Data Mining Within DBMS Functionality

Maciej Zakrzewicz

Poznan University of Technology, Poland
mzakrz@cs.put.poznan.pl

Abstract

Data mining slowly evolves from simple discovery of frequent patterns and regularities in large data sets toward interactive, user-oriented, on-demand decision supporting. Since data to be mined is usually located in a database, there is a promising idea of integrating data mining methods into database management systems (DBMS). In this paper we present the results of developing our research prototype for DBMS-integrated data mining. We focus on two main contributions: query language for data mining and constraints-driven algorithm for association rules discovery.

Keywords**:** data mining algorithms, data mining architectures.

1. Introduction

Data mining, also referred to as database mining or knowledge discovery in databases (KDD), is a new research area that aims at the discovery of useful information from large datasets. Data mining uses statistical analysis and inference to extract interesting trends and events, create useful reports, support decision making etc. It exploits the massive amounts of data to achieve business, operational or scientific goals. One of the most promising data mining applications is affinity analysis, which provides a user with the knowledge about item co-occurrences in item sets. The most common representation of item co-occurrences are association rules.

The problem of association rules discovery is an interesting subfield of data mining. By an association rule, that holds in a database of item sets (e.g. big-store transactions), we mean a formula of the form $X \rightarrow Y$, where X and Y are two sets of items. We refer to the left hand side of the rule as the *body* and to the right hand side as the *head*. Additionally, each rule has two associated measures of statistical significance and strength: *support* and *confidence*. The support is the joint probability to find X and Y in the same items group. The rule confidence is the conditional probability to find in the item group Y having found X.

A number of association rules discovery algorithms has been proposed. Most of them discover rules that have support and confidence greater than given minimum values, however, the data mining process is more interactive and more iterative in nature. It requires not only high-performance and rapid-response algorithms, but also the environmental support that assists users in data selection, rule generation and rule filtering.

In this paper we present the architecture of our research prototype, called RD2, which extends DBMS functionality toward on-demand association rules discovery. The prototype implements our improved algorithm for association rules discovery and provides a user with an SQL-like query language which is used to control the algorithm. The paper illustrates the need and the solution for integrating data mining methods with core DBMS functions.

1.1 BASIC DEFINITIONS

Let $L=\{l_1, l_2, ..., l_m\}$ be a set of literals, called items. Let a non-empty set of items T be called an *itemset*. Let D be a set of variable length itemsets, where each itemset $T \subseteq L$. We say that an itemset T *supports* an item $x \in L$ if x is in T. We say that an itemset T *supports* an itemset $X \subseteq L$ if T supports every item in the set X.

An *association rule* is an implication of the form $X \rightarrow Y$, where $X \subset L$, $Y \subset L$, $X \cap Y = \emptyset$. Each rule has associated measures of its statistical significance and strength, called *support* and *confidence*. The support of the rule $X \rightarrow Y$ in the set D is:

$$support(X \rightarrow Y, D) = \frac{\left|\left\{T \in D \mid T \text{ supports } X \cup Y\right\}\right|}{|D|}$$

In other words, the rule $X \rightarrow Y$ holds in the set D with support s if $(100*s)$% of itemsets in D support $X \cup Y$. The confidence of the rule $X \rightarrow Y$ in the set D is:

$$confidence(X \rightarrow Y, D) = \frac{\left|\left\{T \in D \mid T \text{ supports } X \cup Y\right\}\right|}{\left|\left\{T \in D \mid T \text{ supports } X\right\}\right|}$$

In other words, the rule $X \rightarrow Y$ has confidence c if $(100*c)$% of itemsets in D that support X also support Y.

Example. Consider a supermarket with a large collection of products. When a customer buys a set of products, the whole purchase is stored in a database and referred to as a transaction having a unique identifier, date, and a customer code. Each transaction contains the set of purchased products together with their quantity and price. An example of the database of customer transactions is depicted below. The attribute *trans_id* represents the transaction identifier, *cust_id* - the customer code, *product* - the purchased product, *qty* - the quantity and *price* - the price.

trans_id	**cust_id**	**product**	**date**	**Qty**	**Price**
1	908723	soda_03	02/22/98	6	0.20
1	908723	potato_chips_12	02/22/98	3	0.99
2	032112	beer_10	02/22/98	4	0.49
2	032112	potato_chips_12	02/22/98	1	0.99
2	032112	diapers_b01	02/22/98	1	1.49
3	504725	soda_03	02/23/98	10	0.20
4	002671	soda_03	02/24/98	6	0.20
4	002671	beer_10	02/24/98	2	0.49
4	002671	potato_chips_12	02/24/98	4	0.99
5	078938	beer_10	02/24/98	2	0.49
5	078938	potato_chips_12	02/24/98	4	0.99
5	078938	diapers_b01	02/24/98	10	1.49

The strongest association rules that can be found in the example database are listed below:

beer_10 → potato_chips_12	support=0.60	confidence=1.00
potato_chips_12 → beer_10	support=0.60	confidence=0.75
beer_10 ∧ diapers_b01 → potato_chips_12	support=0.40	confidence=1.00
diapers_b01 ∧ potato_chips_12 → beer_10	support=0.40	confidence=1.00
diapers_b01 → beer_10 ∧ potato_chips_12	support=0.40	confidence=1.00
diapers_b01 → beer_10	support=0.40	confidence=1.00
diapers_b01 → potato_chips_12	support=0.40	confidence=1.00
beer_10 ∧ potato_chips_12 → diapers_b01	support=0.40	confidence=0.67
beer_10 → diapers_b01 ∧ potato_chips_12	support=0.40	confidence=0.67
beer_10 → diapers_b01	support=0.40	confidence=0.67
soda_03 → potato_chips_12	support=0.40	confidence=0.67
potato_chips_12 → beer_10 ∧ diapers_b01	support=0.40	confidence=0.50
potato_chips_12 → diapers_b01	support=0.40	confidence=0.50
potato_chips_12 → soda_03	support=0.40	confidence=0.50

For example, the association rule "`beer_10 → potato_chips_12 (support=0.60, confidence=1.00)`" states that every time the product *beer_10* is purchased, the product *potato_chips_12* is purchased too and that this pattern occurs in 60 percent of all transactions. Knowing that 60 percent of customers who buy a certain brand of beer also buy a certain brand of potato chips can help the retailer determine appropriate promotional displays, optimal use of shelf space, and effective sales strategies. As a result of doing this type of association rules discovery, the retailer might decide not to discount potato chips whenever the beer is on sale, as doing so would needlessly reduce profits.

1.2 BASIC ALGORITHM FOR ASSOCIATION RULES DISCOVERY

The first algorithm for association rules discovery was presented in the paper of Agrawal, Imielinski and Swami [1]. The algorithm discovered all association rules whose support and confidence were greater than some user specified minimum values. In [5], an algorithm called *SETM* was proposed to solve this problem using relational operators. In [2], two new algorithms called *Apriori* and *AprioriTID* were proposed. These algorithms achieved significant improvements over the previous algorithms and became the core of many new ones [9, 4, 11, 10, 12, 3].

The algorithm called *Apriori* discovers in a given database all association rules with support and confidence above some minimum values. We assume that items in each itemset are kept sorted in their lexicographic order. In this algorithm, the problem of association rules discovery is decomposed into two subproblems:

1. Iteratively find all possible itemsets that have support greater or equal to a given minimum support value (*minsup*). Itemsets satisfying the above constraint are called *large* itemsets, and all others are *small* itemsets. The first pass of the algorithm counts item occurrences to determine the large 1-itemsets (each 1-itemset contains exactly one item). In each of the next passes, the large itemsets L_{k-1} found in the (k-1)th pass are used to generate the candidate itemsets C_k, using *apriori-gen* function described below. Then, the database is scanned and the support of candidates in C_k is counted. The output of the first phase of the *Apriori* algorithm consists of a set of k-itemsets (k=1, 2, ...), that have support greater or equal to a given minimum support value. Figure 1 presents a formal description of this part of the algorithm.
2. Use the large itemsets to generate the desired rules. For each large itemset *l*, find all non-empty subsets *a* of *l*. For each subset *a*, output a rule of the form $a \rightarrow (l - a)$ if *support(l)*/*support(a)* is greater or equal to a given minimum confidence value (*minconf*). Notice, that if a rule $a \rightarrow (l - a)$ has the confidence value less than *minconf*, then any rule $b \rightarrow (l - b)$, where $b \subset a$, also has the confidence values less than *minconf*. Thus, the rule generation begins with the empty head that is being expanded unless the confidence value falls below *minconf*.

```
$L_1$ = {large 1-itemsets};
for ( $k$ = 2; $L_{k-1} \neq 0$; $k$++) do begin
  $C_k$ = apriori_gen ($L_{k-1}$ );
  forall transactions $t \in D$ do begin
          $C_t$ = subset ($C_k$ , $t$);
          forall candidates $c \in C_t$ do
                  $c$.count ++;
  end
  $L_k$ = { $c \in C_k$ | $c$.count $\geq$ minsup};
end
Answer = $\cup_k$ $L_k$;
```

Figure 1. Large itemset generation phase of Apriori algorithm

In the algorithm *Apriori*, candidate itemsets C_k are generated from previously found large itemsets L_{k-1}, using the *apriori-gen* function. The *apriori-gen* function works in two steps: 1. join step and 2. prune step. First, in the join step, large itemsets from L_{k-1} are joined with other large itemsets from L_{k-1} in the following SQL-like manner:

```
insert into $C_k$
select p.item$_1$, p.item$_2$, ..., p.item$_{k-1}$, q.item$_{k-1}$
from $L_{k-1}$ p, $L_{k-1}$ q
where p.item$_1$ = q.item$_1$
  and p.item$_2$ = q.item$_2$
  ...
  and p.item$_{k-2}$ = q.item$_{k-2}$
  and p.item$_{k-1}$ < q.item$_{k-1}$;
```

Next, in the prune step, each itemset $c \in C_k$ such that some (k-1)-subset of c is not in L_{k-1} is deleted:

```
forall itemsets $c \in C_k$ do
  forall (k-1)-subsets $s$ of $c$ do
    if ($s \notin L_{k-1}$) then delete $c$ from $C_k$;
```

The set of candidate k-itemsets C_k is then returned as a result of the function *apriori-gen*.

2. RD2 System Architecture

RD2 is our prototype extension to *Oracle DBMS* which implements association rules discovery methods. From a user's point of view, *RD2* is a transparent layer, located on top of *Oracle DBMS*, extending *SQL* language with a set of new statements and functions, which can be used to discover association rules in database tables. From a programmer's point of view, *RD2* delivers a universal *API* (Application Programmer Interface), which can be used to build data mining applications.

A user, or an application, specifies data mining problems in the form of declarative queries, similar to those in *SQL*. The queries are processed by *RD2* and the association rules satisfying given constraints are discovered and presented to the user. The user can analyse the rules or store them in the same database for future retrieval.

The general architecture of *RD2* is given in Figure 2. The user application sends the queries via the *RD2* network layer, using *TCP/IP* protocol. The queries are parsed by the extended *SQL* parser, and then either processed by a data mining algorithm or sent directly to the *DBMS* for further processing. Finally, the discovered association rules are returned to the user.

RD2 core modules are located on the machine running the *DBMS*. It results in fast database access, without the need to transfer large data volumes through networks.

We have also developed a user tool for ad-hoc rule queries execution. The tool window is presented in Figure 3 – the user specifies the rule query from keyboard and watch the resulting association rules on the screen.

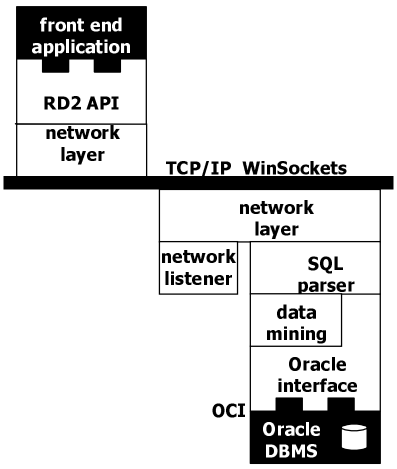
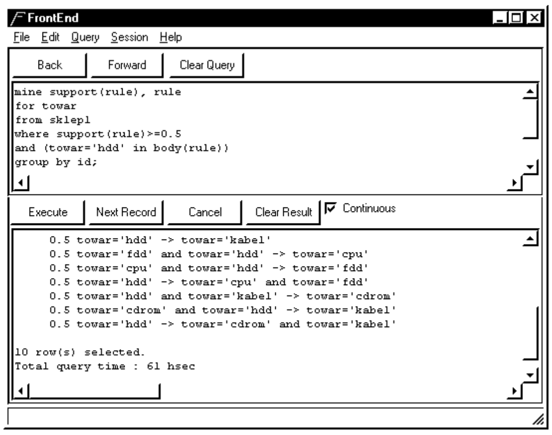


Figure 2. RD2 internal architecture



```
mine support(rule), rule
for towar
from sklep1
where support(rule)>=0.5
and (towar='hdd' in body(rule))
group by id;
```

```
     0.5 towar='hdd' -> towar='kabel'
     0.5 towar='fdd' and towar='hdd' -> towar='cpu'
     0.5 towar='cpu' and towar='hdd' -> towar='fdd'
     0.5 towar='hdd' -> towar='cpu' and towar='fdd'
     0.5 towar='hdd' and towar='kabel' -> towar='cdrom'
     0.5 towar='cdrom' and towar='hdd' -> towar='kabel'
     0.5 towar='hdd' -> towar='cdrom' and towar='kabel'

10 row(s) selected.
Total query time : 61 hsec
```

Figure 3. The user tool for ad-hoc rule queries

2.1 SQL LANGUAGE EXTENSION

We have proposed the extension of industry-standard SQL language to handle data mining, called *MineSQL*. In *MineSQL,* users express their specific problems by means of rule queries. The rule queries are processed by a data mining engine, which generates demanded rules from the given relational database. Sets of discovered rules are then returned to the users.

The main difference between industry-standard *SQL* language and the *MineSQL* language is the addition of the *MINE* statement, which is used to extract rules from database tables. The *MINE* statement can also be used as a query or subquery in another statement (e.g. *SELECT*, *INSERT*). The general syntax of the *MINE* statement is defined as follows:

```
MINE rule_expr [,...]
[FOR {data_expr [USING tax_name][,...]| *}]
[TO  {data_expr [USING tax_name][,...]| *}]
FROM table [, table] ...
[WHERE   {data_condition|rule_condition}
[{AND|OR}{data_condition|rule_condition}]
 ... ]
[GROUP BY data_expr [, data_expr]... [HAVING condition]]
[ORDER BY rule_expr [{ASC|DESC}][,...]]
```

The *MINE* statement produces a set of rules or rule expressions. It defines the structure of rules: the body is defined as a subset of attribute expressions in the *FOR* clause, the head is defined as a subset of attribute expressions in the *TO* clause. The *AS* keyword can specify a taxonomy (conceptual hierarchy) used in generalizing item values. The clause *FROM* specifies which tables or views to explore. The selection constraints for both table data and rules extracted are specified in the *WHERE* section. The *MINE* statement inspects table records grouped by attributes indicated in the *GROUP BY* clause: records belonging to a group are characterized by the same value of the grouping attribute. The result rules may be sorted by the *ORDER BY* expressions in ascending or descending order. We also support user-defined discretization of attribute values by means of stored database procedures.

Rule expressions used in the *MINE* statement are combinations of rules, constants and rule functions. The *MineSQL* language supports a wide collection of rule functions. The rule functions operate on rules. The most commonly used rule functions are:

- `support(r)` - returns the support value of the rule *r*,
- `confidence(r)` - returns the confidence value of the rule *r*,
- `body(r)` - returns the rule *r* body,
- `head(r)` - returns the rule *r* head,
- `bodylen(r)` - returns the number of rule *r* body elements,
- `headlen(r)` - returns the number of rule *r* head elements,
- `rulelen(r)` - returns the number of rule *r* body and head elements.

The *MineSQL* language also allows *IN* and *NOT IN* set operators to be used on a rule, rule body and rule head items. For two sets of items *A* and *B*, the expression *A* IN *B* is true if: $A \neq \emptyset$ and $A \subseteq B$.

To illustrate the *MINE* statement, let us consider the following example.

Example. Assume that a user looks for all association rules about products of today's big-store transactions such that: the body of the rule contains the element *product='milk'* or *product='butter'*, the confidence of the rule is at least 10%, the support is at least 20% and the head of the rule contains only one element. Additionally, the user would like to explore only this year's transactions.

The query specified above is represented in MineSQL as follows:

```
mine rule, support(rule)
for product
from shoppings P
where (‘product=‘‘milk’’’ in body(rule)
       or ‘product=‘‘butter’’’ in body(rule))
  and confidence(rule) > 0.1
  and support(rule) > 0.2
  and headlen(rule)=1
  and p.date >= ‘01.01.97’
group by transaction_id
order by support(rule)
```

Let us assume that the customer's purchase table *Shoppings* is organized as follows:

```
TRANS_ID CUST PRODUCT DATE     PRICE QUANT
-------- ---- ------- -------- ----- ------
1        101  BREAD   17/02/97 0.75  1
1        101  MILK    17/02/97 0.40  2
2        100  BUTTER  17/02/97 1.80  1
2        100  BREAD   17/02/97 0.75  3
2        100  MILK    17/02/97 0.40  1
3        105  BUTTER  28/12/96 1.80  1
4        100  WINE    20/12/96 9.90  1
4        100  MILK    20/12/96 0.40  2
4        100  PAPER   20/12/96 2.50  1
```

Then the result of the query is:

```
RULE                                           SUPPORT(RULE)
--------------------------------------------   -------------
PRODUCT='MILK'->PRODUCT='BREAD'                  1.00
PRODUCT='BUTTER'->PRODUCT='BREAD'                0.50
PRODUCT='BUTTER'->PRODUCT='MILK'                 0.50
PRODUCT='BUTTER' & PRODUCT='MILK'->PRODUCT='BREAD' 0.50
PRODUCT='BREAD' & PRODUCT='BUTTER'->PRODUCT='MILK' 0.50
```

3. Constraints-Driven Algorithm For Association Rules Discovery

In this Section, our constraints-driven algorithm for on-demand mining of association rules is presented. It is built as an extension of the basic *Apriori* algorithm. However, it introduces the general techniques that can be applied to a wide variety of association rules mining algorithms.

The key idea of the constraints-driven algorithm is following. User expresses the rule query using MineSQL language. The rule query contains selection constraints that should be satisfied by extracted rules. These constraints are then converted into *an external selection formula*, which is a logical combination of binary functions, representing simple selection conditions. In the next step, the external selection formula is transformed into an *internal selection formula*, which is directly applied in rule generation phase. For each conjunction of the internal selection formula, a set of rules is generated.

The algorithm reduces the processing time as well as temporary storage requirements by:

- filtering out input item groups that contain unnecessary items (such constraints are called *dataset filtering constraints*), e.g. if only rules involving an item x in the body are of interest, then it is sufficient to process only those record groups that contain x,
- not accepting the candidate itemsets that do not satisfy the extended constraints (not only the minimum support constraint), called *itemset preaccepting constraints*, e.g. if rules of length d are of interest, then it is unnecessary to accept candidates that are longer than d,

- pruning the itemsets accepted in the previous iteration of the algorithm (*itemset accepting constraints*). We can prune only those itemsets that will not be needed to determine rules' confidence factors. For example, if only rules involving an item *x* are of interest, then we can prune all itemsets that do not contain *x*, on the condition that they will not be needed to determine other rules' confidence,
- controlling the rules generation process - pruning the rules that do not satisfy the extended constraints, called *rules accepting constraints*, e.g. if only rules containing *x* in the body are of interest, then it is unnecessary to continue generating rules that do not contain *x* in the body (remember that the rule head can only expand in a rule generation process).

3.1 SELECTION CONSTRAINTS

We first introduce some definitions necessary to define the algorithm.

Definition. External selection predicate σ(class, value) is a binary function that evaluates to 1 if the condition described by *class* is satisfied by a *value*. The *value* is a constant of integer, real, character or set type.
We consider the following external selection predicate types:

- support predicates, concerning the number of data groups that support a rule:
  σ(SG, α) = 1, if: `support(rule) =<` α
  σ(SL, α) = 1, if: α `=< support(rule)`
- confidence predicates, concerning the measure of rule's strength:
  σ(CG, α) = 1, if: `confidence(rule)=<` α
  σ(CL, α) = 1, if: α `=<confidence(rule)`
- contents predicates, concerning restrictions on items that can appear in a rule:
  σ(IH, *A*) = 1, if: `A IN head(rule)`
  σ(IB, *A*) = 1, if: `A IN body(rule)`
  σ(IR, *A*) = 1, if: `A IN rule`
  σ(EB, *A*) = 1, if: `body(rule) = A`
  σ(EH, *A*) = 1, if: `head(rule) = A`
  σ(ER, *A*) = 1, if: `rule = A`
  σ(BI, *A*) = 1, if: `body(rule) IN A`
  σ(HI, *A*) = 1, if: `head(rule) IN A`
  σ(RI, *A*) = 1, if: `rule IN A`
  σ(NIH, *A*) = 1, if: `A NOT IN head(rule)`
  σ(NIB, *A*) = 1, if: `A NOT IN body(rule)`
  σ(NIR, *A*) = 1, if: `A NOT IN rule`
  σ(NEH, *A*) = 1, if: `head(rule) <> A`
  σ(NEB, *A*) = 1, if: `body(rule) <>A`
  σ(NER, *A*) = 1, if: `rule <> A`
  σ(HNI, *A*) = 1, if: `head(rule) NOT IN A`
  σ(BNI, *A*) = 1, if: `body(rule) NOT IN A`
  σ(RNI, *A*) = 1, if: `rule NOT IN A`
- cardinality predicates, concerning the length of the rule, of the body or of the head:
  σ(RLL, α) = 1, if: α `< rulelen(rule)`
  σ(BLL, α) = 1, if: α `< bodylen(rule)`
  σ(HLL, α) = 1, if: α `< headlen(rule)`
  σ(RLS, α) = 1, if: `rulelen(rule) <` α

σ(BLS, α) = 1, if: `bodylen(rule) < α`
σ(HLS, α) = 1, if: `headlen(rule) < α`

- complex predicates, as none of the above:
  σ(CPX, *C*) = 1, if condition *C* is satisfied

Definition. External selection formula *E* is a logical combination of external selection predicates, involving the connectives: ∧, ∨ , (,)

Example. For the following *WHERE* clause of the *MINE* statement:

```
WHERE support(rule) > 0.8
  AND confidence(rule) > 0.5
  AND 'product=''bread''' in body(rule)
  AND 'product=''butter''' in body(rule)
```

the external selection formula is the following:

```
E =   σ(SG, 0.8) ∧ σ(CG, 0.5) ∧ σ(IB, "product='bread'") ∧
      σ(IB, "product='butter'")
```

Definition. Internal selection predicate γ(class, subclass, value) is a binary function that evaluates to 1 if the condition described by *subclass* is satisfied by a *value*. The *value* is a constant of integer, real, character or set type. The *class* determines the verification point in the data mining algorithm.

We consider the following internal selection predicate types:

- dataset filtering predicates (DF), for filtering input item groups that contain unnecessary items:
  γ(DF, FI, *A*) = 1, for all groups that contain all items of *A*,
  γ(DF, PI, *A*) = 1, for all groups that contain at least one item of *A*,
  γ(DF, L, α) = 1, for all groups that contain more than α items,
- itemset preaccepting predicates (IP), for pruning the candidate itemsets:
  γ(IP, ILS, α) = 1, for all itemsets that have less than α items,
  γ(IP, SG, α) = 1, for all itemsets that have support greater or equal to α
- itemset accepting predicates (IF), for pruning the itemsets pre-accepted in the previous iteration of the algorithm:
  γ(IA, SL, α) = 1, for all itemsets that have support less than α,
  γ(IA, FI, *A*) = 1, for all itemsets that contains all items of *A*,
  γ(IA, PI, *A*) = 1, for all itemsets that contain at least one item of *A*,
  γ(IA, ILL, α) = 1, for all itemsets that have more than α items,
- rules accepting predicates (RP), for pruning the rules during rules generation step:
  γ(RA, HLS, α) = 1, for all rules whose head has less than α items,
  γ(RA, BLL, α) = 1, for all rules whose body has more than α items,
  γ(RA, CG, α) = 1, for all rules that have confidence greater or equal to α,
  γ(RA, FIB, *A*) = 1, for all rules whose body contains all items of *A*,
  γ(RA, PIB, *A*) = 1, for all rules whose body contains at least one item of *A*,
  γ(RA, NFI, *A*) = 1, for all rules whose head does not contain all items of *A*,
  γ(RA, NPIH, *A*) = 1, for all rules whose head does not contain any item of *A*,
- rules filtering predicates (RF), for the final pruning of the rules:
  γ(RF, CPX, *C*) = 1, if condition *C* is satisfied

In the above definitions, by *filtering* we mean ‘not taking into account’ and by *pruning* - ‘removing from memory’.

Definition. Internal selection formula *I* is a disjunctive normal form of combination of internal selection predicates (disjunction of conjunctions), involving the connectives: ∧, ∨.

Example. As we will explain it later, the internal representation of the external selection formula *E* from the previous example:

```
E =   σ(SG, 0.8) ∧ σ(CG, 0.5) ∧ σ(IB, "product='bread'") ∧
      σ(IB, "product='butter'")
```

is the following:

```
I =   γ(IP, SG, 0.8, itemset) ∧ γ(RA, CG, 0.5, rule) ∧
      γ(DF, FI, 'bread', itemset) ∧ γ(RA, FIB, 'bread', rule) ∧
      γ(IA, FI, 'bread', itemset) ∧ γ(RP, FI, 'bread', itemset) ∧
      γ(DF, FI, 'butter', itemset) ∧ γ(RP, FI, 'butter', itemset) ∧
      γ(RF, CPX, 'body(rule) contains 'butter'', rule)
```

As it was mentioned before, an external selection formula *E* is transformed into an internal selection formula *I*. The transformation is done according to the transformation table *T*, given in Figure 4. The structure of the transformation table *T* is the following. Each row *k* corresponds to an external selection predicate and each column *l* corresponds to as internal selection predicate. The sign ‘+’ on a crossing (*k*, *l*) means that the external selection predicate *k* is replaced with the internal selection predicate *l*. If more than one internal selection predicate corresponds to an external selection predicate, then the external selection predicate is replaced with a conjunction of those internal selection predicates.

internal / external	DF,FI	DF,PI	DF,IL	IP, ILS	IP,SG	IP,FI	IA, ILL	IA, SL	IA, FI	RP,FI	RP,PI	RP,RL	RP,RS	RA, PI	RA, HS	RA, BL	RA, CG	RA, FIB	RA, PIB	RA, NFIH	RA, NPIH	RA, HI	RF, CPX
SG					+																		
SL																							+
CG																	+						
CL																							+
IH	+									+													+
IB	+								+	+								+					
IR	+									+													+
EB	+								+	+								+					+
EH	+									+													+
BI		+									+								+				+
HI		+									+												+
NIH																				+			+
NIB																							+
NIR						+																	
NEH																				+			+
NEB																							+
HNI																					+		+
BNI																							+
HS															+								
BS																							+
RS				+									+										
HL			+									+											+
BL			+									+				+							
RL			+									+											
CPX																							+

Figure 4. Transformation table

3.2 ALGORITHM

Constraints-driven algorithm for mining association rules satisfying given selection constraints is presented below.

Input: A set of database relations r_1, r_2, ..., r_n, containing item sets, data and rule selection conditions expressed in SQL-like language, minimum confidence value for generated rules - *minconf*,
Output: The set of demanded association rules.
Method:

```
1.  Express the given selection constraints in the form of external selection formula E
2.  Transform E into internal selection formula I
3.  forall conjunctions i of I do
4.    result(i) = ∅
5.      C_1 = {all 1-itemsets from item sets satisfying all DF predicates of  i};
6.      A_1 = {c ∈ C_1 | c satisfies all IP predicates of  i};
7.      for ( k = 2; A_{k-1} ≠ 0; k++) do begin
8.        C_k = apriori_gen (A_{k-1} );
9.        forall candidates c ∈ C_k  do
10.         forall item sets satisfying all DF predicates of  i do
11.           if  contains ( t, c ) then c.support ++;
12.       A_k = { c ∈ C_k | c satisfies all IP predicates of  i};
13.       L_{k-1}  = { a ∈ A_{k-1} | a satisfies all IA predicates of  i};
14.     end
15.     L_k  = { a ∈ A_k | a satisfies all IA predicates of  i};
16.     compute
17.     for (m = 2; m<= k; m++) do
18.       forall itemsets S ∈ L_m do
19.         if S satisfies all IA predicates of i then genrule(S, ∅);
20.   end

procedure genrule (body, head)
begin
  forall item ∈ body do
    if item > maxitem(head) then
      begin
        newbody = body - item;
        newhead = head + item;
        if (newbody→newhead) satisfies all RF predicates od i then
          return (newbody→newhead);
        if (newbody→newhead) satisfies all RA predicates od i then
          genrule (newbody, newhead);
      end;
end;
```

The most important points of the algorithm are the following. In the lines (5) and (10) we filter out those item groups that contain unnecessary items (verifying *dataset filtering constraints*). In (6) and (12) the candidate pre-accepting is done - candidates that do not satisfy *itemset preaccepting constraints* are pruned. In the lines (13) and (15) we prune the unnecessary itemsets of the ones accepted in the previous iteration (according to *itemset accepting constraints*).

The procedure *genrule* receives an accepted itemset and generates all possible body-head combinations. The generation always starts with an empty head and then expands the head recursively with the succeeding body elements. The rules satisfying *rule filtering constraints* are returned. The generation process can be stopped (not continued) if a rule that do not satisfy *rule accepting constraints* is built.

3.3 EXPERIMENTAL RESULTS

To assess the performance of the implemented algorithm, we performed several experiments on PC Pentium 150MHz, with 128 MB of main memory, running *Windows NT*. We used a synthetic database, created by *GEN* generator from *QUEST* project [13]. Several parameters affect the distribution of the synthetic data. These parameters are shown in Table 1 (see [13] for details).

parameter	**value**
n_{trans}	number of item sets, 50,000
n_{items}	number of different items, 2000
t_{len}	average items per set, 5
n_{pats}	number of patterns, 500 and 10000
patlen	average length of maximal pattern, 4
corr	correlation between patterns, 0.25

Table 1. Synthetic data parameters

label	**statement**

A	`MINE rule`
	`IN item`
	`FROM data_table`
	`WHERE support(rule)>=0.0033`

B	`MINE rule`
	`IN item`
	`FROM data_table`
	`WHERE support(rule)>=0.0033`
	`AND body(rule) CONTAINS 10`

C	`MINE rule`
	`IN item`
	`FROM data_table`
	`WHERE support(rule)>=0.0033`
	`AND length(rule)>=4`

D	`MINE rule`
	`IN item`
	`FROM data_table`
	`WHERE support(rule)>=0.0033`
	`AND length(rule)<=2`

E	`MINE rule`
	`IN item`
	`FROM data_table`
	`WHERE support(rule)>=0.0033`
	`AND body(rule)='10 & 20'`
	`AND head(rule)='30 & 40'`

Table 2. Experimental *MineSQL* statements

In the scope of our experiment, we compared the performance of different rule queries. The queries were expressed in *MineSQL* language (Table 2). Figure 5 illustrates execution times of the example rule queries.

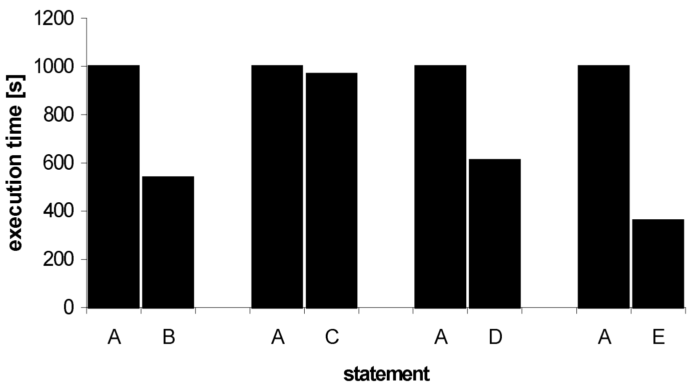


Figure 5 Execution times of example experimental statements

The rule query *A* does not use any advanced features of our constraints-driven algorithm – it plays the role of a reference here. Its execution time presents the capabilities of the original Apriori algorithm. The statements *B, C, D, E* are optimized using additional steps in our algorithm.

4. Conclusions and Future Work

In this paper we have presented the architecture of our prototype DBMS – Data Mining integration. The new SQL-like language for data mining from relational databases, called *MineSQL*, has been presented together with the constraints-driven algorithm used to process its rule queries.

Our future research directions are oriented toward database sequences (ordered sets) processing. We plan to extend *MineSQL* in order to allow sequence processing and frequent subsequence discovery and to develop a similar constraint-driven algorithm for processing new *MineSQL* constructs.

References

1. Agrawal, R., Imielinski, T., Swami, A. Mining Association Rules Between Sets of Items in Large Databases. *Proc. ACM SIGMOD*, pp. 207-216, Washington DC, USA, May 1993.
2. Agrawal, R., Srikant, R. Fast Algorithms for Mining Association Rules. *Proc. 20th Int'l Conf. Very Large Data Bases*, pp. 478-499, Santiago, Chile, 1994.
3. Cheung, D.W., Han, J., Ng, V., Wong, C.Y. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. *Proc. Int'l Conf. Dana Eng.*, New Orleans, USA, February 1996.
4. Han, J., Fu, Y. Discovery of Multiple-Level Association Rules from Large Databases. *Proc. 21th Int'l Conf. Very Large Data Bases*, pp. 420-431, Zurich, Switzerland, Sept. 1995.
5. Houtsma, M., Swami, A. Set-Oriented Mining of Association Rules. *Research Report RJ 9567*, IBM Almaden Research Center, San Jose, California, USA, October 1993.
6. Manilla, H., Toivonen, H., Inkeri Verkamo A. Efficient Algorithms for Discovering Association Rules. *Proc. AAAI Workshop Knowledge Discovery in Databases*, 1994.
7. Morzy, T., Zakrzewicz, M. Constraints-Driven Algorithm for Mining Association Rules On Demand. *Technical Report RA-004/97*, Poznań University of Technology, 1997.
8. Morzy, T., Zakrzewicz, M. SQL-Like Language For Database Mining. *1st Int'l Conference on Advances in Databases and Information Systems*, pp. 311-317, St. Petersburg, 1997.
9. Srikant, R., Agrawal, R. Mining Generalized Association Rules. *Proc. 21th Int'l Conf. Very Large Data Bases*, pp. 407-419, Zurich, Switzerland, Sept. 1995.
10. Srikant, R., Agrawal, R. Mining Quantitative Association Rules in Large Relational Tables. *Proc. 1996 ACM SIGMOD Int'l Conf. Management Data*, pp. 1-12, Montreal, Canada, 1996.
11. Savasere, A., Omiecinski, E., Navathe, S. An Efficient Algorithm for Mining Association Rules in Large Databases. *Proc. 21th Int'l Conf.VLDB*, Zurich, Switzerland, 1995.
12. Toivonen, H. Sampling Large Databases for Association Rules. *Proc. 22nd Int'l Conf. Very Large Data Bases*, Bombay, India, 1996.
13. Agrawal, R., Mehta, M., Shafer, J., Srikant, R., Arning, A., Bollinger, T. The Quest Data Mining System. *Proc. of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, 1996.