# View Materialization vs. Indexing: Balancing Space Constraints in Data Warehouse Design

Stefano Rizzi and Ettore Saltarelli

DEIS - University of Bologna
Viale Risorgimento, 2
40136 Bologna - Italy
{srizzi, esaltarelli}@deis.unibo.it

**Abstract.** View materialization and indexing are the most effective techniques adopted in data warehouses to improve query performance. Since both materialization and indexing algorithms are driven by a constraint on the disk space made available for each, the designer would greatly benefit from being enabled to determine a priori which fractions of the global space available must be devoted to views and indexes, respectively, in order to optimally tune performances. In this paper we first present a comparative evaluation of the benefit (saving per disk page) brought by view materialization and indexing for a single query expressed on a star scheme. Then, we face the problem of determining an effective trade-off between the two space fractions for the core workload of the warehouse. Some experimental results are reported, which prove that the estimated trade-off is satisfactorily near to the optimal one.

## 1 Introduction

Among the techniques adopted in relational implementations of data warehouses (DW's) to improve query performance, view materialization and indexing are surely the most effective ones.

View materialization consists in precomputing and storing a set of partial aggregates useful to solve, with decreased cost, frequent and/or crucial queries within the workload. Several approaches to view materialization were devised in the literature (see [16] for a survey and a general statement of the problem), mostly aimed at determining the subset of views which allows to minimize the execution cost of a given workload under a given space constraint.

The other technique universally adopted to reduce query execution costs is, of course, indexing. Though a number of papers were devoted to proposing or adapting indexing techniques for DW's [12–14], only a few works focus on the selection of indexes for DW's. Since indexes may be built on any view materialized, in order to reduce the problem complexity materialization and indexing are often faced separately, meaning that the optimal indexing scheme is chosen, under a space constraint, *after* the set of views to be materialized has been determined [4].

It is apparent that, in most approaches, both materialization and indexing are driven by a rough indication of the disk space made available for each. Since the warehouse administrator can reasonably constrain only the *global* space available, $S$, as stated in [1] the designer must be capable of estimating which fractions of $S$ should be devoted to views and indexes, $S_V$ and $S_X$ respectively, in order to optimally tune performances. Due to the high computation complexity of the algorithms for selecting the optimal sets of views and indexes, it is highly desirable that a good balancing of $S_V$ and $S_X$ is decided a priori, since a trial-and-error approach would require to execute the optimization algorithms several times under different space constraints. To this end, since the benefit of both materialization and indexing strongly depends on the characteristics of the queries formulated on the DW, we believe that the workload must be necessarily taken into account.

Let the workload be composed of GPSJ queries, typical of OLAP applications: essentially, queries consisting of a selection and an aggregation operated over a join. The key factors which impact the optimization benefit for a GPSJ query are its aggregation level (defined by its grouping set) and its selectivity (defined by the HAVING/WHERE clause). It is reasonable to expect that materialization will offer great advantage for queries with coarse aggregation, which compute a few groups out of a huge number of tuples, since accessing a small view is much cheaper than accessing a huge table. On the other hand, indexes will give their best when solving queries with high selectivity, which select only a few tuples, since accessing lots of useless tuples will be avoided. Thus, intuitively, queries with fine aggregation and high selectivity encourage indexing, while queries with coarse aggregation and low selectivity encourage materialization. On the other hand, as sketched in Figure 1, it is difficult to predict even qualitatively which of the two optimization techniques will fit best for queries falling outside these two regions.

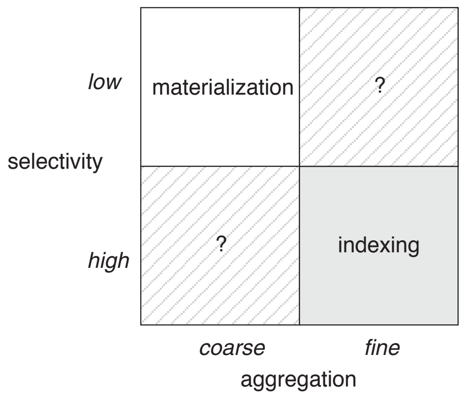

**Fig. 1.** Recommended optimization techniques depending on the query selectivity and aggregation level

In this paper we face the problem of determining a priori an effective trade-off between $S_V$ and $S_X$ based on the core workload of the DW. The approach we follow is rooted on the accurate estimation of the saving per disk page (*benefit*) of materialization and indexing for a single GPSJ query expressed on a star scheme; the estimates are based on a detailed analysis of the optimization strategies adopted by a commercial DBMS. The benefits computed for all the queries in the workload are then compared to heuristically estimate the space fractions $S_V$ and $S_X$.

The paper is organized as follows. Section 2 briefly discusses the related literature, while Section 3 introduces the necessary background on star schemes, GPSJ queries, and views. Sections 4 and 5, respectively, outline a model for query execution plans and describe the cost model adopted. Section 6 discusses and compares the benefits of view materialization and indexing in function of the query characteristics, while Section 7 shows how these benefits can be used to heuristically estimate an effective space balancing to be used for design. Finally, Section 8 draws the conclusions.

## 2 Related Work

A huge work on view materialization has been done during the last few years, starting with [9] in which a lattice was used for the first time to capture the relationships between aggregate views. Other basic approaches are described in [2, 6, 8]; in all these cases materialization is workload-driven, and a space constraint is considered. In all these approaches index selection is not considered, except in [6] which adopts a fixed indexing scheme.

In [4], a heuristic algorithm for selecting tid-list and bitmap indexes under a space constraint is proposed; it is assumed that views have been previously selected. In [7] the problem of simultaneously choosing views and B$^+$-tree indexes for a workload given a global constraint on the space available is investigated; the "two-steps" approach (first materialize views and then choose indexes) is criticized, but no criteria for balancing space between views and indexes are proposed. In [11] a set of heuristic criteria for selecting views and indexes is proposed; the problem of space trade-off is tackled, but no really useful conclusion is offered.

## 3 Background

### 3.1 Star Scheme and Working Example

Relational implementations of DW's are typically based on *star schemes*. The star scheme for a fact of interest is composed by a set of $n$ denormalized *dimension tables* $DT_1^{(0)}, \ldots, DT_n^{(0)}$, one for each *dimension* of analysis, and a *fact table* $FT^{(0)}$ whose primary key is obtained by composing the $n$ foreign keys referencing the dimension tables. The fact table also includes one non-key attribute for each *measure* which quantitatively describes the fact. Each dimension table typically models, besides a dimension, a *hierarchy* of attributes functionally determined by the dimension itself: thus, it includes non-key attributes for the dimension and for the other attributes of the hierarchy; the key is typically a surrogate generated by the DBMS.

**Example 1** The working example adopted in this paper is derived from the well-known TPC-H benchmark [15]; its star scheme is as follows:

PART (<u>PartId</u>, Part, Brand, MFGR, Type, Container, Size)
SUPPLIER (<u>SupplierId</u>, Supplier, SNation, SRegion)
ORDER (<u>OrderId</u>, Order, ODate, OMonth, OYear, Customer)
LINEITEM (<u>PartId</u>, <u>SupplierId</u>, <u>OrderId</u>,
Qty, ExtPrice, Discount, DiscPrice, UnitPrice, Tax)

where LINEITEM is the fact table and the others are dimension tables. Note that dimension tables are denormalized: for instance, in SUPPLIER, the supplier nation SNation functionally determines his region SRegion. □

## 3.2 The Workload

The workloads we consider are sets of GPSJ queries. A GPSJ (Generalized Projection-Selection-Join) query $q$ is a generalized projection over a selection over a join [5]; as such, it may be expressed in relational algebra over a star scheme as follows:

$$q = \pi_{G,M}\sigma_{Pred}(FT^{(0)} \bowtie DT_1^{(0)} \bowtie \ldots \bowtie DT_n^{(0)})$$

where $Pred$ is a conjunction of simple range predicates on dimension table attributes, $G$ is a set of dimension table attributes (*grouping set*), and $M$ is a set of aggregated measures each defined by applying an aggregation operator to a measure in $FT^{(0)}$. Generalized projection $\pi_{G,M}$ is an extension of duplicate eliminating projection [5]; from the SQL point of view, it corresponds to grouping by the attributes in $G$ and inserting $G, M$ in the selection clause.

**Example 2** A possible GPSJ query on the LINEITEM scheme is the one which returns the total quantity ordered during 2001 and the average unit price for each type of product and each supplier nation in the western region:

$$\pi_{\mathsf{Type},\mathsf{SNation},\mathsf{OYear},SUM(\mathsf{Qty}),AVG(\mathsf{UnitPrice})}\ \sigma_{\mathsf{SRegion}='West'\ AND\ \mathsf{OYear}='2001'}\ (R)$$

where $R$ = PART $\bowtie$ SUPPLIER $\bowtie$ ORDER $\bowtie$ LINEITEM. The equivalent SQL formulation is as follows:

```
SELECT P.Type, S.SNation, O.OYear, SUM(L.Qty), AVG(L.UnitPrice)
FROM LINEITEM AS L, PART AS P, SUPPLIER AS S, ORDER AS O
WHERE L.PartId=P.PartId
AND    L.SupplierId=S.SupplierId
AND    L.OrderId=O.OrderId
AND    S.SRegion='West' AND O.OYear='2001'
GROUP BY P.Type, S.SNation, O.OYear  □
```

We will assume that all selection predicates are *external*, i.e. that they are formulated on attributes functionally determined by an attribute in the grouping set $G$.[1] Besides, we will assume for simplicity that at most one selection predicate is formulated on each dimension table. A dimension table (or, equivalently, a hierarchy) on which $q$ formulates a predicate is said to be *conditioned* in $q$.

### 3.3 Views

Materializing a view from the base fact table $FT^{(0)}$ may be seen as consolidating the result of a query. Here we will only consider views consolidated from GPSJ queries in which no selection predicate is formulated and the measures returned are all those in $FT^{(0)}$, each aggregated by the most appropriate aggregation operator. Thus, each view is fully characterized by its grouping set; the base fact table $FT^{(0)}$ can be seen as a particular case of a view, whose grouping set $G_0$ is the set of the dimensions of analysis ($G_0 =${Part,Supplier,Order} in the LINEITEM example).

View materialization obviously involves a modification of the DW logical schema. Among the different alternatives proposed in the literature and in the practice, in this paper we adopt the variant of the classic star scheme in which one separate fact table is created for each materialized view and a separate dimension table is created for each attribute belonging to the grouping set of at least one view. Thus, view $FT^{(G)}$ with grouping set $G = \{a_1, \ldots, a_{|G|}\}$ will be associated to dimension tables $DT_1^{(G)}, \ldots, DT_{|G|}^{(G)}$ where $DT_i^{(G)}$ contains one surrogate key and a field for each attribute functionally dependent on $a_i$, including $a_i$ itself.

**Example 3** Materializing the view with grouping set {Part,OMonth} means enriching the LINEITEM scheme with the following tables:

OMONTH (<u>OMonthId</u>, OMonth, OYear, Customer)
LINEITEM1 (<u>PartId</u>, <u>OMonthId</u>, SumQty, SumExtPrice, SumDiscount, SumDiscPrice, AvgUnitPrice, SumTax) □

Materializing view $FT^{(G)}$ brings benefit to all the queries whose grouping set is "coarser" or equal to $G$: in fact, they all can be solved with reduced costs by rewriting them on $FT^{(G)}$ instead of $FT^{(0)}$. The notion of "coarseness" of a grouping set is formally described as a partial ordering over the (exponential) set of possible grouping sets, represented by the well-known *multidimensional lattice* proposed in [9].

## 4 Modeling Query Execution Plans

Evaluating the benefit of materializing a view or creating an index for a single query requires to estimate the costs for executing that query when the

[1] A predicate which does not satisfy this requirement can be easily made external by refining the grouping set of the query.

view/index is absent and when it is present. Thus, a reference model for query execution plans must be considered. The execution plans considered in this work are derived from the rule-based optimizer model described in [4], which was determined by carrying out a black-box analysis on the optimizer of Red Brick 6.0 [10].

Relational DBMSs necessarily require that an index is built at least on the primary key of each table; thus we will assume that, for each view materialized (including the base fact table), indexes are always built on the primary key of both the fact table and the dimension tables. Besides these primary indexes, other secondary indexes may optionally be built, if it is convenient, on non-key attributes of dimension/fact tables. In this work, indexes built on numerical measures of the fact table are not considered for simplicity.

A query execution plan is a sequence of elementary operators, each modeling a function carried out by the DBMS on either tables or indexes. According to the rule-based model adopted in [4], the execution plan for query $q$ in presence of B$^+$-tree and bitmap indexes is mainly determined by the number $c$ of dimension tables conditioned in $q$:

- If $c = 0$, the fact table is sequentially scanned then joined with all the dimension tables involved in $q$ through a nested-loop on their primary key indexes.
- If $c = 1$, the conditioned dimension table is accessed by the index on the conditioned attribute if such index has been created, by sequential scan otherwise. The join with the fact table is based on nested-loop if the fact table is indexed on the corresponding foreign key, on hybrid hash otherwise. The result of the join is then joined with all the other dimension tables requested in output.
- If $c > 1$, each conditioned dimension table is joined separately with the fact table. A conditioned dimension table is accessed by the index on the conditioned attribute if such index has been created, by sequential scan otherwise. The join with the fact table is based on nested-loop if the fact table is indexed on the corresponding foreign key, on hybrid hash otherwise. The tid sets obtained from the different conditioned dimension tables are then intersected, and the corresponding tuples of the fact table are accessed. Finally, the result is joined with all the dimension tables requested in output.

Of course, after the join has been completed, the grouping is executed.

**Example 4** The graphical representation of two possible execution plans on the base fact table LINEITEM for the query in Example 2 are depicted in Figure 2 (the group-by operator is not reported). In the first one (top of figure), only primary indexes have been created; thus, conditioned dimension tables are accessed by sequential scan and joined with the fact table by hybrid hash. In the second plan (bottom of figure), indexes on the conditioned attribute SUPPLIER.SRegion and ORDER.OYear, as well as indexes on the foreign keys LINEITEM.SupplierId and LINEITEM.OrderId, have been built; thus, conditioned dimension tables are accessed via index and joined by nested loop with the fact table, accessed via index as well. The rest of the plan does not change in the two cases. □

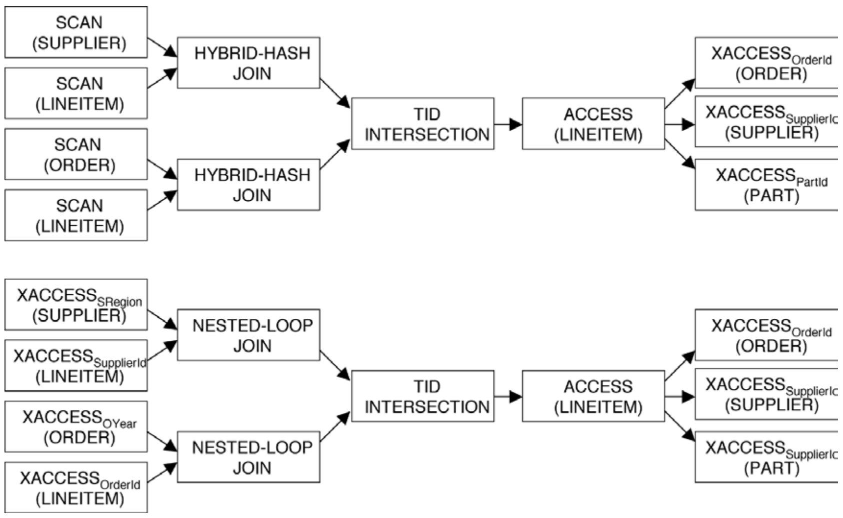

**Fig. 2.** Execution plans for the query in Example 2 (ACCESS denotes accessing a table via tuple identifiers, XACCESS via an index)

## 5 The Cost Model

Let $q$ be a query over the star scheme composed of tables $FT^{(0)}$, $DT_1^{(0)}$, ..., $DT_n^{(0)}$, and let $G$ and $G_0$ be, respectively, the grouping sets of $q$ and $FT^{(0)}$ ($|G_0| = n$). As detailed in Section 6, our approach to evaluate the benefits of materialization and indexing for $q$ is based on the comparison between the costs for executing $q$ over three notable configurations of the logical/physical scheme:

1. No view is materialized and only primary indexes are built on the base fact table $FT^{(0)}$ and on each $DT_i^{(0)}$. In this case $q$ is solved by hybrid-hash join on $FT^{(0)}$ (*reference plan*);
2. The view $FT^{(G)}$ yielding the least cost for $q$, i.e. the view having grouping set $G$, is materialized; only primary indexes are built on $FT^{(G)}$ and on each $DT_i^{(G)}$. In this case $q$ is solved by hybrid-hash join on $FT^{(G)}$ (*view plan*);
3. No view is materialized but, besides primary indexes, also all potentially useful secondary indexes on $FT^{(0)}$ (those on foreign keys) and on each $DT_i^{(0)}$ (those on conditioned attributes) are built. In this case $q$ is solved by nested-loop join on $FT^{(0)}$ (*index plan*).

Note that reference and view plans are structured as in Figure 2 at the top, while index plans are structured as in Figure 2 at the bottom.

The notation used in defining the cost model for these plans is summarized in Table 1; the simple formulae adopted for estimating the numbers of table pages and index leaves are omitted for brevity. According to the execution model described in Section 4 and assuming that hierarchies are ordered in such a way that the conditioned ones come first, the cost in disk pages of the reference, view, and index plans for $q$ (*not* comprising the cost of group-by) may be estimated respectively as follows:

$$cost_{ref}(q) = \begin{cases} NPF(G_0) + \\ \quad \sum_{i=1}^{|G|}(NPD(a_i^{(G_0)}) + NLD(a_i^{(G_0)})) & \text{if } c = 0 \\ NPF(G_0) + NPD(a_1^{(G_0)}) + \\ \quad f \cdot \sum_{i=2}^{|G|}(NPD(a_i^{(G_0)}) + NLD(a_i^{(G_0)})) & \text{if } c = 1 \\ (c+f) \cdot NPF(G_0) + \sum_{i=1}^{c} NPD(a_i^{(G_0)}) + \\ \quad f \cdot \sum_{i=1}^{|G|}(NPD(a_i^{(G_0)}) + NLD(a_i^{(G_0)})) & \text{if } c > 1 \end{cases} \quad (1)$$

$$cost_{view}(q) = \begin{cases} NPF(G) + \\ \quad \sum_{i=1}^{|G|}(NPD(a_i^{(G)}) + NLD(a_i^{(G)})) & \text{if } c = 0 \\ NPF(G) + NPD(a_1^{(G)}) + \\ \quad f \cdot \sum_{i=2}^{|G|}(NPD(a_i^{(G)}) + NLD(a_i^{(G)})) & \text{if } c = 1 \\ (c+f) \cdot NPF(G) + \sum_{i=1}^{c} NPD(a_i^{(G)}) + \\ \quad f \cdot \sum_{i=1}^{|G|}(NPD(a_i^{(G)}) + NLD(a_i^{(G)})) & \text{if } c > 1 \end{cases} \quad (2)$$

$$cost_{ix}(q) = \begin{cases} cost_{ref}(q) & \text{if } c = 0 \\ f \cdot (NL(a_1^{(cond)}) + NPD(a_1^{(G_0)}) + NLS + NPF(G_0) + \\ \quad \sum_{i=2}^{|G|}(NPD(a_i^{(G_0)}) + NLD(a_i^{(G_0)}))) & \text{if } c = 1 \\ \sum_{i=1}^{c} f_i \cdot (NL(a_i^{(cond)}) + NPD(a_i^{(G_0)}) + NLS) + \\ \quad f_1 \cdot NPF(G_0) + \\ \quad f \cdot \sum_{i=1}^{|G|}(NPD(a_i^{(G_0)}) + NLD(a_i^{(G_0)})) & \text{if } c > 1 \end{cases} \quad (3)$$

Some considerations on the assumptions and approximations introduced:

– All fact and dimension tables are assumed to be ordered on their primary keys.
– All selection predicates are assumed to select a continuous range of values.
– The cost for tid-intersection is neglected; when estimating the cost of an index access, the cost for descending the tree is neglected. The cost of group-by is not considered here: the reason for this will be made clear in Section 6.
– For simplicity, when estimating the number of accesses to the fact table in the presence of selection predicates, we assume that the tuples to be read are adjacent; a more precise evaluation would require to use the Cardenas formula [3].

**Table 1.** Notation for the cost model

| **Query** | |
|---|---|
| $G$ | grouping set |
| $\|G\|$ | number of attributes in $G$ |
| $a_i^{(G)}$ | attribute of the $i$-th hierarchy in $G$, $1 \leq i \leq \|G\|$ |
| $a_i^{(G_0)}$ | attribute of the $i$-th hierarchy in $G_0$, $1 \leq i \leq \|G\|$ |
| $c$ | number of conditioned hierarchies $(0 \leq c \leq \|G\|)$ |
| $a_i^{(cond)}$ | conditioned attribute in the $i$-th hierarchy, $1 \leq i \leq c$ |
| $f_i$ | selectivity of the selection predicate on the $i$-th hierarchy, $1 \leq i \leq c$ $(0 \leq f_i \leq 1$, $f_i = 1$ if not conditioned) |
| $f$ | global selectivity $(f = \prod_{i=1}^{c} f_i)$ |
| **Tables** | |
| $card(G)$ | cardinality of $FT^{(G)}$ |
| $NPF(G)$ | number of pages of $FT^{(G)}$ |
| $NPD(a_i^{(G)})$ | number of pages of $DT_i^{(G)}$ |
| **Indexes** | |
| $NL(a_i^{(cond)})$ | number of leaves of index on attribute $a_i^{(cond)}$ over $DT_i^{(G_0)}$ |
| $NLD(a_i^{(G)})$ | number of leaves of primary index on $DT_i^{(G)}$ |
| $NLF(G)$ | number of leaves of primary index of $FT^{(G)}$ |
| $NLS$ | number of leaves of secondary index on a foreign key of $FT^{(0)}$ |

These formulae have been validated by comparing the costs they estimate with the corresponding costs measured on Red Brick for different values of the parameters; the error is always less then 10%.

The results presented in this section are computed on a star scheme including 3 equal hierarchies of 10 attributes each; the domain cardinality is 1000. The base fact table includes $10^6$ tuples with 8 measures. Each measure and attributes takes, respectively, 8 and 20 bytes. Each disk page is 8 KB. Figure 3 shows how $cost_{view}(q)$ depends on the grouping set $G$ and on the global selectivity $f$ of $q$ for $c = 2$; the cost of the reference plan is very similar to $cost_{view}(q)$ for $G = G_0$. The cost of the view plan is linear in both $card(G)$ and $f$; this is due to the fact that its most significant term is $(c + f) \cdot NPF(G)$.

Figure 4 shows how $cost_{ix}(q)$ depends on $G$ and $f$ for $c = 2$. The cost does not significatively depend on $card(G)$, since $q$ is executed on $FT^{(0)}$; the dependence on $f$ is slightly parabolic.

## 6 Benefit Evaluation

In this section we will discuss and compare the benefits, meant as savings per disk page, of view materialization and indexing for different classes of queries.

We define the *benefit of materialization* for $q$ as the difference between the total costs for executing $q$ on $FT^{(0)}$ and on $FT^{(G)}$ with only primary indexes

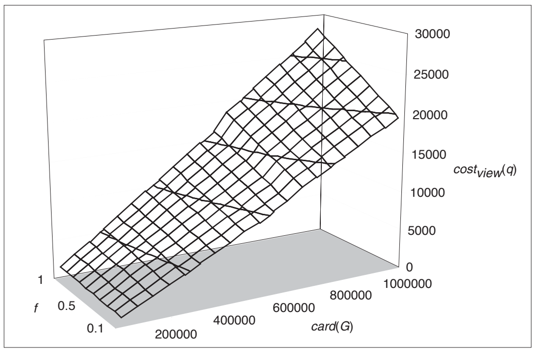

**Fig. 3.** Cost of the view plan for $c = 2$ (expressed in disk pages)

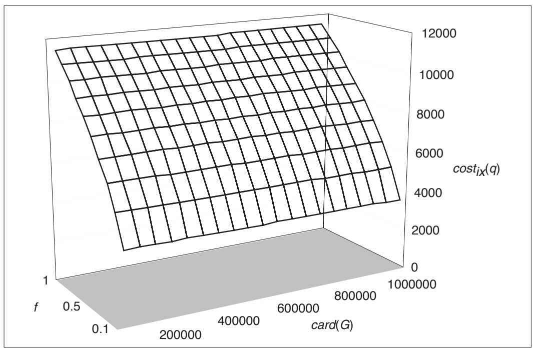

**Fig. 4.** Cost of the index plan for $c = 2$

built, divided by the space overhead of materialization:

$$bf_V(q) = \frac{cost_{ref}(q) + costGB_{ref}(q) - (cost_{view}(q) + costGB_{view}(q))}{space_{view}(q)} \tag{4}$$

where $costGB_{ref}(q)$ and $costGB_{view}(q)$ are the costs of grouping for the reference and the view plans, while

$$space_{view}(q) = NPF(G) + NLF(G) + \sum_{a \in G} (NPD(a) + NLD(a)) \tag{5}$$

By definition, for a query $q$ whose grouping set is $G^{(0)}$ it is $space_{view}(q) = 0$ and $bf_V(q) = 0$.

As to grouping, consistently with the Red Brick implementation, we assume that a hash-based algorithm is used. The experimental tests revealed that, while the cost of grouping depends substantially on the number of groups in output, it depends only marginally on the number of tuples to be grouped. Thus, since the number of groups is the same for both plans (it only depends on the grouping set

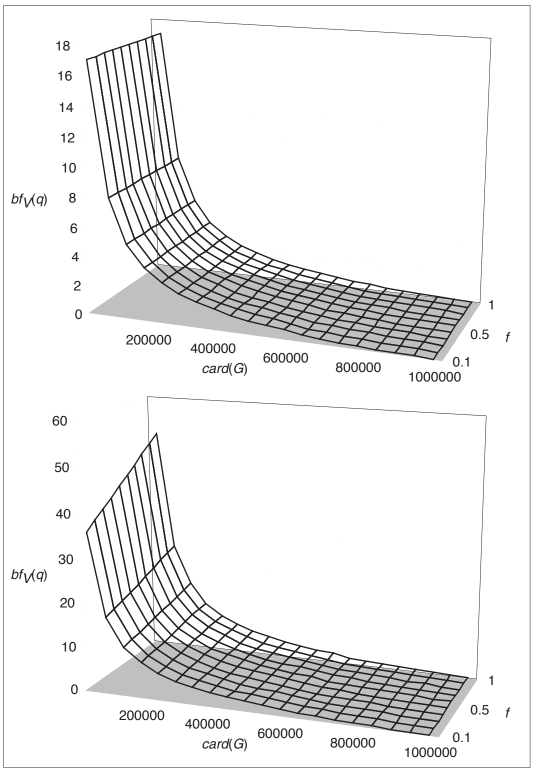

**Fig. 5.** Benefit of materialization for $c = 1$ (top) and $c = 2$ (bottom)

and on the selectivity of $q$), we will assume for simplicity that $costGB_{ref}(q) = costGB_{view}(q)$. [2]

Figure 5 shows the benefit of materialization for $c = 1$ and $c = 2$ (case $c = 0$ is not shown since very similar to $c = 1$). While in the first case the benefit is largely independent of $f$ [3], in the second it varies linearly with it. The dependence on $card(G)$ is always hyperbolic (roughly, it is $bf_V(q) = O(card(G_0)/card(G))$).

We define the *benefit of indexing* for $q$ as the difference between the costs for executing $q$ on $FT^{(0)}$ when only primary indexes are built and when all potentially useful indexes are built, divided by the space overhead of indexing; the group-by cost is not considered since it is the same for both the reference

[2] The error introduced is obviously higher for low values of $card(G)$, since the difference between the number of tuples to be grouped in the reference and the view plans is more relevant. On the other hand, in this case the benefit is high since the view on $G$ is very small. Thus, a possible way of taking the group-by cost into account would be to multiply the benefit of materialization by a corrective factor greater than 1.

[3] This is due to the fact that no indexes are built and hybrid-hash join is used. The main term determining the cost is that related to the sequential scan of the fact table, which is independent of $f$.

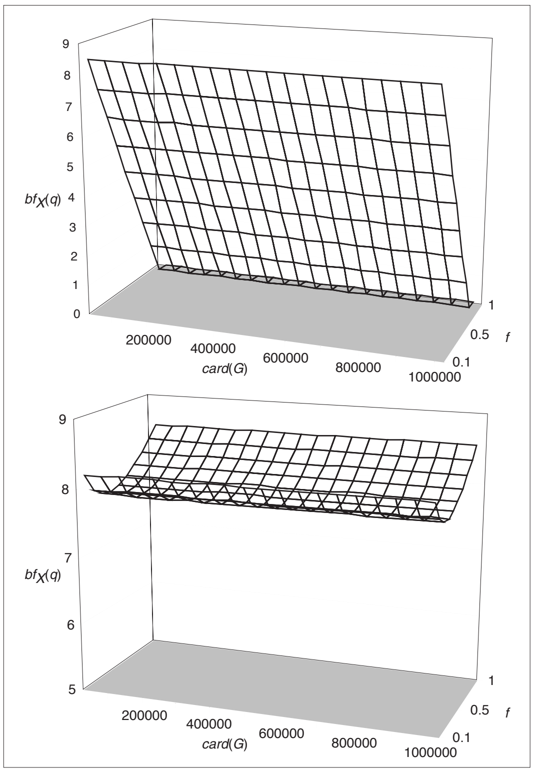

**Fig. 6.** Benefit of indexing for $c = 1$ (top) and $c = 2$ (bottom)

and the index plans (both the tuples in input and the groups in output are the same for the two plans):

$$bf_X(q) = \frac{cost_{ref}(q) - cost_{ix}(q)}{space_{ix}(q)} \tag{6}$$

where [4]

$$space_{ix}(q) = \sum_{i=1}^{c} NL(a_i^{(cond)}) + c \cdot NLS \tag{7}$$

Figure 6 shows the benefit of indexing for $c = 1$ and $c = 2$ (for $c = 0$ it is obviously $bf_X(q) = 0$). While in the first case the benefit is linear in $f$, in the second it is nearly constant; in both cases it does not depend on $card(G)$.

It is now very interesting to compare the two benefits. Figure 7 shows the contour lines of the surface representing $bf_V(q) - bf_X(q)$ in function of $card(G)$ and $f$; the white and the grey areas correspond, respectively, to $bf_V(q) > bf_X(q)$

[4] Assuming that, for indexing foreign keys of fact tables, $B^+$-trees are always used.

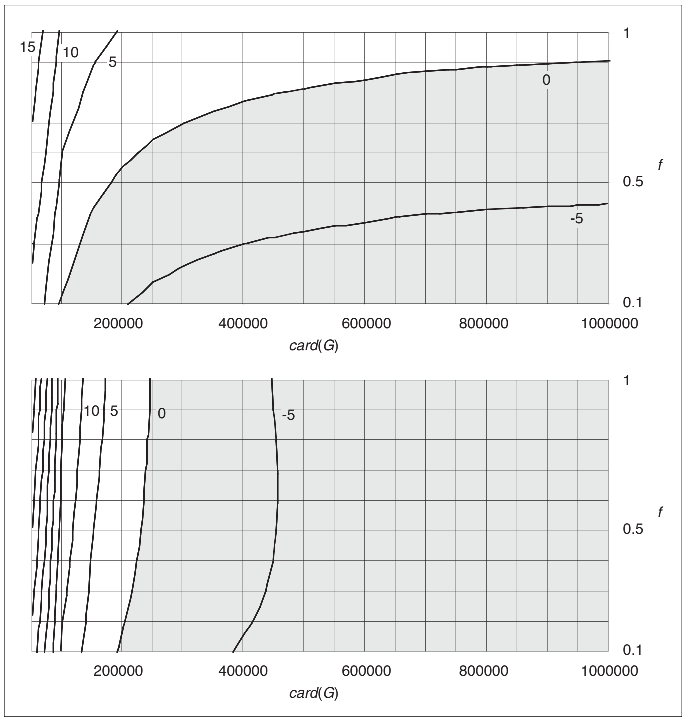

**Fig. 7.** Contour lines for $bf_V(q) - bf_X(q)$ in case $c = 1$ (top) and $c = 2$ (bottom)

and $bf_V(q) < bf_X(q)$. Compare this diagram with the qualitative one sketched in Figure 1: the influence of selectivity is less relevant than we expected; besides, though the benefit of materialization reaches higher values than indexing, the convenience area for indexing is much larger.

## 7 Balancing Space Constraints

Given a constraint $S$ (expressed in disk pages) on the total disk space available for optimization, in this section we propose an approach to estimate the fractions of $S$ to be devoted to materialization and indexing, $S_V$ and $S_X$ respectively, with reference to a given workload $W$. We will assume that the space $S_V^{(0)}$, required to store the base fact table $FT^{(0)}$ together with its dimension tables and primary indexes, is *not* included in $S$; thus, even case $S = 0$ corresponds to a feasible constraint (no optimization at all). On the other hand, $S_V$ and $S_X$ are to be meant, respectively, as the space to be allocated for views (not including the base fact table) plus their related dimension tables and primary indexes, and the space to be allocated for indexes on foreign keys of fact tables (including the base one) and non-key attributes of dimension tables.

We will first introduce two extreme optimization situations, which we will call full materialization and full indexing:

- Given a workload $W$, *full materialization* occurs when, for each query $q \in W$, an ad hoc view $FT^{(G)}$ (where $G$ is the grouping set of $q$) is materialized. Letting $S_V^{full} = \sum_{q \in W} space_{view}(q)$, the total disk space required for full materialization is $S_V^{(0)} + S_V^{full}$.
- A materialized view is *fully indexed* when, for each $q \in W$ to be executed on it, all the useful indexes have been created. Given a set of views taking space $S_V$, we will denote with $(S_V)_X^{full}$ the total space for full indexing all of them plus the base fact table; thus, $(0)_X^{full}$ is the space to fully index the base fact table only, while $(S_V^{full})_X^{full}$ is the space to fully index also all the views in case of full materialization. We estimate $(S_V)_X^{full}$ by assuming that

$$\frac{S_V + S_V^{(0)}}{(S_V)_X^{full}} \approx \frac{S_V^{full} + S_V^{(0)}}{(S_V^{full})_X^{full}} \tag{8}$$

$$\Rightarrow \quad (S_V)_X^{full} \approx \frac{S_V + S_V^{(0)}}{S_V^{full} + S_V^{(0)}} \cdot (S_V^{full})_X^{full} \tag{9}$$

In the remainder we will assume that $S \leq S_V^{full} + (S_V^{full})_X^{full}$; if not, the space constraint is redundant and the space trade-off can be easily found.

As a first observation, an index may only be built on a view which has been materialized: thus, for a set of materialized views taking space $S_V$, $(S_V)_X^{full}$ is an upper bound to the space which can be realistically filled with indexes. For instance, given $S = 10000$ disk pages, let $S_V = 6000$ and $S_X = 4000$ define the space trade-off for a workload which encourages indexing. Now, suppose that fully indexing the views materialized within $S_V$ plus the base fact table requires only 1000 pages: 3000 pages uselessly reserved to indexes will be wasted, while they could have been more profitably used for materialization.

As a second observation, the space reserved for materialization should not overcome $S_V^{full}$. For instance, let $S = 10000$, $S_V = 8000$, and $S_X = 2000$ for a workload which encourages materialization. Now, suppose that full materialization requires only 6000 pages besides the base fact table: 2000 pages uselessly reserved to views will be wasted, while they could have been more profitably used for indexing.

As depicted in Figure 8, this situation can be summarized by constraining the feasible trade-off solutions, for each value of $S$, as follows:

$$S_V + S_X = S \tag{10}$$

$$0 \leq S_X \leq (S_V)_X^{full} \tag{11}$$

$$0 \leq S_V \leq S_V^{full} \tag{12}$$

Thus the grey area shown in Figure 8, delimited by

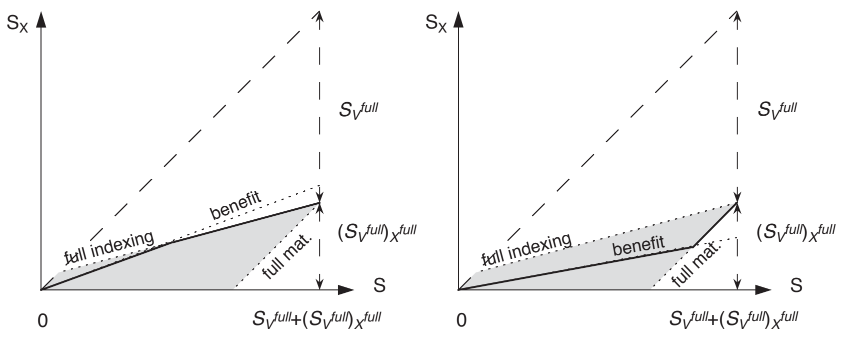

**Fig. 8.** Estimated values for $S_X$ in function of $S$ when the workload encourages indexing (left) and materialization (right)

$$S_X^{upper} = \min\left\{(S_V)_X^{full}, S\right\} = \min\left\{\frac{(S_V^{full})_X^{full} \cdot (S + S_V^{(0)})}{S_V^{full} + (S_V^{full})_X^{full} + S_V^{(0)}}, S\right\} \tag{13}$$

$$S_X^{lower} = \max\{0, S - S_V^{full}\} \tag{14}$$

represents the space of the feasible solutions ($S_X^{lower} \leq S_X^{upper}$ when $0 \leq S \leq S_V^{full} + (S_V^{full})_X^{full}$). Within such space, the optimal values for $S_V$ and $S_X$ are estimated by considering the ratio between the benefits of materialization and indexing, computed on the whole workload. For a given global constraint $S$ and with reference to workload $W = \{q_1, \ldots, q_p\}$, let

$$S_X^{bf} = \frac{S}{1 + \frac{\sum_{q_i \in W} bf_V(q_i)}{\sum_{q_i \in W} bf_X(q_i)}} \tag{15}$$

Then, the estimate for the optimal space trade-off is defined as follows:

$$S_X = \begin{cases} S_X^{lower}, & \text{if } S_X^{bf} < S_X^{lower} \\ S_X^{bf}, & \text{if } S_X^{lower} \leq S_X^{bf} \leq S_X^{lower} \\ S_X^{upper}, & \text{if } S_X^{bf} > S_X^{upper} \end{cases} \tag{16}$$

$$S_V = S - S_X \tag{17}$$

## 8 Experimental Tests and Conclusions

In this paper we have presented a comparative evaluation of the benefits brought by view materialization and indexing in DW's in function of the query characteristics. Then, we have proposed a heuristic approach to estimate, for a given workload and a global space constraint, the optimal trade-off between the space devoted to view materialization and that devoted to indexing.

The experimental tests presented in this section were conducted on the same star scheme used in Section 5. Three different workloads were considered, each including 25 queries: $W_1$, which encourages indexing, $W_2$, which encourages materialization, and $W_3$ in which queries are uniformly distributed in the space of $card(G)$ and $f$.

Figure 9 shows how the global cost of workload $W_3$ (expressed in disk pages) varies depending on the relative space amount devoted to materialization and indexing, for different values of $S$. Since each disk page takes 8 KB, the space constraint ranges approximatively between 100 MB and 1 GB. The algorithms used for view materialization and indexing are those proposed, respectively, in [2] and [4]. The cost for a non-optimal space trade-off may even be three times that of the optimal trade-off; the irregular shape of the curves is due to the sub-optimality introduced by the heuristic approaches to materialization and indexing.

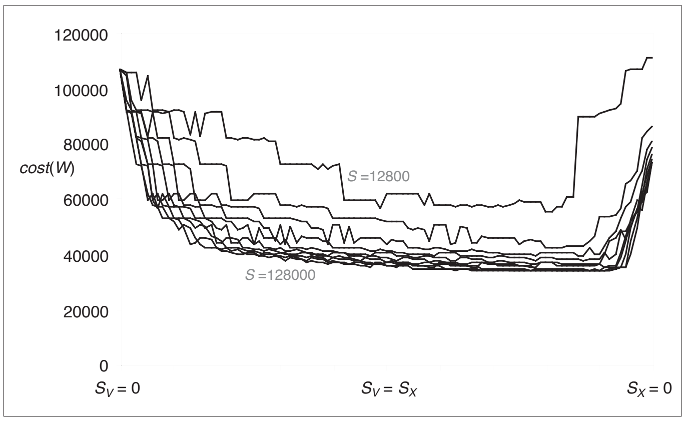

**Fig. 9.** Cost of workload $W_3$, as a function of the relative space amount devoted to materialization and indexing, for increasing values of $S$

Figure 10 compares the estimated and optimal values for $S_X$ in function of $S$ for all three workloads, emphasizing the relative position of the full indexing and the full materialization lines, as well as the values of $S_X^{bf}$ suggested by the ratio between the materialization and the indexing benefits.

Finally, Table 2 reports the percentage difference between the workload costs for the optimal solution and for the solution estimated by our approach. It is apparent that the costs measured when applying the estimated trade-off are very close to those yielded by the optimal trade-off, which demonstrates the utility of our approach.

Our future work on this topic will be aimed at overcoming the main limitations of our approach:

– The benefits of materialization and indexing are not really independent of each other, since only a view which has been materialized can be indexed. In our present approach this is considered through the thresholding mechanism

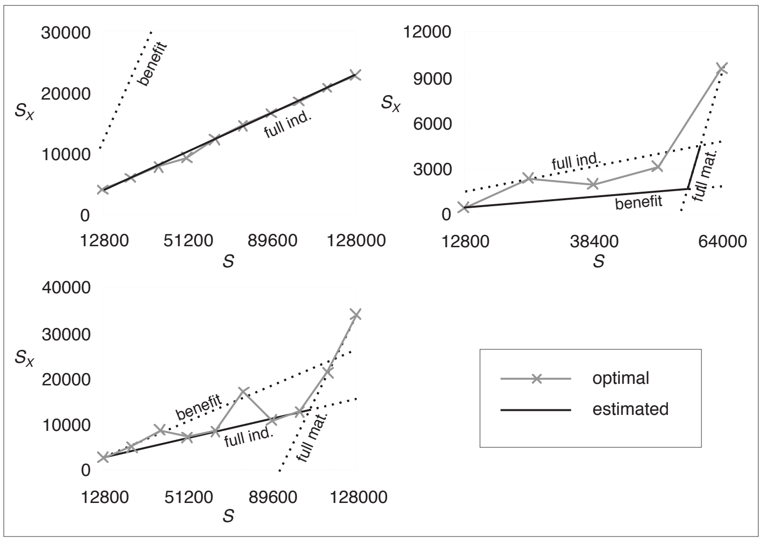

**Fig. 10.** Estimated and optimal values for $S_X$ in function of $S$ for $W_1$ (top left), $W_2$ (top right), and $W_3$ (bottom)

introduced by full indexing; we will try to directly model the dependence of indexing on materialization within the definition of benefit, instead.

- Presently, our benefit estimates are independent of whether the grouping sets of the queries are scattered or clustered within the multidimensional lattice. Intuitively, if grouping sets are very "near" to each other, the actual benefit of materialization is higher than our estimate, since one common view may dramatically decrease the cost for several queries.
- Rule-based optimizers present some well-known drawbacks due to the fact that they do not take query costs into account. Developing and adopting a detailed cost model also for a cost-based optimizer will widen the usability of our approach.
- In this approach we considered only the workload cost as an indicator of the quality of optimization, while also the maintenance costs of both views and indexes should be considered.

Another interesting evolution of our approach is related to the absence of a specific workload. In this case, $S_X^{bf}$ could be estimated by considering a uniform distribution of queries, i.e. by computing the integrals of $bf_V(q)$ and $bf_X(q)$ over the $card(G) \times f$ space.

## References

1. S. Agrawal, S. Chaudhuri, and V. Narasayya. Automated selection of materialized views and indexes for SQL databases. In *Proc. 26th VLDB*, pages 496–501, Cairo, Egypt, 2000.

**Table 2.** Percentage difference between the workload costs for the optimal and the estimated solutions. In parentheses the situations in which the space constraint is redundant

| $S$ | $W_1$ | $W_2$ | $W_3$ |
|---|---|---|---|
| 12800 | 0.0% | 0.1% | 0.0% |
| 25600 | 0.6% | 1.4% | 1.7% |
| 38400 | 0.5% | 0.2% | 1.0% |
| 51200 | 0.4% | 1.5% | 0.3% |
| 64000 | 0.0% | (0.0%) | 0.0% |
| 76800 | 0.0% | (0.0%) | 2.5% |
| 89600 | 0.0% | (0.0%) | 0.0% |
| 102400 | 0.0% | (0.0%) | 0.0% |
| 115200 | 0.0% | (0.0%) | (0.0%) |
| 128000 | 0.0% | (0.0%) | (0.0%) |

2. E. Baralis, S. Paraboschi, and E. Teniente. Materialized view selection in multidimensional database. In *Proc. 23rd VLDB*, pages 156–165, Athens, 1997.
3. A.F. Cardenas. Analysis and performance of inverted database structures. *Communications of the ACM*, 18(5):253–263, 1975.
4. M. Golfarelli, S. Rizzi, and E. Saltarelli. Index selection for data warehousing. In *Proc. DMDW*, Toronto, 2002.
5. A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *Proc. 21st VLDB*, Zurich, 1995.
6. H. Gupta. Selection of views to materialize in a data warehouse. In *Proc. ICDT*, pages 98–112, 1997.
7. H. Gupta, V. Harinarayan, A. Rajaraman, and J.D. Ullman. Index selection for OLAP. In *Proc. ICDE*, pages 208–219, 1997.
8. H. Gupta and I.S. Mumick. Selection of views to materialize under a maintenance cost constraint. In *Proc. ICDT*, Jerusalem, Israel., 1999.
9. V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *Proc. ACM SIGMOD Conf.*, Montreal, 1996.
10. Informix. *Informix Red Brick Decision Server administrator's guide, Version 6.0*, November 1999.
11. W.J. Labio, D. Quass, and B. Adelberg. Physical database design for data warehouses. In *Proc. ICDE*, pages 277–288, 1997.
12. V. Markl, F. Ramsak, and R. Pieringer. The TransBase HyperCube RDBMS: multi-dimensional indexing of relational tables. In *Proc. 17th ICDE*, Heidelberg, Germany, 2001.
13. P. O'Neil and G. Graefe. Multi-table joins through bitmapped join indices. *SIGMOD Record*, 24(3):8–11, 1995.
14. P. O'Neil and D. Quass. Improved query performance with variant indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Tucson, AZ, 1997.
15. M. Poess and C. Floyd. New tpc benchmarks for decision support and web commerce. *ACM SIGMOD Record*, 29(4), 2000.
16. D. Theodoratos and M. Bouzeghoub. A general framework for the view selection problem for data warehouse design and evolution. In *Proc. DOLAP*, McLean, 2000.