A New SQL-like Operator for Mining Association Rules

Rosa Meo
Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy
rosimeo@polito.it
Giuseppe Psaila
Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy
psaila@elet.polimi.it
Stefano Ceri
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
ceri@elet.polimi.it

Abstract

Data mining evolved as a collection of applicative problems and efficient solution algorithms relative to rather peculiar problems, all focused on the discovery of relevant information hidden in databases of huge dimensions. In particular, one of the most investigated topics is the discovery of association rules.

This work proposes a unifying model that enables a uniform description of the problem of discovering association rules. The model provides SQL-like operator, named *MINE RULE*, which is capable of expressing all the problems presented so far in the literature concerning the mining of association rules. We demonstrate the expressive power of the new operator by means of several examples, some of which are classical, while some others are fully original and correspond to novel and unusual applications. We also present the operational semantics of the operator by means of an extended relational algebra.

1 Introduction

Data Mining is a novel research area that develops techniques for *knowledge discovery* in massive amounts of data. In the last years, an increasing number of researchers has concentrated on the solution of a variety of data mining problems, ranging from classification of data into disjoint groups [2, 19], to discovery of associations [3, 16, 12, 7, 5, 15], sequential patterns [8] and similarities in ordered data [1, 10, 4, 6]. The common approach given to research in the field is to concentrate on the development of specialized, efficient techniques for solving specific data mining problems. This emphasis on algorithmic solutions is well motivated by the concrete problem of managing inside data collections, however has some drawbacks; in particular, we believe that not enough emphasis has been placed on the equally important problem of specifying data mining problems from a purely logical and linguistic perspective.

We have observed that many data mining problems consist in finding association rules among data grouped by some common characteristics. As the problem was introduced in the application domain of the basket data analysis, purchase data were collected grouped by the purchase transaction, and associations between two sets of bought products (referred as *items*) were found. But in general, data may be grouped equally well by some different attribute. For example in the sequential pattern problem, data are grouped by customer; then each group is partitioned by date. In both cases, the data mining process consists in discovering associations between two sets of data found in the same group.

An association rule has the form $\mathcal{X} \Rightarrow \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are two sets of items. In this paper we refer to the left hand side of the rule as the *body* and to the right hand side as the *head*. The aim of rules is to provide an observation *a posteriori* on the most common links between data. The frequency of such an observation in the data gives the measure of its relevance. As the number of produced associations might be huge, and not all the discovered associations are meaningful, two

Proceedings of the 22nd VLDB Conference Mumbai(Bombay), India, 1996

probability measures, called *support* and *confidence*, are introduced to discard the less frequent associations in the database. The *support* is the joint probability to find in the same group $\mathcal{X}$ and $\mathcal{Y}$; the *confidence* is the conditional probability to find in a group $\mathcal{Y}$ having found $\mathcal{X}$. Two thresholds, respectively for support and confidence, are given by the user in order to discard those association rules judged less meaningful, as they are less frequent in the amassed data.

We have pointed out the similarities among these problems and defined a unique operator, named **MINE RULE**, that captures the majority of them. This new operator is designed as an extension of the SQL language. The operator has similar objectives as the **DATA CUBE** operator which was introduced in [11]; although **DATA CUBE** and **MINE RULE** refer to distinct problems they respond to the same need of giving a unified framework and proposing a standard formulation for problems that have become very popular, concerning multi-dimensional databases (for **DATA CUBE**) and data mining (for **MINE RULE**), before being given a standard, unitary formulation.

The suitability of the **MINE RULE** operator requires to demonstrate two features. First, the operator must capture most of the data mining problems which were so far informally formulated as well as many other problems, whose formulation is made possible by the operator itself. Second, the operator must be associated to efficient evaluation techniques, that ensure the possibility of solving the specific data mining problems.

This paper is concerned with the first issue, namely the expressive power of the **MINE RULE** operator, which is demonstrated by means of a very large number of examples. In order to guarantee that these problems are unambiguously formulated, we give an inefficient operational semantics, based on a relational algebra.

The paper is organized as follows: Section 2 introduces the operator by means of examples that span from the classical association rules, to rules for sequential patterns and for taxonomic databases; Section 3 defines the semantics of the **MINE RULE** operator; finally Section 4 draws the conclusions, and the Appendix A reports the full syntax of the operator.

1.1 Related Work

The problem of discovering of association rules was introduced in [3], in which associations between a set of items in the body of the rule and a single item in the head are considered. Association rules are slightly generalized in [7], in order to enable more than one item in the head. Both these works inspect data in a flat file. In [13, 14] data is contained in a relational database and rules are discovered by means of the creation of temporary tables and the manipulation of them using SQL expressions. In [8] the problem of discovering of sequential patterns is introduced: a sequential pattern is an association between sets of items, in which some temporal properties between items in each set and between sets are satisfied. In particular, items in a set have the same temporal reference, and an order between sets is established by means of the temporal reference.

Association rules were extended to taxonomic databases in [16]. A taxonomic database describes a hierarchy of the items stored in the database. In presence of such a hierarchy, rules associate not only items, but also classes of items. An algorithm that discovers association rules in taxonomic databases is provided by [16]. Other algorithms, in [12], differ form the previous one for the fact that they discover associations between classes inside a level of the hierarchy, i.e. a rule associates only classes of the hierarchy that have the same distance from the root of the hierarchy.

2 Language by Examples

In this section, we introduce our mining operator **MINE RULE**, showing its application to mining problems based on a practical case. The practical case is the classical database collecting purchase data of a big-store. When a customer buys a set of products (also called *items*), the whole purchase is referred to as a *transaction* having a unique identifier, a date and a customer code. Each transaction contains the set of bought items with the purchased quantity and the price. The simplest way to organize this data is the table **Purchase**, depicted in Figure 1. The transaction column (`tr.`) contains the identifier of the customer transaction; the other columns correspond to the customer identifier, the type of the purchased item, the date of the purchase, the unitary price and the purchased quantity (`q.ty`).

tr.	*customer*	*item*	*date*	*price*	*q.ty*
1	$cust_1$	ski_pants	12/17/95	140	1
1	$cust_1$	hiking_boots	12/17/95	180	1
2	$cust_2$	col_shirts	12/18/95	25	2
2	$cust_2$	brown_boots	12/18/95	150	1
2	$cust_2$	jackets	12/18/95	300	1
3	$cust_1$	jackets	12/18/95	300	1
4	$cust_2$	col_shirts	12/19/95	25	3
4	$cust_2$	jackets	12/19/95	300	2

Figure 1: The **Purchase** table for a big-store.

2.1 Simple Association Rules

In literature, association rules were introduced in the context of the analysis of purchase data, typically organized in a way similar to that of the **Purchase** table.

A rule describes regularities of purchased items in customer transactions. For example, the rule

$$\{brown\_boots, jackets\} \Rightarrow col\_shirts$$

states that if **brown_boots** *and* **jackets** *are bought together in a transaction, also* **col_shirts** *is bought in the same transaction.* In this simple kind of association rules, the body is a set of items and the head is a single item. Note that the rule $\{brown\_boots, jackets\} \Rightarrow brown\_boots$ is not interesting because it is a tautology:

in fact if the head is implicated by the body the rule does not provide new information. This problem has the following formulation:

```
MINE RULE SimpleAssociations AS
SELECT DISTINCT 1..n item AS BODY,
                1..1 item AS HEAD,
                SUPPORT, CONFIDENCE
FROM Purchase
GROUP BY transaction
EXTRACTING RULES WITH SUPPORT: 0.1,
                      CONFIDENCE: 0.2
```

The **MINE RULE** operator produces a new table, called **SimpleAssociations**, where each tuple corresponds to a discovered rule. The **SELECT** clause defines the structure of rules: the body is defined as a set of **items** whose cardinality is any positive integer as specified by **1..n**; the head is defined as a set containing one single item, as specified by **1..1** [1] . The **DISTINCT** keyword states that no replications are allowed inside body or head. This keyword is mandatory because rules are meant to point out the presence of certain kind of items, independently of the number of their occurrences. Furthermore, the **SELECT** clause indicates that the resulting table has four attributes: **BODY**, **HEAD**, **SUPPORT** and **CONFIDENCE**.

The **MINE RULE** operator inspects data in the **Purchase** table grouped by **transaction**, as specified by the **GROUP BY** clause. Figure 2 shows the **Purchase** table after the grouping. Rules are extracted from within groups; their support is the number of groups satisfying the rules divided by the total number of groups; their confidence is the number of groups satisfying the rule divided by the number of groups satisfying the body.

The clause **EXTRACTING RULES WITH** indicates that the operator produces only those rules whose support is greater than or equal to the minimum support and the confidence is greater than or equal to the minimum confidence. In this case, we have a minimum threshold for support of **0.1** and a for confidence of **0.2**.

Figure 3 shows the resulting **SimpleAssociations** table; observe that if we change the minimum support to **0.3**, we then loose almost all rules of Figure 3 except those having **0.50** as support.

Variants of Simple Association Rules Several variants of the basic case of simple association rules are possible; in the following, we discuss them.

If we are interested only in extracting rules from a portion of the source table instead of the whole table, a selection on the source table is necessary. .Similarly to the classical SQL **FROM** clause, in our language it is possible to specify an optional **WHERE** clause associated to the **FROM** clause. This clause creates a temporary table by selecting tuples in the source table that satisfy the **WHERE** clause; then, rules are extracted from this temporary table. For example, if we are interested only in purchases of items that cost no more than $150, we write:

```
MINE RULE SimpleAssociations AS
SELECT DISTINCT 1..n item AS BODY,
 1..1 item AS HEAD, SUPPORT, CONFIDENCE
FROM Purchase WHERE price <= 150
GROUP BY transaction
EXTRACTING RULES WITH SUPPORT: 0.1,
                      CONFIDENCE: 0.2
```

If rules must be extracted only from within groups with a certain property, it is possible to use the classical SQL **HAVING** clause associated to the **GROUP BY** clause. Inside this clause, either aggregate functions (such as **COUNT**, **MIN**, **MAX**, **AVG**) or predicates on the grouping attributes can be used. For instance, if we like to extract rules from purchases of no more than six items, we write:

```
MINE RULE SimpleAssociations AS
SELECT DISTINCT 1..n item AS BODY,
 1..1 item AS HEAD, SUPPORT, CONFIDENCE
FROM Purchase
GROUP BY transaction
   HAVING COUNT(*) <= 6
EXTRACTING RULES WITH SUPPORT: 0.1,
                      CONFIDENCE: 0.2
```

In [17] the case of simple association rules is extended to *generalized association rules*, i.e. rules with an arbitrary number of elements in the head. Our operator treats also this case, by means of a different specifica-

tr.	*customer*	*item*	*date*	*price*	*q.ty*
1	$customer_1$	ski_pants	12/17/95	140	1
	$customer_1$	hiking_boots	12/17/95	180	1
2	$customer_2$	col_shirts	12/18/95	25	2
	$customer_2$	brown_boots	12/18/95	150	1
	$customer_2$	jackets	12/18/95	300	1
3	$customer_1$	jackets	12/18/95	300	1
4	$customer_2$	col_shirts	12/19/95	25	3
	$customer_2$	jackets	12/19/95	300	2

Figure 2: The **Purchase** table grouped by **transaction**.

BODY	*HEAD*	*S.*	*C.*
{ski_pants}	{hiking_boots}	0.25	1
{hiking_boots}	{ski_pants}	0.25	1
{col_shirts}	{brown_boots}	0.25	0.5
{col_shirts}	{jackets}	0.5	1
{brown_boots}	{col_shirts}	0.25	0.5
{brown_boots}	{jackets}	0.25	1
{jackets}	{col_shirts}	0.5	0.66
{jackets}	{brown_boots}	0.25	0.33
{col_shirts,brown_boots}	{jackets}	0.25	1
{col_shirts,jackets}	{brown_boots}	0.25	0.5
{brown_boots,jackets}	{col_shirts}	0.25	1

Figure 3: The **SimpleAssociations** table containing association rules valid for data in **Purchase** table.

[1] Note that the annotations **1..n** and **1..1** are optional in the syntax of Appendix A; this cardinalities are assumed by default when they omitted.

tion for the cardinality of the head, that becomes `1..n` instead of `1..1`.

```
MINE RULE GenAssociations AS
SELECT DISTINCT item AS BODY,
1..n item AS HEAD, SUPPORT, CONFIDENCE
FROM Purchase
GROUP BY transaction
EXTRACTING RULES WITH SUPPORT: 0.1,
                      CONFIDENCE: 0.2
```

With the **MINE RULE** operator it is possible to group the source table by whichever attributes; this fact changes the meaning of extracted rules. For example, if the **Purchase** table were grouped by **customer** instead of the usual **transaction**, rules would describe regularities among customers, independently of the purchase transactions. Thus, we analyze the customer behaviour without paying attention to the transactions in which items are purchased. The problem is formalized as follows:

```
MINE RULE CustomerAssociations AS
SELECT DISTINCT item AS BODY,
 1..n item AS HEAD, SUPPORT, CONFIDENCE
FROM Purchase
GROUP BY customer
EXTRACTING RULES WITH SUPPORT: 0.1,
                      CONFIDENCE: 0.2
```

2.2 Association Rules with Clustering

We said that rules are extracted from within groups: tuples belonging to a group are characterized by the same value of the grouping attributes. Thus, extracted rules are irrespective of other properties described by the remaining attributes. We now extend this simple model by assuming that tuples in a group are partitioned into sub-groups by some non-grouping attributes; we refer to each sub-group as a *cluster*, and to the attributes that define clusters as *clustering attributes*. All tuples in a cluster have the same values of the clustering attributes. With clusters, we extract rules so that their body and head refer to clusters within the same group. Support and confidence are still computed on groups, since rules still describe regularities among groups. This way, the user can specify more sophisticated requirements which are not expressible by means of the simple grouping.

cust	*date*	*item*	*tr.*	*price*	*q.ty*
$cust_1$	12/17/95	ski_pants	1	140	1
		hiking_boots	1	180	1
	12/18/95	jackets	3	300	1
$cust_2$	12/18/95	col_shirts	2	25	2
		brown_boots	2	150	1
		jackets	2	300	1
	12/19/95	col_shirts	4	25	3
		jackets	4	300	2

Figure 4: The **Purchase** table grouped by **customer** and clustered by **date**.

For instance, consider the **Purchase** table; we restrict the association rules of the last example of the previous Section, so that *items purchased by some customer give rise association rules only if they are bought in the same day.* This problem can be specified as follows:

```
MINE RULE ClusteredByDate AS
SELECT DISTINCT 1..n item AS BODY,
   1..n item AS HEAD, SUPPORT, CONFIDENCE
FROM Purchase
GROUP BY customer
CLUSTER BY date
EXTRACTING RULES WITH SUPPORT: 0.01,
                      CONFIDENCE: 0.2
```

The rule extraction process proceeds in the following way. At first, the **Purchase** table is grouped by **customer**; second, after grouping, groups are clustered by **date**, obtaining the table of Figure 4.

At this point, in each group, the cross product of clusters is created, obtaining the table of Figure 5; observe that clusters still maintain the tuples they contain.

For each group, rules are now extracted only from within couples of clusters, the left cluster for the body and the right cluster for the head: the effect is that the body (or the head) of a rule contain items purchased in the same date. For instance, consider the second couple of clusters contained in the group of customer $customer_1$; from this couple, it is possible to extract the rules $\{ski\_pants\} \Rightarrow \{jackets\}$, $\{hiking\_boots\} \Rightarrow \{jackets\}$ and $\{ski\_pants, hiking\_boots\} \Rightarrow \{jackets\}$. Observe that the body of the last rule contains two items bought in the same date.

Tautologies are possible when rules are extracted from a couple of the same cluster. For instance, consider the first couple of clusters contained in the group of $customer_1$; from this couple, it is possible to extract the rules $\{ski\_pants\} \Rightarrow \{ski\_pants\}$ and $\{hiking\_boots\} \Rightarrow \{hiking\_boots\}$ which are tautologies, since they do not provide new information because body and head refer to the same date. In contrast, the rule $\{col\_shirts, brown\_boots, jackets\} \Rightarrow \{col\_shirts, jackets\}$, extracted from the second couple of clusters contained in the group of customer $customer_2$, is not a tautology. In fact, it informs us that items in the head refer to a different date w.r.t the items in the body.

The extraction process can be resumed as follows. The source table is grouped and clustered. Then ordered couples of clusters coming from the same group are created. Given a couple of clusters, the first cluster is used to extract bodies, while the second one is used to extract heads. Finally, all extracted rules are collected; for each rule, its support and confidence are computed on groups (and not clusters) that contain the rule. Only rules with sufficient support and confidence are kept.

Let us consider a rule produced as a variant of the previous example. We are now interested only in rules

group	*body cluster*					*head cluster*				
customer	*date*	*item*	*tr.*	*price*	*q.ty*	*date*	*item*	*tr.*	*price*	*q.ty*
$cust_1$	12/17/95	ski_pants	1	140	1	12/17/95	ski_pants	1	140	1
		hiking_boots	1	180	1		hiking_boots	1	180	1
	12/17/95	ski_pants	1	140	1	12/18/95	jackets	3	300	1
		hiking_boots	1	180	1		ski_pants	1	140	1
	12/18/95	jackets	3	300	1	12/17/95	hiking_boots	1	180	1
	12/18/95	jackets	3	300	1	12/18/95	jackets	3	300	1
$cust_2$	12/18/95	col_shirts	2	25	2	12/18/95	col_shirts	2	25	2
		brown_boots	2	150	1		brown_boots	2	150	1
		jackets	2	300	1		jackets	2	300	1
	12/18/95	col_shirts	2	25	2	12/19/95	col_shirts	4	25	3
		brown_boots	2	150	1		jackets	4	300	2
		jackets	2	300	1	12/18/95	col_shirts	2	25	2
	12/19/95	col_shirts	4	25	3		brown_boots	2	150	1
		jackets	4	300	2		jackets	2	300	1
	12/19/95	col_shirts	4	25	3	12/19/95	col_shirts	4	25	3
		jackets	4	300	2		jackets	4	300	2

Figure 5: Table of Figure 4 after the associations between clusters

that *describe temporally ordered purchases, i.e. the items in the body are purchased previously than the items in the head.* This is similar to the problem of finding *sequential patterns* introduced in [8].

The temporal constraint is a condition on the clustering attributes; it can be specified in the **MINE RULE** operator by means of an optional **HAVING** clause associated to the **CLUSTER BY** clause. This predicate is used to discard couples of clusters before forming rules. Inside this predicate, we can use correlation variables **BODY** and **HEAD** to denote the left and right cluster, as described by Figure 5. The refined problem is described as follows:

```
MINE RULE OrderedSets AS
SELECT DISTINCT 1..n item AS BODY,
 1..n item AS HEAD, SUPPORT, CONFIDENCE
FROM Purchase
GROUP BY customer
CLUSTER BY date
  HAVING BODY.date < HEAD.date
EXTRACTING RULES WITH SUPPORT: 0.01,
                      CONFIDENCE: 0.2
```

The **HAVING** clause following the **CLUSTER BY** clause specifies which couples of clusters must be kept; for the particular case, it produces the table of Figure 6, from which the rules of Figure 7 are produced. Note that these association rules are a subset of the association rules produced in the table **ClusteredByDate**.

If clusters are not specified, each group contains only the trivial cluster. Thus, for each group, the trivial cluster is coupled with itself, and rules are extracted from within this single couple. This shows that the semantics of rule extraction without clustering is a particular case of the semantics with clustering.

2.3 Association Rules with Mining Condition

Let us further refine the example discussed in the previous Section. For example, we are interested in rules such that *the body contains only items whose price is greater than or equal to $100, and the head contains only items whose price is less than $100.* It is not possible to express this requirement by means of the other clauses that the operator provides. In fact, the **WHERE** predicate associated to the **FROM** clause changes the structure of the source table, modifying also the definition of support and confidence; thus, selection predicates are not appropriate for expressing conditions upon rules. The **HAVING** predicate of the **GROUP BY** clause discards groups, and this is not the case. The **CLUSTER BY** clause cannot be used, because the requirement does not specify that items in the body and in the head must have the same price; consequently, also the associated **HAVING** clause is useless. Thus, it is necessary to introduce another selection predicate, called *mining condition*, that must be applied when rules are actually mined. In fact, for each couple of clusters, a rule is extracted if there is a cross product of tuples of the left and right cluster that projected on the body and head attributes gives the rule; the new predicate selects tuples from each cross-product, thereby reducing their cardinality and the extracted rules.

In our operator, the mining condition is specified by means of an optional **WHERE** clause placed between the **SELECT** and the **FROM** clauses. As inside the **HAVING** predicate associated to the **CLUSTER BY** clause, we can use correlation variables **BODY** and **HEAD** to denote tuples of the left and right cluster. This predicate differs from the **HAVING** clauses because it can refer to attributes which are neither grouping nor clustering attributes, and not even in the schema of body and head; this is the case of **price**. Indeed, this is a *tuple predicate*, while **HAVING**

group	*body cluster*					*head cluster*				
customer	*date*	*item*	*tr.*	*price*	*q.ty*	*date*	*item*	*tr.*	*price*	*q.ty*
$cust_1$	12/17/95	ski_pants	1	140	1	12/18/95	jackets	3	300	1
		hiking_boots	1	180	1					
$cust_2$	12/18/95	col_shirts	2	25	2	12/19/95	col_shirts	4	25	3
		brown_boots	2	150	1		jackets	4	300	2
		jackets	2	300	1					

Figure 6: Table of Figure 5 after the selection of couples of clusters.

BODY	*HEAD*	*S.*	*C.*
{ski_pants}	{jackets}	0.5	1
{hiking_boots}	{jackets}	0.5	1
{ski_pants,hiking_boots}	{jackets}	0.5	1
{col_shirts}	{col_shirts}	0.5	1
{col_shirts}	{jackets}	0.5	1
{col_shirts}	{col_shirts, jackets}	0.5	1
{brown_boots}	{col_shirts}	0.5	1
{brown_boots}	{jackets}	0.5	1
{brown_boots}	{col_shirts, jackets}	0.5	1
{jackets}	{col_shirts}	0.5	0.5
{jackets}	{jackets}	0.5	0.5
{jackets}	{col_shirts, jackets}	0.5	0.5
{col_shirts,brown_boots}	{col_shirts}	0.5	1
{col_shirts, brown_boots}	{jackets}	0.5	1
{col_shirts, brown_boots}	{col_shirts, jackets}	0.5	1
{col_shirts,jackets}	{col_shirts}	0.5	1
{col_shirts, jackets}	{jackets}	0.5	1
{col_shirts, jackets}	{col_shirts, jackets}	0.5	1
{brown_boots,jackets}	{col_shirts}	0.5	1
{brown_boots,jackets}	{jackets}	0.5	1
{brown_boots, jackets}	{col_shirts, jackets}	0.5	1

Figure 7: The output table `OrderedSets`.

clauses introduce *group and cluster predicates*. The problem can be specified as follows:

```
MINE RULE FilteredOrderedSets AS
SELECT DISTINCT item AS BODY,
 1..n item AS HEAD, SUPPORT, CONFIDENCE
WHERE BODY.price >= 100 AND HEAD.price < 100
FROM Purchase
GROUP BY customer
CLUSTER BY date
   HAVING BODY.date<HEAD.date
EXTRACTING RULES WITH SUPPORT: 0.01,
                       CONFIDENCE: 0.2
```

For understanding mining conditions, consider again the table in Figure 6; we said that rules are actually mined from this intermediate table. It is not possible to extract rules satisfying the requirement from the couple of clusters in the group of customer $customer_1$, because the item in the right cluster costs more than \$100, and it is not allowed to appear in the head. Consider now the couple of clusters in the group of customer $customer_2$. In the left cluster, only items *brown_boots* and *jackets* cost more than \$100, and are allowed to appear in the body; in the right hand side cluster, only item *col_shirts* cost less than \$100, and is allowed to appear in the head. Thus, the resulting table `FilteredOrderedSets` contains only three rules, as described in Figure 8.

BODY	*HEAD*	*S.*	*C.*
{brown_boots}	{col_shirts}	0.5	1
{jackets}	{col_shirts}	0.5	0.5
{brown_boots,jackets}	{col_shirts}	0.5	1

Figure 8: The output table `FilteredOrderedSets`.

The mining condition can be also used without clusters. For example, let us suppose now that *we are interested in rules such that the items in the body are purchased previously than the items in the head.* Observe that we do not want that items in the body or in the head be bought in the same date; hence, clusters are not useful. The problem can be specified as follows:

```
MINE RULE OrderedItems AS
SELECT DISTINCT 1..n item AS BODY,
 1..1 item AS HEAD, SUPPORT, CONFIDENCE
WHERE BODY.date < HEAD.date
FROM Purchase
GROUP BY customer
EXTRACTING RULES WITH SUPPORT: 0.1,
                       CONFIDENCE: 0.2
```

The resulting table `OrderedItems` is a superset of table `OrderedSets`: since clusters are missing, new sets of items for the body and the head are allowed which were not allowed previously.

2.4 Association Rules with Generalization

Given a table from which association rules are extracted, by means of a taxonomy on the tuples in the table, association rules can be specified by means of the properties described in the taxonomy. The taxonomy can be represented as a hierarchy tree, where each node corresponds to a class of items and its sons are its sub-classes; leaf nodes correspond to items in the database. If there is an ordered path from a node a to a node b, a is cal-

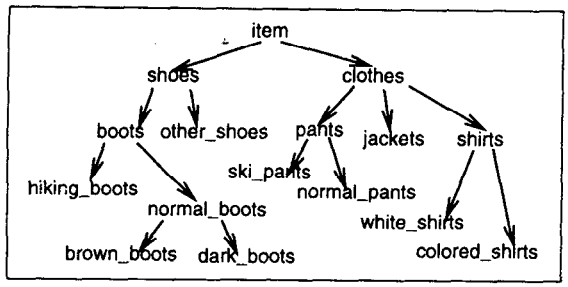

Figure 9: Hierarchy tree of items.

led *ancestor* of b, and b is called *descendant* of a. Using as an example the case of purchases, let us suppose to have a hierarchy of classes on the attribute **item** of the **Purchase** table, as shown in Figure 9. The hierarchy is described by table **ItemHierarchy**, shown in Figure 10.

Each tuple stables a pair connecting a node to one of its ancestors; the attribute **level** indicates the number of levels that separate the node from the ancestor in the hierarchy. Observe that each node is considered ancestor of itself, with the corresponding level set to 0.

2.4.1 Hierarchies in the Mining Condition

Hierarchies can be used in the mining condition to restrict the association rules that can be extracted from the source table, in such a way that the rules refer to specific portions of the hierarchy.

For example, consider the **Purchase** table. Suppose that you want to extract rules that *associates items which are boots with items which are pants*. This is similar to the examples discussed in Section 2.3, with the additional problem that the information about the hierarchy is not contained in the source table. The idea is that the mining condition queries the table **ItemHierarchy** to select only items for the body having *boots* as ancestor, and items for the head having *pants* as ancestor; these queries can be specified by means of the standard SQL **IN** predicate, as we show below.

```
MINE RULE BootsPantsRules AS
SELECT DISTINCT item AS BODY,item AS HEAD,
                SUPPORT, CONFIDENCE
WHERE HEAD.item IN (SELECT node
  FROM ItemHierarchy WHERE ancestor ='pants')
  AND BODY.item IN (SELECT node
  FROM ItemHierarchy WHERE ancestor ='boots')
FROM Purchase
GROUP BY transaction
EXTRACTING RULES WITH SUPPORT: 0.2,
                       CONFIDENCE: 0.5
```

For the data in the **Purchase** table of Figure 1, only the rule $\{hiking\_boots\} \Rightarrow \{ski\_pants\}$, having $support = 0.25$ and $confidence = 1$, is extracted.

2.4.2 Hierarchies in the Source Table

Association rules can be generalized, in order to obtain rules that associate classes. A generalized rule can be obtained by a rule that associates leaves of the hierarchy, by replacing each leaf with one of its ancestors. For example, the simple association rule:
$\{hiking\_boots\} \Rightarrow \{ski\_pants\}$ can be generalized as:
$\{hiking\_boots\} \Rightarrow \{pants\}$,
$\{hiking\_boots\} \Rightarrow \{clothes\}$,
$\{boots\} \Rightarrow \{ski\_pants\}$, $\{shoes\} \Rightarrow \{ski\_pants\}$,
$\{boots\} \Rightarrow \{pants\}$, $\{boots\} \Rightarrow \{clothes\}$,
$\{shoes\} \Rightarrow \{pants\}$ and $\{shoes\} \Rightarrow \{clothes\}$

node	*ancestor*	*level*
hiking_boots	hiking_boots	0
hiking_boots	boots	1
hiking_boots	shoes	2
brown_boots	brown_boots	0
brown_boots	normal_boots	1
brown_boots	boots	2
brown_boots	shoes	3
...	...	...

Figure 10: **ItemHierarchy** table containing a description of the hierarchy defined on **item**.

The generalized rules are characterized by values of support which are greater than the values of the specialized rules. For this fact, it is not possible to obtain all the generalized rules contained in the source table from the specialized rules, for the fact that generalized rules with support and confidence greater than the minimum thresholds can derive from specialized rules that are not extracted since their support is too low.

To extract generalized association rules, we need to add information in the hierarchy table to the source table, by joining this two tables in the **FROM** clause of the **MINE RULE** operator. For example, the following specification *extracts generalized association rules* from within the **Purchase** table using the **ItemHierarchy** table.

```
MINE RULE GeneralizedRules AS
SELECT DISTINCT ancestor AS BODY,
  1..n ancestor AS HEAD, SUPPORT, CONFIDENCE
FROM (SELECT  *
      FROM Purchase, ItemHierarchy
      WHERE node=item)
GROUP BY transaction
EXTRACTING RULES WITH SUPPORT: 0.3,
                       CONFIDENCE: 0.5
```

The first thing that can be noticed, is the **SELECT** clause inside the **FROM** clause: it computes the join of the two tables, and the resulting table is the source table from which rules are extracted, shown in Figure 11.

The second major point to note is that rules contain the attribute **ancestor** in the body and in the head instead of the attribute **item**; this way the leaves of the hierarchy are not loosed, because in the **ItemHierarchy** table each node is ancestor of itself with level 0. A portion of the resulting table is shown in Figure 12.

tr.	*customer*	*date*	*item*	*price*	*q.ty*	*ord_num*	*ancestor*	*level*
1	$customer_1$	12/17/95	ski_pants	150	1	1	ski_pants	0
1	$customer_1$	12/17/95	ski_pants	150	1	1	pants	1
1	$customer_1$	12/17/95	ski_pants	150	1	1	clothes	2
1	$customer_1$	12/17/95	hiking_boots	180	1	2	hiking_boots	0
1	$customer_1$	12/17/95	hiking_boots	180	1	2	boots	1
1	$customer_1$	12/17/95	hiking_boots	180	1	2	shoes	2
...	...	...	...	...	...	...	...	

Figure 11: Source table for extracting generalized association rules.

BODY	*HEAD*	*S.*	*C.*
{ski_pants}	{hiking_boots}	0.25	1
{ski_pants}	{boots}	0.25	1
{ski_pants}	{shoes}	0.25	1
{clothes}	{hiking_boots}	0.25	0.25
{clothes}	{boots}	0.5	0.5
{clothes}	{shoes}	0.5	0.5
{clothes}	{boots, shoes}	0.25	0.5
...	...	...	...

Figure 12: Table `GeneralizedRules` containing generalized association rules.

2.4.3 Generalized Use of Hierarchies

The hierarchy can be used in the mining condition and in the source table at the same time; the effect is that the problem of finding generalized association rules can be refined, obtaining a smaller number of rules. For example, suppose we are interested in *extracting generalized association rules that have sub-classes of boots in the body and sub-classes of pants in the head.* This problem requires that the hierarchy is used at first in source table to produce generalized association rules, and second in the mining condition to reduce the number of extracted rules. Its specification by means of the `MINE RULE` is:

```
MINE RULE GeneralizedBootsPantsRules AS
SELECT DISTINCT ancestor AS BODY,
  1..n ancestor AS HEAD, SUPPORT, CONFIDENCE
WHERE  HEAD.ancestor IN (SELECT node
  FROM ItemHierarchy WHERE ancestor ='pants')
  AND BODY.ancestor IN (SELECT node
  FROM ItemHierarchy WHERE ancestor ='boots')
FROM (SELECT  * FROM Purchase, ItemHierarchy
     WHERE node=item)
GROUP BY transaction
EXTRACTING RULES WITH SUPPORT: 0.3,
                      CONFIDENCE: 0.5
```

2.5 Final Example

We show a final example in which we specify an unconventional problem that can be solved by extracting association rules. Let us suppose we are interested in rules that associate a customer with a set of customers, such that *customers in the head buys the same product previously bought by the customer in the body.* We call this problem the *word of mouth* effect. By means of the `MINE RULE` operator, we can write the following specification.

```
MINE RULE WordOfMouth AS
SELECT DISTINCT 1..1 customer AS BODY,
 1..n customer AS HEAD, SUPPORT, CONFIDENCE
WHERE BODY.date <= HEAD.date
FROM Purchase
GROUP BY item
EXTRACTING RULES WITH SUPPORT: 0.01,
                      CONFIDENCE: 0.05
```

At first, the `Purchase` table is grouped by `item`, because we want to extract rules that describe a repeated behaviour w.r.t purchased items. Second, the table is clustered by `date`, and the `HAVING` predicate specifies that rules must be extracted from couples of clusters temporally ordered. Finally, rules having one single customer in the body and multiple customers in the head are extracted. A hypothetical rule might be: $\{Jennifer\} \Rightarrow \{Janet, Barbara\}$.

Observe that the minimum thresholds for support and confidence are very low; this is due to the fact that the *word of mouth* effect is not expected to be very evident in the global population.

3 Semantics of the Operator

The aim of this section is to provide a formal semantics for the `MINE RULE` operator. The semantics is procedurally described by means of an extended relational algebra: this technique is able to describe how to transform the source table in order to discover association rules.

3.1 New Relational Operators

In the following sections, we use relational tables with complex attributes, i.e. attributes which are themselves relations. An example of nested relation is the table shown in Figure 2, whose schema is

```
(transaction,Group:table(customer,item,date,
                         price,quantity)).
```

In order to operate on extended relations, we use the traditional relational algebra (described in [18]) extended with special operators. The relational algebra provides operators like selection, projection, union, intersection, difference, cartesian product, join and natural join; their semantics is adapted to extended relations, as e.g. in [9]. Hereafter, we introduce the new operators.

Group by: $\Gamma(grouping\ attrs; new\ attr)\ T$ partitions the relation by distinct values of the grouping attributes.

The schema of the result contains the *grouping attributes* and a new set valued attribute (called *new attribute*) whose schema contains all other attributes of T. Values in the *new attribute* are structured as one subtable for each distinct value of the *grouping attributes*.

Unnest: $\eta(attribute\_name)\ T$, is the opposite of the *group by* operator.

Extend: $\mathcal{E}(attribute\ name; expression)\ T$, extends the schema of the operand with a new attribute called *attribute name*; for each tuple, the value to be assigned to the new attribute is obtained evaluating the generic algebraic *expression*.

Substitute: $\Sigma(attribute\ name; expression)\ T$, substitutes the value of the attribute indicated by *attribute name* with the result of the algebraic *expression* evaluated for each tuple.

Rename: $\rho(old\ attr\ names; new\ attr\ names)\ T$, changes the names of the attributes listed as *old attribute names* into the names listed as *new attribute names*.

Powerset: $\mathcal{P}(powerset\ name)\ T$ produces a relation whose schema is obtained by introducing a single attribute, named *powerset name*, that is in turn a relation with the same schema of the original relation T. Tuples of the results are relations which correspond to the power-set of T, hence each relation corresponds to a non-empty subset of the T.

As an example, consider table **Purchase** of Figure 1; Figure 2 shows the table after it is grouped by **transaction**. This operation can be algebraically described as $Grouped = \Gamma(transaction; Group)\ Purchase$. By unnesting attribute *Group* with the expression $\eta(Group)\ Grouped$, table **Purchase** is obtained again.

3.2 Algebraic Semantics of the MINE RULE Operator

In this section, the semantics of the **MINE RULE** operator is formally defined by means of the algebraic operators introduced in Section 3.1. The idea is that the source relation which rules have to be extracted from passes through several transformations; the final result of this process is the relation containing a rule for each tuple, with associated attributes for support and confidence.

In order to simplify the description of the semantics and improve its clarity, we divide the transformation process in distinct steps. Each step is defined as a function that is given a name and a list of input relations, and produces either a derived relation or a number; an algebraic equation assigns the result of a function to a variable, that can be either a relation or a numeric variable. The collection of equations necessary to describe the procedural semantics of the operator is the following algebraic system.

$$\begin{cases} AllGroups = CountAllGroups(Table) \\ Clustered \equiv MakeClusterPairs(Table) \\ Bodies \equiv ExtractBodies(Clustered, AllGroups) \\ Rules \equiv ExtractRules(Clustered, Bodies, AllGroups) \end{cases}$$

The meaning of the equations in the system can be summarized as follows: at first, function *CountAllGroups* computes the number of groups (w.r.t the attribute list in the **GROUP BY** clause) in the source relation; the second equation transforms the source relation into a new one containing couples of clusters (defined by the **CLUSTER BY** clause). At this point, the third equation extracts all possible rule bodies, that are used to evaluate the confidence of each extracted rule; finally, the fourth equation extracts all rules that have sufficient support and confidence.

In the following, we use of the example in Section 2.2 producing table **OrderedSets** as running example.

Counting Groups: the first equation in the system derives the number of groups *AllGroups* present in the source relation: this number is necessary to evaluate the support of rules. Function *CountAllGroups* projects the source table on the grouping attributes (the list of grouping attributes is called *GBAttrList* and it contains the attributes appearing in the **GROUP BY** clause of the **MINE RULE** operator); then, it counts the tuples remained after the projection.

$$CountAllGroups(\ Table\ ) \equiv$$
$$\equiv COUNT(\pi(GBAttrList)\ Table)$$

Making Couples of Clusters: the second equation of the system transforms the source relation into a new one such that each tuple corresponds to a couple of clusters. Recall that if the **CLUSTER BY** clause is not specified, for each group the trivial cluster is coupled to itself; otherwise, the list of coupling attributes appearing in that clause (in the following this list is indicated as *ClAttrList*) is used to obtain clusters and couple them.

$$\begin{aligned} &MakeClusterPairs(Table) \equiv \\ \equiv\ &\pi(GBAttrList, BClAttrList, HClAttrList, \\ &\quad BGroup, HGroup) && (6) \\ &\mathcal{E}(HGroup; \sigma(foreach\ a \in ClAttrList: \\ &\quad HEAD.a = GROUP.a)\ Group) && (5) \\ &\mathcal{E}(BGroup; \sigma(foreach\ a \in ClAttrList: \\ &\quad BODY.a = GROUP.a)\ Group) && (4) \\ &\eta(ClusterPairs) && (3) \\ &\mathcal{E}(ClusterPairs; MakePairs(Group)) && (2) \\ &\Gamma(GBAttrList; Group)\ Table && (1) \end{aligned}$$

Function *MakeClusterPairs* at first groups the source relation by the grouping attributes, obtaining a new intermediate relation where each tuple has the grouping attributes and a complex attribute called *Group*: this is a relation that contains all the tuples in the source relation belonging to that group; its schema is obtained from the schema of the source relation removing the grouping attributes. In the running example, only **customer** is specified as grouping attribute; then, Line (1) obtains a relation with the following schema:

```
(customer,Group:table(transaction,item,date,
                      price,quantity)).
```

Then at line (2), by means of sub-function *MakePairs*, each tuple is further extended with a complex attribute, called *ClusterPairs*: this is the set of

couples of clusters contained in the group corresponding to that tuple. Let us describe the sub-function.

$MakePairs(Group) \equiv$
$\equiv$ ($\rho(foreach\ a \in ClAttrList; BODY.a)$
$\pi(ClAttrList)\ Group$)
$\bowtie[ClusterCondition]$
($\rho(foreach\ a \in ClAttrList; HEAD.a)$
$\pi(ClAttrList)\ Group$)

Sub-function *MakePairs* receives the table *Group* as input parameter, i.e. the tuples contained in a group without the grouping attributes. The sub-function is divided in two parts. The first part projects table *Group* on the clustering attributes, in order to obtain the set of clusters contained in that group; since these are clusters from which rule bodies might be extracted, each clustering attribute $a \in ClAttrList$ is renamed as $BODY.a$. The second part of the function obtains clusters for rule heads, naming each clustering attribute $a \in ClAttrList$ as $HEAD.a$. Finally, the two intermediate relations are joined to form all couples of clusters satisfying the coupling condition *ClusterCondition* possibly coming from the `HAVING` clause associated to the `CLUSTER BY` clause. For instance, in the running example the clustering attribute is `date`; thus, the schema of the relation produced from this function is: `(BODY.date,HEAD.date)`.

Coming back to the description of function *MakeClusterPairs*, at Line (3) the unnest operator unnests the attribute *ClusterPairs*: this operation puts cluster pairs to the topmost level, obtaining a tuple for each group and cluster pair. After that, at Lines (4) and (5) two complex attributes, called *BGroup* and *HGroup*, are added to the schema: they contain the tuples in the group that belong to the cluster for the body and to the cluster for the head, respectively. The final schema of relation *Clustered* for the running example is the following:

```
customer,BODY.date,HEAD.date,
BGroup:table(transaction,item,date,
             price,quantity),
HGroup:table(transaction,item,date,
             price,quantity) )).
```

Extracting Bodies: relation *Bodies* contains all possible bodies contained in the couples of clusters of relation *Clustered*. We need to know the set of bodies in order to evaluate the confidence of rules.

$ExtractBodies(Clustered, AllGroups) \equiv$
$\equiv$ $\sigma(COUNT(BGROUPS)/AllGroups > mSUP)$ (7)
$\Gamma(BODY; BGROUPS)$ (6)
$\pi(GBAttrList, BODY)$ (5)
$\mathcal{E}(BODY; \pi(BSchema)\ Subset)$ (4)
$\eta(Subsets)$ (3)
$\mathcal{E}(Subsets; \mathcal{P}(Subset)\ BGroup)$ (2)
$\pi(GBAttrList, BGroup)\ Clustered$ (1)

Function *ExtractBodies* transforms relation *Clustered*, where each tuple corresponds to a couple of clusters. Line (2) extends the schema with a complex attribute, called *Subsets*, containing all the subsets of attribute *BGroup*, i.e. the tuples contained in the cluster which bodies are extracted from; this work is done by the *power set* operator $\mathcal{P}(Subset)$. An example of the schema at this point is the following:

```
(customer,BGroup:table(transaction,item,date,
                       price,quantity),
 Subsets:table( Subset:table(transaction,item,
                       date,price,quantity))).
```

This new complex attribute is then unnested (Line (3)), in order to put its internal attribute *Subset* to the topmost level. Then, attribute *Subset* is projected on the attributes appearing in the body schema *BSchema* to obtain bodies. Finally, after at Line (5) the relation is projected on the grouping attributes and the attribute *BODY*, at Line (6) the relation is grouped by *BODY*, in order to have a body for each tuple, and the set of groups containing the body in the new complex attribute *GROUPS*. Bodies with insufficient support are discarded at Line (7). The schema of relation *Bodies* for the running example is the following: `(BODY:table(item), BGROUPS:table(customer))`.

Extracting Rules: the last equation of the system extracts rules by means of function *ExtractRules*. This function uses sub-functions in order to simplify the description of the process.

$ExtractRules(Clustered, Bodies, AllGroups) \equiv$
$\equiv$ $AddConfidence(Bodies,$
$AddSupport(AllGroups,$
$CollectRules(DiscardTautologies($
$MakeRules(MakeSubsets(Clustered)))))$

$MakeSubsets(Clustered, Bodies) \equiv$
$\equiv$ $\eta(PairsOfSubsets)$ (3)
$\pi(GBAttrList, BClAttrList,$
$HClAttrList, PairsOfSubsets)$ (2)
$\mathcal{E}(PairsOfSubsets;$
$MakePairsOfSubsets(BGroup, HGroup))$ (1)
$Clustered$

At first, function *MakeSubsets* extracts from relation *Clustered* the subsets of tuples contained in clusters. For each tuple, corresponding to a couple of clusters, at Line (1) it adds a new complex attribute, called *PairsOfSubsets*, that contains ordered pairs of subsets of tuples contained in the couple of clusters. Then, at Line (3) this new attribute is unnested, in order to have one pair of subsets for each tuple. The resulting schema for the running example is:

```
(customer,BODY.date,HEAD.date,
 BODY:table(transaction,item,date,
            price,quantity),
 HEAD:table(transaction,item,date,
            price,quantity) ).
```

Observe that this schema is similar to the schema of relation *Clustered*, except for the fact that attributes *BGroup* and *HGroup* become *BODY* and *HEAD*.

Couples of subsets are computed by sub-function *MakePairsOfSubsets*: it extracts subsets for body and

head by means of the power set operator, and produces the cartesian product of these subsets.

$MakePairsOfSubsets(BGroup, HGroup) \equiv$
$\equiv \mathcal{P}(BODY)\ BGroup \times \mathcal{P}(HEAD)\ HGroup$

After function *MakeSubsets* terminates, function *ExtractRules* calls sub-function *MakeRules* that actually extracts rules from the pairs of subsets.

$MakeRules(Subsets) \equiv$
$\equiv \Sigma(HEAD; \pi(HSchema)\ HEAD)$ (4)
$\Sigma(BODY; \pi(BSchema)\ BODY)$ (3)
$\sigma(BadBH = \emptyset)$ (2)
$\mathcal{E}(BadBH; BODY \bowtie[\neg MiningCond]\ HEAD)$ (1)
$Subsets$

This function extracts rules only from couples of subsets that satisfy the *mining condition*. For each tuple, corresponding to a couple of subsets, at Line (1) the subset $BODY$ is joined with the subset $HEAD$ in order to check for the presence of tuples that do not satisfy the mining condition. At Line (2), if the resulting attribute $BadBH$ is empty, the tuple is selected, because the corresponding couple of subsets satisfies the mining condition. Finally, the actual body and head are computed (Lines (3) and (4)), by means of a projection on body and head schema, respectively. Observe that after Lines (3) and (4), attribute $BODY$ has the body schema (in our running example `BODY:table(item)`), and attribute $HEAD$ has the head schema (in the example `HEAD:table(item)`).

$DiscardTautologies(ClustersWithRules) \equiv$
$\equiv \pi(GBAttrList, BODY, HEAD)$ (5)
$(\sigma(Taut = \emptyset)$ (4)
$\mathcal{E}(Taut; \pi(CSchema)\ BODY$
$\cap\ \pi(CSchema)\ HEAD)$ (3)
$\sigma(CSchema = HSchema \wedge$
$foreach\ a \in ClAttrList:\ BODY.a = HEAD.a)$ (2)
$ClustersWithRules)\ \cup$
$(\sigma(CSchema \neq HSchema \vee$
$thereis\ a \in ClAttrList:\ BODY.a \neq HEAD.a)$ (1)
$ClustersWithRules)$

Tautological rules are discarded by sub-function *DiscardTautologies*. Tautologies are possible when a rule comes from a couple of the same cluster and the head schema is contained in the body schema. If we indicate the intersection of the two schemas as $CSchema$ ($CSchema = BSchema \cap HSchema$), this fact can be indicated with $CSchema = HSchema$. After this premise, it is obvious that the function is divided in two groups: Line (1) takes rules which are certainly not tautological (i.e. the head schema not contained in the body schema or the rule does not come from a couple of the same cluster); Line (2) to (4) take possibly tautological rules and discard rules containing tautologies (tautological rules have non-empty intersection of body and head).

$CollectRules(GroupsWithRules) \equiv$
$\equiv \Gamma(BODY, HEAD; GROUPS)\ GroupsWithRules$

After rules are extracted, sub-function *CollectRules* associates to each rule the set of group identifiers that contain the rule. This is done grouping the input relation by attributes $BODY$ and $HEAD$. The resulting schema for the example is:

```
(BODY:table(item), HEAD:table(item),
 GROUPS:table(customer))
```

Finally, it is necessary to add support and confidence to rules and discard rules with insufficient support and confidence.

$AddSupport(RulesWithoutSupport,\ AllGroups) \equiv$
$\equiv \pi(BODY, HEAD, SUPPORT)$
$\sigma((SUPPORT \geq minSUPPORT)$
$\mathcal{E}(SUPPORT; COUNT(GROUPS)/AllGroups)$
$RulesWithoutSupport$

$AddConfidence(RulesWithSupport,\ Bodies) \equiv$
$\equiv \pi(BODY, HEAD, SUPPORT, CONFIDENCE)$
$\sigma(CONFIDENCE \geq minCONF)$
$\mathcal{E}(CONFIDENCE;$
$COUNT(GROUPS)/COUNT(BGROUPS))$
$(RulesWithSupport \bowtie Bodies)$

For each rule, function *AddSupport* simply adds the new attribute $SUPPORT$ counting the groups containing the rule and dividing it by the number of groups in the source relation (*AllGroups*). Conversely, function *AddConfidence* is a bit more complicated, because for each rule it needs know the groups containing only the body; observe that it is possible to do that by joining relation *RulesWithSupport* with relation *Bodies*, computed by the third line of the algebraic system.

4 Conclusions

This paper has proposed a unifying model describing the problem of discovering association rules, one of the most investigated topics in data mining. The model is based on a new operator, named *MINE RULE*, designed as an extension of the SQL language. The operator is introduced by means of examples: the classical problems are reformulated using the new operator and provide examples of its applications; then novel examples are proposed showing unconventional cases of association rules. Finally the procedural semantics of the operator is given by means of an extended relational algebra.

Acknowledgement. The idea of designing an operator for defining data mining rules originates from stimulating discussions with, and a tutorial given at EDBT Summer School by, Rakesh Agrawal.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *4th International Conference On Foundations of Data Organization and Algorithms*, Chicago, October 1993

[2] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *VLDB-92*, pages 560–573, Vancouver, August 1992.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD-93*, pages 207–216, Washington, D.C., May 1993. British Columbia.

[4] R. Agrawal, K. I. Lin, H. S. Sawhney, and K. Shim. Fast similarity serach in the presence of noise, scaling, and transaltion in time-series databases. In *VLDB-95*, Zurich, Switzerland, September 1995.

[5] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In Padhraic Smyth Usama M. Fayyad, G. Piatetsky-Shapiro and Ramasamy Uthurusamy (Eds), editors, *Knowledge Discovery in Databases*, volume 2. AAAI/MIT Press, Santiago, Chile, September 1995.

[6] R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zait. Querying shapes of histories. In *VLDB-95*, Zurich, Switzerland, September 1995.

[7] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB-94*, Santiago, Chile, September 1994.

[8] R. Agrawal and R. Srikant. Mining sequential patterns. In *International Conference on Data Engineering*, Taipei, Taiwan, March 1995.

[9] P. Atzeni and V. De Antonellis. *Relational Database Theory: A Comprehensive Introduction.* Bejamin Cummings, 1993.

[10] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. of the ACM SIGMOD-94*, May 1994.

[11] J. Gray, A. Bosworth, A. Layman, and H. Piranesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *ICDE96 12th International Conference on Data Engineering*, pages 560–573, New Orleans, Louisiana, USA, February 1996.

[12] J. Han and Fu. Discovery of multiple-level association rules from large databases. In *VLDB-95*, Zurich, Switzerland, September 1995.

[13] M. A. W. Houtsma and A. Swami. Set-oriented mining for association rules in relational databases. In *11th International Conference on Data Engineering*, Taipei, Taiwan, March 6-10 1995.

[14] M. A. W. Houtsma and A. Swami. Set-oriented mining in relational databases. *Data and Knowledge Engineering*, To Appear 1996.

[15] J.S.Park, M.-S.Shen, and P.S.Yu. An effective hash based algorithm for mining association rules. In *Proc. of the ACM SIGMOD-95*, San Jose, California, May 1995.

[16] R. Srikant and R. Agrawal. Mining generalized association rules. In *VLDB-95*, Zurich, Switzerland, September 1995.

[17] R. Srikant and R. Agrawal. Mining generalized association rules. Technical Report RJ 9963, IBM Almaden Research Center, San Jose, California, June 1995.

[18] J. D. Ullman. *Priciples of Database and Knowledge-Base Systems*, volume 1 of *Principles of Computer Science Series.* Computer Science Press, Rockvill, Maryland (USA), 1988.

[19] S. M. Weiss and C. A. Kulikowski. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan-Kaufmann, 1991.

A Syntax

This appendix presents the full syntax of the **MINE RULE** statement. In the syntax, square brackets denote optionality; productions **<FromList>** and **<WhereClause>** denote the standard SQL clauses **FROM** and **WHERE** which are not further expanded (the keywords **BODY** and **HEAD** can be used as correlation variables in the **WHERE** clause for the **SELECT** clause and in the **HAVING** clause for the **CLUSTER BY** clauses); **<TableName>** and **<AttributeName>** denote identifiers, **<Number>** denotes a positive integer, **<real>** denotes real numbers.

```
<MineRuleOp> := MINE RULE <TableName> AS
 SELECT DISTINCT <BodyDescr>, <HeadDescr>
                 [,SUPPORT] [,CONFIDENCE]
 [WHERE <WhereClause>]
 FROM <FromList> [WHERE <WhereClause>]
 GROUP BY <Attribute> <AttributeList>
   [HAVING <HavingClause>]
 [ CLUSTER BY <Attribute> <AttributeList>
   [HAVING <HavingClause>] ]
 EXTRACTING RULES WITH SUPPORT:<real>,
                       CONFIDENCE:<real>

<BodyDescr>:=
 [<CardSpec>] <AttrName> <AttrList> AS BODY
 /* default cardinality for Body: 1..n */
<HeadDescr>:=
 [<CardSpec>] <AttrName> <AttrList> AS HEAD
 /* default cardinality for Head: 1..1 */
<CardSpec>:=<Number> .. (<Number> | n)
<AttributeList>:={,<AttributeName>}
```