Materialized Views in Data Mining

Vahida Attar and Vandana Inamdar

College of Engineering , Pune, India

Summary — Relational views are used both as a specification technique and as an execution plan for the derivation of the warehouse data. Currently available mining algorithms suffer from long processing times depending mainly on the size of the dataset. One observation is that the results of consecutive data mining queries are usually very similar. This observation leads to the idea of reusing materialized results of previous data mining queries in order to improve performance of the system.The views created are used by the data mining algorithms to accelerate the process of analysis on the data.
In this paper, we summarize relational views, how the results stored in these views can be used to accelerate processing of data mining queries. Looking at a major issue which deals with refreshing of the views which are needed to maintain accuracy that may cost some time to maintain the updates view.
Keywords: Materialized Views, Data Mining, Query Execution, OLAP,

1. Introduction

Data mining aims at discovery of useful patterns from large databases or warehouses. It is a non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in large volumes of data.

The basic problem in data mining is the processing time of data mining queries. Data mining algorithms often require minutes or hours to answer a simple query. On the other hand, mining practice shows that the majority of data mining queries are only minor modifications of previous queries. Given these circumstances, data mining systems should try to exploit the results of previous queries, instead of running a complete mining algorithm for each query.

The results of the previous queries along with the queries are stored as a materialized view which is reused to answer the datamining queries, theses views need to be maintained with the corresponding objects.

In this paper we discuss relational views and their various forms along with processing of views. We will look at creation of datamining views.

Refreshing is an important concept for maintaining accuracy, it deals with the refreshing or sync of views to their objects. The ability to sync with the object increases the accuracy but it may cause delays to answer queries, so a trade off must be established for optimal results

2. Related Work

The work on materialized views started in the 80s. The basic concept was to use materialized views as a tool to speed up queries and serve older copies of data.

The problem of association rule discovery was introduced in [1]. In the paper, discovery of frequent itemsets was identified as the key step in association rule mining. In [2], the authors proposed an efficient frequent itemset discovery algorithm called Apriori that became the basis for many other mining algorithms. The idea of sequential pattern discovery was first presented in [3]. In [4], time constraints were incorporated into the problem and a sequential pattern discovery algorithm called GSP was introduced.

Incremental mining was first discussed in [5] in the context of association rules. A novel algorithm called FUP was proposed to efficiently discover frequent itemsets in an incremented dataset, exploiting previously discovered frequent itemsets. Incremental mining was also analysed in the context of sequential patterns (e.g. [8]). The notion of interactive and iterative knowledge discovery first appeared in [7]. The authors postulated to create a knowledge cache that would keep recently discovered frequent itemsets along with their support value. Besides presenting the notion of knowledge cache the authors introduced several maintenance techniques for such cache, and discussed using the cache contents when answering new frequent set queries.

To facilitate interactive and iterative sequential pattern discovery, [8] proposed to materialize patterns discovered with the least restrictive selection criteria, and answer incoming queries by filtering the materialized pattern collection.

The concept of Knowledge and Data Management Systems was first introduced in [6]. In the opinion of the authors, KDMS should replace contemporary database management systems by integrating data and knowledge related activities in one central place. The authors also defined the notion of a data mining query and suppressed the need to tightly integrate knowledge discovery systems

Manuscript received December 5, 2007
Manuscript revised December 20, 2007

with the existing database and data warehouse infrastructure to provide a framework for advanced applications

2.1 Form of Views

Relational views have several forms:
Pure program: an unmaterialized view is a program specification, "the intension" that generates data. Query modifications and compiled queries were the first techniques exploiting views- their basic difference is that the first is used as macro that does not optimize until run-time while the second stores optimized execution plans. Such a view form is a pure program with no extensional attachments. Each time the view program is invoked, it generates (materializes) the data at a cost that is roughly the same for each invocation.

Derived Data: A materialized view is "the extension" of pure program form and has the characteristics of data like any other relational data. Thus, it can be further queried to build views-on-views or collectively grouped to build super-views. The derivation operations are attached to materialize views. These procedural attachments along with some "delta" relational algebra are used to perform incremental updates on the extension.

Pure Data: When materialized views are converted to snapshots, the derivation procedure is detached and the views become pure data that is not maintainable (pure data is the opposite end of the spectrum from pure program).

Pure Index: View indexes and View Caches illustrate this flavour of views. Their extension has only pointers to the underlying data which are de referenced when the values are needed. Like all indexing schemes, the importance of indexes lies in their organization, which facilitates easy manipulation of pointers and efficient single-pass dereferencing, and thus avoids thrashing.

Hybrid Data & Index: A partially materialized view stores some attributes as data while the rest are referenced through pointers. This combines data and indexes. B-trees, Join indexes, star-indexes and most of the other indexing schemes belonging to this category, with appropriate schema mapping for translating pointers to record field values. Note that in this form, the data values are drawn directly from the underlying relations and no transformation to these values is required.

OLAP aggregate/indexing: A data cube is a set of materialized or indexed views. They correspond to projections of the multi-dimensional space data to leser dimensionality subspaces and store aggregate values in it. In this form, the data values are aggregated from a collection of underlying relation values. Summary tables and Star Schema belong in this form (the latter belongs here as much as in the previous category).

2.2 Processing of Views

Let's examine view processing for all the view forms except for the pure data (snapshots) which are not maintainable. View processing involves view scanning, incremental update, or both applied simultaneously. Scanning and incremental update of views imply special locks, locking protocols, authorization, and consistency protocols for asynchronous updates from multiple sources. We will concentrate here on performance issues. View scanning in the pure program view form is typically the same as re-execution of the query that created the view. There is no performance benefit for unmaterialized views other than predicting re-execution cost more accurately after the first time. The performance is bad but predictable. Scanning a materialized view has a cost that depends on the ratio of the useful tuples in it to answer a given query, called density of the view. For a 100% ratio, scanning a materialized view is optimal because it has all the data for answering the query compacted in a tight storage space. If the density is low, the noise can be more than the amount of useful data. For the index view form, scanning cost can range from near optimal, when the pointers are aligned and point to a tight space, to very high, when pointer dereferencing causes thrashing (similar to unclustering indexes in RDBMS or in OODBMS). For this reason, in the index form, it is very important that the pointers be well organized and use a tailored buffer manager which avoids thrashing caused by the multidimensionality of the view. ViewCache uses a form of puzzle shaped packed R-trees and tailored cache replacement strategies. Cube trees utilize multi-dimensional compressed and packed R-trees. Again, for performance, the organization is the only thing that matters.
Incremental update techniques for views are mature, as they go back for more than a decade of research. The same techniques were the foundation for the management of replicated data which found its way to the log-based replication tools of commercial RDBMS.
Incremental update of a view depends again on its underlying form. In its un-materialized form the cost of an incremental update is the cost of re-execution. For other forms we must distinguish two cases. The first case occurs when the incremental update is done in real-time during the query execution. In this case, the update is

combined with scanning and therefore, the cost of incremental update is subsumed by the scanning cost. This was the main objective of the one pass incremental update algorithms of View Cache. The subsumed cost savings are significant and this was shown by comparing worst case analysis estimations against actual timed experiments. This was especially true for views-on-views because of the elimination of storing and accessing intermediate results.

The second case is when the incremental update of a view is done at times other than scanning. This is the typical case in a data warehouse where updates from multiple sources are applied asynchronously either when they arrive or at scheduled (often off-line) times. The benefit of combining scanning and updating is not a factor any more. Therefore, minimal dereferencing is a good target optimization. Partially materialized views which materialize only the subset of the attributes useful for the incremental update, outer joins instead of joins, or other appropriate attribute caching techniques are best suited. On the other hand, fully materialized views are cumbersome and generate a lot of unnecessary I/O and data movement for just updating views that are to be used in the future.

It should be mentioned here that the issue of self maintenance of views is important. However, the additional information necessary for the incremental update and its storage organization must be well designed since this affects performance. For example, the storage organization of the deltas may have equivalent adverse effects to thrashing if their tuples are scattered in an unclustered space.

2.3 Creation of Data Mining Views

Traditional views are used mainly to hide difficult query structures from a user. Views also provide independence of applications from the schema changes occurring in the database. All changes must be reflected only in the definition of the view and no modification is required in end-user applications. Every access to the view triggers the execution of the query that defines the view. Data mining is an interactive and iterative process and data mining queries tend to be fairly complicated. Data mining views hide the complexity of the algorithm from an application and simplify access to discovered patterns. The following MineSQL statement creates a data mining view V SEQ PATS. The view presents sequential patterns discovered in the CUST TRANSACTIONS table, having the support exceeding 0.2, using the following time constraints: max-gap of 100, min-gap of 1, and no window-size (the default value of 0 is used).

```
CREATE VIEW V SEQ PATS AS
MINE PATTERN MAXGAP 100 MINGAP 1
FROM (SELECT SEQUENCE(T TIME, ITEM)
FROM CUST TRANSACTIONS
GROUP BY C ID)
WHERE SUPPORT(PATTERN)>0.2;
```

Data mining views provide additional independency layer between the database and the end-user application. Slight modifications of algorithm parameters or explored dataset are reflected only in the view definition while the application does not notice any changes. Besides, the user is separated from the technical details of the algorithm and can perform repetitive data mining tasks without knowing the details of syntax of the MINE statement. As with traditional views, every access to the data mining view triggers the execution of the underlying algorithm.

2.4 Refreshing of Materialized Views

The algorithms for pattern discovery are usually time-consuming. Processing time of a data mining query could easily become unacceptable from the point of interactive mining. The solution to this problem is materialization of previously obtained results of data mining queries. A materialized data mining view is a database object storing the results of a data mining query (frequent sets, association rules, sequential patterns). With every materialized view a time period can be associated, after which the view is automatically refreshed. The following statement creates the materialized data mining view MV SEQ PATS. The view is to be refreshed automatically once a week.

```
CREATE MATERIALIZED VIEW MV SEQ PATS
REFRESH 7 AS
MINE PATTERN MAXGAP 100 MINGAP 1
FROM (SELECT SEQUENCE(T TIME, ITEM)
FROM CUST TRANSACTIONS
GROUP BY C ID)
WHERE SUPPORT(PATTERN)>0.2;
```

Materialized data mining views can be refreshed either automatically or on user's demand. In most cases such refresh can be performed by one of the incremental mining algorithms instead of running the complete discovery algorithm. Additional advantage of materialized views is the fact that data mining usually takes place in a data warehouse where changes to base relations (and thus to the stored patterns) do not happen continually over time but are accumulated and loaded to the data warehouse during data warehouse refresh process. The patterns discovered and stored in the materialized view remain valid for a long period of time until next data

warehouse refresh. Validation of patterns can be postponed until next warehouse refresh event

2.4.1 Refreshment According to Quality of Data

2.4.1.1 Accuracy

To attain maximum possible accuracy it is required to refresh MV frequently. The frequency can be varied from intervals between updating to several numbers of days depending on the type of the database object.
eg. The data should be refreshed at least once in a day.

2.4.1.2 Quick Retrieval

To get the data available at the time when required, data should be available as a Materialized View on time, by increasing the refreshment interval parameter will increase the availability of the materialized View. But in this case accuracy will sustain inversely proportional to the interval.

2.4.1.3 Optimized Retrieval

As per the network condition the refreshment parameter should be moderate enough to accommodate accuracy and availability. When speed matters, the time duration between consecutive refreshment can be delayed according to bandwidth status. So depending upon the requirement we can sacrifice over accuracy, speed, burden on network. According to property of database,
Number of read on database object corresponding to the Materialized View = x
Number of write on database object corresponding to the Materialized View = y
Case 1: $x >> y$
In this situation we can use big interval for refreshment so we can have more accuracy even with less refreshment. So, in this case refreshment cost will be greatly reduced.
Case 2: $x << y$
Since updation on database exceeds so we need more frequently refreshment it will greatly increase refreshment cost but it is really necessary to maintain the accuracy.
Case 3: $x =\sim y$
In this case as per our requirement we can vitiate refreshment interval as per the accuracy or network bandwidth. Refreshing time can be decided as per the network traffic, for example in some e countries traffic in day time is more than night hours, we can prefer nights to have maximum possible network speed.

2.5 Concurrency

In generic concurrency control mechanisms, immediate materialized aggregate join view maintenance becomes extremely problematic—the addition of a materialized aggregate join view can introduce many lock conflicts and/or deadlocks that did not arise in the absence of this materialized view.
The V locking protocol is designed to support concurrent, immediate updates of materialized aggregate join views without engendering the high lock conflict rates and high deadlock rates that could result if two-phase locking with S and X lock modes were used. This protocol borrows from the theory of concurrency control for associative and commutative updates, with the addition of a latch pool to deal with insertion anomalies that result from some special properties of materialized view updates. Perhaps surprisingly, due to the interaction between locks on base relations and locks on the materialized view, this locking protocol, designed for concurrent update of aggregates, also supports direct propagate updates and materialized non aggregate join view maintenance.

2.6 Using Materialized View in Query Execution

In many cases contents of the materialized view can be used to answer a query that is similar to the query defining the view. For example, if the query defining the view includes a given query Q then the latter can be answered by simply reading the contents of the view and pruning those patterns that do not meet the conditions formulated in. The key issue is identification of syntactic differences leading to situations in which one query can be efficiently answered using the results of another query. In our analysis we consider only materialized views containing frequent sets and sequential patterns. Even if the final goal is discovery of association rules, we propose to materialize frequent sets for two reasons. As it was also observed by other researchers: generation of rules from item sets is straightforward, and materialized item sets can be used to answer more item set and rule queries.

3. Conclusions

Views are the most important asset of the relational model. They provide a uniform conceptual and implementation model of relational programs, derived data, indexes, and aggregated derived data. Selecting views to materialize is one of the most important decisions in designing a data warehouse. Materialized data mining views are physical data warehouse structures, created explicitly or implicitly, used to store pre computed results of selected data mining queries. By assigning moderate value to refreshment parameter we can make efficient data retrieval. Data mining algorithms can utilize this concept to create Materialized Views with maximum efficient retrieval in favorable conditions.

References

[1] R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large databases. In Proc. Of the 1993 ACM SIGMOD Conference, 1993.

[2] R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules. In Proc. of the 20th VLDB Conference, 1994.

[3] R. Agrawal, R. Srikant. Mining Sequential Patterns. In Proc. of the 11th ICDE Conference, 1995.

[4] R. Agrawal, R. Srikant. Mining Sequential Patterns: Generalizations and Performance Improvements. In Proc. of the 5th EDBT Conference, 1996.

[5] D. W.-L. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In Proc. of the 12th ICDE Conference, 1996.

[6] T. Imielinski, H. Mannila. A Database Perspective on Knowledge Discovery. Communications of the ACM, 39(11), 1996.

[7] B. Nag, P. Deshpande, D. J. DeWitt. Using a Knowledge Cache for Interactive Discovery of Association Rules. In Proc. of the 5th KDD Conference, 1999.

[8] S. Parthasarathy, M. J. Zaki, M. Ogihara, S. Dwarkadas. Incremental and interactive sequence mining. In Proc. of the 1999 ACM CIKM Conference, 1999