Incremental View Maintenance: An Algorithmic Approach

1Abdulaziz S. Almazyad, 2Mohammad Khubeb Siddiqui
1College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia
2College of Computer Engineering and Sciences, Al Kharj University, Al Kharj, Saudi Arabia
1almazyad@ksu.edu.sa, 2khubeb@ksu.edu.sa

Abstract—To maintain the materialized view is one of the crucial tasks in a warehousing environment. The results of incremental computation are affected by interfering updates and compensation is required. The conventional approaches used incremental algorithm causes some anomalies. To solve such anomalies we proposed the novel approach an incremental view maintenance approach by using some existing approach with the use of Version Store and Transaction ID. The information stored in the warehouse is in the form of materialized views. Materialized views are the derived relation, which are stored as the relation in the database, when some updates occur in the parent relation all its child relations also get updated by viewing to maintain the consistency and convergence of the database. In this paper we proposed an algorithm for incremental view maintenance with the inclusion of some existing approaches. We utilized the concept of version store for older versions of tables that have been updated at the source and we are also able to detect the update notification messages that are lost during updating the view. Through the concept of Version store we can retrieve the correct data of corresponding state.

Index Term-- Data Warehouse Materialized View, Version Store, Transaction ID, View Manager and View Maintenance

I. INTRODUCTION

Data warehouse means storage of data (may be in the size of terabytes of disk storage), data warehouse is a copy of transaction data specifically structured for querying and reporting, which stores volume of historical data. A data warehouse can be normalized or denormalized. It can be a relational database, multidimensional database, hierarchical database, object database, etc. It should be: Subject-oriented, Integrated, Non-volatile, Time-Variant, Accessible, and Process-Oriented. In data warehouses, materialized views act as a cache, a copy of data that can be quickly accessed because indexes are built up over that, you can use materialized views to pre compute and store aggregated data. In these environments Materialized Views are often referred to as summaries, because they store summarized data. They can also be used to pre compute joins with or without aggregations, replicating distributed data and it quickly accesses for complex joins. A materialized view eliminates the overhead associated with expensive joins and aggregations for a large or important class of queries. Materialized views are of three types Materialized Views with Aggregates, Materialized Views Containing Only Joins and Nested Materialized Views. Maintaining a view is one of the most important tasks in warehousing environment.

A. Architecture of Data warehouse with view manager:

As mentioned in fig 1. the blocks of the data warehouse with view manager are described as below:

SOURCE:

A database, application, file, or other storage facility from which the data in a data warehouse is derived. The source contains the operating data, flat files and stage files. The stage file receives the data from source process and it verifies its creditability and the required data files will be passed to warehouse through view manager. Source division also termed as top tier of architecture.

STAGE FILE:

A place where data is processed before entering into data warehouse, it includes the following operations data cleaning, integration, transformation, load and refresh.

VIEW MANAGER:

After staging the View manager transaction is initiated and the file is transferred to version store.

VERSION STORE:

Version store is storage of updated data. It stores old versions of tables that have been updated. A transaction number TXN is given to each transaction in the table so as to maintain the versioning. After matching the Query Transaction the answer is fetched as per the Transaction ID.

WAREHOUSE:

A relational database that is designed for query and analysis rather than transaction processing. A data warehouse usually contains historical data that is derived from transaction data, but it can include data from other sources. It separates analysis workload from transaction workload and enables a business to consolidate data from several sources. It contains the Summary data, raw data, metadata, mined data etc. Warehouse division also termed as middle tier of architecture.

USERS:

Users may be end users and make use of the data warehouse view maintenance in the Analysis of Data mining, Data reporting etc, and User division also termed as top tier of the architecture.
The task of keeping materialized view up-to-date and consistent with changes to its underlying data is called as *Materialized View maintenance*. On each and every update it can be recomputed but it is better to maintain it by *Incremental Approach*. Changes to the database relation are used to compute changes to materialized view, which is already updated. View maintenance can be done by manual defining triggers on insert, delete, and update of each relation in the view definition or by manually writing code to update the view whenever database relations are updated. The main task of Incremental view maintenance is to efficiently compute the row of view.

The survey and literature reveals many innovations of efficient view maintenance of data warehousing, Jingren Zhou et al [2], they presented the lazily maintenance of materialized view that relieves updates of this overhead, they improved the efficiency of lazy maintenance by combining updates from several transactions into a single maintenance operation, by considering multiple updates of the same row into a single update, and by exploiting row versioning, with the help of Microsoft SQL Server they show the faster response time for updates and also significant reduction in maintenance cost. Aggarwal et al [5], presented incremented view maintenance Algorithm for a data warehouse from multiple distributed autonomous data sources. They have developed two Algorithms called SWEEP and NESTED SWEEP former ensure complete consistency of the view at the data warehouse in the presence of concurrent updates the later compute a composite view change for multiple updates that occur currently. Nathan Folkert et al [16], have revealed the updates to the base table using bulk and partition operations, they refresh optimizer in the presence of partitioned table and materialized views, many recognize dependencies between base tables and the Materialized View partitions leading to the generate on of very efficient refresh expressions. This makes Database Server more manageable and user friendly.

Ki Yong Lee et al [14], proposed a strategy to maintain the view called *delta* propagation that can minimize the total size of base relations accessed by analysing the properties of base relations, they usually concentrate on cost of maintaining the view. Gianluca Moro et al [15], worked on algorithm for incremental maintenance on multi-source view that does not need the compensation step and applies to general view expressions of the bag algebra, without limit on the number of base relations per data. Hao He et al [9], proposed the strategy Asymmetric batch View Maintenance. Batch Processing of base table modifications, they tackle the problem of finding the most efficient batch incremental maintenance under the refresh response time constraint. They present a series of analytical result leading to the development of practical algorithms that approximate an “Oracle Algorithm” with perfect knowledge of the future. Latha S. Colby et al [17], focus on Maintenance of View that is known as *Deferred View Maintenance*, it allows a view to become inconsistent with its definition. A refresh operation is used to re-establish consistency, they presented an algorithm to incrementally refresh a view during deferred maintenance, and this algorithm also avoids a state bug that has artificially limited techniques previously used for deferred maintenance. Eng Koon Sze and Tok Wang Ling [18] show an improvement of incremental computation through the use of data source and refreshed version numbers, they simply remove the overhead maintenance of unnecessary queries and their query results as well as avoiding the time required to execute the Cartesian products and detect the updates that will not effect the view so on that the incremental computation is not applied.

II. EXISTING APPROACHES FOR VIEW MAINTENANCE

Various approaches have been introduced for maintaining the view in a warehouse environment.
A. Basic Algorithm:

In Fig. 2 shown that there is communication between Source and the warehouse, when update occurs at source, it sends the notification to warehouse later on warehouse sends the query to source for the corresponding update as source receives the query it sends the answer to warehouse to that corresponding query.

At source:

i. Updating at source (S u_i): execute u_i; send u_i to the warehouse; trigger event W u_i at the warehouse.

ii. Query at source (S qu_i): receive query q_i; Let A_i, Q_i(ss_i); (ss_i is a current source state) Send A_i to warehouse; Trigger event W ans_i to the warehouse.

At warehouse:

i. W u_p_i: receive update u_i; Let Q_i = v(u_i); Send Q_i to the source; trigger event S qu_i at the source

ii. W ans_i: receive A_i; Update view: MV = MV + A_i;

The above algorithm states that:

1. When an update occurs at the source, it sends the update notification to the warehouse.
2. Warehouse receives the notification and sends back the query to the source about the update.
3. Source receives the query sent by the warehouse and returns the answer to that query.

The basic algorithm is neither convergent nor weakly consistent in warehouse environment [3].

B. Recompute the View:

RV does not rely on incremental view maintenance approach. It is based on recomputation of materialized view from the scratch. When ever the update occurs at the source it recomputes the view from the scratch. In RV approach warehouse sends the Query to the source asking it to recompute the view from the scratch after certain number of updates. RV sends 2 messages for each update. The bytes transferred are much higher in RV than the relative algorithms. This degrades the performance of RV [3].

C. ECA (Eager Compensating Algorithm):

COLLECT = ∅

W up_i: receive U_i;

Let Q_i = v(U_i) - Σ_0<sub>i</sub>C_{qsQ_j(U_i)}

send Q_i to the source;

trigger event S qu_i at the source

W ans_i: receive A_i;

let COLLECT = COLLECT + A_i;

if UQS = ∅

then { MV ← MV + COLLECT;

COLLECT ← ∅}

else do nothing.

ECA is an incremental view maintenance algorithm. It is a method for fixing the view maintenance problem that occurs due to the decoupling between base data and the view maintenance manager at the warehouse. The key idea of the ECA algorithm is that it cannot rely on the state of the base information that is continuously being updated/modified by the sources. It must keep track of the updates received from the source and then filter out i.e., compensate any information that will duplicate the resulting queries. By subtracting (or adding) the results it knows that will (not) get in future queries, it will create an accurate end result for the view.

The above algorithm states that:

Initially the COLLECT will be empty, source executes an update (U_i) and the notification sent to the warehouse. Warehouse receives the source update (U_i) and the query (Q_j) based on (U_i), for each query in UQS(Unanswered Query Set: the set of query set that were sent by the warehouse, but answers have not been received) formulates a compensating query Q_i based on U_i and Q_j with Q_i. Warehouse receive the query result and update the Materialized View(MV), the result of the query should be applied to the Materialized View(MV) only after the answer to this query and all related compensating query have been received.

To avoid invalid state ECA collects the intermediate answers in relation denoted as COLLECT (initially its empty).

D. Lazy Approach:

Lazy approach maintains the view in a lazy manner that relieves the updates of the maintenance overhead as in the incremental view maintenance approaches. View maintenance is postponed until the system has free cycles or it is referenced by any query. These free cycles are utilized for the view maintenance that relieves the updates and queries from the overhead. The updates are combined from different transactions into a single maintenance task. It also exploits row versioning. In lazy maintenance the updates do not maintain the view it just stores the required information so that the affected views can be maintained later. It actually uses system free cycles to maintain the views, in this no updates or
queries pay for the maintenance task. But, in case the view is not up to date and query is sent over it, then the particular query has to pay for all part of the view maintenance and some delay also. However, it pays only the view maintenance that it uses and not for other views [2].

III. THE PROPOSED SOLUTION
The maintenance of structural modifications in data warehouses is a crucial point for keeping track of structural modifications. We have proposed a solution in which we used the concept of some exiting approaches like version store [2]. Our solution is based upon incremental view maintenance utilizing the functionality of version store. Assume a user updates some data in the source. In turn the source sends the update (Ui) notification to the warehouse. Simultaneously, the copy of the updated data or that particular record(s) with the TXN number gets stored in the version store by view manager as it is shown in fig.1. The warehouse after receiving the notification prepares a query (Qi) and sends it back to the source. The source receives the query and returns back the answer (Ai) to that query to the warehouse. This is the normal functioning of the system if the synchronization between the warehouse and source is maintained. But, in actual the scenario is completely different there is no synchronization between source and the warehouse as stated above in section II. In our approach we had just tried to overcome the problems with the traditional approaches in section II. Below we have described our detailed solution.

A. As per our Proposed System.
We have classified our work in the following steps and presented a flow diagram of the system in fig. 3.

A.1 Our Algorithm:
READ UPDATE, SOURCE, TXN, ANSWER, QUERY;
IS Query = null
Yes: update source
No: Query = Answer
IF Query = null
  Do update into Source
ELSE
  IF Query_i = Answer_i
    MV = MV + Answer_i
  ELSE
    Get Data from Version
    Query_i = TXN_i
    TXN_i = Answer_i
    MV = MV + Answer_i
  END IF
END IF

A.2 Steps:
1. A user makes some change in the source (update, insertion, deletion etc).
2. The updated data (table) gets stored with TXN (Transaction number) in version store.
3. Source notifies the warehouse about the Update.
4. Warehouse checks the current sending Query (Qi).
4.1. If it is null, it sends back the query to the warehouse goto 9.
4.2. If not null, goto 5.
5. If the current Qi is equal to Ai goto 10.
6. If the current query Qi is not equal to Ai, then the updated data is fetched from version store.
7. The version store returns the answer by matching the TXN (Transaction number), goto 4.
8. Warehouse receives the answer goto 5.
9. Source acknowledges the warehouse request and returns the answer to that query.
10. Warehouse updates the view with the received answer.
11. End.

A.3 View Manager:
After staging stage the data first the data goes to view manager then data goes to warehouse. The main task of View manager is to maintain the materialized views. The view manager keeps track of all the maintenance tasks for different views. It uses hash table to store an entry of a view that has an active maintenance task. With each entry a list is maintained with the transaction numbers (TXN) for the number of tasks that are in sorted order.
To update a view, view manager gets the maintenance task from the list and contacts the version store. The versions store has all the updated data with transaction numbers (TXN). The view manager matches the TXN numbers and retrieves the specific data from the version store and updates the view accordingly.

A.4. Version Store:
To maintain the View, Version store plays an important task by match the TXN number with the query pass by data warehouse as it is mentioned in section A.1, it stores old version of tables that have been updated till now, it is the storage of updated data [2]. A transaction number i.e. TXN which is unique number that is generated for each transaction in the table so as to maintain the versioning, the TXN numbers are sequentially generated that may be incremented by sequentially or randomly. Even each statement in a transaction is given a statement number STMTN. So that it is possible to figure out which statement belongs to which transaction. It maintains a version chain of the records so that it can recover the older versions of the records until they are available. Actually, this is not a persistent but the transient storage. It stores the older versions of tables for a short period of time. A version has to be kept only until all transactions that may require it have terminated. Then garbage collector reclaims the memory used by the stored data. Version tables greatly simplify the view maintenance tasks. If in the case of two or more then two consecutive updates at the source, the traditional systems are unable to get all the updated states from the source which develops the anomaly in the view maintenance. In our system version tables helps view manager to overcome this anomaly.

IV. RESULTS AND DISCUSSION: PERFORMANCE EVOLUTION
In section II we have outlined several approaches from view maintenance in a warehousing environment i.e., Basic Approach, re-computing view (RV), eager compensating algorithm (ECA), Lazy Maintenance etc. All these approaches provide consistent and non-redundant materialized views at the warehouse. In this approach and in this section we are addressing the performance based on the messages transferred in maintaining a view.

A. Characteristics of Proposed Approach:
The proposed approach has following characteristics:

1. Approach is totally based on strategy incremental view maintenance approach no need to update the view from scratch.

2. Version store creates a number for each transaction that is known as TXN so it maintains the synchronization between source and warehouse, and it is free from locks.

3. TXN number also used to detect the update notification messages that are lost during updating the view.

B. Performance based on Number of Messages
Let us suppose that there are n numbers of updates. In case of Re-computing View (RV) assume the warehouse sends a query to the source asking it to re-compute the view after m number of updates where m ≤ n. If we compute both query and answer messages the total numbers of messages are:

\[
MV = \left(\frac{n}{m}\right) \times 2
\]

Therefore it shows that RV generates at least 2 messages if the view is recomputed once i.e., if m = n. Now suppose if the view is recomputed after every m number of updates where m ≤ n then RV generates 2n number of messages. In ECA if there are n updates there are also n number of queries and the same number of answers, so there are 2n messages.

In our system if the updates are sufficiently spaced so that each query is processed before a new update occurs our solution works same as ECA i.e., 2n number of messages. In the worst case the total numbers of messages are higher than the RV and ECA as shown in the fig. 3. The answer from the source is matched with the query sent i.e. Q_1 = A_i if the Q_k ≠ A_i, then the answer is fetched from the version store. In this case the total numbers of messages transferred are 4m. So, the worst case is not desirable in our solution as the total transfers are much higher than the existing approaches.

CONCLUSION
Data warehousing is an emerging and already very popular technique used in many applications for retrieval and integration of data from autonomous sources. However, warehousing typically is implemented in an ad hoc way. We have shown that the standard algorithm for maintaining the materialized view at a warehouse can lead to anomalies and inconsistent modification to the views. The anomalies are due to the fact that view maintenance at the warehouse is decoupled from the updates at the data sources, and we cannot expect the data sources to perform sophisticated functions in support of view management. Consequently previously proposed view maintenance algorithm cannot be used in this environment.

We have presented a novel approach that correctly maintains the materialized view in a warehousing environment. We have used the version store to store the old states of the updated records with the transaction id, so as to figure out which update belongs to which transaction. This greatly helps in our new incremental view maintenance approach to overcome the anomalies in the previous standard approaches and would greatly help in warehousing application like Information processing, Analytical processing and Data mining.

ACKNOWLEDGMENT
We would like to acknowledge and give thanks to King Saud University for the support while completing this research work and to all my colleagues who have taken the time to review the paper and comment on this paper prior to publication. Special thanks to our friends, who have been unselfishly extending their efforts and understanding.
To our parents who have always been very understanding and supportive. And above all, to the Almighty God, who never cease in loving us and for the continued guidance and protection.

REFERENCES


Prof. Abdulaziz S. Almazayd received his MS and PhD in Computer Engineering from Syracuse University, US in year 1995. His main area of Research is Computer Security, Databases, and Data mining. He is currently working as an Assistant Professor in College of Computer and Information Sciences, King Saud University, Riyadh and also holding the position of Dean in CCES, King Saud University, Al Kharj, Saudi Arabia. He also worked as a consultant for Ministry of Education, Ministry of Justice, and Royal Saudi Naval Forces Saudi Arabia. Prof. Almazayd member of International Association of Computer Science and Information Technology (IACSIIT), and has published multiple paper in IEEE/ACM.

Eng. Mohammad Khubeb Siddiqui receives his Engineering degree in Computer Science and Engineering from UP Technical University, India and his main area of interest in research is Databases and Data Mining. He is currently working as a Researcher in CCES, Al Kharj University, Saudi Arabia. He has works as an Oracle DBA for various telecom based project. Eng. Siddiqui member of International Association of Computer Science and Information Technology (IACSIIT), published paper in IEEE, ASME press.