

Ad-Hoc Association-Rule Mining within the Data Warehouse

Svetlozar Nestorov
Department of Computer Science
The University of Chicago
Chicago, IL, USA
evtimov@cs.uchicago.edu

Nenad Jukić
School of Business Administration
Loyola University Chicago
Chicago, IL, USA
njukic@luc.edu

Abstract

Many organizations often underutilize their already constructed data warehouses. In this paper, we suggest a novel way of acquiring more information from corporate data warehouses without the complications and drawbacks of deploying additional software systems. Association-rule mining, which captures co-occurrence patterns within data, has attracted considerable efforts from data warehousing researchers and practitioners alike. Unfortunately, most data mining tools are loosely coupled, at best, with the data warehouse repository. Furthermore, these tools can often find association rules only within the main fact table of the data warehouse (thus ignoring the information-rich dimensions of the star schema) and are not easily applied on non-transaction level data often found in data warehouses. In this paper, we present a new data-mining framework that is tightly integrated with the data warehousing technology. Our framework has several advantages over the use of separate data mining tools. First, the data stays at the data warehouse, and thus the management of security and privacy issues is greatly reduced. Second, we utilize the query processing power of a data warehouse itself, without using a separate data-mining tool. In addition, this framework allows ad-hoc data mining queries over the whole data warehouse, not just over a transformed portion of the data that is required when a standard data-mining tool is used. Finally, this framework also expands the domain of association-rule mining from transaction-level data to aggregated data as well.

1. Introduction

Data warehousing has become a standard practice for most large companies worldwide. The data stored in the data warehouse captures many different aspects of the business process such as manufacturing, distribution, sales, and marketing. This data reflects explicitly and

implicitly customer patterns and trends, business practices, strategies, know-how and other characteristics. Therefore, this data is of vital importance to the success of the business whose state it captures, which is why companies choose to engage in the relatively expensive undertaking of creating and maintaining the data warehouse (a recent study [13] reports the median cost of \$1.5 million for creating a data warehouse with additional \$0.5 million for annual operating cost).

While some information and facts can be gleaned from the data warehouse directly, much more remains hidden as implicit patterns and trends. The discovery of such information often yields important insights into the business and its customers and may lead to unlocking hidden potentials by devising innovative strategies. The discoveries go beyond the standard on-line analytical processing (OLAP) which mostly serves reporting purposes (albeit in an increasingly complex and sophisticated manner).

One of the most important and successful methods for finding new patterns is association-rule mining. Typically, if an organization wants to employ association-rule mining on their data warehouse data, it has to acquire a separate data mining tool. Before the analysis is to be performed, the data must be retrieved from the database repository that stores the data warehouse, which is often a cumbersome and time-consuming process. The vendors of data management software are becoming aware of the need for integration of data mining capabilities into database engines, and some companies are already allowing for tighter integration of their database and data mining software (e.g. IBM's DB2 Database and Intelligent Miner or NCR's Teradata Database and Teradata Warehouse Miner) [9]. While this does represent an improvement, it still does not eliminate the need for additional software. In this paper we describe a direct approach to association-rule data mining within data warehouses that utilizes the query processing power of the data warehouse itself without using a separate data mining tool. In addition, this approach is capable of answering a variety of questions based on the entire set of

data stored in the data warehouse, which is in contrast to regular association-rule mining that is limited to portions of the data warehouse. Another limitation of regular association-rule mining is that it requires transaction-level data, even though much of the data stored in business-related data warehouses is in aggregated form. In this paper, in addition to presenting a direct approach to association-rule data mining of entire data warehouses, we present a methodology for association-rule mining of aggregated data.

The remainder of this paper is organized as follows: Section 2 gives a brief overview of association-rule mining and illustrates its applicability in a data warehouse environment; Section 3 introduces the new approach of Extended Association-Rule Mining; Section 4 discusses the issue of association-rules mining from aggregate (non-transaction level) data in data warehouses; Section 5 introduces the implementation architecture for Extended Association-Rule Mining; and Section 6 describes the results of an experimental study. Finally, Section 7 gives the conclusions and indicates the future work.

2. Association-Rule Data Mining in Data Warehouses

Dimensional modeling [8], which is the most prevalent technique for modeling data warehouses, organizes tables into fact tables containing basic quantitative measurements of a business subject and dimension tables that provide descriptions of the facts being stored. The data model that is produced by this method is known as a *star-schema* [5] (or a star-schema extension such as *snowflake* or *constellation*). Figure 1 shows a simple star-schema model of a data warehouse for a retail company.

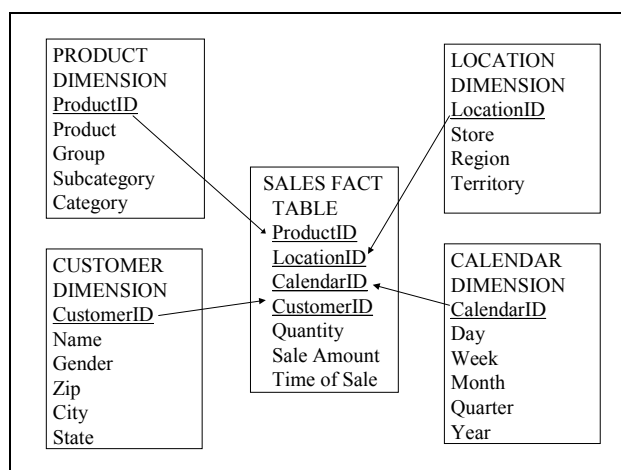


Figure 1. Example Star-Schema

The fact table contains the sale figures for each sale transaction and the foreign keys that connect it to the four dimensions: Product, Customer, Location, and Calendar.

Standard association-rule mining [1][2] discovers correlations among items within transactions (the prototypical example of utilizing association-rule mining is determining what things are found together in a basket at a checkout line at the supermarket; hence the often-used term: market basket analysis [4]). The correlations are expressed in the following form:

Transactions that contain X are likely to contain Y as well.

Letters X and Y represent sets of items. There are two important quantities measured for every association rule: *support* and *confidence*. The support is the fraction of transactions that contain both X and Y. The confidence is the fraction of transactions containing X, which also contain Y. Intuitively, the support measures the significance of the rule, so we are interested in rules with relatively high support. The confidence measures the strength of the correlation, so rules with low confidence are not meaningful. In practice, the discovery of association rules has two phases. In the first phase, all sets of items with high support (often called *frequent itemsets*) are discovered. In the second phase, the association rules among the frequent itemsets with high confidence are constructed. Since the computational cost of the first phase dominates the total computational cost [2], association-rule mining is often defined as the following question over transactional data, where each transaction contains a set of items:

What items appear frequently together in transactions?

A typical data warehouse, however, has multiple dimensions, which are completely ignored by the above single-dimension question. Consider the data warehouse depicted by Figure 1. The standard association-rule mining question for this environment would be:

What products are frequently bought together?

This question examines the fact table as it relates to the product dimension only. An analyst at the corporate headquarters may ask the following question:

What products are frequently bought together in a particular region and at a particular month?

This question examines multiple dimensions of the fact table. Unfortunately, standard association-rule mining (ARM) algorithms cannot find the answer to this question directly. In fact, ARM may not discover any association rules in situations when there are several meaningful associations that involve multiple dimensions. The following example illustrates an extreme case scenario of this situation.

Example 1 A major retail chain operates stores in two regions: *South* and *North*; and divides the year into two

seasons: *Winter* and *Summer*. The retail chain sells hundreds of different products including sleeping bags and tents. Table 1 shows the number of transactions for each region in each season as well as the percentage of transactions that involved a sleeping bag, a tent, or both.

Table 1.

		North			South				
		Total x 1000	% Bags	% Tents	% Both	Total x 1000	% Bags	% Tents	% Both
W		200	4	1	0	100	2	4	1
S		100	2	4	2	200	4	2	1

Suppose we are looking for association rules with 1% support and 50% confidence. Let's first consider the transactions in all regions and all seasons. There are 600 thousand transactions and only 5 thousand of them involve both a sleeping bag and a tent, so the support of the pair of items is .83% (less than 1%). Thus, no association rule involving sleeping bags and tents will be discovered. Let's consider the transactions in the North and the South regions separately. There are 300 thousand transactions in each region. In the North sleeping bags and tents appear together in 2 thousand transactions, so their support is .66%. Thus, no association rule involving both of them will be discovered. In the South, sleeping bags and tents appear together in 3 thousand transactions, so their support is 1%. The confidence of **sleeping bag** → **tent** is 30% and the confidence of **tent** → **sleeping bag** is 37.5% (both less than 50%), so no rules involving both sleeping bag and tent will be discovered in the South either. Similarly, no associations will be discovered between sleeping bags and tents when we consider all the transactions in each season separately. These conclusions (no association rules) are rather surprising because *if each region and season were considered separately* the following association rules would be discovered:

- In the North during the summer: **sleeping bag** → **tent** (sup=2%, conf=100%)
- In the North during the summer: **tent** → **sleeping bag** (sup=2%, conf=50%)
- In the South during the summer: **tent** → **sleeping bag** (sup=1%, conf=50%)
- In the South during the winter: **sleeping bag** → **tent** (sup=1%, conf=50%)

Standard ARM cannot discover these association rules directly. Instead, the data need to be preprocessed or transformed in one of the following ways:

Solution 1. Expand the definition of an item. Define an item to be not just a product, but a region or a season as well. Each transaction (basket) contains the products as

well as two additional items: the season when the purchase was made and the region where the store is located. Now we can apply any standard ARM algorithm and filter out all rules that do not contain a season, a region, and at least two products.

There are obvious problems with this approach. The initial number of rules generated by the algorithm will be several orders of magnitude larger than the number of rules of the required form. Consequently, the amount of time and computational resources spent to eliminate spurious rules will be rather excessive.

Solution 2. Partition the data into several subsets according to the season and the region of the transaction. Apply ARM to each of the subsets.

The drawback of this approach is the need for partitioning of the data warehouse every time a question is asked. Each subsequent new request will require a pass over all the data in order to re-partition the data warehouse. Thus, this approach will also require an excessive amount of time and computational resources.

Solution 3. Redefine the definition of an item. Define an item to be a triple consisting of a product, a region, and a season. Apply ARM the new market basket data.

While this approach does not require filtering out spurious rules or partitioning the data warehouse, it still has a major disadvantage of requiring an additional step (creating the redefined items – triples in this case), which again requires significant additional time and computational resources, due to the explosion of the number of different items.

As illustrated by the above example-solutions, regular ARM based approaches for finding answers to questions involving multiple dimensions can overwhelm the system with requests for resources. In the following section, we describe a different and more feasible approach.

3. Extended Association Rules

This section introduces a new concept of extended association rules. Standard association rules can express correlations between values of a single dimension of the star schema. However, as illustrated in Example 1, values of the other dimensions may also be correlated. Furthermore, some associations become evident only when multiple dimensions are involved. In Example 1, sleeping bags and tents appear uncorrelated in the sales data as a whole, or even when the data is considered separately by region or season. Yet, several association rules are discovered if the focus is on the data for a particular region during a particular season. One such rule is **sleeping bag** → **tent** for the transactions that occurred in the North region during the Summer season. This association rule involves more than just a pair of

products; it also involves the location as well as the time of the transaction. In order to capture this additional information, we augment the association rule notation as follows:

sleeping bag \rightarrow **tent** (region=north, season=summer)

Formally, we define extended association rules as follows:

Definition: Let T be a data warehouse where X and Y sets of values of the item-dimension. Let Z be a set of equalities assigning values to attributes of other (non-item) dimensions of T . Then, $X \rightarrow Y (Z)$ is an extended association rule with the following interpretation: Transactions that satisfy Z and contain X are likely to contain Y .

While extended association rules are natural extensions of standard association rules, they contain more detailed and diverse information that offer several distinct advantages.

First, the phenomenon that underlines an extended rule can be explained more easily. For example if we change the percentages shown in the last column of Table 1 from 1% to 2%, standard ARM would find the rule **tent** \rightarrow **sleeping bag** within the specified confidence and support threshold (1% and 50%) for all sales. However, only the extended rules would give the explanation that the correlation of sales between tents and sleeping bags holds exclusively in warmer weather conditions (because all found extended rules would contain either *south* region or *summer* season).

Second, extended rules tend to be local, in a sense of location or time, and thus more actionable. For example, it is much simpler to devise a marketing strategy for sleeping bags and tents that applies to a particular region during the a particular season than a general strategy that applies to all regions during all seasons.

Third, for the same support threshold, the number of discovered extended association rules is likely to be much less than the number of standard rules. The reason is that extended rules involve more attributes and thus transactions that support them fit a more detailed pattern than standard rules. Even though the overall number of extended rules is likely to be much less than the number of regular rules, there may be some items that are involved only in extended rules (as shown in Example 1). Intuitively, standard rules show associations of a coarser granularity while extended rules show finer associations.

Augmenting standard association rules has previously been proposed in [6]. This approach applies attribute-oriented generalization algorithm to characteristic attributes of discovered association rules. Our approach differs from [6] in that we involve all attributes (not just items) in finding the frequent itemsets. Thus, there are

more opportunities for optimizations (see Section 5.2) and the discovered rules have finer-granularity.

Before we describe the mining process for extended association rules, let us recall standard association-rule mining. Given a set of transactions, and support and confidence thresholds, it discovers association rules among items that have support and confidence higher than the chosen thresholds. With this setting, the only parameters that the user controls directly are the support and confidence thresholds. Very often the result of the mining is one of two extremes. The number of discovered rules is either overwhelmingly large (tens of thousands) or very small (dozens) [12]. In both cases, the user has no other recourse but to revise the support and confidence and run the mining process again (as a post-processing step, the user can choose to focus on certain items but that goes against the notion of mining; in fact, given the unlimited amount of time, the user may very well compute the support and confidence of the items directly using OLAP.)

Since the mining process often takes hours [10], most users are not likely to continually refine the thresholds. Thus, the mining process is often seen as not particularly useful to non-technical users.

In contrast, the process of mining extended association rules requires more input from the user and thus affords more control and lends itself to customization. Since extended association rules involve attributes of non-item dimensions, the user needs to specify these attributes. For example, the user can ask for association rules that involve a region and a season in addition to products. Note that the user *does not* need to specify the *particular values* of the attributes. The actual values are discovered as part of the mining. This mining process differs from the one-size-fits-all approach for standard association rules. Different users can ask mining questions involving different attributes of the dimensional hierarchies. For example, a marketing executive who is interested in setting up a targeted marketing campaign can ask the following:

Find products bought frequently together by customers of a particular gender in a particular region.

A sample association rule discovered by this question may be: **ice cream** \rightarrow **cereal** (region=California, gender=female). Discovering that ice cream and cereal sell well together in California with female customers can facilitate creating a more focused and effective marketing campaign that is at the same time less costly (e.g. adds for ice cream and cereal can appear on consecutive pages of a local female-audience oriented magazine).

On the other hand, a vice president in charge of distribution may ask the following:

Find products bought frequently together in a particular city at a particular month.

A sample association rule discovered by this question may be: **sleeping bag** → **tent (city=Oakland, month=May)**. Such a rule is of more value for managing and planning the supply-chain than the equivalent standard association rule (e.g. it can indicate that the shipping of sleeping bags and tents to the certain stores should be synchronized, especially during certain time periods).

In Section 5 we will describe the implementation details of Extended Association Rule Mining. However, before that, in the following section we will examine yet another way of finding out more information from data warehouses.

4. Mining from Aggregated Data

Standard association-rule mining works with transaction-level data that reflects individual purchases. However, the industry practice for many large corporations is to keep such data in the data warehouse only within a certain limited time horizon (e.g. 60 days, 1 quarter, etc.) Afterwards transaction-level data is stored off-line on a medium suited for bulk management of archival data, such as magnetic tape or optical disk [7]. Such data is still available electronically but it is not on-line because it does not reside in the data warehouse. The data warehouse continues to store summarizations of the transaction-level data; for example data aggregated by day. Figure 2 shows the schema of the data warehouse from Figure 1 with the product sales data aggregated by day for each store.

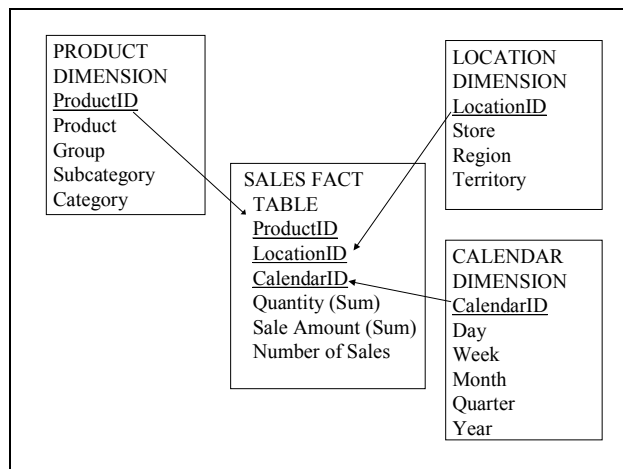


Figure 2. Example Star-Schema for Aggregated Data

This situation is not pertinent for most association-rule mining tools (if the transaction-level data is available on off-line media) because the tools do not use the query

processing capabilities of the data warehouse. Instead, as we mentioned in introductory part of the paper, most such tools require loading of the data (which can be an especially slow process with off-line data). Our approach, however, is to leverage the query-processing power of the database management system (DBMS). Thus, we need to define the concept of association rules on aggregated data.

First, let's consider the granularity of the data in the data warehouse. Aggregated data is created by rolling up the transaction-level data by one or more attributes. One of the most common cases is aggregating data by some measure of time. In the example case shown in Figure 2 the, data contains information about the daily sales of items instead of individual purchases. Thus, the base unit in the data warehouse is no longer an individual transaction but rather the daily sales at a store.

Next, in order to define association rules for aggregated data we need to re-examine the meaning of *together* in questions of type:

Find products bought frequently together.

Note that we can find out the total number of times each item appears in transactions. However, we no longer know the exact number of transactions that contained two or more particular items from the daily sales data. We can approach this problem in several different ways.

Approach 1

The approach that most closely follows the original definition of association rules is to assume that a certain fraction (percentage) of all possible transactions at each store during each time period (e.g. day, if the data is aggregated by day) contain the two items. A reasonable approach is to take $f * \min(S_x, S_y)$ where S_x and S_y are the total sales for items X and Y on a given day in a give store. The actual selected value of fraction f can even be somewhat arbitrary as long as it is constant for all stores and days. In fact, we can incorporate the chosen value of f into the *support* measure since the *support* inequality will be:

$$\sum_{store, day} (f * \min(S_x, S_y)) \geq support\ threshold$$

So if we adjust the *support threshold* accordingly (divide by f) we can get the results without changing the association rule algorithm.

The above-described approach (Approach 1) is based on approximating togetherness. A different strategy involves redefining the meaning of *together* where we, instead of looking for pairs of items that are sold during the same transaction, look for pairs of items that are sold on the *same occasion*. In a case when the data is aggregated by day, an occasion represents a day in a store. The information that a store sells many items of X and Y on the same day, even if the items are not sold together in

every instance, is quite meaningful and useful (e.g. for improving the distribution and shipping process).

Approach 2

One possible occasion-based approach involves breaking the support into two parts to better reflect the granularity of the mined data. Intuitively, we are looking for pairs of items that sell well together on many occasions. The meaning of "sell well together" is that both items must have been sold more than a certain threshold at the same store on the same day. Formally, we define the condition as:

$$\sum_{store, day} (1 \text{ if } \min(S_x, S_y) \geq \text{sales threshold, else } 0) \geq \text{occasion threshold}$$

In principle, we first find, for every pair of items, all day-store pairs (corresponding to occasions) for which the items were both sold more than the sales threshold. Then, we count the number of such pairs, and if the number exceeds the occasion threshold we add the pair to the discovered rules. For example, a pair of items is bought frequently if there are at least 100 different occasions (*occasion threshold = 100*) on which both items are bought separately in more than 10 transactions (*sales threshold = 10*).

Approach 3

Another approach is to split the support condition into three parts. Intuitively, we are looking for pairs of items that sell well together on many occasions in many stores. Formally, we have the following three conditions:

- Let *O* be the set of occasions such that $\min(S_x, S_y) \geq \text{sales threshold}$,
- Let *P* be the set of stores such that the number of occasions in *O* for each store $\geq \text{day threshold}$
- Consider *X* and *Y* frequent only if the number of stores in *P* is $\geq \text{store threshold}$.

For example, find all pairs of items such that they are both sold more than 10 times (*sales threshold = 10*) in 10 stores (*store threshold = 10*) on 10 different days (*day threshold = 10*). Note that this approach is a specialization of the occasion-based strategy. In other words, any pair of items that are bought frequently, based on this definition, will also be considered bought frequently according to the occasion definition. In practice, the choice of approach will be based on its relevance for the organization and user and its actionability.

After defining the association rules for aggregate data, we can define the extended rules in the same way as we did with transaction-level data. For example, if we are looking for items sold together in a particular month we can restate the previous example as follows: Find all

pairs of items such that they are both sold more than 10 times in 10 stores on 10 different days of the same month.

In the last two sections we discussed new approaches to association-rule mining of data warehouses. The following section will discuss the implementation of these approaches within an operational system.

5. System Architecture and Implementation

In this section we describe our implementation of extended association rules within the data warehouse. This implementation is granularity-independent as it can accommodate both transaction and non-transaction level data. The basic architecture of our system is shown in Figure 3.

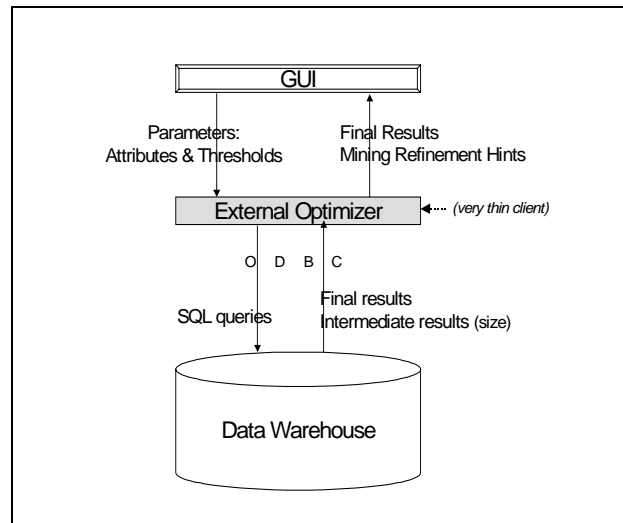


Figure 3. System Architecture for the Ad-Hoc Data Mining System

The most notable feature of the system architecture is the tightly coupled integration with the relational database that powers the data warehouse. The benefits of this integration are threefold:

1. No data leaves the data warehouse so we reduce the redundancy and avoid any privacy, security, and confidentiality issues related to data movement.
2. The relational database does all query processing, so we leverage the computational and storage resources of the data warehouse
3. Extended association rules can be mined from any set of tables within the relational database that is storing the data warehouse, so we enable wide-range ad-hoc mining.

5.1 The Mining Process

The mining process starts with the user defining the extended association rule by choosing the non-item attributes to be involved (the system can accommodate regular ARM as well - if a user chooses zero non-item attributes, regular ARM will be performed).

The choice can be made through a simple interface with pull down menus for each dimension. In cases when non-transaction level data is being mined, the user can also choose an appropriate function for implementing “frequently together”. The default is Approach 1 (presented in Section 4) that mimics the original definition of standard association rules, but the user can also choose one of the other approaches presented in Section 4.

Finally, the user also specifies the support threshold (and possibly other threshold arising from the choice of “frequently together” function.) However, our choice of architecture allows certain flexibility about choosing threshold. Later in this section, we show how the threshold can be changed midstream.

Once all parameters are chosen, the next step of the mining process is the creation of a sequence of SQL queries that implements the extended association rule specified by the user’s choice of parameter values. There are many different SQL sequences that can implement the same rule, so the choice of the sequence is made according to an optimization algorithm (which will be outlined in sub-section 5.2).

Once the optimal sequence is selected, the relational database starts to execute its SQL queries one by one. The results of each query remain in the data warehouse in temporary tables. Only the sizes of the intermediate results are sent to the external optimizer (as shown in Figure 3). Herein lies an opportunity to make the mining process interactive. If the size of certain intermediate results is too large then we can expect that the complete mining process will take a long time and possibly generate too many rules. In this case, the system can alert the user and suggest revising the support threshold upwards. Similarly, if the size of certain intermediate results is too small and we expect that the number of mined rules is going to be very small, we can advise the user to decrease the support threshold.

The last query of the sequence computes the discovered rules. The rules are sent to the external optimizer, which displays them for the user.

The example of final result (set of extended association rules) will be shown in Section 6.

First, however, we will describe the SQL sequence optimization algorithm.

5.2 Optimization Algorithm

The algorithm involves several different parameters that are database specific. The following example illustrates the general optimization principles without going into system specific issues.

Example 2: Consider mining extended association rules from the data warehouse shown in Figure 1 based on the following question: *Find products bought frequently together by customers from a particular zip-code in a particular month.*

This question involves four tables, namely Product, Customer, Calendar, and Sales. Typically, the size of a fact table (such as Sales) will be several orders of magnitude bigger than the size of any of the dimension tables. In order to make the example concrete, suppose that the sizes and attribute cardinalities for these tables are as follows:

- Table Product has 10 thousand tuples (records)
- Table Customer has 10 thousand tuples within 100 different zip-codes
- Table Calendar has 300 tuples within 12 different months
- Table Sales has 1 million tuples

Furthermore, suppose that the support threshold is 100. Ultimately, in order to find pairs of products bought frequently together we have to join some portion of the Sales table with itself. A naïve approach, writing the query directly in SQL is shown in Figure 4.

```
SELECT A.ProductID, B.ProductID, Zip,
Month, COUNT(*)
FROM Sales A, Sales B, Customers,
Calendar
WHERE A.CustomerID = B.CustomerID
AND A.CalendarID = B.CalendarID
AND Calendar.CalendarID = A.CalendarID
AND Customers.CustomerID = A.CustomerID
GROUP BY A.ProductID, B.ProductID, Zip,
Month
HAVING COUNT(*) >= 100
```

Figure 4: Naïve SQL Query

The cost of this join is likely to dominate the cost of the mining process so the optimization goal is to reduce the size of the portion of Sales before we do the self-join. This optimization is crucial for the success of our tightly coupled approach. Mining without such optimization (by running the naïve query) will be extremely slow and may require massive additional storage space for the internal

intermediate results (as large as the entire data warehouse).

Standard association rule employs the so-called *a-priori technique* [3] to accomplish a similar optimization goal. The crux of a-priori is the observation that a set of items can be frequent only if all proper subsets are also frequent. Based on this observation, a-priori applies step-wise pruning to the sets of items that need to be counted, starting by eliminating single items that are infrequent.

The a-priori technique works for association rules but we can augment it, similar to *query-flocks* method [11], to accommodate the extended association rules. Consider the following three questions:

Question A: *Find all products bought in at least 100 transactions.*

Question B: *Find all products bought in at least 100 transactions in the same month.*

Question C: *Find all products bought in at least 100 transactions by customers with the same zip.*

We can use the result of any of the three questions to reduce the size of the portion of the Sales table by semi-joining Sales with the result (semi-join of table A with table B produces all tuples of A that would participate in join of A and B.) The problem is which one of the three to choose.

Consider question A. The expected number of transactions per product is 100 (size of Sales / size of Product) so it is unlikely that many products will be eliminated by question A.

Consider question B. We can expect that there will be on average less than 9 transactions that contain a particular product in a particular month (size of Sales / (size of Product * number of months) = 8.33), so question B is likely to reduce the number of relevant Sales records by a significant factor.

Consider question C. The expected number of transactions per product, zip-code combination is 1 (size of Sales / (size of Product * number of zip-codes)) so the reduction will be even more significant.

Thus, if we only consider the extent of the reduction, we will choose question C. However, the problem is more complicated since both question B and question C involve joining the fact table (Sales) with a dimension (Calendar and Customer respectively). On the other hand, question A can be computed directly from Sales. The exact tradeoff between the cost of the join and the amount of reduction can be made explicit by taking into account various system parameters such as the type of database system, amount of main memory and buffer pool, processor and disk speed, etc. (The number of reduction choices is at least exponential in the number of

dimensions involved in the question, which can make the optimization problem intractable for large number of dimensions. In practice, however, data warehouse models typically have less than 20 dimensions [8]. Furthermore, data-mining questions involving more than a small number of dimensions are unlikely to be useful.)

In this example case, it is likely that question B is the best choice since the reduction is very significant and the join involves a small table (300 tuples.) Figure 5 shows the optimized sequence of SQL queries (optimization based on question B), result-equivalent to the Naïve SQL query shown in Figure 4.

```
1.
INSERT INTO FrequentProductIdMonth
SELECT ProductId, Month
FROM Sales, Calendar
WHERE Sales.CalendarID = Calendar.CalendarID
GROUP BY ProductId, Month
HAVING COUNT(*) >= 100

2.
INSERT INTO ReducedSales1
SELECT <all attributes of Sales>
FROM Sales S, Calendar C,
FrequentProductIdMonth F
WHERE S.ProductId = F.ProductId
AND S.CalendarID = Calendar.CalendarID
AND C.Month = F.Month

3.
INSERT INTO FrequentProductIdMonthZip
SELECT ProductId, Month, Zip
FROM ReducedSales1 S, Calendar, Customers
WHERE S.CalendarID = Calendar.CalendarID
AND Customers.CustomerID = S.CustomerID
GROUP BY ProductId, Month, Zip
HAVING COUNT(*) >= 100

4.
INSERT INTO ReducedSales2
SELECT <all attributes of Sales>
FROM ReducedSales1 S, Calendar C,
FrequentProductIdMonthZip F, Customers D
WHERE S.ProductId = F.ProductId
AND S.CalendarID = Calendar.CalendarID
AND C.Month = F.Month
AND D.Zip = F.Zip
AND D.CustomerID = S.CustomerID

5.
SELECT A.ProductId, B.ProductId, Zip, Month,
COUNT(*)
FROM ReducedSales2 A, ReducedSales2 B,
Customers, Calendar
WHERE A.CustomerID = B.CustomerID
AND A.CalendarId = B.CalendarId
AND Calendar.CalendarId = A.CalendarId
AND Customers.CustomerID = A.CustomerID
GROUP BY A.ProductId, B.ProductId, Zip, Month
HAVING COUNT(*) >= 100
```

Figure 5: Equivalent Optimized Sequence of SQL Queries

Query 1 creates a table of product-month pairs for which there are more than 100 sales transactions in the Sales table. Query 2 creates a table ReducedSales1, which extracts from the Sales fact table only those sale transactions that involve product-month pairs resulting from Query 1. The result of Query 2, a reduced Sales table ReducedSales1, is based on question B. This reduced table is used in Query 3 to create a table of product-month-zip triples for which there are more 100 sales transactions in the Sales fact table and later in Query 4 to create a table ReducedSales2 that extracts all sale transactions that involve product-month-zip triples resulting from Query 3. We did not have to use the original Sales fact table, because we know that all possible transactions that satisfy conditions of Query 3 and 4 are replicated in much smaller ReducedSales1 table. Finally, Query 5 performs a join of the ReducedSales2 table with itself in order to find the final result.

6. Experiments

In this section we describe an experimental performance study of mining real world data that uses the system and methods described in previous sections of this paper.

For our experiments we used Oracle 9i relational DBMS running on a 128KB Linux machine. The data we used reflects the aggregated daily purchases for a major US-based retailer during a six-month period and conforms to the schema shown in Figure 2. The sizes of the tables are as follows:

- The Product table has around 5000 tuples.
- The Location table has around 2000 tuples with about 150 different Regions.
- The Calendar table has around 200 tuples.
- The Sales table has around 3 million tuples.

As we discussed in Section 4, standard association rules are not defined for such data. However, as we also outlined in Section 4, we can still mine association rules by approximating or redefining the meaning of “frequently together.” In our experiments we mined both association rules and extended association rules. In this paper, we present the results of two queries that are typical for our experiments.

The first query finds regular association rules based on the following question:

Find all pairs of items that appear together in more than 9000 transactions

Our implementation of this query (code available by contacting the authors) involves a pruning of all

infrequent items as described in Section 5.2. The result of the query is thirteen pairs of items shown in Table 2.

Table 2

PRODUCTID1	PRODUCTID2	SUM
600	8130	16853
3060	8130	10226
7740	8130	9811
8130	8280	12473
8130	8310	13717
8130	8550	9057
8130	13380	11192
8130	13890	11906
8130	15150	9642
8130	15240	11541
13890	15150	9498
13890	15240	11157
15150	15240	10749

The second query finds extended association rules using the Approach 1 (Section 4) to approximating “frequently together”, and it is based on the following question

Find all pairs of items that appear together in more than 300 transactions in the same region.

Our implementation of this query (code available by contacting the authors) involves pruning all item-region pairs that do not appear in more than 300 transactions. The result of this query, shown in Table 3, is a set of fifteen rules, where each rule involves two items and a region.

Table 3

PRODUCTID1	PRODUCTID2	REGIONID	SUM
600	8130	48	521
8130	13380	64	521
13890	15240	64	561
15150	18420	62	562
123660	123690	54	565
123660	123720	53	556
123660	123780	54	507
123690	123720	53	736
123690	123720	54	803
123690	123840	53	535
123720	123840	53	1081
164490	168420	62	599
167520	167640	17	530
168120	168420	62	555
168420	169650	62	575

Comparing the two queries and their implementation in our system we make the following observations. The extended association rule mining takes significantly less

time to produce about the same number of rules as association rule mining. Furthermore, in order to find a similar (manageable) number of rules, the support threshold for the association rules had to be set significantly larger than the threshold for extended rules. This fact supports our claim that association rules find coarser granularity correlations among items while extended rules discover finer patterns.

The experiments also validate the viability of our tightly-coupled integration with the relational DBMS. The running times for all of our queries are measured in seconds and minutes; and there is still a room for significant performance improvement by, for example, upgrading hardware or through the addition of indexing. In practice, in cases when a data warehouse is heavily utilized with OLAP and reporting requests, a separate data mart dedicated exclusively to data mining can be a good alternative in order to minimize the hits on the enterprise data warehouse and improve overall performance.

7. Conclusions

In this paper, we presented a new data-mining framework that is tightly integrated with the data warehousing technology. In addition to integrating the mining with database technology by keeping all query-processing within the data warehouse, our approach introduces the following two innovations:

- Extended association rules using the other non-item dimensions of the data warehouse, which results in more detailed and ultimately actionable rules.
- Defining association rules for aggregated (non-transactional) data.

We have shown how extended association rules can enable organizations to find new information within their data warehouses relatively easily, utilizing their existing technology. We have also defined several exact approaches to mining repositories of aggregated data, which allows companies to take a new look at this important part of their data warehouse.

We have conducted experiments that implement our approach on real-life aggregated data and the results support the viability of our integration approach as well as the appropriateness of extended association rules.

In our future work we plan to elaborate on the optimization algorithm. We also intend to undertake a further performance study with larger data sets, using different hardware platforms and various types of indexes.

References

[1] Agrawal R., Imielinski T. and A. Swami. Mining Association Rules Between Sets of Items in Large Databases.

Proceeding of ACM SIGMOD International Conference. (1993), 207-216.

[2] Agrawal R. and Srikant R. Fast Algorithms for Mining Association Rules. *Proceeding of International Conference On Very Large Databases VLDB.* (1994), 487-499.

[3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. Fast Discovery of Association Rules. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press (1996).

[4] Berry, M. and Linoff, G. *Data Mining Techniques for Marketing, Sales and Customer Support.* Wiley (1997).

[5] Chaudhri, S. and Dayal, U. An overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record.* 26 (1), (1997), 65-74.

[6] Hilderman, R.J., Carter, C.L., Hamilton, H.J., and Cercone, N. Mining Association Rules from Market Basket Data Using Share Measures and Characterized Itemsets. *International Journal of Artificial Intelligence Tools.* 7 (2), (1998), 189-220.

[7] Inmon, W. H. *Building the Data Warehouse.* Wiley (1996).

[8] Kimball, R., Reeves, L., Ross, M., and Thornthwhite, W. *The Data Warehouse Lifecycle Toolkit.* Wiley (1998).

[9] Leavitt, N. Data Mining for the Corporate Masses. *IEEE Computer.* 35 (5), (2002), 22-24.

[10] S. Sarawagi, S. Thomas, R. Agrawal. Integrating Mining with Relational Database Systems: Alternatives and Implications. *Proceedings of ACM SIGMOD Conference* (1998), 343-354

[11] S. Tsur, J. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, A. Rosenthal. Query Flocks: A Generalization of Association-Rule Mining. *Proceedings of ACM SIGMOD Conference,* (1998), 1-12.

[12] Wang, K., He Y., and Han J. Mining Frequent Itemsets Using Support Constraints. *Proceedings of International Conference on Very Large Databases VLDB,* (2000), 43-52

[13] Watson, H. J., Annino, D. A., and Wixom, B. H. Current Practices in Data Warehousing. *Information Systems Management.* 18 (1), (2001).