View Materialization for Nested GPSJ Queries

Matteo Golfarelli
DEIS – University of Bologna
Viale Risorgimento 2, Bologna, Italy
mgolfarelli@deis.unibo.it

Stefano Rizzi
DEIS, CSITE CNR – University of Bologna
Viale Risorgimento 2, Bologna, Italy
srizzi@deis.unibo.it

Abstract

View materialization is a central issue in logical design of data warehouses since it is one of the most powerful techniques to improve the response to the workload. Most approaches in the literature only focus on the aggregation patterns required by the queries in the workload; in this paper we propose an original approach to materialization in which the workload is characterized by the presence of complex queries which cannot be effectively described only by their aggregation pattern. In particular, we consider queries represented by nested GPSJ (Generalized Projection / Selection / Join) expressions, in which sequences of aggregate operators may be applied to measures and selection predicates may be formulated, at different granularities, on both dimensions and measures. Other specific issues taken into account are related to the need for materializing derived measures as well as support measures to make algebraic operators distributive. Based on this query model, an efficient algorithm to determine a restricted set of candidate views for materialization, to be fed into an optimization algorithm, is proposed. Finally, the effectiveness of our approach is discussed with reference to a sample workload.

1 Introduction

A *data warehouse* (DW) is a repository which provides an integrated environment for decision support. From a logical point of view, a DW can be seen as a multidimensional cube where each cell contains information useful for the decision process, i.e., a set of numerical attributes called *measures*, while each axis represents a possible *dimension* for analysis. For example, in a DW modeling a chain store, time, product and store are typical dimensions while quantity sold and retail price are measures.

Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2000)
Stockholm, Sweden, June 5-6, 2000
(M. Jeusfeld, H. Shu, M. Staudt, G. Vossen, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-28/

A DW implemented on a relational DBMS is usually organized according to the so-called *star scheme* [Kim96], composed by one *fact table* containing the measures and one denormalized *dimension table* for each dimension of analysis. The primary key of each dimension table is imported into the fact table; the primary key of the fact table is defined by the set of these foreign keys. Each dimension table contains a set of *dimension attributes* defining a hierarchy of aggregation levels for the corresponding dimension.

The basic mechanism to extract useful information from elemental data in DWs is aggregation. In principle, all queries could be solved by directly aggregating the elemental data in the *base* fact table; for efficiency reasons, a DW typically stores, besides the elemental values of measures, also values summarized according to some *aggregation pattern*, i.e., to a set of dimension attributes defining the coarseness of aggregation. Even summarized data are organized into a star scheme, whose fact table is usually called a *view*; measure values are obtained by applying an aggregation operator to the data in the base fact table or in another view with a finer pattern. Of course, the larger the number of queries a view can be used to answer, the more effective materializing it [Coh99, Yan95].

Since pre-computing all the possible views is unfeasible, several techniques to select an appropriate subset of views to materialize have been proposed. While most approaches only focus on the aggregation patterns required by queries, very few works analyze how the presence of different operators affects aggregation. On the other hand, using only one operator (typically, the sum) to aggregate measures when materializing views, makes them useless for several queries.

In this paper we propose an original approach to materialization in which the workload is characterized by the presence of complex queries which cannot be effectively described only by their aggregation pattern. In particular, we consider queries represented by Nested Generalized Projection / Selection / Join (NGPSJ) expressions, in which sequences of aggregate operators may be applied to measures and selection predicates may be defined, at different granularities, on both dimensions and measures. As a result, the correspondence between measures in the views and in the base fact table is not necessarily one-to-one: some measures in the base fact table may not be present at all in a view, while others may correspond to several measures in the view; furthermore, additional (either support or derived) measures may be included in the view in order to correctly compute aggregations. Particular emphasis is given to the effects on materialization of the contemporary presence of multiple aggregation operators in a query.

Under these assumptions, an efficient algorithm to determine the restricted set of views which could be usefully materialized (*candidate views*) is proposed. The algorithm builds a *query view graph* whose vertices represent candidate views and whose arcs denote the possibility of computing a view from another. The query view graph may then be fed into an optimization algorithm like the ones proposed in [Gup97, Gup99] which select, from the set of the candidate views, the subset which minimizes the response to the workload under a given space constraint.

After discussing NGPSJ expressions in Section 2 and classifying aggregate operators in Section 3, in Section 4 we define how views will be structured. The algorithm for determining candidate views is outlined in Section 5, while Section 6 discusses the experimental results.

1.1 Motivating Example

The simple *Sales* star scheme used as a working example is defined below:

STORE (<u>StoreId</u>, SName, SCity)
TIME (<u>TimeId</u>, TDay, TMonth, TYear)
PRODUCT (<u>ProductId</u>, PName, PType, PCategory)
SALE (<u>TimeId</u>, <u>StoreId</u>, <u>ProductId</u>, Qty, Prc)

where the denormalized dimension tables leave out the following hierarchies of functional dependencies:

$$\text{SName}\rightarrow\text{SCity},\ \text{TDay}\rightarrow\text{TMonth}\rightarrow\text{TYear},$$
$$\text{PName}\rightarrow\text{PType}\rightarrow\text{PCategory}$$

Tuples in dimension tables are typically identified by surrogate keys; in the rest of the paper, for the sake of simplicity, surrogate keys will never be considered, assuming that tuples are identified by an attribute of the hierarchy.

As to the workload, consider the following queries on the *Sales* star scheme:

> q_1: "*For each product, find the average selling price and the maximum total quantity sold for the stores which sold more then 1000 units of that product*"
>
> q_2: "*For each month and for each type of food product, find the average selling price and the total revenue*"
>
> q_3: "*For product of type beverage and for each year, find the total quantity sold and the average of the total monthly revenues*"

Query q_1 requires to first aggregate data on pattern {SName, PName}, calculating the average price and the sum of the quantities sold, then to select the stores for which this sum is greater than 1000, finally to further aggregate on {PName} calculating the maximum quantity and the average price. Query q_2 aggregates foodstuff sales on {TMonth, PType}, calculating for each month the average price and the sum of the derived measure R = Qty×Prc. Query q_3 requires to aggregate on {TMonth, PName} calculating the monthly sums of the quantities and that of the revenues, then to aggregate on {TYear, PName} calculating the sums of the monthly quantities and the averages of the monthly revenues.

These examples give rise to some considerations:

1. Different aggregate operators may be applied to the same measure (e.g., SUM(Qty), MAX(Qty)).
2. Sequences of aggregate operators may be applied to a measure (e.g., the maximum of the sums).
3. Some aggregations may require support measures in order to be correctly calculated from partial aggregates (e.g., the average operator requires the cardinality of each partial aggregate).
4. Calculating a derived measure from its component measures may determine a wrong result if the query is not solved directly on the base fact table (e.g., SUM(Qty)*AVG(Prc) ≠ SUM(Qty*Prc)).

In all these cases, in order to take full advantage of materialization, it may be necessary to include additional measures in views. Besides:

5. The results of aggregation are affected by the selections which operate on its source data.

1.2 Related Literature

Several works adopt graph-like structures to represent views, but a few focus on how to build it. In [The97] an algorithm to build a *multiquery graph* is outlined; the approach deals with the problem of building base fact tables from operational databases. Besides, since only join, selection and classical projection operators are involved, and no aggregation semantics is introduced, the approach cannot be used to determine aggregate views.

The approach which is most closely related to ours is the one described in [Aki99]. That work is improved by ours with reference to the class of queries considered (GPSJ queries with distributive operators vs. NGPSJ queries with distributive, algebraic and holistic operators) and to the computational cost of the graph-building algorithm. In [Aki99] the complexity is always exponential since, for each query, all the possible evaluation strategies are built and organized into a graph; the query view graph is then obtained by applying a set of rules that prune non-candidate views. The complexity of our algorithm is exponential only in the worst case, to become polynomial in the best case.

Query nesting is considered in [Ros98], where subqueries involve multiple dependent aggregates at multiple granularities (*multi-feature cubes*). Though interesting results concerning distributive and algebraic cubes are provided, the queries considered are quite different from NGPSJ queries, since no nesting of distinct aggregation operators at different granularities is possible.

2 Query Modeling

In principle, the workload for a DW is dynamic and unpredictable. A possible approach to cope with this fact consists in monitoring the actual workload while the DW is operating. Otherwise, the designer may try to determine a core workload a priori: in fact, on the one hand, the user typically knows in advance which kind of data analysis (s)he will carry out more often for decisional or statistical purposes; on the other, a substantial amount of queries are aimed at extracting summary data to fill standard reports.

Basically, the approaches to view materialization differ in the level of detail they adopt to model the queries in the workload. Some approaches only consider the aggregation patterns [Bar97], while others analyze the query features in more detail [Gup97].

Our approach falls in the second group; in particular, we consider queries represented by *nested GPSJ expressions*. A GPSJ (Generalized Projection / Selection / Join) expression [Gup95] is a selection σ_1 over a generalized projection π over a selection σ_2 over a set of joins χ: $\sigma_1\pi\sigma_2\chi$. The *generalized projection* operator, $\pi_{P,M}(R)$, is an extension of duplicate eliminating projection, where *P* denotes an aggregation pattern, i.e., the set of group-by attributes, and *M* denotes a set of aggregate operators applied to the attributes in *R*. Thus, GPSJ expressions extend select-joins expressions with aggregation grouping and group selection.

Nesting GPSJ expressions means using the result from an expression as the input for another; it adds expressive power to GPSJ expressions since it allows sequences of aggregate operators to be used on a measure. For instance, the queries in Section 1 can be formulated as follows:

q_1: $\pi_{\text{PName,WAVG(P,C),MAX(Q)}}(\sigma_{\text{Q>1000}}$
$(\pi_{\text{PName,SName,Q=SUM(Qty),P=AVG(Prc),C=COUNT(*)}}$
$(\text{SALE} \bowtie \text{PRODUCT} \bowtie \text{STORE})))$

q_2: $\pi_{\text{PType,TMonth,AVG(Prc),SUM(Qty.Prc)}}(\sigma_{\text{PCategory='Foodstuffs'}}$
$(\text{SALE} \bowtie \text{PRODUCT} \bowtie \text{TIME}))$

q_3: $\pi_{\text{TYear,PName,SUM(Q),AVG(R)}}$
$(\pi_{\text{TYear,TMonth,PName,Q=SUM(Qty),R=SUM(Qty.Prc)}}$
$(\sigma_{\text{PType='Bev'}}(\text{SALE} \bowtie \text{PRODUCT} \bowtie \text{TIME})))$

where WAVG(m,w) computes the weighted average of measure *m* based on the weights *w*. The SQL formulation of q_1 is the following:

```
SELECT PN,SUM(P*C)/SUM(C),MAX(Q)
FROM ( SELECT PRODUCT.PName AS PN,
         STORE.SName AS SN,SUM(SALE.Qty) AS Q,
         AVG(SALE.Prc) AS P,COUNT(*) AS C
       FROM SALE,PRODUCT,STORE
       WHERE <...join conditions...>
       GROUP BY PRODUCT.PName,STORE.SName )
WHERE Q>1000
GROUP BY PN
```

The expressions above are in *normal form* [Gup95], i.e.:

- the necessary joins are pushed below all projections and selections;
- projection and selections are coalesced as much as possible;
- selections on dimensional attributes are pushed below all projections and selections on measures[1].

In the rest of the paper we will consider NGPSJ expressions in normal form applied to a star scheme $\mathcal{S}$ with base fact table *FT* and dimension tables $DT_1,...DT_n$, structured as follows:

e: $\sigma_{\text{PredM}_h}(\pi_{\text{P}_h,\text{M}_h}(\ldots\ \sigma_{\text{PredM}_1}(\pi_{\text{P}_1,\text{M}_1}(\sigma_{\text{PredA}\ \wedge\ \text{PredM}_0}$
$(\text{FT} \bowtie \text{DT}_1\ \ldots\ \bowtie \text{DT}_n)))\ldots))$

where:

- M_j (j=1,...h) is a set of aggregate measures, each defined as $\text{OP}(\text{m}_1,\ldots\text{m}_{\text{max}_{\text{OP}}})$ where OP is an aggregate operator, max_{OP} is the number of arguments of OP and m_i is an algebraic expression involving measures in M_{j-1}; M_0 denotes the set of measures in *FT*.
- $PredM_j$ (j=0,...h) is a conjunction/disjunction of Boolean predicates expressed on measures in M_j;
- P_j (j=1,...h) is an aggregation pattern.
- *PredA* is a conjunction/disjunction of Boolean predicates expressed on dimension attributes of $DT_1,...DT_n$.

We wish to emphasize that, in this paper, we are not interested in determining optimal execution plans. Expressions are written in normal form for the sake of clarity and in order to simplify their management in the algorithm for building the query view graph; the order in which operators appear within the expressions *does not* reflect the execution plans that will be used to calculate them. For instance, pushing all the joins below the other operators would obviously be inefficient in several cases, and pushing some projections/selections down might be highly preferable.

For the same reason, we will assume for simplicity that the fact table is always joined to *all* the dimension tables (though in some queries, depending on the aggregation pattern required, it may be possible to omit one or more of them; see q_1, q_2 and q_3 for instance). As to aggregation patterns, we will write them in a concise non-redundant form by dropping all the attributes functionally determined by other attributes in the pattern. For instance, q_3 can be rewritten in concise form as

$$\pi_{\text{TYear,PName,SUM(Q),AVG(R)}}(\pi_{\text{TMonth,PName,Q=SUM(Qty),R=SUM(Qty.Prc)}}$$
$$(\sigma_{\text{PType='Bev'}}(\text{SALE} \bowtie \text{PRODUCT} \bowtie \text{TIME})))$$

by dropping attribute TYear from the intermediate pattern since TMonth→TYear.

[1] While in general moving a selection on a measure in different positions of an expression affects the expression semantics, selections on dimensional attributes can be placed indifferently in any position outside the joins. Placing them in the innermost possible position makes the materialization algorithm simpler since no push-downs or pull-ups are required to compare couples of expressions.

Definition 1. Given two NGPSJ expressions e_i, e_j defined on star scheme $\mathcal{S}$, we say that e_j can be *derived* from e_i ($e_i \geq e_j$) if, by applying a sequence of generalized projections and selections to e_i, it is possible to obtain a NGPSJ expression which, put in normal form, is equivalent to e_j.

Derivability holds when every measure returned in e_j can be calculated from those returned in e_i. In evaluating derivability, three factors must be taken into account: aggregation patterns, selections on both dimension attributes and measures, and the aggregation operators applied to measures. For instance, expression

$$\pi_{\text{PType,TYear,SUM(Q)}}(\sigma_{\text{Q>2000}}(\pi_{\text{PName,TMonth,Q=SUM(Qty)}}(\sigma_{\text{PCategory='Foodstuffs'} \wedge \text{TYear>'1997'}}(\text{SALE} \bowtie \text{PRODUCT} \bowtie \text{TIME}))))$$

can be derived from

$$R = \sigma_{\text{Q>1000}}(\pi_{\text{PName,TMonth,Q=SUM(Qty)}}(\text{SALE} \bowtie \text{PRODUCT} \bowtie \text{TIME}))$$

by computing

$$\pi_{\text{PType,TYear,SUM(Q)}}(\sigma_{\text{PCategory='Foodstuffs'} \wedge \text{TYear>'1997'} \wedge \text{Q>2000}}(R)).$$

On the other hand, expression

$$\sigma_{\text{Q>1000}}(\pi_{\text{PName,TMonth,Q=SUM(Qty)}}(\sigma_{\text{TDay>'Feb15,97'} \wedge \text{Prc>10}}(\text{SALE} \bowtie \text{PRODUCT} \bowtie \text{TIME})))$$

cannot be derived from R since it specifies additional selections on a dimension attribute which did not appear in R due to aggregation (TDay) and on a measure which in R is not available at the granularity required (Prc).

3 Aggregate Operators

An aggregate operator is a function mapping a multiset of values into a single value. An attempt to define rules for aggregation of measures is presented in [Len97] and [Sho96]: measures are classified into *flow*, *stock* and *value-per-unit*, and a table defines when each class of measures can be aggregated depending on the operators used and on the type of hierarchy (temporal, non-temporal). A classification of aggregation functions which is relevant to our approach is presented in [Gra95]:

- *Distributive*, that allows aggregates to be computed directly from partial aggregates.
- *Algebraic*, that require additional information (*support measures*) to calculate aggregates from partial aggregates.
- *Holistic*, that do not allow aggregates to be computed from partial aggregates through a finite number of support measures.

Other works demonstrating the importance of aggregation operators are [Ros98b], which studies the problem of determining if a conjunction of aggregation constraints is feasible, and [Cab99], which presents a theoretical framework for studying aggregations from a declarative and operational point of view.

3.1 Support Measures

Query q can be solved on a view v if the NGPSJ expression representing q can be derived from the one which defines v ($v \geq q$). If a distributive operator is used, aggregation can be equivalently carried out by steps; thus,

$$\pi_{\text{TYear,SUM(Qty)}}(\text{SALE} \bowtie \text{STORE} \bowtie \text{PRODUCT} \bowtie \text{TIME})$$

can be derived from

$$R = \pi_{\text{TMonth,Q=SUM(Qty)}}(\text{SALE} \bowtie \text{STORE} \bowtie \text{PRODUCT} \bowtie \text{TIME})$$

by computing $\pi_{\text{TYear,SUM(Q)}}(R)$.

On the other hand, if an algebraic operator is used for aggregation, derivability holds only if the necessary support measures are added to the view. For instance, in order to calculate the average of a set of elemental values from a set of partial aggregates of those values, the cardinality of each single partial aggregate must be known. Thus,

$$\pi_{\text{TYear,AVG(Qty)}}(\text{SALE} \bowtie \text{STORE} \bowtie \text{PRODUCT} \bowtie \text{TIME})$$

can be derived from

$$R = \pi_{\text{TMonth,Q=AVG(Qty),C=COUNT(*)}}(\text{SALE} \bowtie \text{STORE} \bowtie \text{PRODUCT} \bowtie \text{TIME})$$

by computing $\pi_{\text{TYear,WAVG(Q,C)}}(R)$. Obviously, computing $\pi_{\text{TYear,AVG(Q)}}(R)$ leads to a different result.

Finally, for a holistic operator, aggregation can never be carried out by steps since the number of necessary support measures is not bounded.

3.2 Aggregation Sequences

Aggregation sequences increase the query language expressivity in two ways: by allowing two or more different aggregate operators to be applied in sequence on a measure, and by allowing selections to be applied on partially aggregated measures.

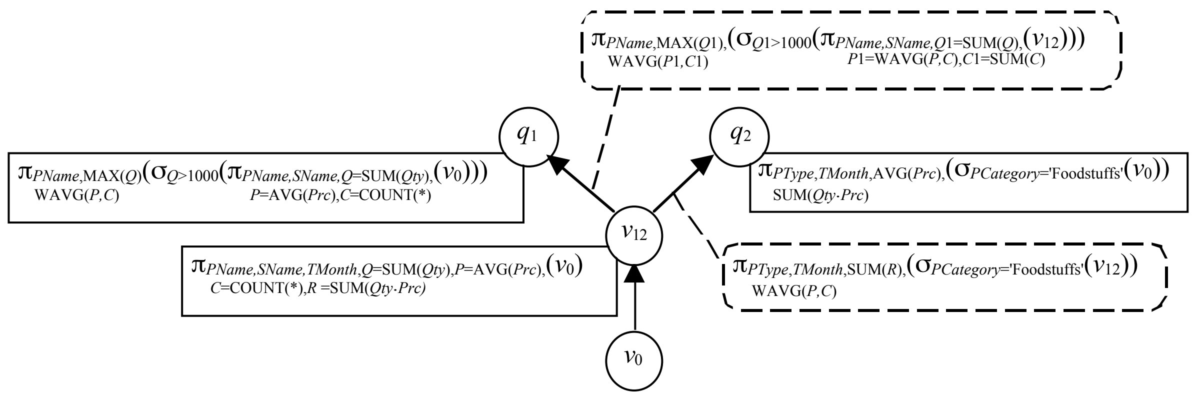
Consider the following query:

$$\pi_{P_2,m_2=OP_2(m_1)}(\pi_{P_1,m_1=OP_1(m)}(\pi_{P_0,m}(\ldots)))$$

where $OP_1 \neq OP_2$. In general, there is no way of computing m_2 directly from m by coalescing OP_1 and OP_2. Besides, the aggregation sequence determines the query result: in fact, swapping OP_1 and OP_2 in q would change the semantics of m_2; the semantics also depends on the intermediate pattern P_1, which determines the partial aggregates on which OP_2 operates.

As to the use of selections, consider the query

$$\pi_{P_1,m_1=OP(m)}(\sigma_{predm}(\pi_{P_0,m}(\ldots)))$$



$\pi_{PName,\text{MAX}(Q1),\text{WAVG}(P1,C1)}(\sigma_{Q1>1000}(\pi_{PName,SName,Q1=\text{SUM}(Q),P1=\text{WAVG}(P,C),C1=\text{SUM}(C)}(v_{12})))$
q_1
q_2
$\pi_{PName,\text{MAX}(Q),\text{WAVG}(P,C)}(\sigma_{Q>1000}(\pi_{PName,SName,Q=\text{SUM}(Qty),P=\text{AVG}(Prc),C=\text{COUNT}(*)}(v_0)))$
$\pi_{PType,TMonth,\text{AVG}(Prc),\text{SUM}(Qty\cdot Prc)}(\sigma_{PCategory=\text{'Foodstuffs'}}(v_0))$
v_{12}
$\pi_{PName,SName,TMonth,Q=\text{SUM}(Qty),P=\text{AVG}(Prc),C=\text{COUNT}(*),R=\text{SUM}(Qty\cdot Prc)}(v_0)$
$\pi_{PType,TMonth,\text{SUM}(R),\text{WAVG}(P,C)}(\sigma_{PCategory=\text{'Foodstuffs'}}(v_{12}))$
v_0

Figure 1: QVG for queries q_1 and q_2 (v_0 = SALE⋈STORE⋈PRODUCT⋈TIME).

in which the selection predicate on m reduces the set of partial aggregates on which OP operates. In general, both pushing down and pulling up the selection operator alter the query semantics.

The different semantics which may be induced by aggregation sequences play a central role in determining derivability between NGPSJ expressions; as shown in Section 5, this may lead to a dramatic increase in the number of candidate views.

4 Views

View materialization consists in selecting, among all the possible views, those that optimize the response to the workload under both a disk space and a maintenance cost constraint; this can be formulated as a knapsack problem, and is obviously NP-hard. Most methods in the literature aim at determining the subset of views that could be useful with reference to the workload (*candidate views*), then adopt heuristic algorithms to find sub-optimal solutions [Gup97, Gup99]. In [Aki99], this is done with reference to distributive aggregate functions; neither derived/support measures nor nesting of GPSJ queries are considered. Within this paper we focus on determining candidate views; the exponential dimension of the space of the possible views makes this step more complex and crucial than that of selecting the candidate views to be materialized.

Definition 2. Given a star scheme $\mathcal{S}$ and a workload $Q=\{q_1,...q_m\}$ on $\mathcal{S}$, a *candidate view* is a fact table defined by a NGPSJ expression e on $\mathcal{S}$ such that:

$$(\exists\, q_i \in Q \mid e = q_i) \vee (\exists\, q_i, q_j \in Q \mid (e \geq q_i) \wedge (e \geq q_j) \wedge (\nexists e' \mid (e \geq e') \wedge (e' \geq q_i) \wedge (e' \geq q_j)))$$

In other words, a candidate view is a fact table which either includes exactly the tuples returned by one query or is the 'coarsest' table including all the tuples allowing two or more queries to be solved. In [Bar97] the authors prove that candidate views defined as above are the only relevant views for materialization, i.e., that materializing non-candidate views may never decrease the workload cost. In our approach, candidate views may:

- contain a subset of the tuples at a given aggregation pattern as a consequence of selections on dimension attributes and measures;
- contain only a subset of the measures in the base fact table as a result of projections;
- include measures obtained by applying different aggregation sequences to the same measure;
- include derived measures and support measures necessary to support queries based on algebraic operators.

The widened definition of views given in our approach may lead to proliferation of measures; very large tables may entail poor performance. In order to avoid this, it may be convenient to adopt vertical partitioning techniques capable of splitting views into smaller fragments aimed at optimizing performance for a given workload [Gol00].

The candidate views and the relationships between them can be represented in a graph:

Definition 3. The *query view graph* (QVG) for workload Q is the directed acyclic graph $\mathcal{G}=(V,A)$ such that:

- $V=\{v_0,...v_p\}$, where v_0 is the base fact table joined with the dimension tables and $v_1,...v_p$ are the candidate views.
- Each arc $a_{ij} \in$ A denotes that $v_i \geq v_j$. Only the elemental derivability relationships are represented in A; those which can be obtained by transitivity are omitted.

The QVG for a given workload is necessarily unique, since the set of candidate views is univocally determined by the set of queries[2]. Figure 1 represents the QVG for queries q_1 and q_2 in Section 1; to help the reader, also the expressions which allow one view to be computed from another are shown (in dashed call-outs).

[2] Actually, since some aggregation operators can be expressed in terms of others (e.g., AVG can be computed from SUM and COUNT), some candidate views can be rewritten using different operators; nevertheless, the QVG is still unique since the set of queries which can be computed on each candidate view does not change.

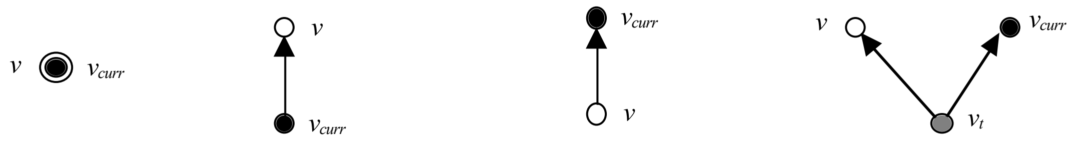


Figure 2: Different relationships between couples of views.

5 Building the Query View Graph

The QVG for star scheme $\mathcal{S}$ and workload $Q=\{q_1,...q_m\}$ is built incrementally by merging each query with the QVG according to the expression which defines it:

```
QVGBuilder(Q,v₀):
{ V={v₀,q₁};
  A={(v₀,q₁)};
  QVG=(V,A);
  for each qᵢ∈Q, i>1 do
  { S={vⱼ∈V| Child(vⱼ)=∅};
    // S set of the leaves in QVG;
    // function Child(vⱼ)
    // returns {vₖ∈V|(vⱼ,vₖ)∈A}
    V∪={qᵢ};
    Merge(S,qᵢ,NULL);
  }
}
```

Merging rules are aimed at determining if and how a query can be answered on a view belonging to the QVG. In general, the four possible relationships between two candidate views v and v_{curr} as entailed by Definition 1 are presented in Figure 2. In case A, v and v_{curr} are equivalent; in cases B and C, one of the views can be derived from the other; in case D no derivation relationship can be established. In case D, v_t is defined as follows:

Definition 4. Given two candidate views v_i and v_j, the *ancestor* of v_i and v_j is the candidate view v_t such that (1) both v_i and v_j can be derived from v_t and (2) for each other candidate view $v_{t'}$ which satisfies (1), it is $v_{t'} \geq v_t$ (in the worst case, it is $v_t \equiv v_0$).

The pseudo-code for the `Merge` algorithm is shown in Figure 3. The algorithm finds the correct position for inserting v in the QVG, starting from its leaves and descending towards its root v_0. If v is already present (case A), no additional views must be inserted and `Merge` terminates. Otherwise, if while descending a path a derivability relationship (case B) with v is found, the correct insertion position on that path has been reached; if a D relationship (no derivability) among v and some view v_{curr} is found, the path is abandoned and v_{curr} is inserted into a set *Drel* to be further examined. At the end, for each view in *Drel*, a new candidate view v_t is determined by function `Ancestor` and recursively merged into the QVG.

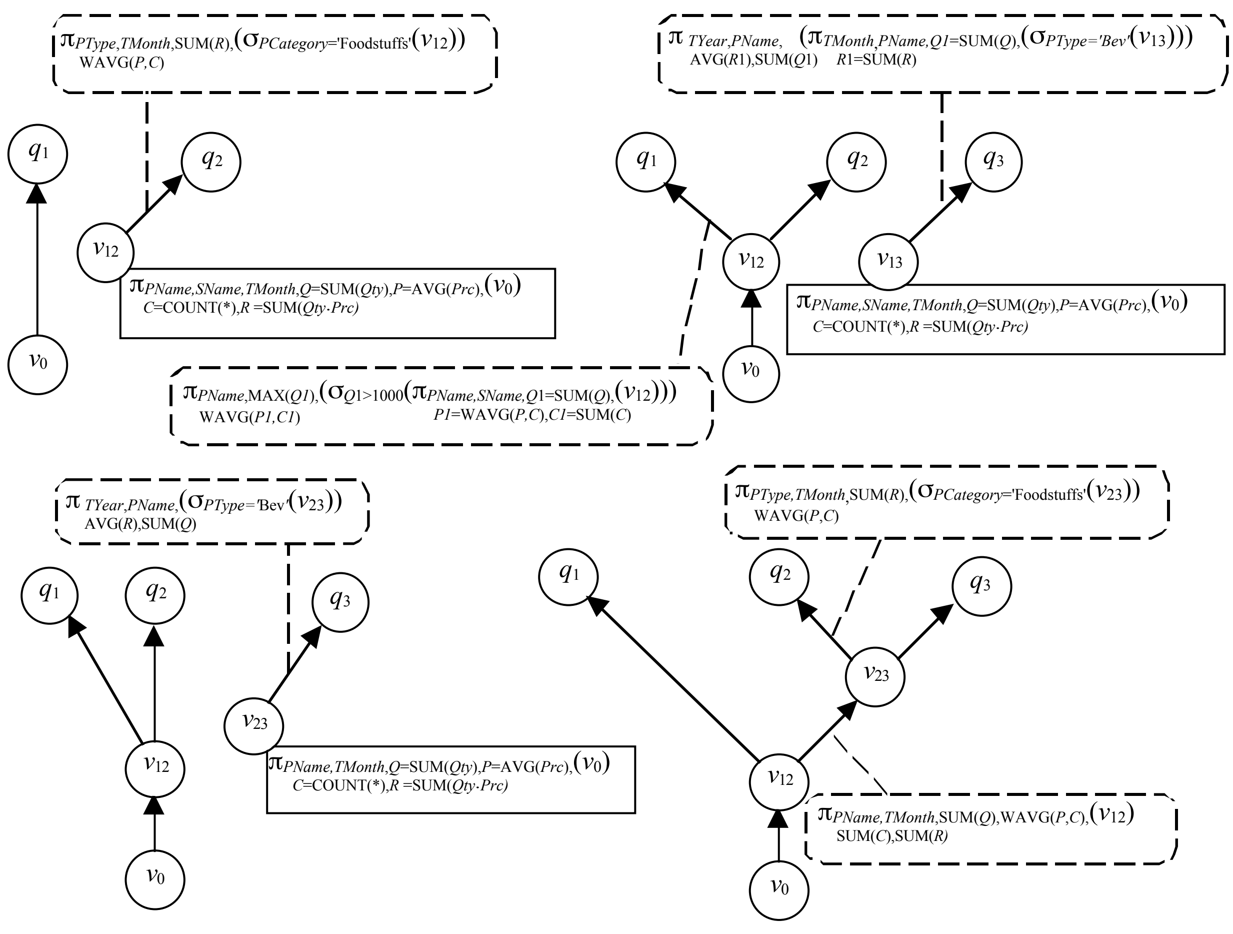
The `ExploreSubGraph` function carries out a depth-first exploration of the sub-graph including all the predecessors of v_{curr}; its pseudo-code is shown in Figure 4.

```
Merge(S,v,v_curr):
{ Drel=∅; // vertices in relationship D with v
  for each v_curr∈S do
  { inserted=FALSE;
    if ExploreSubGraph(v,v_curr,NULL,inserted)
    // case A holds
      then return;
    if not inserted then Drel∪={v_curr};
  }
  if (Drel≠∅) then          // case D holds
    for each vⱼ∈Drel do
    { vₜ=Ancestor(vⱼ,v);
      V∪={vₜ};
      A∪={(vₜ,v)};
      S'={vₖ|vⱼ∈Child(vₖ)};
      A∪={(vₜ,vⱼ)};
      Merge(S',vₜ,vⱼ);        // merge vₜ
    }
}
```

Figure 3: The `Merge` algorithm.

```
Boolean ExploreSubGraph(v,v_curr,v_succ,inserted):
// returns TRUE if v is already present
// among the predecessors of v_curr
{ Case Compare(v,v_curr) of:
  { 'A': { V-={v};
          for each vⱼ∈Child(v) do
          // v already exists as v_curr
          { A-={(v,vⱼ)}; A∪={(v_curr,vⱼ)}; }
          RemoveRedundantArcs();
          return TRUE;
        }
    'B': { A∪={(v_curr,v)};       // v_curr≥v
          if v_succ≠NULL then A-={(v_curr,v_succ)};
          inserted=TRUE;
        }
    'C': { A∪={(v,v_curr)};       // v≥v_curr
          if v_succ≠NULL then A-={(v,v_succ)};
          for each vⱼ|v_curr∈Child(vⱼ) do
           if ExploreSubGraph(v,vⱼ,v_curr)
             then return TRUE;
        }
    'D':         // no derivability relationship
  }
  return FALSE;
}
```

Figure 4: The `ExploreSubGraph` function.



π_{PType,TMonth},SUM(R),WAVG(P,C)(σ_{PCategory='Foodstuffs'}(v12))
q1
q2
v12
v0
π_{PName,SName,TMonth,Q=SUM(Qty),P=AVG(Prc),C=COUNT(*),R=SUM(Qty·Prc)}(v0)
π_{TYear,PName,AVG(R1),SUM(Q1)}(π_{TMonth,PName,Q1=SUM(Q),R1=SUM(R)}(σ_{PType='Bev'}(v13)))
q1
q2
q3
v12
v13
v0
π_{PName,SName,TMonth,Q=SUM(Qty),P=AVG(Prc),C=COUNT(*),R=SUM(Qty·Prc)}(v0)
π_{PName,MAX(Q1),WAVG(P1,C1)}(σ_{Q1>1000}(π_{PName,SName,Q1=SUM(Q),P1=WAVG(P,C),C1=SUM(C)}(v12)))
π_{TYear,PName,AVG(R),SUM(Q)}(σ_{PType='Bev'}(v23))
q1
q2
q3
v12
v23
v0
π_{PName,TMonth,Q=SUM(Qty),P=AVG(Prc),C=COUNT(*),R=SUM(Qty·Prc)}(v0)
π_{PType,TMonth},SUM(R),WAVG(P,C)(σ_{PCategory='Foodstuffs'}(v23))
q1
q2
q3
v23
v12
v0
π_{PName,TMonth,SUM(Q),WAVG(P,C),SUM(C),SUM(R)}(v12)

Figure 5: Construction of the QVG for queries q_1, q_2, q_3.

In Figure 5, the steps of the `QVGBuilder` algorithm for the queries proposed in Section 1 are shown. View v_{13} does not appear in the final QVG since it is equivalent to v_{12} (case A). As to v_{23}, the search is restricted to arc (q_2,v_{12}) since a derivability relationship (case B) is found and the other can be inferred.

If we restrict to considering only the aggregation patterns, the QVG resulting from the algorithm degenerates to the MDred Lattice presented in [Bar97]. With reference to that approach, ours has a lower computation complexity (in terms of comparisons between expressions) since the derivability relationships expressed by the QVG are actively used to restrict the search to the sub-graph where the new vertex can be inserted. In fact, the complexity of `Merge` for adding the (i+1)-th query to the QVG reaches $O(2^i)$ only in the worst case, in which (1) the QVG already contains 2^i views and (2) all the candidate views solving both q_{i+1} and each element in the power set of the i queries in the QVG must be created, i.e., relationship D holds for all the views in the QVG. In the best case the QVG degenerates into a path, and the complexity drops to O(1). In the other cases, the complexity depends on the specific characteristics of the queries and on the order in which they are merged. The algorithm is efficient since the search space is pruned by neglecting all the vertices for which a derivability relationship with the new vertex to be positioned can be inferred from the graph structure.

6 Experimental Tests

We tested our approach on the TPC-D benchmark; the workload includes four standard TPC-D queries and three additional NGPSJ queries. In order to demonstrate the effectiveness of our approach, we compared the results obtained by applying the same materialization heuristics [Bar97] to our QVG and to the lattice defined in [Bar97] (considering the same global space constraint). The cost function adopted expresses the total number of disk pages which must be accessed in order to solve each query, taking both the view size and the query selectivity into account. The results are summarized in Figure 6.

The low execution cost for the QVG-based approach is mainly due to the fact that, since NPSGJ views can express the nesting of aggregation operators, queries can be executed on views aggregated at a “higher” pattern. Besides, since NPSGJ views are more closely tailored on queries than classical views, more NPSGJ views typically fit the same space constraint. These considerations are further supported by observing how, in Figure 6, materializing the maximum number of useful NPSGJ views (7, one for each query in the workload) requires only 1.1 GB, and entails a lower cost than materializing the same number of classical views.

7 Conclusions

In this paper we have presented an original approach to view materialization in which NGPSJ expressions are used to model both queries and candidate views. An algorithm for selecting the minimal set of candidate views by incrementally inserting queries into a directed acyclic graph has been proposed; the algorithm is efficient since, every time a query is inserted, only a portion of the graph is visited.

Our approach relies on the existence of a method for comparing two relational algebra expressions and determining their ancestor [Nut98]. The procedure for comparing two NGPSJ expressions can be seen as an iteration of the comparison between two GPSJ expressions, which was proposed in [Gup95]. Comparison proceeds from inside the two nested expressions, considering at each step two units having the form $\pi(\sigma(R))$; while in [Gup95] R denotes a set of joins, here it denotes the result of the NGPSJ expressions analyzed up to the previous step. Since the problem of comparing two GPSJ expression is undecidable [Ros98b], the ancestor determined will sometimes be sub-optimal. Our future work will focus on devising a comparison algorithm specifically oriented to NGPSJ expressions, by taking their peculiarities into account.

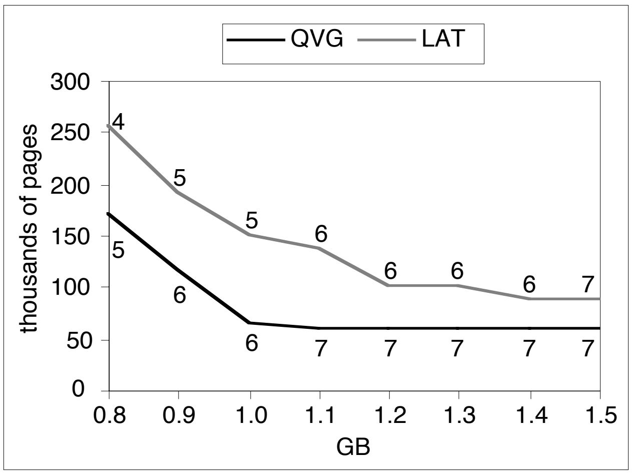


Figure 6: Execution costs for the QVG-based and the lattice-based approaches, on a sample workload, in function of the global space constraint. Besides each value point, the number of views materialized.

References

[Aki99] M. Akinde and M. Böhlen. Constructing GPSJ view graphs. *Proc. Int. Workshop on Design and Management of Data Warehouses*, Heidelberg, Germany, 1999.

[Bar97] E. Baralis, S. Paraboschi and E. Teniente. Materialized view selection in multidimensional database. *Proc. 23rd Int. Conf. on Very Large Data Bases*, Athens, Greece, pp. 156-165, 1997.

[Cab99] L. Cabibbo and R. Torlone. A framework for the investigation of aggregate functions in database queries. *Proc. Int. Conf. on Database Theory*, Jerusalem, Israel, 1999.

[Coh99] S. Cohen, W. Nutt and A. Serebrenik. Algorithms for rewriting aggregate queries using views. *Proc. Int. Workshop on Design and Management of Data Warehouses*, Heidelberg, Germany, 1999.

[Gol00] M. Golfarelli, D. Maio and S. Rizzi. Applying Vertical Fragmentation Techniques in Logical Design of Multidimensional Databases. To appear in ***Proc. 2nd Int. Conf. on Data Warehousing and Knowledge Discovery***, Greenwich, 2000.

[Gra95] J. Gray, A. Bosworth, A. Lyman and H. Pirahesh. Data-Cube: a relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Technical Report* MSR-TR-95-22, Microsoft Research, 1995.

[Gup95] A. Gupta, V. Harinarayan and D. Quass. Aggregate-query processing in data-warehousing environments. *Proc. 21st Int. Conf. on Very Large Data Bases*, Zurich, Switzerland, 1995.

[Gup97] H. Gupta. Selection of views to materialize in a data warehouse. *Proc. Int. Conf. on Database Theory*, Delphi, Greece, pp. 98-112, 1997.

[Gup99] H. Gupta and I.S. Mumick. Selection of views to materialize under a maintenance cost constraint. *Proc. Int. Conf. on Database Theory*, Jerusalem, Israel, 1999.

[Kim96] R. Kimball. *The data warehouse toolkit*. John Wiley & Sons, 1996.

[Len97] H.J. Lenz and A. Shoshani. Summarizability in OLAP and statistical databases. *Proc. Statistical and Scientific Database Management,* Washington, 1997.

[Nut98] W. Nutt, Y. Sagiv and S. Shurin. Deciding equivalences among aggregate queries. *Proc. 17th Symposium on Principles of Database Systems*, 1998.

[Ros98] K. A. Ross, D. Srivastava, and D. Chatziantoniou. Complex aggregation at multiple granularities. *Proc. Int. Conf. on Extending Database Technology*, 1998.

[Ros98b] K. A. Ross, D. Srivastava, P. J. Stuckey and S. Sudarshan. Foundations of aggregation constraints. *Theoretical Computer Science*, 193(1-2):149-179, 1998.

[Sho96] A. Shoshani. OLAP and statistical databases: similarities and differences. *Proc. Int. Conf. on Information and Knowledge Management*, Rockville, Maryland, 1996.

[The97] D. Theodoratos and T. Sellis. Data warehouse configuration. *Proc. 23rd Int. Conf. on Very Large Data Bases*, Athens, Greece, pp. 126-135, 1997.

[Yan95] W.P. Yan and P. Larson. Eager and lazy aggregation. *Proc. 21st Int. Conf. on Very Large Data Bases*, Zurich, Switzerland, pp. 345-357, 1995.