Data Mining and Knowledge Discovery, 2, 195–224 (1998)
© 1998 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

An Extension to SQL for Mining Association Rules

ROSA MEO rosimeo@polito.it

GIUSEPPE PSAILA psaila@elet.polimi.it

Politecnico di Torino - Dip. Automatica e Informatica
C.so Duca degli Abruzzi 24 - 10129 Torino, Italy

STEFANO CERI ceri@elet.polimi.it

Politecnico di Milano - Dip. Elettronica e Informazione
P.za Leonardo da Vinci 32 - 20133 Milano, Italy

Editor: Usama M. Fayyad

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 a 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.

Keywords: association rules, data mining and relational databases

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 (Agrawal et al., 1992, Weiss and Kulikowski, 1991), to discovery of associations (Agrawal, Imielinski and Swami, 1993; Agrawal et al., 1995; Agrawal and Srikant, 1994; Han and Fu, 1995; Park, Shen and Yu, 1995; Srikant and Agrawal, 1995) , sequential patterns (Agrawal and Srikant, 1995) and similarities in ordered data (Agrawal, Faloutsos and Swami, 1993; Agrawal et al., 1995a; Agrawal et al., 1995c; Faloutsos, Ranganathan and Monolopoulos, 1994). 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 efficiently managing large 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.

One of the most relevant problems in data mining is the discovery of association rules. 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 probability measures, called *support* and *confidence*, are introduced to discard the less frequent associations in the database.

The problem was introduced in the application domain of the basket data analysis, purchase data were 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, when the same data set is examined for searching sequential patterns in the sales of each individual customers, data are grouped by customers; then each group is further partitioned by date. In both cases, the data mining process consists in discovering associations between two sets of data found in the same group. 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 the less frequent association rules.

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 SQL syntax used in this paper is illustrative of the features that are added in order to specify data mining operations. However, the proposed language design was carefully studied; syntactic elements of the `MINE RULE` operator are all justified by our study of the expressive needs of rules and of the syntactic clauses in SQL; thus, well known SQL clauses (such as `GROUP BY`, `FROM`, `WHERE`, `HAVING`) are extensively used in our operator, since its semantics is based on concepts such as grouping, tuple selection predicates, etc.. The operator has similar objectives as the `DATA CUBE` operator which was introduced in (Gray et al., 1996); 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. We do not propose to *extend* a SQL optimizer, but rather to integrate a SQL server with a specific data mining engine. A different

paper ((Meo, Psaila and Ceri, 1998)) is focused upon an architecture and implementation of the data mining operator, briefly summarized in the conclusions.

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 reports the full syntax of the operator.

1.1. Related Work

The problem of discovering of association rules was introduced in (Agrawal, Imielinski and Swami, 1993), 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 (Agrawal and Srikant, 1994), in order to enable more than one item in the head. Both these works inspect data in a flat file. In (Houtsma and Swami, 1995, Houtsma and Swami, 1996) 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 (Agrawal and Srikant, 1995) 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 (Srikant and Agrawal, 1995). 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 (Srikant and Agrawal, 1995). Other algorithms, in (Han and Fu, 1995), differ from 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.

The definition of an operator for mining association rules in relational environments is mentioned in (Imielinski and Mannila, 1996), where an operator, named `MINE` is added to the classical `SELECT` clause. However, the proposal is not comparable with our work, since the semantics of the `MINE` operator is less general and considers association rules as a composition of Horn clauses.

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

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.

of the purchased item, the date of the purchase, the unitary price and the purchased quantity (`q.ty`).

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` (note that the annotations `1..n` and `1..1` are optional in the syntax reported in Appendix; this

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

cardinalities are assumed by default when they are omitted). 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 kinds 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 attribute `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 minimum threshold 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 `0.5` 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`, `SUM`) 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 (Srikant and Agrawal, 1995) 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 specification for the cardinality of the head, that becomes `1..n` instead of `1..1`.

```
MINE RULE GenAssociations AS
SELECT DISTINCT 1..n 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
```

The set of attributes chosen for grouping must be disjoint from the sets of attributes that are selected for body and head. Otherwise, no rule with valid support would be found, since each rule would be present in one single group. For example look at the following wrong instruction:

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

In this specification, body is made by a set of `items` while head is made by a single `item` followed by its `date`. Grouping is also done by `date` and the rules will contain `items` selected from within sets of purchases made in the same dates. Since the grouping attribute appears as head attribute, no rule will be found in more than one group. The result is that all the rules have the same support, which is equal to 1 divided by the total number of groups.

2.2. Association Rules with Clustering

So far, rules were extracted from within groups. We now extend this simple model by assuming that tuples in a group are further 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. 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.

For instance, consider the `Purchase` table; we restrict the association rules of the example `CustomerAssociations` of the previous Section, so that *items purchased by some customer may generate 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`; next, groups are clustered by `date`, obtaining the table of Figure 4.

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`.

group	*body cluster*					*head cluster*				
cust	*date*	*item*	*t.*	*pr.*	*q*	*date*	*item*	*t.*	*pr.*	*q*
$cust_1$	12/17/95	ski_pants	1	140	1	12/17/95	ski_pants	1	140	1
		h_boots	1	180	1		h_boots	1	180	1
	12/17/95	ski_pants	1	140	1	12/18/95	jackets	3	300	1
		h_boots	1	180	1					
	12/18/95	jackets	3	300	1	12/17/95	ski_pants	1	140	1
							h_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
		b_boots	2	150	1		b_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
		b_boots	2	150	1					
		jackets	2	300	1		jackets	4	300	2
	12/19/95	col_shirts	4	25	3	12/18/95	col_shirts	2	25	2
							b_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

At this point, in each group, the cross product of clusters is created, obtaining the table of Figure 5; the figure shows the two groups with their cluster pairs. For instance, the first group, corresponding to customer $cust_1$, has four cluster pairs resulting from the cross-product of its two clusters, respectively corresponding to the dates '12/17/95' and '12/18/95'.

For each group, rules are now extracted only from within pairs of clusters, the left cluster for the body and the right cluster for the head: the effect is that both the body and the head of a rule contain items purchased in the same date. For instance, consider the second

pair of clusters contained in the group of customer $cust_1$; from this pair, it is possible to form the rules associating the only possible head $\{jackets\}$ coming form the cluster '12/18/95' to the bodies $\{ski\_pants\}$, $\{hiking\_boots\}$, $\{ski\_pants, hiking\_boots\}$. These ones come from all the possible subsets of the tuples extracted from the first cluster of the pair. So the rules $\{ski\_pants\} \Rightarrow \{jackets\}$, $\{hiking\_boots\} \Rightarrow \{jackets\}$, $\{ski\_pants, hiking\_boots\} \Rightarrow \{jackets\}$ are extracted (since they also have enough support and confidence).

Tautologies are possible when rules are extracted from identical clusters which are paired. For instance, consider the first pair of clusters contained in the group of $cust_1$; from this pair, 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 pair of clusters contained in the group of customer $cust_2$, is not a tautology, because items in the head refer to a different date w.r.t. the items in the body.

Observe that the set of clustering attributes must be disjoint from the set of grouping attributes and from the attributes used for the body and head of the rules. Indeed, the notions of groups and clusters are preliminary to the construction of rules: groups and clusters are both formed before the creation of bodies and heads with the purpose of defining the actual partitions from which rules must be extracted. Thus, no overlapping may exists between grouping, clustering and rule attributes.

Let us consider a variant of the previous example, extracting rules 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 (Agrawal and Srikant, 1995): the temporal constraint is a condition on the clustering attributes; it is specified 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. These are a subset of the association rules produced by the `ClusteredByDate` mining expression.

The *cluster condition* is allowed to contain not only predicates on the *clustering attributes*, but also aggregate functions on clusters. As an example of use of aggregate functions in the

group	*body cluster*					*head cluster*				
cust	*date*	*item*	*t.*	*pr.*	*q*	*date*	*item*	*t.*	*pr.*	*q*
$cust_1$	12/17/95	ski_pants	1	140	1	12/18/95	jackets	3	300	1
		h_boots	1	180	1					
$cust_2$	12/18/95	col_shirts	2	25	2	12/19/95	col_shirts	4	25	3
		b_boots	2	150	1					
		jackets	2	300	1		jackets	4	300	2

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

cluster condition, consider the following instruction, which refines the extraction of ordered sets by discarding body clusters whose average price is less than $50.

```
MINE RULE RefinedOrderedSets 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 AND AVG(BODY.price)>=50
EXTRACTING RULES WITH SUPPORT: 0.01, CONFIDENCE: 0.2
```

Note that aggregate functions in the cluster condition are allowed to refer to all attributes except for *grouping* or a *clustering attributes*.

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. We recall that *clusters* are indeed *sub-groups* inside groups.

2.3. Association Rules with Mining Condition

Let us further refine the example `OrderedSets` 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 clauses that were introduced so far. 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 as well, 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,

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`.

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 this is a *tuple predicate*, while `HAVING` 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
```

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`.

```
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 $cust_1$, 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 $cust_2$. 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 costs 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.

The mining condition should refer to all attributes except for grouping or clustering attributes, thus it can refer to attributes used in the body or head of the rules, or to other attributes (in the above case, attribute `price`).

Observe that the mining condition cannot be expressed as a selection on the extracted rule set, because:

- It may refer to attributes not appearing as rule attributes.
- It is applied to pairs of tuples before the extraction, thus changing the rule's support and confidence. Then tuples are considered for extracting rules, and if the condition is false, rules are not extracted; as an effect, the support and confidence are affected.

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.

2.4. Association Rules with Generalizations

Items can be structured hierarchically, forming complex taxonomies. Each 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 called *ancestor* of b, and b is called *descendant* of a. Using the case of purchases as an example, 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 represents 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 *associate 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 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:

```
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 from a rule that

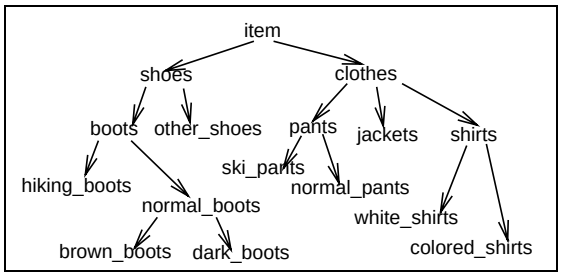

Figure 9. Hierarchy tree of items.

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:

$\{hiking\_boots\} \Rightarrow \{pants\}$, $\{hiking\_boots\} \Rightarrow \{clothes\}$,
$\{boots\} \Rightarrow \{ski\_pants\}$, $\{boots\} \Rightarrow \{pants\}$, $\{boots\} \Rightarrow \{clothes\}$,
$\{shoes\} \Rightarrow \{ski\_pants\}$, $\{shoes\} \Rightarrow \{pants\}$ and $\{shoes\} \Rightarrow \{clothes\}$

The generalized rules are characterized by values of support which are greater than the values of the specialized rules. Consequently, it is not possible to obtain all the generalized rules from the specialized rules. Instead, in order to extract generalized association rules, we need to join the taxonomic information with the source table 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 join of the `Purchase` and `ItemHierarchy` tables is shown in Figure 11, and a portion of the resulting table is shown in Figure 12; note 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 lost, because in the `ItemHierarchy` table each node is ancestor of itself with level 0.

Refining Rules. The above formulation presents some problems. For instance, in Figure 12 consider the rule $\{clothes\} \Rightarrow \{boots, shoes\}$. This rule comes from the third, fifth and sixth tuple of the table in Figure 11; in particular the head comes from the fifth and the sixth tuple. However, if we look at attribute `item` of these two tuples, this is the same ($hiking\_boots$): this fact is not surprising, since the considered table is obtained joining

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`.

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

Figure 11. Source table for extracting generalized association rules.

the `Purchase` and the `ItemHierarchy` tables in the `FROM` clause. Now observe that the head of the rule contains two ancestors ($boots$ and $shoes$) of the same item $hiking\_boots$. Thus, the head (and the rule) is *redundant*.

Also hidden tautologies are possible: for instance, rule $\{boots\} \Rightarrow \{shoes\}$ comes from the same two tuples of Figure 11, which derive from the same item $hiking\_boots$. Thus, the rule is *tautological*.

The problem of avoiding redundant and tautological rules in presence of hierarchies can be solved in two ways: first, by introducing the `CLUSTER BY` clause and the mining condition; second, by adding the `HAVING` clause to the `CLUSTER BY` clause, without using the mining condition. These formulations, however, extract different rules.

- *Adding the* `CLUSTER BY` *clause and the mining condition.*
  First, we observe that the different ancestors of the same item are in the hierarchy tree at different levels. Consequently, if we cluster by `level`, we force each body or head to be composed of `ancestors` at the same level in the hierarchy tree, avoiding redundancies. In the end, the *mining condition*

  ```
  BODY.item~=HEAD.item
  ```

  discards tautological rules, in which bodies and heads refer to the same item. Consequently, we obtain the following formulation:

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.

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

- *Adding the* `CLUSTER BY` *and* `HAVING` *clauses.*
  First of all, observe that clustering by `level`, and introducing a *cluster condition* like

  `BODY.level = HEAD.level,`

  if we are under the hypothesis that the tree is balanced, we fall into the case of the so called *multiple level association rules* (Han and Fu, 1995). In this case, rules only associate elements of the same level of the hierarchy (e.g. $\{boots\} \Rightarrow \{pants\}$); thus, redundancies and tautologies are automatically avoided, and the mining condition is no longer necessary. The complete specification for this case is the following.

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

2.4.3. Hierarchies in a HAVING clause Hierarchies may be also used in the `HAVING` clause of the `GROUP BY` clause or of the `CLUSTER BY` clause. Consider the new table `PurchasedWithBrands`, that is defined on the same attributes of table `Purchase` plus an extra attribute called `brand`, which denotes the brand of the purchased item; in our database, the same item can appear with different brands.

Consider now the following problem: *extract all rules on brands, such that brands in the body refer to a pant, and brands in the head refer to a shoe; keeps only rules with support 0.3 and confidence 0.5.*

The following specification captures the case:

```
MINE RULE BrandRules  AS
SELECT 1..n brand AS BODY, 1..n brand AS HEAD, SUPPORT, CONFIDENCE
FROM  PurchaseWithBrands
GROUP BY customer
CLUSTER BY item
       HAVING BODY.item IN (SELECT node
                            FROM ItemHierarchy
                            WHERE ancestor='pants')
          AND HEAD.item IN (SELECT node
                            FROM ItemHierarchy
                            WHERE ancestor='shoes')
EXTRACTING RULES WITH SUPPORT: 0.3 , CONFIDENCE: 0.5
```

Observe that the hierarchy is used to filter cluster pairs such that the body cluster is a descendant of `'pants'`, while the head cluster is a descendant of `'shoes'`.

2.4.4. Generalized Use of Hierarchies The use of hierarchies in the mining condition and in the source table, permits to refine the previous problem of generalized association rules, 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 the source table to produce generalized association rules, and second in the mining condition to reduce the number of extracted rules.

```
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 some final examples 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: $\{Jennifer\} \Rightarrow \{Janet, Barbara\}$. Note that we have set lower minimum thresholds for support and confidence, since *word of mouth* effect is not expected to be very evident in the global population.

2.6. Discussion

In conclusions, we have considered four kinds of rules.

- *Simple Association Rules.* They require the `FROM` clause, the `GROUP BY` clause and the `SELECT` clause.
- *Association Rules from Filtered Groups.* They add to simple association rules the `HAVING` clause inside the `GROUP BY` clause, which filters groups from which rules are extracted.
- *Association Rules with Clustering.* They add to simple association rules the `CLUSTER BY` clause, and possibly the associated `HAVING` clause, which partitions each group in clusters and couples them to each other, keeping only those cluster couples satisfying the `HAVING` clause, if specified.
- *Association Rules with Mining Condition.* They add to simple association rules the `WHERE` clause associated to the `SELECT` clause, also called *mining condition*, a tuple predicate that must be satisfied by every tuple from which rules are extracted.

The four classes of problems discussed above put in evidence that the `HAVING` clause of the `GROUP BY` clause, the `CLUSTER BY` clause, and the `WHERE` clause for the mining

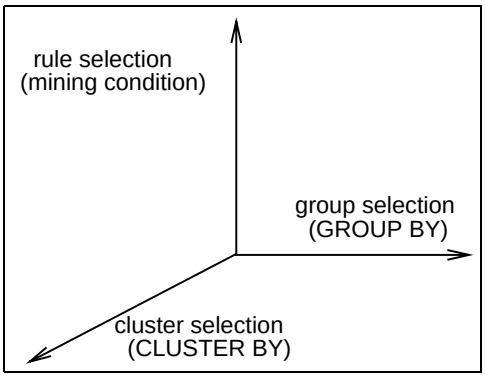

Figure 13. Orthogonality of the MINE RULE clauses.

condition express *orthogonal* aspects of the semantics of extracted association rules, w.r.t. the mining problems they are designed for (see Figure 13). Observe that the case of *simple association rules* is a particular case of the other three classes of discussed problems.

Due to the orthogonality, a mining application can be designed as an arbitrary composition of the mandatory and optional clauses, thus it can be viewed as a point in the 3-dimensional space depicted in figure.

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 table T_2 in Figure 14, whose schema is

```
(a, b, s:table(c, d)).
```

The schema contains three attributes, where `a` and `b` are simple attributes, while `s` is in turn a table whose schema contains two simple attributes, named `c` and `d`. Observe the instance of T_2 depicted in figure: in the first two tuples, attribute `s` contains two tuples; in the last tuple of T_2, attribute `s` has only one tuple, even though it is still a table.

In order to operate on extended relations, we use the traditional relational algebra (described in (Ullman, 1988)) extended with special operators. The relational algebra provides operators like selection, projection, union, intersection, difference, cartesian product, join

Table 1. Traditional Relational Operators.

Selection	$\sigma(predicate)\ T$	Difference	$T_1 - T_2$
Projection	$\pi(attributeList)\ T$	Cartesian Prod.	$T_1 \times T_2$
Union	$T_1 \cup T_2$	Join	$T_1 \bowtie [predicate]\ T_2$
Intersection	$T_1 \cap T_2$	Natural Join	$T_1 \bowtie T_2$

and natural join (summarized in Table 1); their semantics is adapted to extended relations, as in (Atzeni and De Antonellis, 1993). Hereafter, we introduce the new operators.

- **Group by**: $\Gamma(grouping\ attributes; new\ attribute)\ T$, 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 a subtable for each distinct value of the *grouping attributes*.
- **Unnest**: $\eta(attribute\_name)\ T$, is the opposite of the *group by* operator; if the unnested complex attribute is empty in a tuple t, there are no tuples for t in the resulting table.
- **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 denoted as *expression*.
- **Substitute**: $\Sigma(attribute\ name; expression)\ T$, substitutes the value of the attribute indicated by *attribute name* with the result of *expression*, evaluated for each tuple. The resulting type of *attribute name* may be different with respect to the initial type, due to *expression*.
- **Rename**: $\rho(old\ attribute\ names; new\ attribute\ names)\ T$, changes the names of the attributes listed as *old attribute names* into the names listed as *new attribute names*; attribute types do not change.
- **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 T.

As an example, Figure 14 reports some extended relations that are derived from relation T_1, by means of the following expressions.

$$
\begin{aligned}
T_2 &= \Gamma(a, b; s)\ T_1 \\
T_3 &= \mathcal{E}(t; COUNT(s))\ T_2 \\
T_4 &= \Sigma(s; \pi(d)\ s)\ T_3 \\
T_5 &= \rho(a, b, s; x, y, z)\ T_4 \\
T_6 &= \mathcal{P}(p)\ \sigma(d = d_1)\ T_1
\end{aligned}
$$

Relation T_2 is obtained grouping relation T_1 by attributes a and b. Relation T_3 is obtained from relation T_2 extending its schema with a new attribute t; for each tuple, attribute t is set

to the value obtained counting the elements contained in the complex attribute s of the same tuple. Relation T_4 is derived from relation T_3 by substituting the complex attribute s with the result of the expression; for each tuple, the expression projects the same complex attribute s on its internal attribute d, obtaining a table with one column instead of the previous two columns. Relation T_5 is obtained by relation T_4 renaming attributes a, b and s as x, y and z. In the end, relation T_6 is obtained from T_1 by means of the power-set operator, applied on the 2 tuples having d_1 as value of the attribute d; observe that each tuple of T_6 contains an element of the power-set, i.e. attribute p is a relation defined on the schema of T_1.

As a practical 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 $Grouped = \Gamma(transaction; Group)\ Purchase$, thus obtaining the following schema, which is the same of the table in Figure 2:

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

Observe that by unnesting attribute $Group$, the expression $\eta(Group)\ Grouped$, obtains table `Purchase` 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.

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

The meaning of the equations in the system can be summarized as follows:

1. at first, function $CountAllGroups$ computes the number of groups (w.r.t. the attribute list in the `GROUP BY` clause) in the source relation;
2. the second equation transforms the source relation into a new one containing couples of clusters (defined by the `CLUSTER BY` clause);
3. at this point, the third equation extracts all possible rule bodies, that are used to evaluate the confidence of each extracted rule;
4. finally, the fourth equation extracts all rules that have sufficient support and confidence.

T_1 =

a	b	c	d
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2
a_1	b_2	c_1	d_1
a_1	b_2	c_1	d_3
a_1	b_2	c_3	d_3

T_2 =

a	b	$s:table$			
		(	c	d	)
a_1	b_1	{	c_1	d_1	
			c_2	d_2	}
a_1	b_2	{	c_1	d_1	
			c_1	d_3	}
a_2	b_2	{	c_3	d_3	}

T_3 =

a	b	$s:table$				t
		(	c	d	)	
a_1	b_1	{	c_1	d_1		2
			c_2	d_2	}	
a_1	b_2	{	c_1	d_1		2
			c_1	d_3	}	
a_2	b_2	{	c_3	d_3	}	1

T_4 =

a	b	$s:table$			t
		(	d	)	
a_1	b_1	{	d_1		2
			d_2	}	
a_1	b_2	{	d_1		2
			d_3	}	
a_2	b_2	{	d_3	}	1

T_5 =

x	y	$z:table$			t
		(	d	)	
a_1	b_1	{	d_1		2
			d_2	}	
a_1	b_2	{	d_1		2
			d_3	}	
a_2	b_2	{	d_3	}	1

T_6 =

$p:table$					
(	a	b	c	d	)
{	a_1	b_1	c_1	d_1	}
{	a_1	b_2	c_2	d_1	}
{	a_1	b_1	c_1	d_1	
	a_1	b_2	c_1	d_1	}

Figure 14. Examples of Extended Relations.

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

Counting Groups: the first equation in the system counts the 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 the clause (in the following this list is indicated as $ClAttrList$) is used to obtain clusters and couple them.

$$
\begin{array}{ll}
MakeClusterPairs(Table) \equiv & \\
\equiv\ \eta(ClusterPairs) & (3)\\
\quad \mathcal{E}(ClusterPairs; MakePairs(Group)) & (2)\\
\quad \sigma(GroupCondition)\ \Gamma(GBAttrList; Group)\ \ Table & (1)
\end{array}
$$

1. At first, line (1) of function $MakeClusterPairs$ groups the source relation by the grouping attributes, in order to apply the predicate $GroupCondition$ that discards groups not satisfying it. It obtains 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)).`

2. Then at line (2), by means of sub-function $MakePairs$ hereafter reported, 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 the tuple.

   $$
   \begin{array}{l}
   MakePairs(Group) \equiv\\
   \equiv\ \rho(BODY, HEAD; BCluster, HCluster)\\
   \quad (\ \rho(foreach\ a \in ClAttrList; BODY.a)\\
   \quad\ \ \Gamma(ClAttrList; BODY)\ \ Group\ )\\
   \quad \bowtie[ClusterCondition]\\
   \quad (\ \rho(foreach\ a \in ClAttrList; HEAD.a)\\
   \quad\ \ \Gamma(ClAttrList; HEAD)\ \ Group\ )
   \end{array}
   $$

   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 groups table $Group$ by the clustering attributes, in order to obtain the clusters contained in the group; in the resulting table, each topmost tuple corresponds to a cluster with associated the tuples it contains in the new complex attribute $BODY$. Afterwards, 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, and renames 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. The renaming of clustering attributes is made with the purpose to use the predicate $ClusterCondition$ in making couples of clusters: in fact $ClusterCondition$ refers to *body* clusters with the correlation name `BODY` and to *head* clusters with `HEAD`. In the following we will refer to the set of clustering attributes renamed with prefix $BODY$ and $HEAD$ as $BClAttrList$ and $HClAttrList$ respectively.
   Finally, after the join, the complex attributes $BODY$ and $HEAD$ are renamed as $BCluster$ and $HCluster$, respectively.

   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,
 BCluster:table(transaction,item,price,quantity),
 HCluster:table(transaction,item,price,quantity)).
```

3. 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. The final schema of relation $Clustered$ for the running example is the following:

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

Extracting Bodies: relation $Bodies$ contains all possible bodies contained in the clusters actually present in the source relation $Table$. We need to know the set of bodies in order to evaluate the confidence of rules.

$$
\begin{aligned}
&ExtractBodies(Table, AllGroups) \equiv \\
\equiv\ &\sigma(COUNT(BodyGroups)/AllGroups > minSupport) && (9)\\
&\Gamma(BodySet; BodyGroups) && (8)\\
&\pi(GBAttrList, BodySet) && (7)\\
&\Sigma(BodySet; \pi(BSchema)\ BodySet) && (6)\\
&\eta(BCluster) && (5)\\
&\Sigma(BCluster; \mathcal{P}(BodySet)\ BCluster) && (4)\\
&\Gamma(GBAttrList, ClAttrList; BCluster) && (3)\\
&\eta(BGroup) && (2)\\
&\sigma(GroupCondition)\ \Gamma(GBAttrList; BGroup)\ Table && (1)
\end{aligned}
$$

Function $ExtractBodies$ proceeds as follows.

1. Line (1) groups again the source table by the grouping attribute, in order to apply the predicate on groups, named $GroupCondition$, i.e. the `HAVING` clause associated to the `GROUP BY` clause; then, line (2) unnests the complex attribute $BGroup$, in order to obtain the source relation without invalid groups.

2. Line (3) groups again the result of line (2) by the grouping and clustering attributes, together, in order to partition valid groups into clusters, from which bodies are extracted: a new complex attribute, `BCluster`, is created. The schema, for the running example, obtained at line (3) is:

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

3. Line (4) substitutes each cluster with all the subsets that can be formed from its tuples, i.e. its power set computed by the *power set* operator, $\mathcal{P}(BodySet)$. An example of the schema at this point is the following:

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

   The complex attribute `BCluster` is then unnested (Line (5)), in order to put its internal attribute $BodySet$ to the topmost level. Then, attribute $BodySet$ is projected on the

attributes appearing in the body schema $BSchema$ to obtain bodies. For the running example, the body schema is simply $item$ and the schema of the table, obtained at this point, is the following:

```
(customer,date, BodySet:table(item)).
```

4. Finally, at Line (7) the relation is projected on the grouping attributes, denoted as $GBAttrList$, and the attribute $BodySet$. At Line (8) the relation is grouped by $BodySet$, in order to have in each tuple only one body and in the complex attribute, $BodyGroups$, the set of groups in which the body is present. Bodies with insufficient support are discarded at Line (9) in which the value of minimum support inserted by the user is denoted by $minSupport$. The schema of relation $Bodies$, result of the function $ExtractBodies$, is the following for the running example:

   ```
   Bodies:table(BodySet:table(item), BodyGroups: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.

$$
\begin{aligned}
&ExtractRules(Clustered, Bodies, AllGroups) \equiv \\
&\equiv AddConfidence(Bodies, \\
&\quad AddSupport(AllGroups, \\
&\quad\ CollectRules(DiscardTautologies(\\
&\quad\quad MakeRules(MakeSubsets(Clustered)))))
\end{aligned}
$$

1. At first, function $MakeRules$ calls $MakeSubsets$ to extract from the relation named $Clustered$ the subsets of tuples contained in clusters.

$$
\begin{aligned}
&MakeSubsets(Clustered) \equiv \\
&\equiv \eta(PairsOfSubsets) && (3)\\
&\quad \pi(GBAttrList, BClAttrList, HClAttrList, PairsOfSubsets) && (2)\\
&\quad \mathcal{E}(PairsOfSubsets; MakePairsOfSubsets(BCluster, HCluster)) && (1)\\
&\quad Clustered
\end{aligned}
$$

   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.

   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.

$$
\begin{aligned}
&MakePairsOfSubsets(BCluster, HCluster) \equiv \\
&\equiv \mathcal{P}(BODY)\ BCluster \times \mathcal{P}(HEAD)\ HCluster
\end{aligned}
$$

   At Line (2) a projection is made in order to get rid of attributes $BCluster$ and $HCluster$ no longer needed.

   At Line (3) the complex attribute $PairsOfSubsets$ is unnested, in order to have one pair of subsets for each tuple. The resulting table is named $CoupledSubsets$ and, for the running example, is:

```
CoupledSubsets:table(customer,BODY.date,HEAD.date,
                     BODY:table(transaction,item,price,quantity),
                     HEAD:table(transaction,item,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 $HEAD$ due to the function $MakePairsOfSubsets$.

2. After function $MakeSubsets$ terminates, function $ExtractRules$ makes a call to sub-function $MakeRules$ that actually extracts rules from the pairs of subsets.

$$
\begin{array}{ll}
MakeRules(CoupledSubsets) \equiv & \\
\equiv \Sigma(HEAD; \pi(HSchema)\ HEAD) & (4)\\
\quad \Sigma(BODY; \pi(BSchema)\ BODY) & (3)\\
\quad \sigma(BadBH = \emptyset) & (2)\\
\quad \mathcal{E}(BadBH; BODY \bowtie [\neg MiningCond]\ HEAD) & (1)\\
\quad CoupledSubsets &
\end{array}
$$

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 couples of tuples that do not satisfy the mining condition. At Line (2), if the resulting attribute $BadBH$ is empty, the couple of subsets is selected, because the mining condition is satisfied. 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)`).
The result of function $MakeRules$ is table $ClustersWithRules$ whose schema is the following:

```
ClustersWithRules:table(customer, BODY.date, HEAD.date,
                        BODY:table(item), HEAD:table(item)).
```

3. Tautological rules are now 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 denote the intersection of the two schemas as $CSchema$ ($CSchema = BSchema \cap HSchema$), the condition for schema containment mentioned above can be indicated with $CSchema = HSchema$.

$$
\begin{array}{ll}
DiscardTautologies(ClustersWithRules) \equiv & \\
\equiv \pi(GBAttrList, BODY, HEAD) & (5)\\
\quad (\sigma(Taut = \emptyset) & (4)\\
\quad \mathcal{E}(Taut; \pi(CSchema)\ BODY \cap \pi(CSchema)\ HEAD) & (3)\\
\quad \sigma(CSchema = HSchema \wedge \forall a \in ClAttrList : BODY.a = HEAD.a) & (2)\\
\quad ClustersWithRules) \quad \cup & \\
\quad (\sigma(CSchema \neq HSchema \vee \exists a \in ClAttrList : BODY.a \neq HEAD.a) & (1)\\
\quad ClustersWithRules) &
\end{array}
$$

The function is divided in two groups: Line (1) takes rules which are certainly not tautological (i.e. the head schema is 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 those ones containing tautologies (tautological rules have non-empty intersection of body and head).
The result at this point is the table $GroupsWithRules$ that has the following schema:

```
GroupsWithRules:table(customer,BODY:table(item), HEAD:table(item)).
```

4. After rules are extracted, sub-function $CollectRules$ associates to each rule the new complex attribute $GroupSet$ having the set of group identifiers that contain the rule. This is done grouping the input relation by attributes $BODY$ and $HEAD$.

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

The resulting table is $RulesWithoutSupport$ whose schema for the running example is:

```
RulesWithoutSupport:table(BODY:table(item), HEAD:table(item),
                          GroupSet:table(customer)).
```

5. Finally, it is necessary to add support and confidence to rules and discard those ones with insufficient support and confidence. This is done by functions $AddSupport$ and $AddConfidence$.

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

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

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 ($AllGroups$). The resulting table is $RulesWithSupport$ whose schema for the running example is:

```
RulesWithSupport:table(BODY:table(item), HEAD:table(item), SUPPORT)
```

Conversely, function $AddConfidence$ compares the confidence of each rule with the minimum value $minConf$ given by the user; function $AddConfidence$ is a bit more complicated, because for each rule it needs to know the number of groups containing only the body; observe that it is possible to solve the problem by joining relation

$RulesWithSupport$ with relation $Bodies$, computed by the third line of the algebraic system.

The final result is table $Rules$ containing extracted rules with support and confidence. Its schema for the running example is:

```
Rules:table(BODY:table(item),HEAD:table(item),SUPPORT,CONFIDENCE)
```

4. Conclusions

This paper has proposed a unifying model describing the problem of discovering association rules, one of the most relevant 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 applied to the classical case of purchase data, which is taken as running example: these usual problems are formulated using the new operator and provide examples of its applications; then novel examples are proposed showing unconventional cases of association rules. As a second step, criteria of application design are discussed, based on the orthogonality property characterizing the clauses of our operator; the application of these criteria to a practical case is also discussed. Finally the procedural semantics of the operator is given by means of an extended relational algebra.

In (Meo, Psaila and Ceri, 1998) we study the efficient implementation of data mining algorithms. We propose an architecture where data mining is tightly integrated with a SQL server; we identify the computations which are best performed by the SQL server and the computations that require a specialized data mining engine. For the former computations, we describe the syntax-driven translation of data mining expressions into SQL expressions, showing that most of the clauses of our operator are really related to (and can be evaluated by) existing query primitives. For the latter computations, we identify a uniform interface to a collection of data mining algorithms covering all the power of the `MINE RULE` operator. We demonstrate the efficiency of the porposed solution on several case studies.

We are also investigating formal properties of the operator, in order to understand the expressive power and equivalence transformations or containment properties of two `MINE RULE` expressions.

Finally, we are defining a framework which makes the user able to define *templates*, i.e. classes of `MINE RULE` specifications where several elements of the specification, such as table names, attribute names, constants and so on, are left undetermined; by means of a design tool (under development too), the user chooses a particular template, and target the template to its own database, in order to obtain a fully defined `MINE RULE` specification that corresponds to his/her particular needs.

Appendix 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>}
```

References

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

Agrawal,R., Ghosh, S., Imielinski, T., Iyer, B. and Swami, A. An interval classifier for database mining applications. In *18th International Conference on Very Large Databases (VLDB)*, pages 560–573, Vancouver, August 1992.

Agrawal, R., Imielinski, T. and Swami, A. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993. British Columbia.

Agrawal, R., Lin, K.I., Sawhney, H.S. and Shim, K. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, September 1995.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H. and Verkamo, A.I. 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.

Agrawal, R., Psaila, G., Wimmers, E.L. and Zait, M. Querying shapes of histories. In *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, September 1995.

Agrawal, R. and Srikant, R. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, September 1994.

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

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

Faloutsos, C., Ranganathan, M. and Manolopoulos, Y. Fast subsequence matching in time-series databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, May 1994.

Gray, J., Bosworth, A., Layman, A. and Piranesh, H. 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.

Han, J. and Fu, Y. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, September 1995.

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

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

Imielinski, T. and Mannila, H. A database perspective on knowledge discovery. *Coomunications of the ACM*, 39(11):58–64, November 1996.

Meo, R., Psaila, G. and Ceri, S. A tightly coupled architecture for data mining. In *Proceedings of the IEEE International Conference on Data Engineering*, Orlando, Florida, February, 1998.

Park, J.S., Shen, M. and Yu, P.S. An effective hash based algorithm for mining association rules. In *Proceedings of the ACM-SIGMOD International Conference on the Management of Data*, San Jose, California, May 1995.

Srikant, R. and Agrawal, R. Mining generalized association rules. In *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, September 1995.

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

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

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