Optimizing Partition-Selection Scheme for Warehouse Aggregate Views[*]

C.I. Ezeife
School of Computer Science
University of Windsor
Windsor, Ontario
Canada N9B 3P4
cezeife@cs.uwindsor.ca
Tel: (519) 253-3000 ext 3012; FAX: (519) 973-7093.

Abstract

Business applications for decision support or online analytical processing (OLAP) are largely interested in discovering patterns in transactions data stored over time, on which executive decisions can be made. Data warehouses store such transactions data as integrated, historical and subject-oriented data which are made available for aggregate querying by business executives. Since volume of data in warehouse tables is huge, one technique for improving warehouse query response time is to pre-compute and store (materialize) all needed aggregate tables (or views).

Data cube provides a conceptual *n*-dimensional representation of 2^n aggregate views for a set of *n* groupby attributes or subjects in the main warehouse fact table. Storing all the 2^n data cube views may pose storage space problems as well as increase maintenance cost. Many algorithms have been proposed for selecting a set of views of the data cube most beneficial to materialize. One such algorithm is the Partition-Selection scheme which recommends storing a materialized view as a set of horizontal fragments of the view. The objective of this paper is to enhance the performance of this Partition-Selection scheme for views by defining an algorithmic component which recommends the best set of fragments of a materialized view that answers any warehouse aggregate query.

Keywords: Data Warehouse, Views, Fragmentation, Performance benefit

1. Introduction

A telephone company may want to generate statistics which displays the total monthly long distance bill by each of its customers for the past one or 2 years. From this statistics, it becomes easy to identify the customers who make the most long distance calls and these customers could be target for a new and specialized service package that may best benefit the customers and by finding a way to please the customers, the

[*] This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

company gains competitive advantage as few customers would be willing to switch to a new company. An emerging database technology that provides the means for business organizations to store and manage these types of snap shots of business data gathered over long periods of time and which integrates data from all distributed branches and departments is data warehousing. Widom in [Wi95] defines data warehousing system as a single data repository which integrates information from different data sources like relational databases, object-oriented databases and others. Chaudhuri and Dayal in [ChDa97] report that data warehousing technologies have been successfully applied in many industries including telecommunications, financial services, retail stores and health care.

In relational OLAP systems, warehouse data are designed to have a star schema which allows a main fact table that holds all the integrated, time-variant data. The attributes of the fact table are foreign keys while the other tables in the warehouse called dimension tables are associated to the fact table through the foreign key attributes of the fact table. An example simple banking warehouse system consisting of a fact table and its dimension tables is given below:

Fact table is:
b-activity (cid (C), accttype (A), transtype (R), time-m(T), Amount)

Dimension tables are:
customer (cid, cname, ccity, cphone)
account (accttype, date-opened)
time (time-m, hour, day, month, year)

Thus, the fact table has attributes customer id (cid), account type (accttype), transaction type (transtype), time in minutes (time-m) and amount in dollars (Amount) involved in a transaction executed by a customer. With this fact table, all transactions on all four types of accounts available at the bank by any customer any minute are recorded. The dimension tables or dimension hierarchies allow queries that need customer name instead of his/her id provided by the fact table to be answered. Dimension hierarchies are also used for roll-up and drill-down analysis. A roll-up analysis presents an aggregation first in the lowest level detail but proceeds to present it in a more general level detail. An example of a roll-up analysis is a query to first get the total amount of dollars deposited in each account every minute, then from this aggregate we now want the total deposited

every hour, then every day, month and year. A drill-down analysis presents summaries from the coarsest level detail to the lowest level detail. The example warehouse given above does not have any aggregate tables or views yet stored. The implication is that every aggregate query seeking the total amount of dollars by each customer, by each transaction, by each account type or by each minute needs to be answered using the fact table and the dimension tables. These tables have millions of rows and queries may take long to be answered. One way to improve warehouse query response time is to pre-compute the needed aggregate tables and store. A data cube with n attributes is called an *n*-dimensional data cube and in relational OLAP is a table with 2^n subtables representing the 2^n subviews of the data cube. The *n* attributes are the subjects of interest to the organization around which the aggregate may grouped. With our warehouse example, the four fact table attributes, cid(C), accttype(A), transtype(R) and time-m(T) make the 4 dimensions of the data cube and 16 subviews can be represented by this 4-dimensional data cube. The subviews arise from different combinations of the attributes. The 16 subviews are labeled CART, CAR, CAT, ART, CRT, CA, CR, CT, AR, AT, RT, C, A, R, T, {}. The subview labeled CART corresponds to the following SQL query:

Create View CART-trans(cid, accttype, transtype, time-m, TotalAmt) AS
Select cid, accttype, transtype, time-m, Sum(Amount) AS TotalAmt
From b-activity
groupby cid, accttype, accttype, transtype, time-m;
The view labeled {} corresponds to:

Create View {}-trans AS
Select Sum(Amount) AS TotalAmt
From b-activity;

Thus, the label of the subview indicates which subjects or groupby attributes are involved in creating the view. Storing all these 2^n huge subviews may lead to storage space problems and increase in maintenance cost since all stored views need to be refreshed as updates are being made in the source databases. [HaRaUl97] proposed a greedy algorithm for selecting a set of subviews of the data cube to materialize in order to reduce the time needed to answer the queries given some storage space. Meredith and Khader in [MeKh96] argue that aggregate view partitioning can be used to improve warehouse performance. [EzBa98] propose a partition-selection scheme for partitioning any selected view and using the re-computed size of the partitioned view based on the actual

fragments of the view scanned by queries, the greedy algorithm is applied to further selection of views. The effectiveness of the approach proposed by the partition-selection scheme is enhanced if the system is able to give directions on what constitutes the best execution path for any warehouse query. In other words, given any query, which of the materialized views, and which of the fragments of this materialized view would best answer this query to give best response time?

1.1 Related Work

Gray *et al*. In [Gretal96] presented the concept of the data cube, a multidimensional representation of a set of aggregate measures. A lot of algorithms have been proposed for selecting views and indexes of cube views [Ez97a, Guetal97, HaRaUl96, LaQuAd97]. Harinayaran *et al.* in [HaRaUl96] defined the relationship between subviews using a lattice framework and defined a greedy algorithm for selecting a set of views of the data cube. The greedy algorithm starts by selecting the top level view into the set *S*. Then, each of the cube views is checked for the one that yields the maximum benefit considering the views that are already in the set *S*. The benefit of a view, *v* not in the set *S* is computed as the (number of rows in the smallest parent *u* of view already in *S* minus the number of rows in *v*) multiplied by the number of *v*'s descendant views in the cube including the view *v* itself, which can be created using view *v*. The benefit of all remaining views in the lattice are computed each time and the view with the highest benefit is included in the set *S*. Gupta *et al*. [Guetal97] extended the greedy algorithm to select both views and indexes. [Ez97a] defines a uniform scheme based on a comprehensive cost model for selecting both views and indexes. [Ez97b] extended this uniform scheme to handle dimension hierarchies. [OzVa91] presented horizontal fragmentation schemes for relational databases based on simple predicates and with no access frequencies taken into consideration. [In96, MeKh96, TiCh96] have all expressed the need for data partitioning schemes in the data warehouse aggregate materialization problem. [EzBa98] defined a scheme based on the greedy algorithm for selecting views but which fragments every selected view horizontally using application access pattern and frequency. The percentage of all rows of the view accessed on the average by queries and the frequency of their access is used to recalculate a new size for the

partitioned view. The new size is used when making future selection. For this approach to deliver the expected outcome, queries need to be directed to the appropriate fragments of some view and the work in [EzBa98] falls short of providing techniques for guiding queries on what constitutes the best execution path.

1.2 Contributions

The objective of this paper is to provide some optimization to the partition-selection scheme by incorporating a fragment-advisor algorithm. The responsibility of the fragment-advisor is to use the query access information and their access frequencies together with the fragments of the selected materialized views to recommend the set of fragments of a view the query needs to execute. The use of the size of the partitioned view in selecting future views with greedy algorithm is more justified if this component is included. The fragment-advisor gathers and provides information needed for calculating the size of partitioned views and provides the basis for dynamic re-fragmentation and re-selection of views. The benefits and performance of the proposed scheme are demonstrated using elaborate examples.

1.3 Outline of the Paper

The rest of the paper is organized as follows. Section 2 presents motivating example based on a banking warehousing system which shows selected views of the system and some queries as well as how decisions are made about what makes the best execution path. Section 3 presents formal discussion of the fragment-advisor scheme, section 4 discusses some performance justification while section 5 presents conclusions.

2. Motivating Example

This section gives an example to show how a set of fragments of a materialized view can be selected as the best for answering a warehouse query, although the formal presentation and discussion of the algorithm designed for this purpose is made in section 3.

Example 2.1 Consider a banking data warehouse with historical records of every transaction customers have asked for in all four types of accounts available (savings1(S1), savings2(S2), chequeing1(C1) and chequeing2 (C2)) from across many branches.

The data warehouse has the following fact and dimension tables:

b-activity (cid(C), accttype(A), transtype(R), time-m(T), amount(M))
customer (cid, cname, ccity, cphone)
account (accttype, date-opened)
time (time-m, hour, day, month, year)

The domain of cid is c0001, c0002, .., c1000. The domain of transtype is deposit (dep), withdrawal (wd), transfer, billpay and balance display. A sample fact table data is given in Figure 1 for only 10 tuples although table holds millions of rows typically.

cid	accttype	transtype	time-m	Amount
C0001	S1	dep	199603210003	200
C0518	C1	wd	199603210100	500
C1000	C2	wd	199603210200	300
C0001	S1	dep	199603221300	500
C0518	S2	dep	199603230600	300
C0411	S2	dep	199603230600	400
C1000	C2	wd	199603231000	100
C0300	S1	dep	199603231200	300
C0411	C2	dep	199603240500	400
C0001	C1	wd	199603241100	600

Figure 1: Sample Warehouse Fact table data

The time the transaction took place is recorded as year/month/day/minute. Since in a day there are 1440 minutes (24 * 60), the last four digits of time is used to represent both minute and hour. Some warehouse queries on this table are:

- Q_1: Get the number of customers who have made more than 2 withdrawals in savings account S1 in any month.

For the purposes of our design, we decompose every warehouse query to consist of three attribute components namely (1) Partition attributes (PA), (2) analysis attributes (AA) and (3) measure attributes (MA). Partition attributes are the attributes involved in the ``where clause'' of the SQL version of the query. Analysis attributes are those involved

in the ``group-by" clause and measure attributes are aggregates of interest. The first step in our approach is to be able to define simple predicates using the partition attributes. Simple predicates are of the form ``PA (relational operator) Value". We decompose each query into PA, AA and MA, then, from the PA we identify the set of simple predicates and the following attributes and predicate have been identified from the query Q_1.

PA = Accttype (A)

AA = Month (T), transtype (R)

MA = Number of customers or Count (C)

Predicates: P_1: A = ``S1"

- Q_2: Get all customers who have deposited some money in the morning minutes. The attributes and predicate from Q_2 are:

PA = Time-m (T)

AA = none

MA = Number of customers or Count(C)

Predicates: P_2: T $\leq$ 0720

- Q_3: Find the total amount of dollars involved in each transaction type and account type in the minutes between 0720 and 0780 (lunch hour). We have from this query:

PA = Time-m (T)

AA = Accttype (A) and transtype (T)

MA = Total amount of dollars or Sum(Amount)

Predicates: P_3: T $\geq$ 0720 AND T < 0780

- Q_4: Find the total amount of dollars deposited by each customer every minute in account C1. From this query, we obtain:

PA = Accttype (A)

AA = Customer (C), Time (T)

MA = total amount of dollars

Predicates: P_4: A = ``C1'`

Applying the Partition-Selection algorithm to this example entails defining the importance value (IP) of each of the predicates and selecting some *n* most important predicates which have the highest IP values. The IP or importance value of each

predicate is obtained by multiplying the cardinality of the predicate (number of rows from the fact table or view that are true for this predicate) by application access frequency (the number of times the predicate is accessed by an application). Further, it is assumed the queries Q_1 to Q_4 access the warehouse at the following frequencies respectively: 100, 40, 20 and 60 times. We can then see, reading from our sample fact table that $| P_1| = 3$, $| P_2| = 6$, $| P_3| = 0$ and $| P_4| = 2$. Therefore, IP of $P_1 = 3 * 100 = 300$, IP of $P_2 = 6 * 40 = 240$, IP of $P_3 = 0 * 20 = 0$, and IP of $P_4 = 120$. The two predicates selected P_1 and P_2 were used to generate the following four horizontal fragments of the top level view CART.

$M_1 = P_1 \wedge P_2 \Rightarrow A = \text{"S1"} \wedge T \leq 0720$

$M_2 = \neg P_1 \wedge P_2 \Rightarrow A \neq \text{"S1"} \wedge T \leq 0720$

$M_3 = P_1 \wedge \neg P_2 \Rightarrow A = \text{"S1"} \wedge T > 0720$

$M_4 = \neg P_1 \wedge \neg P_2 \Rightarrow A \neq \text{"S1"} \wedge T > 0720$

This has provided fragmentation of only the top level view, and to select the rest of the views, we re-compute the size of the fragmented CART view using only the total number of rows actually accessed by queries through its fragments. For example, since CART has 4 fragments, if all views are accessed 10 times by queries and each query on the average accesses only 2 fragments of the view (which are about the same size) 75% of the time, then the average number of rows accessed in the view by a query is (100/2 * .75) + (100 * .25) = 62.5 rows or approximately 63 rows. Thus, the initial 100 million row size of view CART is now taken as 63 million rows. This new size is used in running the greedy algorithm to select three additional views from the main cube lattice of this warehouse given as Figure 2. Every selected view is in turn fragmented and the final result from the partition-selection scheme is that the set of views to materialize are S={CART, CA, CT, C} and the lattice showing the materialized views with their fragments is given in Figure 3.

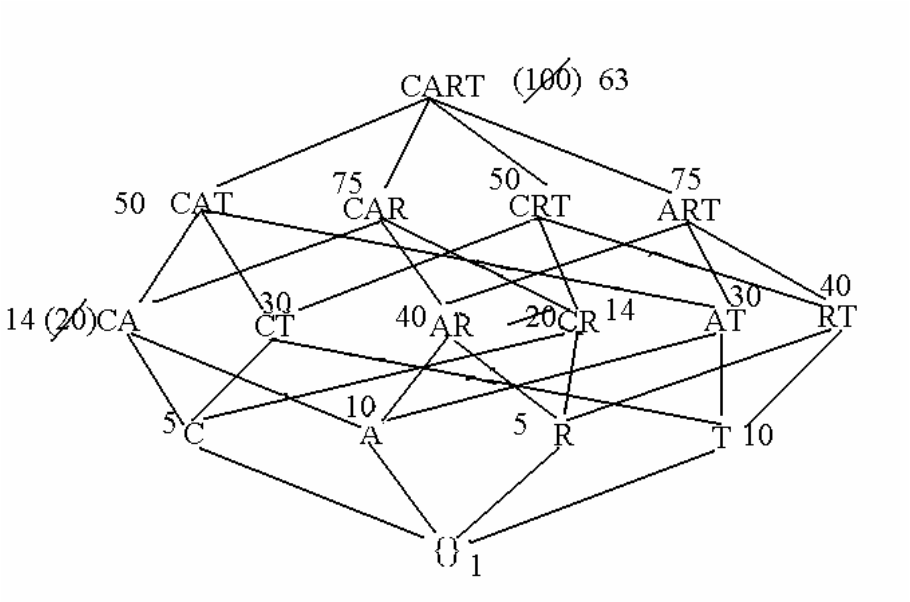

Figure 2: The Main Cube View Lattice With Re-computed Size for CART

In the cube lattice of Figure 2, the parent-child relationship between the views show that if a view is not materialized, it can be computed from any of its ancestor views.

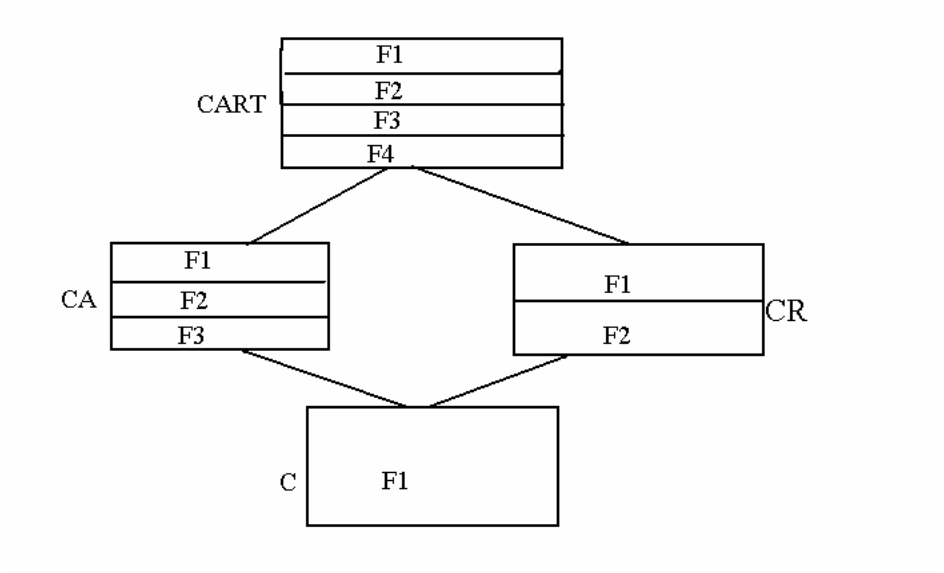

Figure 3: The materialized views with their Fragments

Figure 3 shows the selected views and their horizontal fragments. Information needed in addition by the fragment advisor includes the minterm predicates that defines each of these fragments and table 1 shows the minterm fragments for all the fragments of each selected view.

Materialized View	Fragment	Minterm Predicate	Num of rows
CART	F_1	A = "S1" $\wedge$ T $\leq$ 0720	1 units
	F_2	A $\neq$ "S1" $\wedge$ T$\leq$ 0720	5
	F_3	A = "S1" $\wedge$ T > 0720	2
	F_4	A $\neq$ "S1" $\wedge$ T > 0720	2
CA	F_1	A = "S1"	2
	F_2	A = "C1"	2
	F_3	A = "C2" $\vee$ "S2"	4
CT	F_1	T $\leq$ 0720	6
	F_2	T > 0720	4
C	F_1	ALL	5

Table 1: Minterm Fragments of Selected Stored Views

Now that all the input data for the fragment advisor have been determined by the partition-selection scheme and are discussed, the question to be answered by the fragment advisor is: "given any warehouse query like Q_1 to Q_4, which fragments of a view should be used to improve response time?" Taking Q_1 for example, the approach used is to define all attributes needed to answer the query by concatenating the PA, AA and the non-aggregate form of MA. For query Q_1 that will give attributes C,A,R.T. Then, we rule out any stored view which does not have all needed attributes. For Q_1, we are left with only CART but for most cases, we are still left with a set of views. From the set of eligible views, applying the predicates of the query on the minterms of the fragments enables us to count the number of rows visited by a query as the size of every fragment whose minterm fragment includes the predicates of this query. In the end, we select the view with the lowest total count of number of rows visited obtained from the sum of the size of all fragments needed to answer the query. The predicate for Q_1 is A = "S1" and since this predicate is found only in fragments F_1 and F_3 of CART, it means that the two fragments recommended for answering Q_1 are F_1 and F_3 for a total number of rows of (1 + 2) = 3 million rows as opposed to the 10 million rows in the original unfragmented CART.

3. Fragment Advisor Algorithm

This section starts by first presenting some formal definitions of the terms used in presenting the algorithm, then, the algorithm which defines the set of fragments of a view that best answers a given query is presented.

3.1 Definitions

Definition 3.1 *Primary horizontal fragmentation* is the partitioning of a relation based on the values of its attributes such that each fragment contains only a subset of the tuples in this relation. ■

Definition 3.2 *A user query Q_i accessing* an aggregate view V_i is made up of a set of analysis attributes AA, a set of partitioning attributes PA and a set of measure attributes MA. ■

Definition 3.3 *An analysis attribute AA_{ij}* from a user query Q_i represents the a subject or groupby attribute of interest to the application. ■

Definition 3.4 *A partition attribute PA_{ij}* in a user query Q_i defines the subset of records found in view V_k which is relevant to the application. It is an attribute in the ``where clause". A set of simple predicates are usually derived from each partition attribute. ■

Definition 3.5 *A measure attribute MA_{ij}* in a user query Q_i is an aggregation attribute of interest to the application. ■

Definition 3.6 *All Attributes of a query (A_q)* is the concatenation of all the AA, PA and the non-aggregate part of MA. ■

Definition 3.7 *All Attributes of a view (A_v)* is the set of all groupby attributes that define the view *v*. For example, A_v of view CAR is C, A, R. ■

Definition 3.8 *Possible view set (P_v)* for answering a query is the set of views among the materialized views that can adequately answer the query. ■

3.2 Fragment Advisor Scheme

The sequence of steps to execute in order to find the set of fragments of a view which best answers a query are discussed next. Input to the scheme are PA, AA, MA and set of

predicates Pr_q of the query as well as the set of materialized views V with their fragments. Each fragment comes with the conjunctive minterm predicate that defines it and the size of the fragment which is the number of rows of this view for which the minterm predicate is true. The steps in the scheme are:

- Step 1: Define the set of possible views P that can be used to adequately answer this query. To get the possible view set, we first define all attributes needed by the query by concatenating the query's PA, AA and MA sets. Thus, A_q = PA || AA || MA. Then, for every view, v, in the set of materialized views, if the set of all attributes of the query A_q is a subset of the set of all attributes of the view A_v , v is made a member of the possible view set. In other words,

  $$\forall_{v \in V \mid Aq \subseteq Av} \Rightarrow v \in P$$

- Step 2: Once we have defined the possible view set, the next step is to determine which of these competing views should be selected to answer the query. Intuitively, the view that should be chosen is the one that requires scanning of fewest rows. The number of rows of a view scanned can be determined as the sum of the cardinalities of all its fragments that need to be visited in order to answer the query. This means that the view which requires some of its fragments to answer the query which will lead to the lowest total number of rows for answering the query, is the selected view and fragments. Thus, the scheme selects a $v_j \in P$ and the set F_{ij} of fragments of v_j such that the following conditions are satisfied:

  $$v_j \in P \wedge Pr_q \subseteq F_{ij} \wedge \forall_{vk \in P} |F_{ij}| \in v_j < |F_{ik}| \in v_k$$

The formal algorithmic definition of this solution is given as Figure 4.

Algorithm 3.1 (Fragment advisor - Recommends a set of fragments of a view for a query)

Input: Query as PA, AA, MA and set of predicates Pr, Set of materialized views V and their fragments, minterms of each fragment and its cardinality.
Output: A set of fragments F of a view
begin

```
A_q = PA || AA || MA
Define the possible view set as follows:
P = {}
for each v in V do
  if A_q ⊆ A_v
        then P = P ∪ v
end
Smallest-total-row = 0
for each v in P do
   begin
        Needed-Fragments of  v = {}
        Number-of-rows for v = 0
        for each fragment F of v do
           begin
                if the conjunction of all predicates Pr of  query ⊆ minterm of    F
                        then Need-Fragments of v = Needed-Fragments of v ∪ F
                Number-of-rows for v = Number-of-rows for v + |F|
           end
        if Smallest-total-row > Number-of-rows for v
                then Smallest-total-row = Number-of-rows for v
                whichview = v
  end
Recommended = Need-Fragments of v
end
```

Figure 4: The Fragment Advisor Algorithm

4. Performance Analysis

Partitioning of stored views leads to some improvement in system performance because of reduced query response time since most queries will indeed scan fewer fragments than all, and in turn scan fewer rows than are stored in the original view. This section argues the importance of the fragment advisor and its contribution to the benefits of the partition-selection scheme. Partitioning a view leads to both reduced query response time

and maintenance cost. The query response time is reduced because only a fraction *f* of all the rows in the view are accessed on the average by a query. In the worst case, *f* is 1 in which case all fragments are visited and all the rows in the view are accessed on the average by each query. In the best case, f is 0 meaning that no rows of the view are scanned on the average by a query. This suggests the need for a fragment-advisor

algorithm whose responsibility is to recommend the best set of fragments of a view a query needs to visit. Without the addition of the fragment advisor component the benefits of the partition-selection scheme may not be realized. This is because queries may need to scan through the entire view again to ensure that no data is left out. Secondly, computing the new size of the partitioned view for running the greedy algorithm during selection of further views is based on estimated data which the advisor can easily and more accurately supply if it keeps track of the total number of rows visited by each query and that is used to compute the average number of rows visited by queries.

The partition-selection approach for materializing views is better for applications accessing mostly parts of the view and the advisor serves the purpose of identifying when the approach is not very beneficial. On how partitioning of view leads to a reduction in cost, the argument is that if high percentage of source database updates are directed to portions of the view kept in fragments, only a few fragments will be scanned and updated leading to lower maintenance cost. Further proof of the gains of this approach can be accomplished through experimental analysis. This will enable collection of data on both query response time gain, maintenance cost gain with different query patterns, varying view sizes, varying selections and partitioning of the views using the algorithm and this is intended for future work.

5. Conclusion and Future Work

This paper investigated the problem of automating the process of deciding which materialized view and which fragments of the view would provide fastest response to a query. The benefits of such a program includes both reduction in query response time as well as reduction in maintenance cost of stored views.

This work serves an enhancement or optimization to a partition-selection scheme for views which recommends storing selected views as a set of horizontal fragments. The reasoning behind such an approach is that it provides some performance gain if queries do not spend precious CPU time scanning rows of the huge warehouse tables that they do not need to scan.

Examples to demonstrate the workings of these algorithms are given. The foundation work presented in this paper can be extended to accommodate taking the

availability of indexes into consideration while making decisions about views and their fragments that should be recommended. It can also be extended to keep track of changes in query access pattern that could trigger a re-fragmentation and re-selection of view when the query access pattern changes so much that most queries are needing to access all fragments of most stored views. These extensions are currently being investigated.

References

[ChDa97] Surajit Chaudhuri and Umeshwar Dayal. An Overview of Dat Warehousing and OLAP Technology. In *Sigmod Record*, Vol. 26, No. 1, March 1997.

[Ez97a] C.I. Ezeife. A Uniform Approach For Selecting Views and Indexes in a Data Warehouse. In *Proceedings of the 1997 International Database Engineering and Applications Symposium, Montreal, Canada*, IEEE publication, Aug. 1997.

[Ez97b] C.I. Ezeife. Accommodating Dimension Hierarchies in a Data Warehouse View/Index Selection Scheme. In W.G. Wojtkowski, Wita Wojtkowski, S.Wrycza and J.Zupancic, editors, *Systems Development Methods for the Next Century*, pp. 195-211. Plennum Publishing Corporation, New York, 1997.

[EzBa98] C.I. Ezeife. A Partition-Selection Scheme for Warehouse Aggregate Views. Submitted to the *9th International Conference of Computing and Information, Winnipeg, Manitoba, Canada*, June 17 - 20, 1998

[Gretal96] J. Gray, A. Bosworth, A.Layman, and H.Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Proceedings of the 12th International Conference on Data Engineering*, pp. 152-159, 1996.

[Guetal97] Himanshu Gupta, Venky Harinarayan, Anand Rajaraman, and Jeffrey Ullman. Index selection for OLAP. In *International Conference on Data Engineering*, Burmingham, U.K., 1997.

[HaRaUl96] Venky Harinarayan, Anand Rajaraman, and Jeffrey Ullman. Implementing Data Cubes Efficiently. In *ACM SIGMOD International Conference on Management of Data*, June 1996.

[In96] W.H. Inmon. *Building the Data Warehouse*. John Wiley Symand [] Sons, Inc. second edition, 1996.

[MeKh96] M.E. Meredith and A. Khader. Divide and Aggregate: Designing Large Warehouses. *Database Programming and Design*, 9(6), June 1996.

[OzVa91] M.T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1991.

[TiCh96] Susan Tideman and Robert Chu. Building Efficient Data Warehouses: Understanding the Issues of Data Summarization and Partitioning. In *Proceedings of the Twenty-First Annual SAS Users Group International Conference*, SUGI 21, Vol. 1, pp. 520-527, 1996.

[Wi95] J. Widom. Research Problems in Data Warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM)*, Novemeber 1995.