for each day and state.

It is clear that in the previous example, it is fundamental to be able to prove that the sensitive data cannot be derived from the views. A hybrid approach can be used, by storing "safe" data in a public data warehouse, easily accessible, while storing more sensitive data, e.g. the count data that allows to recover all available information, in a secure data warehouse, accessible only to the decision making staff.

The problem of maintaining materialized views is crucial in this discussion since essentially, flat views can be maintained (full paper), while ratio views cannot. This advocates the use of a design with two levels of views as suggested above.

# 7. REFERENCES


[1] S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proceedings of the 22th International Conference on Very Large Data Bases, VLDB'96*, pages 506–521, 1996.

[2] S. Abiteboul, and O.M. Duschka. Complexity of Answering Queries Using Materialized Views. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS'98*, pages 254-263, 1998.

[3] F.N. Afrati, M. Gergatsoulis, and T.G. Kavalieros. Answering Queries Using Materialized Views with Disjunctions. In *Proceedings of the 7th International Conference on Database Theory, ICDT'99*, pages 435–452, 1999.

[4] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 232–243. IEEE Computer Society, 1997.

[5] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[6] D. Barbará and T. Imielinski. Sleepers and workaholics: Caching strategies in mobile environments. *VLDB Journal*, 4(4):567–602, 1995.

[7] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 359–370, 1999.

[8] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 190–200. IEEE Computer Society, 1995.

[9] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 155–166. ACM Press, 1999.

[10] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ullman. Computing iceberg queries efficiently. In *Proceedings of the 24th International Conference on Very Large Data Bases, VLDB'98*, pages 299–310, 1998.

[11] O.M. Duschka, and M.R. Genesereth Answering Recursive Queries Using Views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS'97*, pages 109–116, 1997.

[12] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proceed. 12th Intern. Conference on Data Engineering*, New Orleans, Louisiana USA, February 1996.

[13] S. Grumbach, M. Rafanelli, and L. Tininini. Querying aggregate data. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 174–184. ACM Press, 1999.

[14] H. Gupta. Selection of views to materialize in a data warehouse. In *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, volume 1186, pages 98–112. Springer, 1997.

[15] H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman. Index selection for olap. In *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 208–219. IEEE Computer Society, 1997.

[16] H. Gupta and I. Mumick. Selection of views to materialize under a maintenance cost constraint. In *Proceedings of the 7th International Conference on Database Theory, January 10-12, 1999, Jerusalem, Israel*, pages 453–470. Springer, 1999.

[17] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cube efficiently. In *Proceed. Intern. Conference on Management of Data - SIGMOD '96*, Montreal, Canada, 1996.

[18] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. ACM Symp. on Principles of Database Systems*, pages 95–104, 1995.

[19] F. Malvestuto. A universal-scheme approach to statistical databases containing homogeneous summary tables. *ACM Transactions on Database Systems*, 18(4):678–708, 1993.

[20] F. Malvestuto and M. Moscarini. Computational issues connected with the protection of sensitive statistics by auditing sum queries. In *10th International Conference on Scientific and Statistical Database Management, Proceedings, Capri, Italy, July 1-3, 1998*, pages 134–144. IEEE Computer Society, 1998.

[21] F. Malvestuto, M. Moscarini, and M. Rafanelli. Suppressing marginal cells to protect sensitive information in a two-dimensional statistical table. In *Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 29-31, 1991, Denver, Colorado*, pages 252–258. ACM Press, 1991.

[22] I. Mumick, D. Quass, and B. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 100–111. ACM Press, 1997.

[23] G. Ozsoyoglu, Z. Ozsoyoglu, and V. Matos. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Transactions on Database Systems*, 12(4):566–592, December 1987.

[24] D. Quass. Maintenance expressions for views with aggregation. In *VIEW 1996, Proceedings of the Workshop on Materialized Views: Techniques and Applications, June 7, 1996, Monteal, Canada*, pages 110–118, 1996.

[25] M. Rafanelli, A. Bezenchek, and L. Tininini. The aggregate data problem: a system for their definition and management. *ACM Sigmod Record*, 25(4):8–13, December 1996.

[26] A. Shoshani. Olap and statistical databases: Similarities and differences. In *Proceed. Intern. Confer. on Principles of Database Systems, PODS'97*, Tucson, Arizona, USA, May 1997.

[27] D. Sristava, S. Dar, H. Jagadish, and A. Levy. Answering queries with aggregation using views. In *Proc. of Intl. Conf. on Very Large Data Bases*, pages 318–329, 1996.

[28] J. Widom. Research problems in data warehousing. In *CIKM 1995, Proceedings of the 4th International Conference on Information and Knowledge Management, November, 1995, Baltimore, Maryland, USA*, pages 25–30, 1995.

[29] O. Wolfson, A. Sistla, S. Dao, K. Narayanan, and R. Raj. View maintenance in mobile computing. *SIGMOD Record*, 24(4):22–27, 1995.