Functional Dependency Driven Auxiliary Relations Selection for Materialized Views Maintenance

Mukesh Mohania
IBM India Research Lab
IIT Delhi Campus
New Delhi
+91-11-51292222
mkmukesh@in.ibm.com

P. Radha Krishna, K.V.N.N Pavan Kumar
Institute for Development and Research in Banking Technology
Masab Tank
Hyderabad
+91-40-23534981
prkrishna@idrbt.ac.in, pavankkvnn@mtech.idrbt.ac.in

Kamalakar Karlapalem
International Institute of Information Technology, Gachibowli, Hyderabad
+91-40-230101967
kamal@iiit.net

Millist Vincent
School of Computer and Information Science
University of South Australia, Australia
+61- 8- 302 3166
Millist.Vincent@unisa.edu.au

ABSTRACT

In a data warehouse system, maintaining materialized views can speed up query processing. These views need to be maintained in response to updates in the base relations. This is often done for reasons of data currency, using incremental techniques rather than re-computing the view from scratch. However, when the data source changes, the views in the warehouse can become inconsistent with the base data. Thus, maintenance of materialized views in the warehouse consistent with the base relations is a challenging task. In this paper, we propose an approach to maintain a materialized view without accessing the base relations by materializing and maintaining additional relations, known as auxiliary relations. In our approach, these auxiliary relations are derived based on the functional dependencies that hold on base relations, materialized view, and the key participation of the base relations in the materialized view. This approach helps in reducing the storage space and improves the efficiency of view maintenance. We present an algorithm to derive those auxiliary relations and determine which auxiliary relations need to be materialized in order to maintain a materialized view incrementally. We also present the cost model that enables the evaluation of the total cost and benefit involved in materializing auxiliary relations.

ADVANCES IN DATA MANAGEMENT 2005
Jayant Haritsa, T.M. Vijayaraman (Editors)

1. INTRODUCTION

A data warehouse is an information base that stores a large volume of extracted and summarized data for On-Line Analytical Processing and Decision Support Systems. To reduce the cost of executing aggregate queries in a data-warehousing environment, frequently used aggregates are often pre-computed and materialized into summary views so that future queries can utilize them directly [10]. Materialized views are important in data warehouses for fast retrieval of derived data regardless of the access paths and complexity of view definitions. These materialized views avoid scanning the large data sets for the queries that occurs frequently. However, when the underlying database relations are updated by insertion and deletion of tuples, a materialized view must also be updated to ensure the correctness of answers to queries against it. Updating the materialized view by full re-computation is often expensive. There are several approaches [4][17][5][14] for maintaining materialized views incrementally, whereby only the portions of the view that are affected by the changes in the relevant sources are updated. That is, a new view is computed from the existing view as and when there are changes to the base relations. On the other hand, additional views (often called auxiliary relations) are stored at the data warehouse in order to ensure enough information to maintain the views without accessing the base relations.

A materialized view can be maintained efficiently by maintaining and materializing the auxiliary relations. The approach of materializing auxiliary relations saves on base data sources access, but it may require a large amount of data to be stored and maintained at the warehouse. Furthermore, for a system with limited storage space and/or with thousands of summary views, we may be able to materialize only a small fraction of the views. In addition to reducing the cost of maintaining a view, storing auxiliary relations has additional benefits. Firstly, these auxiliary relations can be used in maintaining multiple views having common sub-expressions, where the auxiliary relations will

correspond to these sub-expressions. Lastly, the relations may be used to maintain a view when the view definition itself is slightly modified. Since auxiliary relations also change in response to updates to the base relations, these relations need to be maintained along with the materialized view. Thus, it is necessary to select the auxiliary views with less storage space, which facilitate less maintenance cost of materialized views.

In this work, we present an algorithm for deriving auxiliary relations as well as determining the best possible set of auxiliary relations to be materialized in order to maintain a view. We considered the functional dependencies of both base relations and materialized views and key participation of the base relations in the materialized view. The derived auxiliary relations make it possible to maintain a materialized view by significantly reducing the total computing and maintenance cost. In [11], auxiliary relations are derived from the intermediate results of the views. This work differs from the earlier work mainly in the way it derives the auxiliary relations. We used functional dependencies (FDs) of base relations for deriving the best possible set of auxiliary relation. The FD based decomposed relations by our method are easy to maintain than query process based shared sub-expressions. Our technique is independent of number of queries that get executed.

The rest of the paper is organized as follows. In section 2, we present an overview of the related work. Section 3 presents our algorithm for deriving auxiliary relations by determining the best possible set of auxiliary relations that are needed to maintain the view. Section 4 contains the cost model and experimented on some sample data and we conclude with section 5.

2. RELATED WORK

The related work that was done for maintaining the materialized views falls into three categories: selecting views to materialize in order to minimize query costs, e.g., [6][7], selecting auxiliary views to materialize in order to minimize the cost of maintaining given primary views, e.g., [12][15], and finally the work in lineage tracing and view maintenance [3][2]]. In [7], a greedy algorithm is proposed for selecting auxiliary views to materialize, with the goal of minimizing the cost of queries over aggregate views given certain constraints such as the maximum number of views that can be materialized. The work considers data-cube views only, and can make certain simplifying assumptions based on this restriction. [6] extends the work in [7] to general relational views, and proves that the auxiliary view selection problem under maintenance cost constraints is NP-hard. [15] proposes an exhaustive algorithm for selecting auxiliary views to optimize view maintenance and suggests simple search space pruning strategies when the view is too complex for exhaustive search. [9] presents an A* algorithm for selecting auxiliary views and indexes on different join combinations for SPJ view maintenance. Both [15] and [9] consider a single algorithm for selecting auxiliary views (and indexes in the case of [9]), designed specifically for optimizing view maintenance. They consider as potential auxiliary views all nodes in all possible relevant query plans, making the search space doubly exponential in the view definition size.

The lineage tracing for relational data warehouses in [3] presents a formal framework and basic algorithms. In [2], the authors introduced the problem of selecting auxiliary views to simultaneously reduce view maintenance and lineage tracing costs, considered the restricted case of SPJ views and suggested several alternative auxiliary view schemes and compared their performance.

Lee and Hammer [10] presented a genetic algorithm based solution to the maintenance cost view selection problem for computing a near optimal set of views. A View Relevance Driven Selections (VRDS) algorithm is presented in [16] to select materialized views, which minimizes the query processing cost and view maintenance cost. In [8], an efficient plan for incremental, transient and permanent materialization for the maintenance of a set of materialized views, by exploiting sub-expressions between different view maintenance expressions is presented. Several algorithms for view selection and maintenance problems while designing data warehousing systems are described in [1]. In [13], a view maintenance algorithm is presented by materializing and maintaining auxiliary relations, which are derived from the intermediate results of the view computation. In this paper, we handle the problem of maintaining the materialized views more efficiently by deriving the auxiliary relations based on the functional dependencies present in the base relations and materialized view as well as the participation of keys.

3. DERIVING NORMALIZED AUXILIARY RELATIONS

In this section, we formalize the notions of functional dependencies and some useful definitions of participation of keys. Consider a database $<\mathbf{R}, \sum>$, where $\mathbf{R}$ is a finite set $\mathbf{R}= \{R_1, R_2...R_n\}$ such that each $R_i \in \mathbf{R}$ is a relation schema and $\sum$ is a set of functional and inclusion dependencies. Let a materialized view, $<V, Fv>$ be defined over $\mathbf{R}$ by an SPJ relational algebra operations such that F_V is the set of functional dependencies that can be inferred from $\sum$ that hold in V. Let the primary key of each relation R_i be defined by a set of attributes K_i.

Functional dependency A functional dependency (FD) over a relation schema R_i is a statement of the form R_i: $X \rightarrow Y$ where $R_i \in \mathbf{R}$ and $X, Y \subseteq R_i$ are sets of attributes.

Full key participation: If the key of R_j is fully contained in the view V, then R_j has a full key participation in V.

Partial key participation: If the key of R_i is not fully contained in the view V, then R_i has a partial key participation in V.

Non-key participation: If the key of R_i is not contained in the view V, then R_i has a non-key participation in V. It is a special case of partial key participation.

No participation: If none of the attributes of R_i is contained in the view V, then R_i has no participation in V

Total view: If the view contains the key attributes of all its participating relations, then this view is called a total view. That is, if all the relations have full key participation in V then this view is called a total view.

Partial view: If all the relations do not have full key participation in V, then this view is called a partial view.

Input: E
Output: B (E)

Start

1. B (E) = {{A_1}...{A_j}} where {A_1...A_j) is the set of attributes appearing in E
2. For every pair of selection conditions in σ_c such that $A_i=c_i$, $A_j=c_j$, and $c_i=c_j$
   - Then merge the sets in B (E) corresponding to A_i and A_j
3. For every selection condition $A_i = B_i$ in $\sigma_=$
   - Then merge the sets in B (E) corresponding to A_i and B_i

End

Figure 1. Equality set finding algorithm

Self-maintainable view: The view V is *n-access-maintainable* if the base relations are accessed at most n times in order to maintain the view V for a given update δ. If n = 0, then the view V is self-maintainable.

3.1 Finding the Functional Dependencies that Hold in the View

The notations used in this section are shown in Table 1. In the proposed procedure if the same attribute name is present in more than one relation, then the name of that attribute is changed so that each attribute name will be present in only one relation schema. In this paper, we considered only SPJ views, which can be written as $V = \prod_Z (E)$, where E is $\sigma_c\sigma_= (r_1 X r_2 X \ldots X r_n)$. We denote the set of all attributes appearing on the left hand side of a selection condition σ_c by ATT (σ_c).

To derive functional dependencies that exist in the view V, we first compute the equality sets of E, denoted by B (E), where each equality set is a set of attributes. The meaning of these sets is that if any pair of attributes A and B are in the same equality set, then t [A] = t[B] for every tuple t in E. This can result from either A and B being equated to the same constant in σ_c, or by A being equated to B in $\sigma_=$. The algorithm [12] for finding the equality set is shown in Figure 1.

The functional dependencies in the view are computed using the equality sets as well as the functional dependencies present in the base relations. For every A ∈B(E), we denote the set in B(E) to which A belongs by L_A. The FDs that hold in E, denoted by F_E, are computed as given below:

$$F_E = F^1 \cup F^2 \cup F^3$$

where $F^1 = F_1 \cup \ldots \cup F_n$;
$F^2 = \{\phi \rightarrow ATT(\sigma_c)\}$ and
$F^3 = \{A \rightarrow L_A \mid A \in ATT(E)\}$.

Here, F^2 indicates nothing determines the conditional attributes present in σ_c and F^3 indicates that an attribute determines itself after renaming, which form canonical trivial functional dependencies as discussed in [12]. All other functional dependencies will present in F^1. We note that the FDs corresponding to any singleton set in B (E) is trivial and so can be removed from F^3. We know that an FD $X \rightarrow Y$ holds in $\Pi_x (r_i)$ iff $X \rightarrow Y \in F_i^+$ (closure of F_i) and X Y ⊆ R_i. Finally, we compute the FDs that hold in the view V, denoted by F_v as

$$F_v = \{X \rightarrow Y \mid X \rightarrow Y \in F_E^+ \text{ and } X\ Y \subseteq V\}.$$

Table 1. Notations

D = Database containing the base relations
R_1, ... R_n denote the relations schemes in the database D
A, B, C and their subscripts denote single attribute names
X, Y and their subscripts denote sets of attribute names.
The Set of FDs which holds in relation scheme R_i is denoted by F_i
ATT (E) denotes the set of attributes in E
r_1, ... r_n are the relations correspond to relational schemas R_1, ... Rn
σ_c denotes a selection statement with the set of selection conditions {$A_1 = c_1$, …, $A_p = c_p$}
$\sigma_=$ denotes a selection statement with the set of selection conditions {$A_1 = B_1$, ..., $A_k = B_k$}
Q = Query on the database D that defines a view
T = Newly modified tuple in D
F = The set of functional dependencies that hold on the data base
F_v = The set of functional dependencies derived from materialized views
SAT (database, Functional dependencies) = Satisfaction function representing functional dependencies associated with the database
AR = Set of Auxiliary relations derived
V = Materialized view under consideration
$C_m(AR)$ = Cost of maintaining auxiliary relations
$C_{mb}(V)$ = Cost of maintaining view from base relations
$C_{mA}(V)$ = Cost of maintaining view from auxiliary relations

Input: Keys of all base relations $K_{Rall} = \cup_{i=1}^{n} \{K_{Ri}\}$,
Set of all attributes in the view **Av.**
Rj is the j^{th} relation in database D
Output: Key of the materialized view K_v

Start
If $K_{Rall}-\{K_{Rall} \cap Av\} = \{\phi\}$
$K_v = \cup K_{Ri} - \cup \{A \mid A \in K_i \cap \exists X \rightarrow A \in R_j, i \neq j\}$
Else
$K_v = \{K_{Rall} \cup \{Av- K_{Rall}\}-\{K_{Rall}-Av\}- \cup \{A \mid A \in K_i \cap \exists X \rightarrow A \in R_j, i \neq j\}\}$
End

Figure 2. Algorithm for deriving the key

3.2 Deriving the Key of the Materialized View

The select-project-join operation on auxiliary relations is performed using the key of the materialized view. The key is derived based on whether the materialized view is partial view or total view. Figure 2 shows the algorithm for deriving the key in a materialized view.

Example 3.1.
Let the data base schema be {R_1 (A, B, C, K), R_2(C′, F, G) and R_3 (C″, D, E)}.

Case 1: - Total view
Let $V = \Pi_{ABCDF}(\sigma_{E=e}(R_1 \bowtie R_2 \bowtie R_3))$. Here R_1, R_2, and R_3 have full key participation in *V*. Using the procedure shown in section 3.1 and algorithm in Figure 2, we can derive F_v = (AB→C, C→D) and the primary key K_v = ABF.

Case 2: Partial view
Let $V = \Pi_{ECDG}(\sigma_{E=e}(R_1 \bowtie R_2 \bowtie R_3))$. Here R_3 has full key participation, R_1 has no key participation and R_2 has partial key participation in V. Using the procedure shown in section 3.1 and algorithm in Figure 2, we can derive F_v = (C→D) and the primary key K_v = ECG

In this work, we derived a set of self-maintainable auxiliary views that can be maintained without accessing the data sources or replicating the base data. We assume that any changes to the database do not violate F. For simplicity, we considered the insertion operation at base relations.

Theorem 3.1. Auxiliary relations derived using functional dependencies are self-maintainable.

Proof. A relation is self-maintainable if it does not access base relations and satisfy the functional dependencies after update. As the auxiliary relations that are derived from view are nothing but the functional dependencies that hold on the view, the functional dependencies of materialized view are derived from the functional dependencies that are present in base relations. So the updates to base relations can be reflected in auxiliary relations only with the tuple that updated without accessing base relations. Thus, after update, the auxiliary relations will satisfy functional dependencies.

$\forall T$ SAT(AR$\cup${T}, F_v)
$\forall T$ SAT (D $\cup${T}, F) and Q (D $\cup${T}) = V

So, the view is consistent after modifications to the base relations and hence, auxiliary relations are self-maintainable.

Theorem 3.2. Benefit of incorporating auxiliary relations is always greater than zero.

Proof: According to Theorem 3.1, the set of auxiliary relations AR is self-maintainable. So the cost of maintaining the Auxiliary relations is zero.

$$C_m(AR) = 0$$

The cost of maintaining view is more due to access to base relations. As we are not accessing the base relations and also the cost of maintaining auxiliary relations is zero, we can say the cost of maintaining the views by using auxiliary relations is less than the cost of maintaining views from base relations.

Since, $C_{mb}(V) >> C_{mA}(V)$ and Q (D $\cup${T}) = V ,

we get benefit > 0.

Thus, the benefit of materializing auxiliary relations is greater than zero. The benefit of adding auxiliary relations is mainly based on the presence of "total views" and "partial views". If all views are not "total views", then the benefit of auxiliary relations is always positive.

We initially considered all functional dependencies derived from views and the key of the views as auxiliary relations. The presented algorithm (see Figure 3) computes the auxiliary relations, which gives more benefit. But, in the case of partial view, the auxiliary relation derived based on the key of the partial view cannot be self-maintainable, because that auxiliary relation requires base relations for its maintenance and hence it will not exist in the best possible set of auxiliary relations which are going to be materialized. So we can say that the benefit of adding the auxiliary relations cannot be negative. However, in the case of total view, the materialized view itself can be self-maintainable.

4. COST MODEL AND DISCUSSION

Figure 3 shows the algorithm to derive a set of auxiliary relations that are self-maintainable by considering all the functional dependencies present in the base relations and the key participation.

Input

Base relations B={b_1, b_2…b_n}

Materialized views M={mv_1, mv_2 … mv_m}

Set of functional dependencies in views F_M={F_{mv1}, F_{mv2} … F_{mvm}}

Set of functional dependencies in Base relations F_B={F_{b1}, F_{b2} … F_{bm}}

F_{all} ={ϕ}, F_{final} = {ϕ}

Output

Set of auxiliary relations AR_{final} = {AR_1, AR_2 … AR_l}

Start

1. $\forall mv_i \in M$
   - Compute equality set using procedure shown in Figure 1
   - Compute Functional dependencies by using the procedure explained in section 3.1
   - Compute the key of the view using the procedure presented in Figure 2
   - $F_{all} = F_{all} \cup \{F_i \mid F_i \in F_v \cap F_i \notin F_{all}\}$
2. Initially consider all the functional dependencies and keys derived in step 1 as auxiliary relations.
3. Calculate the benefit we get by materializing each auxiliary relation by using the cost model presented in the section 4 and arrange them in descending order of benefit in AR as AR_1, AR_2, …AR_k where AR is the set of Auxiliary relations.
4. $\forall AR_i \in AR$ if S (AR_i) < $S_{available}$ where $S_{available}$=available space and S(AR_i)=space required for AR_i
   - $AR_{final} = AR_{final} \cup AR_i$
   - $S_{available} = S_{available} - S(AR_i)$

End

Figure 3. Algorithm for deriving Auxiliary relations

The algorithm first computes the functional dependencies that hold on the view and then derives best possible set of auxiliary relations with maximum benefit. For *n* number of base relations and *m* number of functional dependencies in all relations, the complexity of deriving the best possible set of auxiliary relations is O(nm). Our approach is independent of the number of materialized views. Here, the benefit of materializing auxiliary relations is computed using the cost model and the available storage space.

Let B = {b_1, b_2 ... b_n} set of *n* base relations, Q = {q_1, q_2 ... q_q} be the set of q queries accessing the materialized views, M= {mv_1, mv_2...mv_m} be the set of *m* materialized views and AR = {AR_1, AR_2... AR_l} be the set of *l* auxiliary relations generated by the auxiliary design process. Let fu_1^b, fu_2^b, fu_3^b, ... fu_k^b be frequency of updates to the base relations, and fu_1^A, fu_2^A, fu_3^A, ... fu_l^A be the frequency of update to the auxiliary relations, and fq_1, fq_2, ... fq_n be the frequency of queries accessing the materialized views. S(x) determines the space occupied by the relation x where x can be an auxiliary relation, base relation or a materialized view. The analysis assumes that there is no index or hash key in any of the summary views, therefore linear search and nested loop approach are used for the selection and join operations respectively. In the worst case, the analysis estimates that all the records in a summary view will be scanned once in order to process one user's query. The cost notations are presented in Table 2

Table 2. Cost Notations

C_{bqr}	= Total Cost of answering the *r* queries using base relations
$C_b(q_i)$	= Cost of answering the i^{th} query using base relations
C_{Aqr}	= Total Cost of answering the *r* queries using Auxiliary relations
$C_A(q_i)$	= Cost of answering the i^{th} query using Auxiliary relations
C_{mvb}	= Total cost of maintaining the views using base relations
$C_{mb}(mv_i)$	= Cost of maintaining the i^{th} view using base relations
C_{mvA}	= Total cost of maintaining the views using Auxiliary relations
$C_{mA}(mv_i)$	= Cost of maintaining the i^{th} view using Auxiliary relations

The total cost of answering the *r* queries using *n* base relations is

$$C_{bqr} = \sum_{i=1}^{r} fq_i * C_b(q_i)$$

where $C_b(q_i)$ = S (b_1) X S (b_2) X … X S (b_n).

The total cost of answering the *r* queries using *l* auxiliary relations is

$$C_{Aqr} = \sum_{i=1}^{r} fq_i * C_A(q_i)$$

where $C_A(q_i)$ = S(AR_1) X S(AR_2) X … X S(AR_x) X S (b_1) X S (b_2) X … X S (b_y). Here x<<l and y<<n.

The total cost involved in maintaining (re-computing) the materialized views only from base relations is

$$C_{mvb} = \sum_{i=1}^{n} fu_i^b * \sum_{j=1}^{m} C_{mb}(mv_j)$$

where fu_i^b is the frequency of update to the base relations and $C_{mb}(mv_j)$ is the cost of maintaining the j[th] materialized view using base relations. That is,

C_{mb} (mv_j) = S (b_1) X S (b_2) X … X S (b_x) where x <= n.

The (re-) computation of a materialized view will require the joining of the auxiliary relations as well some base relations, based on whether the view is the total view or the partial view. So, the total cost involved in maintaining the materialized views from auxiliary relations is

$$C_{mvA} = \sum_{i=1}^{l} fu_i^A * \sum_{j=1}^{m} C_{mA}(mv_j)$$

where fu_l^A is the frequency of update to the auxiliary relations and $C_{mA}(mv_j)$ is the cost of maintaining the j[th] materialized view using auxiliary relations. That is,

C_{mA} (mv_j) = S (AR_1) X S (AR_2) X … X S (AR_x) S (b_1) X S (b_2) X … X S (b_y)
where x<<l and y<<n

Benefit $= (C_{bqr} - C_{Aqr}) + (C_{mvb} - C_{mvA})$

Consider an example to calculate the benefit we get by materializing the auxiliary relations with the following base relations each having 1GB of data and each auxiliary relation will have 0.5 GB of data.

Product (pid#, p_name, s_id# ′)
Store (s_id#, s_name, s_city)
Order (pid# ′, cid# ′, quantity, date)
Customer (cid#, c_name, c_city)

On average consider each record will be of 5 bytes so there will be 200000000 records in base relation and 100000000 in auxiliary relations. Prime is used after #, just to show the attribute with different name if the same attribute is presented in other relation.

The calculation to derive auxiliary relations using the two approaches namely (i) considering functional dependencies (proposed approach) and (ii) Materializing intermediate results (auxiliary relations) of the view computation using greedy approach is given below.

Let the materialized view V be

$V = \Pi_{cid, c\_city, s\_id, s\_city} \sigma_c \sigma_= (product \times store \times order \times customer)$

where

σ_c = { pid = ‘p001’, date = ‘May’ }

$\sigma_=$ = { pid# = pid# ′, sid# = sid# ′, cid# = cid# ′ }

(i) Considering Functional Dependencies

F^1 = { pid# →p_name; s_id#→s_name, s_city;
pid#, cid#, date → quantity; cid#→c_name, c_city }

F^2 = {φ → date, pid#}

F^3 = { pid# → pid# ′, sid# → sid# ′, cid# → cid# ′ }

F_v = {sid# → s_city; cid# → c_city}

Key = {sid#, cid#}

The three auxiliary relations are { sid#, s_city }, { cid#, c_city } and {sid#, cid# }.

$C_b(q_i) = (2*10^8)* (2*10^8)* (2*10^8)* (2*10^8)$ units

$C_{bqr} = (2*10^8)* (2*10^8)* (2*10^8)* (2*10^8)$ units where r=1 (i.e. one query)

C_{mb} (V) = $(2*10^8)* (2*10^8)* (2*10^8)* (2*10^8)$ units

$C_{mvb} = (2*10^8)* (2*10^8)* (2*10^8)* (2*10^8)$ units

$C_A(q_i) = (10^8)* (10^8)* (10^8)$ units

$C_{Aqr} = (10^8)* (10^8)* (10^8)$ units where r=1 (i.e. one query)

C_{mA} (V) = $(10^8)* (10^8)* (10^8)$ units

$C_{mvA} = (10^8)* (10^8)* (10^8)$ units

Benefit = $32*10^{32} – 2*10^{24}$

(ii) Intermediary results as auxiliary relations

The auxiliary relations obtained by greedy algorithm are

1. { sid#, cid# ,pid#}
2. { cid#, c_city }
3. { sid#, s_city }

$C_b(q_i) = (2*10^8)*(2*10^8)*(2*10^8)*(2*10^8)$ units

$C_{bqr} = (2*10^8)*(2*10^8)*(2*10^8)*(2*10^8)$ units where r=1 (for one query)

C_{mb} (V) $= (2*10^8)*(2*10^8)*(2*10^8)*(2*10^8)$ units

$C_{mvb} = (2*10^8)*(2*10^8)*(2*10^8)*(2*10^8)$ units

$C_A(q_i) = (10^8)*(10^8)*(10^8)$ units

$C_{Aqr} = (10^8)*(10^8)*(10^8)$ units where r=1 (for one query)

C_{mA} (V) $= (10^8)*(10^8)+(10^8)*(10^8)+(10^8)*(10^8)+(2*10^8)*(2*10^8)*(2*10^8)$ units

$C_{mvA} = (10^8)*(10^8)+(10^8)*(10^8)+(10^8)*(10^8)+(2*10^8)*(2*10^8)*(2*10^8)$ units

Benefit $= 32*10^{32} - 9*10^{24}$

The benefit we got with respect to the other method is $7*10^{24}$ units. If we consider that each unit is one join, then we are saving $7*10^{24}$ joins. If each join will take one unit of time, then our approach saves $7*10^{24}$ units of time.

From the above example, we can find that the benefit is greater than zero for one materialized view, but as the number of base relations, records in base relations and the number of materialized views increases, the benefit will increase exponentially and also the space occupied by the auxiliary relations will also be less due to the common auxiliary relations derived for different materialized views. For a set of base relations with the increasing number of materialized views, we calculated the benefit of deriving the auxiliary relations based on the functional dependencies.

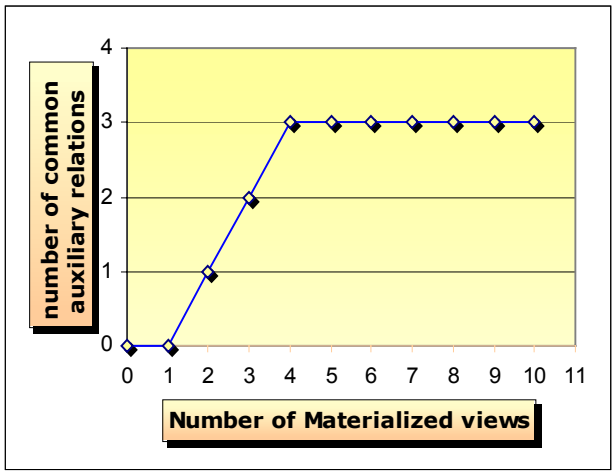

Figure 4: Plot showing common auxiliary relations

To show our approach, we considered 10 queries, which needs to be materialized. The FDs satisfying the materialized views are derived by considering the FDs present in base relations. Each functional dependency present in the materialized view will form one auxiliary relation, which is also satisfied in base relation. Here, same auxiliary relation may be derived for various materialized views. The set of auxiliary relations are derived using the approach described in the Figure 3, which provides a highest benefit. Figure 4 shows the number of common auxiliary relations for the queries under consideration. Here, the derivation of auxiliary relations will depend on the relationship among queries. If larger number of queries is having some common sub-expression, then the common auxiliary relations will be more. As the number of common auxiliary relations increases, the benefit will also increase.

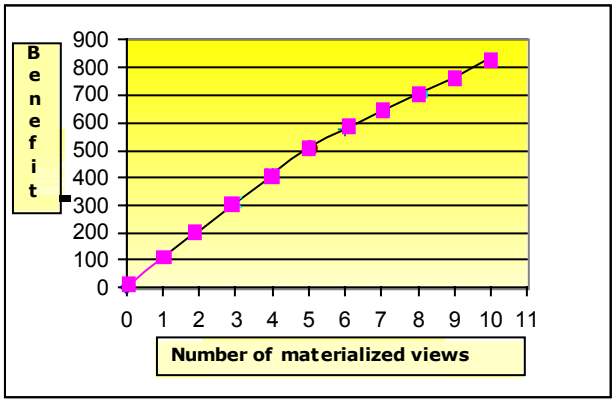

Figure 5: Plot between benefit and number of materialized views

The benefit of materializing auxiliary relations is calculated using the cost model and the results are shown in the Figure 5. Initially the benefit increased highly as the number of queries to be materialized increases and later the benefit will decrease by some amount due the increase in maintenance cost of the materialized views.

4.1. Discussion

Materialized views provide fast access to pre-computed data that results significant performance gain to answer a query. However, if the base relations change frequently, keeping these materialized views updated will incur a high maintenance cost. The two main problems that are addressed in the literature are (i) the view selection problem and (ii) the view maintenance problem. In the present work, we considered the problem of maintaining the materialized views by considering some auxiliary relations. Now, the maintenance problem was shifted to deriving the auxiliary relations that results in effective maintenance of materialized views.

Most of the earlier approaches that deal with auxiliary relations for maintaining the materialized views are based on the intermediary results obtained during the generation of materialized views. These intermediate results consist of several joins and aggregations on base relations to meet the execution of a query. These approaches will not work efficiently if there are no or minimal common sub-expressions present in the different queries. Here, a lot of importance is given for the query processing rather than maintenance of materialized intermediate

relations. But, these approaches are not utilizing the knowledge present in the base relations such as functional dependencies. In this work, we have given more importance to semantic of the schema as expressed by functional dependencies on base relations to design auxiliary views. Our approach also considers the functional dependencies present in the materialized views and key participation of base relations in the materialized views.

As the materialized views are derived from the base relations, there are more chances of having the common functional dependencies among the materialized views that holds on the base relations. We derived functional dependencies of the view based on the FDs present in the base relations, and the derived FDs form auxiliary relations. Since, the FDs present in the base relations are known, the derivation of auxiliary relations is easy due to the less complexity is involved in the selection. Further, we generated a set of best possible auxiliary relations based on the benefit calculated using the cost model. The proposed method is beneficial even if there are no common sub-expressions present in the different queries since the functional dependencies on the views are derived from the same set of FDs present in the base relations.

Our approach derives the auxiliary relations by considering the functional dependencies, which are same for the any query because they are using the same base relations. So the auxiliary views derived are best as the number of materialized views increases. The storage space will also be saved as there are common auxiliary relations derived as number of views increases.

5. CONCLUSION

Materialized views are used to increase the query performance. However, these views need to be maintained in response to updates in the base relations. This is often done, for reasons of accuracy, using incremental techniques rather than recomputing the view from scratch at specified periodic intervals. We have examined the problem of deriving the auxiliary relations to materialize in a data warehousing in order to reduce the cost of view maintenance. The auxiliary relations are derived, by considering the functional dependencies in base relations, materialized views and the key participation of base relations in the materialized views. An algorithm is presented for deriving self-maintainable auxiliary relations. A cost model is also presented with an example. As the number of materialized views increases as shown in Figure 5, initially we found that the cost of maintenance decreases and the benefit will increase highly up to certain level. After that, as auxiliary relations increases, the benefit decreases by some amount due to the maintenance cost of the materialized views.

6. REFERENCES

[1] Bellatreche, L., Kamalakar Karlapalem and Mukesh Mohania, "Some Issues in Design of Data Warehousing systems", *Data Warehousing and Web Engineering*, IRM Press, Hershey, PA (2002).

[2] Cui, Y., and J. Widom. "Practical Lineage Tracing in Data Warehouses", In *Proc. of the Sixteenth International Conference on Data Engineering*, San Diego, California (February, 2000).

[3] Cui, Y., J. Widom, and J.L. Wiener. "Tracing the Lineage of View Data in a Warehousing Environment", *Technical report*, Stanford University Database Group (1997).

[4] Gupta, A., and I. S. Mumick, "Maintenance of Materialized Views: Problems, Techniques, and Applications", *Data Eng. Bull.,* Special Issue on Materialized Views and Data Warehousing 18 (2) 3 – 18 (1995).

[5] Gupta, A., I. S. Mumick, and V. Subrahmanian. "Maintaining views incrementally", In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp. 157–166, Washington, DC (May, 1993).

[6] Gupta, H., "Selection of Views to Materialize in a Data Warehouse", In *Proc. of the Sixth International Conference on Database Theory*, pp. 98–112, Delphi, Greece (January, 1997).

[7] Harinarayan, V., A. Rajaraman, and J. D. Ullman. "Implementing Data Cubes Efficiently", In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp. 205–216, Montreal, Canada (June, 1996).

[8] Hoshi Mistry, Prasan Roy, S. Sudarshan and Krithi Ramamritham, "Materialized View Selection and Maintenance using Multi-query Optimization", *ACM SIGMOD Record*, 30(2), 307-318 (June, 2001).

[9] Labio, W.J., D. Quass, and B. Adelberg. "Physical Database Design for Data Warehousing". In *Proc. of the Thirteenth International Conference on Data Engineering*, pp. 277–288, Birmingham, UK (April, 1997).

[10] Lee, M., and J. Hammer, "Speeding up Materialized view selection in Data Warehouses Using A Randomized Algorithm", *International Journal of Cooperative Information Systems*, 10 (3), 327 –353 (2001).

[11] Michael O. Akinde, Ole Guttorm Jensen, H.Michael Böhlen, "Minimizing Detail Data in Data Warehouses", *EDBT*, pp. 293-307 (1998).

[12] Mohania, M., K. Karlapalem and M. W. Vincent, "Maintenance of Data Warehouse Views using Normalization", *COMAD' 97*, Chennai, India (1997).

[13] Mohania, M., S. Konomi, Y. Kambayahsi and M. Vincent, "Designing View Maintenance Algorithm in Data Warehousing Environment", In *Proc. of the 9th International Conference on Management of Data (COMAD'98),* Hyderabad, India (1998).

[14] Quass, D., "Maintenance Expressions for Views with Aggregation". In *Proc. of the Workshop on Materialized Views: Techniques and Applications*, pp. 110–118, Montreal, Canada (June, 1996).

[15] Ross, K. A., D. Srivastava, and S. Sudarshan, "Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time", In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp. 447–458, Montreal, Canada (June 1996).

[16] Satyanarayana R., Valluri, Soujanya Vadapalli and Kamalakar Karlapalem, "View Relevance Driven Materialized View Selection in Data Warehousing Environment", In *Proc. of 13th Australasian Database Conference (ADC2002)*, Australia (2002).

[17] Zhuge, Y., H. Garcia-Molina, J. Hammer and J. Widom, "View Maintenance in a Warehousing Environment", *SIGMOD Record*, ACM Special Interest Group of Management of Data 24 (2), 316-327 (1995).