SQL-like language for database mining

Tadeusz Morzy, Maciej Zakrzewicz
Institute of Computing Science
Poznań University of Technology
E-mail: morzy@put.poznan.pl
mzakrz@cs.put.poznan.pl

Abstract

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. One of the most interesting and important research problems is discovering of different types of rules (e.g. association, characteristic, discriminant, etc.) from data.

In this work we propose the new SQL-like language for data mining in relational databases, called MineSQL, developed within the scope of the data mining research project led in Poznan University of Technology. MineSQL is the extension of industry standard SQL language developed for expressing rule queries and assisting a user in rule generation, storage and retrieval. We focus on the main features of the language, its syntax and semantics, illustrated by practical examples.

1 Introduction

We are currently witnessing an explosion of interest in data mining technology which is a consequence of growing amount of data collected and warehoused in all industries. 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. The methods of the extraction of useful knowledge from the rapidly growing volumes of data became the subject of the emerging field of knowledge discovery in databases [1, 2, 5, 6]. The discovered knowledge is usually represented by means of rules.

The main step in the *KDD* process is generation of rules from a database. By a rule, informally, we mean a formula of the form $X \rightarrow Y$, where X and Y are conjunctions of expressions $(A = v)$, where A is an attribute and v is a constant. The constant v is either contained in a relation, or is defined by a concept hierarchy, or is defined by a user. For example, the rule shown in Figure 1, generated from a big-store customer's purchase database, states that in 25% of transactions, a customer who purchases bread and milk also purchases butter, with the probability of 75%. 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 all X and Y expressions in one group of records. The rule confidence is the conditional probability to find in the group of records Y, having found X. Data mining distinguishes several types of rules: association rules, characteristic rules, discriminant rules, generalized rules, multiple-level rules etc.

Proceedings of the First East-European Symposium on Advances in Databases and Information Systems (ADBIS'97). St.-Petersburg, September 2–5, 1997.
St.-Petersburg: University of St.-Petersburg, 1997.

The knowledge discovery process is interactive and iterative in nature. A user wants to specify constraints on rules he is looking for, and then to generate the rules. Receiving a result, the user may decide to modify his constraints and to generate the rules again. Such a process requires a high- performance and rapid-response environment that assists a user in data selection, rule generation and rule filtering. The core component of the *KDD* environment is an algorithm for on-demand rules generation.

To assist a user in interactive data mining, several high-level query languages have been proposed [1, 6, 8, 9, 12]. The query language approach assumes that users are capable of expressing their specific problems by means of rule queries, e.g. using a declarative SQL-like language, and that the rules are generated on users demands only. In general, rule queries should generate new rules as well as retrieve the stored ones that were generated before. The query language can play a role of a uniform *API* (Application Programming Interface) for building business applications dealing with knowledge discovery.

RULE	C.	S.
product='bread' & product='milk' → product='butter'	0.75	0.25

Figure 1: Example of a rule

1.1 Problem

In the knowledge discovery process, users can express their specific problems by means of rule queries. The rule queries are processed by a *KDD* Management System (*KDDMS*) [1], which generates demanded rules from given database relations. Sets of generated rules are then returned to the users (see Figure 2). There is a need for an effective and clear query language, in which users could specify their queries, store and retrieve rules, define and utilize concept hierarchies etc.

In this paper we discuss the new SQL-like language for data mining, called MineSQL, developed within the scope of the data mining research project led in Poznan University of Technology. MineSQL is the extension of industry standard SQL language to assist a user in rule generation, storage and retrieval. We focus on the main features of the language, its syntax and semantics, illustrated by practical examples.

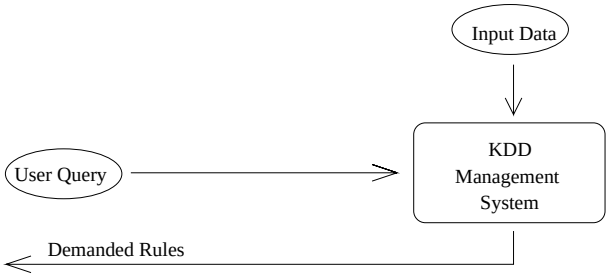

Figure 2: KDD Process

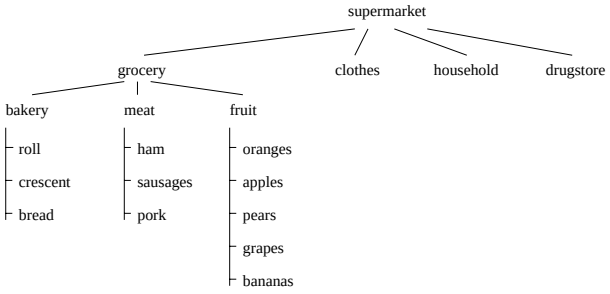

Figure 3: A taxonomy example

1.2 Knowledge rules

Let r be a relation over the scheme $R(A_1, A_2, \ldots, A_n)$, where A_i is an attribute. Records from r are divided into record groups $G_1, G_2, ..., G_k$. Let the expression $(A_i = v)$ mean that the attribute A_i has the value v. Let X and Y be conjunctions of expressions $(A_i = v)$, i.e. $X = (A_a = v_a) \wedge (A_b = v_b) \wedge \ldots \wedge (A_z = v_z)$ and $Y = (A_\alpha = v_\alpha) \wedge (A_\beta = v_\beta) \wedge \ldots \wedge (A_\omega = v_\omega)$, and $X \cap Y = \emptyset$. The knowledge rule over r is an expression of the form $X \rightarrow Y$. We say that a group of records G_i *supports a conjunction* X, if for each $(A_i = v)$ from X, the attribute A_i has in the group G_i the value v. We say that a group of records G_i *supports a rule* $X \rightarrow Y$, if for each $(A_i = v)$ from $X \cup Y$, the attribute A_i has in the group G_i the value v. For a rule $X \rightarrow Y$ we define two measures of statistical significance and strength: *support* and *confidence*:

$$support(X \rightarrow Y) = \frac{\text{num. of record groups supporting } X \rightarrow Y}{\text{num. of all record groups}}$$

$$confidence(X \rightarrow Y) = \frac{\text{num. of record groups supporting } X \rightarrow Y}{\text{num. of record groups supporting } X}$$

We say that a group of records G_i *satisfies* the rule $X \rightarrow Y$, if the group supports the rule. We say that a group of records G_i *violates* the rule $X \rightarrow Y$, if the group supports the conjunction X but does not support the rule. We refer to X as the body and to Y as the head of the rule. Data mining distinguishes several types of rules: association rules, characteristic rules, discriminant rules, generalized rules, multiple-level rules etc. The basic difference between those types of rules is the way of grouping records, over which they are generated. The records may be grouped according to the value of a certain attribute (association rules), or each record can make a separate group (characteristic and discriminant rules).

A *characteristic rule* presents the characteristics of the data set, defined by a specific value for a given attribute. For example, the symptoms of a specific disease can be summarized by a set of characteristic rules. Then, the characteristic rule body represents the symptoms and the head represents the disease. A similar rule type is a *classification rule*, that is used to classify future data into a collection of known data sets.

A *discriminant rule* contains features or properties that distinguish the data set being examined from other data sets. For example, to distinguish one disease from others, a discriminant rule summarizes the symptoms that differentiate this disease from others. The meaning of discriminant rule body and head is the same as of the characteristic rule.

An *association rule* represents connections or associations between items in data groups. For example, one may discover that a set of symptoms often occur together with another set of symptoms, and then study the reasons of this association. To discover association rules, a user need first to define the method of data grouping (e.g. by personal id number, by date, by disease).

In many cases, the data from a database can be additionally described by some background knowledge. For example, products in a big-store are divided into multiple-level is-a groups, e.g. bread is a bakery item, bakery is a grocery item and finally grocery is a product item. To utilize the background knowledge in a rule generation process, the term of conceptual hierarchies was introduced.

A *conceptual hierarchy*, also called *taxonomy*, consists of a set of nodes organized in a tree, where nodes in the tree represent values of an attribute, called *concepts*. The *generalized* value of an attribute is the replacement of its value with its ancestor located on given *level* in the conceptual hierarchy. The example of a conceptual hierarchy is shown in Figure 3. Generalized values for apples are: fruit, grocery, supermarket. Conceptual hierarchies are provided by domain experts or users, or generated automatically.

A *generalized rule* (association, characteristic etc.) is the rule that refers to generalized values of its attributes. All values are generalized on the same level. A rule that refers to values generalized on multiple levels is called a *multiple-level rule*.

1.3 Related work

Several rule query languages and operators have been proposed so far [6, 8, 9, 10, 12]. These languages can be classified as follows: (1) declarative rule query languages that extend traditional SQL, (2) logic programming languages, and (3) template and graphical languages. We briefly review their main features.

Recently, certain declarative rule query languages extending the traditional SQL have been proposed [6, 8, 9]. In [6], a new SQL-like operator, named MINE RULE was presented. The operator is capable of expressing the problems concerning the mining of association rules only. In [8], a data mining query language DMQL for expressing problems related to association, classification, discriminant and generalized rules discovery was introduced. In [9], an additional MINE() operator to SQL was added, allowing propositional rule generation from the database as well as retrieval from the rulebase, created to store some of the previously generated rules.

Shen, et al. [12] consider the logic programming language LDL++ as a metapattern querying language. In [10], the authors introduce a formalism of rule templates, that allow describing the structure of demended rules.

1.4 Outline

The paper is organized as follows. In Section 2 we present the elements of MineSQL language. Section 3 contains practical examples of MineSQL statements. In Section 4, query execution and rule generation issues are addressed. Section 5 contains final conclusions.

2 MineSQL language

2.1 Rule datatype

In this Section we introduce a new datatype, called *RULE*, which is used to store and manipulate rules. Each *RULE* consists of the following components: set of body elements (expressions $(A_i = v)$), set of head elements (expressions $(A_i = v)$), support and confidence values. The *RULE* datatype can be also used in SQL *CREATE TABLE* statement to describe the type of an attribute of a created table.

To help a user in rule managing, we define several rule functions. The rule functions provide access to the rule elements (body, head, support, confidence) as well as convert values from *RULE* datatype to other datatypes, or vice versa. 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 all rule r body and head elements,
- *bodyname(r, n)* - returns the attribute name of the n-th body element,
- *headname(r, n)* - returns the attribute name of the n-th head element,
- *bodyval(r, n)* - returns the attribute value of the n-th body element,
- *headval(r, n)* - returns the attribute value of the n-th head element,
- *to_char(r, fmt)* - converts a value of *RULE* datatype to a value of *CHAR* datatype, using the optional format string fmt,
- *to_rule(char, fmt)* - converts a value of *CHAR* datatype to a value of *RULE* datatype, using the optional format string fmt.

In order to join rules with records or groups of records stored in database tables, two rule operators are defined: *SATISFIED BY* and *VIOLATED BY*. The operator r *SATISFIED BY* rec evaluates to *True* if the record or group of records rec satisfies the rule r. The operator r *VIOLATED BY* rec evaluates to *True* if the record or group of records rec violates the rule r. For example, users can easily find records that violate rules stored in a prepared table.

The main advantage of introducing the new datatype is that rules can be integrated into existing databases of structured data. Hence, discovered rules can be stored in database tables, together with their descriptions, rule numbers and date of discovery. Then, both the structured data and the rules can be searched in a single query. However, rules managed by *KDDMS* may be physically stored in relational database tables as well as in external repositories, such as object databases or plain files.

2.2 Conceptual hierarchies and user-defined attributes

Sets of generalized attribute values are represented in the form of conceptual hierarchies. To store a conceptual hierarchy structure, we introduce a new database object type, called *TAXONOMY*. Each *TAXONOMY* has a unique name and contents. The contents is a directed acyclic graph on the set of attribute values and domain concept values. Each domain concept value is stored in a *NODE* and each attribute value is stored in a *LEAF*. Taxonomies are defined and managed by a user by means of additional MineSQL statements:

- *CREATE TAXONOMY* - defines the conceptual hierarchy name and optionally, its nodes and leaves,
- *ALTER TAXONOMY* - modifies contents of a conceptual hierarchy,
- *DROP TAXONOMY* - removes given conceptual hierarchy,

In many cases, users want to generate rules from discretized data or complex aggregate values. Such values are called *user-defined attributes* (or *virtual attributes*). They are treated as table attributes, but they are not materialized any time. On every external reference to a user-defined attribute, the attribute's value is calculated. The user-defined attributes can be implemented by users by means of standard database stored functions. MineSQL allows generating rules over such user-defined attributes.

2.3 MINE statement

Now, we present a new statement, called *MINE*, which is used to control rule generation from the database. The *MINE* statement can also be used as a query or subquery in another statement (e.g. *SELECT*, *INSERT*). The syntax of the *MINE* statement is defined as follows:

```
MINE rule_expr [[AS] alias][,...]
[FOR {data_expr [USING tax_name][AS
alias][,...]|*}]
[TO  {data_expr [USING tax_name][AS
alias][,...]|*}]
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}][,...]]
```

In the above description, *rule_expr* denotes rule expression, which is a combination of: pseudo-attribute *RULE*, constants and rule functions. The pseudo-attribute *RULE* represents the rule being generated. *Data_expr* denotes data expression, which is a combination of: attributes, constants, user-defined attributes and database functions. *Data_condition* and *rule_condition* denotes constraints on data used to generate rules, and constraints on rules generated. The *rule_condition* section of the *MINE* statement 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 $A \subseteq B$. The *MINE* statement generates 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. If both *FOR* and *TO* clauses are omitted, all attributes are used. If the *TO* clause is omitted, the head contains the same set of attribute expressions that *FOR*clause does. The *USING*

trans_id	customer	product	date	day	hour	quantity
1	100	roll	30-12-96	Monday	9:20	6
1	100	crescent	30-12-96	Monday	9:20	3
1	100	pork	30-12-96	Monday	9:20	5
2	100	roll	30-12-96	Monday	11:10	3
2	100	crescent	30-12-96	Monday	11:10	2
2	100	sausage	30-12-96	Monday	11:10	10
2	100	apple	30-12-96	Monday	11:10	20
2	100	pork	30-12-96	Monday	11:10	10
3	101	bread	30-12-96	Monday	12:40	1
3	101	ham	30-12-96	Monday	12:40	1
3	101	oranges	30-12-96	Monday	12:40	10
4	102	apple	02-01-97	Thursday	16:00	5
4	102	pears	02-01-97	Thursday	16:00	5
4	102	pork	02-01-97	Thursday	16:00	5
4	102	crescent	02-01-97	Thursday	16:00	2
5	103	bananas	03-01-97	Friday	9:50	3
5	103	oranges	03-01-97	Friday	9:50	7

Table 1: The *Shoppings* relation

keyword specifies a taxonomy used in the process of generating generalized rules. Each body and head element as well as a rule expression can be aliased by means of *AS* keyword. The clause *FROM* specifies which tables or views to explore. The selection of both table data and rules extracted depends on conditions specified in the *WHERE* section. The *MINE* statement inspects table records grouped by attributes indicated by 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.

3 Examples

3.1 Example Relation

In this section we illustrate our *MINE* statement showing its application to mining problems based on a big-store relation *Shoppings* collecting purchase data. Customers buy a set of *products*. The whole purchase is referred to as a *transaction* with a unique identifier, a date, a day, an hour and a customer identifier. Each transaction contains the set of bought products with the purchased quantity. The customer's purchase relation *Shoppings* is depicted in Table 1.

3.2 Mining of simple association rules

Let us assume that a user looks for all association rules about products of big-store transactions such that the rule support value is greater than 50%. The MineSQL query is represented as follows:

```
MINE rule, support(rule) s.,
confidence(rule) c.
FOR product
FROM shoppings
WHERE support(rule) > 0.5
GROUP BY trans_id
```

The rule selection condition is formulated in the *WHERE* section of the query. Only rules with confidence value greater than 0.5 will be generated. Records are grouped by the value of *trans_id*. The result of the query is the following:

rule	s.	c.
product='crescent' → product='pork'	0.6	1.0
product='pork' → product='crescent'	0.6	1.0

Two rules with support greater than 50% have been found in the example database.

The MineSQL language provides advanced rule selection techniques. As an example, we present the query that generates all rules containing the product "crescent" in their body. Moreover, let us assume that we are interested only in those rules, whose body contains exactly two elements and whose support is greater than 20%. The query is constructed as follows:

```
MINE rule, support(rule) s.,
confidence(rule) c.
FOR product
FROM shoppings
WHERE 'product=''crescent''' IN body(rule)
AND bodylen(rule) = 2
AND support(rule) > 0.2
GROUP BY trans_id
```

The rule function *bodylen(rule)* is used to determine the number of rule body elements. The condition *'product = "crescent"' IN body(rule)* evaluates to *True* for all rules, whose body contains the element *product = "crescent"*. The query gives the following result:

rule	s.	c.
product='roll' & product='crescent' → product='pork'	0.4	1.0
product='crescent' & product='pork' → product='roll'	0.4	0.66

3.3 Generalized and multiple-level rules

To generate generalized and multiple-level rules, users first define their individual conceptual hierarchies. The hierarchy is defined for a particular attribute. In this example, we present the statement that creates a new *TAXONOMY* object for a part of the conceptual hierarchy shown in Figure 3. The statement is as follows:

```
CREATE TAXONOMY supermarket_taxonomy
(NODE 'supermarket',
NODE 'grocery' REFERENCES 'supermarket',
NODE 'bakery' REFERENCES 'grocery',
LEAF 'roll' REFERENCES 'bakery',
LEAF 'crescent' REFERENCES 'bakery',
LEAF 'bread' REFERENCES 'bakery')
```

After the concept hierarchy has been defined, generalized and multiple-level rules can be generated. The next example presents the *MINE* statement that generates multiple-level rules about prod-

ucts, using previously created taxonomy. Only those rules, whose support value is greater than 50% and confidence value is greater than 80% are returned. The result rules are ordered according to the value of the rule support.

```
MINE rule, suport(rule) s., confidence(rule)
c.
FOR product USING supermarket_taxonomy
FROM shoppings
WHERE confidence(rule) > 0.8
AND support(rule) > 0.5
GROUP BY trans_id
ORDER BY support(rule) DESC
```

In the above statement, the keyword *USING* specifies the name of the conceptual hierarchy to be used for the attribute *product*. To sort the result rules, *ORDER BY* clause has been added. The generated set of multiple-level association rules is as follows:

rule	s.	c.
product='bakery' → product='meat'	0.8	1.0
product='meat' → product='bakery'	0.8	1.0
product='crescent' → product='pork'	0.6	1.0
product='pork' → product='crescent'	0.6	1.0
product='bakery' → product='fruit'	0.6	1.0
product='bakery' & product='fruit' → product='meat'	0.6	1.0
product='pork' → product='bakery'	0.6	1.0
product='crescent' → product='meat'	0.6	1.0

Notice, that the above table of rules contains both one-level rules (e.g. *product = 'bakery' → product = 'meat')* as well as multiple-level rules (e.g. *product = 'pork' → product = 'bakery'*).

In the next example, we show the MineSQL ability to generate generalized rules, whose conceptual level is specified by a user. Let us assume, that the user looks for association rules at the second level of the hierarchy *supermarket_taxonomy*:

```
MINE rule, support(rule) s.,
confidence(rule) c.
FOR TAXNODE(product, 2)
FROM shoppings
WHERE confidence(rule) > 0.8
GROUP BY trans_id
```

The rule query result is the following:

rule	s.	c.
product='bakery' → product='meat'	0.8	1.0
product='meat' → product='bakery'	0.8	1.0
product='bakery' → product='fruit'	0.6	1.0
product='bakery' & product='fruit' → product='meat'	0.6	1.0

All rules returned by the above query contain the values from one conceptual level only.

3.4 Multiattribute association rules

Traditional association rules operate on values of only one attribute. The MineSQL language allows mining of multiattribute association rules, that operate on values of two or more attributes. As an example, we show the MineSQL ability to generate association rules between products and their day of purchase. The query is represented as follows:

```
MINE rule, support(rule) s.,
confidence(rule) c.
FOR product
TO day
FROM shoppings
WHERE support(rule) > 0.3
AND confidence(rule) > 0.8
GROUP BY trans_id
```

The bodies of generated rules will contain the attribute *product*, specified in the *FOR* clause. The heads of the rules will contain the attribute *day*, specified in the *TO* clause. The result of the query is the following:

rule	s.	c.
product='roll' → day='Monday'	0.4	1.0
product='roll' & product='crescent' & product='pork' → day='Monday'	0.4	1.0
product='roll' & product='crescent' → day='Monday'	0.4	1.0
product='roll' & product='pork' → day='Monday'	0.4	1.0

3.5 User-defined attributes

To generate rules over user-defined attributes, we first define the database function *quarter* that returns the quarter's name of the given date:

```
function quarter(date) returns char is
begin
  if month(date) in (1,2,3) then return
'1st'
  elsif month(date) in (4,5,6) then return
'2nd'
  elsif month(date) in (7,8,9) then return
'3rd'
  elsif month(date) in (10,11,12) then
return '4th'
  end if;
end;
```

Now, we generate rules over values of the attribute *product* and the new virtual attribute *quarter*. The attribute *product* will appear in the rule body, whereas the head of the rule will contain the user-defined attribute *quarter*.

```
MINE rule
FOR product
TO quarter(date) AS quarter
FROM shoppings
WHERE support(rule) > 0.3
AND confidence(rule) > 0.8
GROUP BY trans_id
```

The user-defined attribute is specified in *TO* clause, because it will be contained in the rule head. The rules found are the following:

rule
product='roll' → quarter='4th'
product='roll' & product='crescent' & product='pork' → → quarter='4th'
product='roll' & product='crescent' → quarter='4th'
product='roll' & product='pork' → quarter='4th'

The values of the user-defined attributes are calculated by the database function *quarter()*. The above example showed the method of simple discretization of continuous values.

3.6 Mining of characteristic rules

Suppose, we would like to find rules characterizing the value *morning* of the user-defined attribute *time(hour)*. The definition of the user-defined attribute *time* is given below, followed by the rule generation statement.

```
function time(hour) returns char is
begin
  if hour < '06:00' then return 'night'
  elsif hour < '12:00' then return 'morning'
  elsif hour < '18:00' then return
'afternoon'
```

```
  elsif hour < '24:00' then return 'evening'
  end if;
end;

MINE rule, suport(rule) s., confidence(rule)
c.
FOR product, customer
TO time(hour) AS time
FROM shoppings
WHERE head(rule) = 'time=''morning'''
AND support(rule) > 0.1
```

The above statement does not contain a *GROUP BY* clause. It means that the default grouping is used. By default, the *MINE* statement processes groups of one record each. The attributes for the rule body are *product* and *customer*, and the attribute for the rule head is *time*. The result of the query is the following:

rule	s.	c.
customer=100 & product='roll' → time='morning'	0.12	1.0
customer=100 & product='crescent' → time='morning'	0.12	1.0
customer=100 & product='pork' → time='morning'	0.12	1.0
customer=103 → time='morning'	0.12	1.0
customer=100 → time='morning'	0.47	1.0
product='roll' → time='morning'	0.12	1.0
product='crescent' → time='morning'	0.12	0.66
product='pork' → time='morning'	0.12	0.66

All the above rules characterizes the shoppings purchased at the morning.

3.7 Storing rules in tables

Rules generated by means of the *MINE* statement can be stored in database tables. The tables are created with SQL *CREATE TABLE* statement. Let us create the following table:

```
CREATE TABLE my_rules
(r RULE,
description CHAR(20))
```

The table *my_rules* has two attributes: *r* which contains a rule, and *description*, containing additional comments about the rule. The table *my_rules* can be filled with the result of the following example *MINE* statement:

```
INSERT INTO my_rules (r)
MINE rule, suport(rule), confidence(rule)
FOR product, customer
TO time(hour) AS time
FROM shoppings
WHERE head(rule) = 'time=''morning'''
AND support(rule) > 0.1
```

The *MINE* statement above is a subquery of the standard SQL *INSERT* statement. As the result, all the rules generated by the *MINE* statement will be inserted into the table *my_rules*. Having stored the rules, we can find all the tuples that violates the rules. For example, the following statement find all tuples in *shoppings* table that violate rules stored in the table *my_rules*:

```
SELECT s.trans_id, s.customer, s.product
FROM shoppings s, my_rules m
WHERE m.r VIOLATED BY s.*
```

The result of the query is the following:

s.trans_id	s.customer	s.product
4	102	pork
4	102	crescent

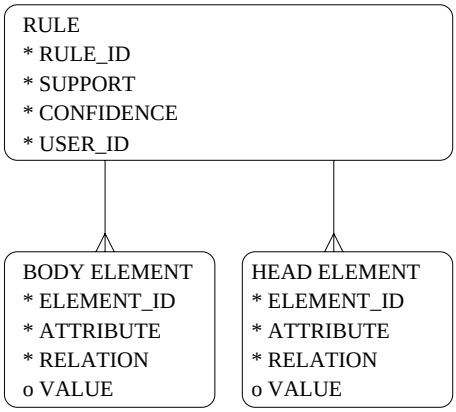

Figure 4: Data model for rule storage

4 Rule generation and storage

The interactive approach to data mining presented in the paper requires fast and effective *KDDMS* that is capable of on-demand generation of rules. Most of known algorithms for mining rules generate the rules that have only support and confidence greater than given minimum values [3, 4, 7, 13, 14, 15]. The application of these algorithms to the on-demand rule generation is straightforward, though very ineffective. It consists in first generating all rules satisfying only support and confidence constraints, and then filtering the rules with the remaining constraints. However, this approach to query execution is time and memory consuming.

Within the scope of the data mining research project led in Poznan University of Technology, we developed a constraints-driven algorithm that employs the specific rule query constraints to control the association rule generation process [11]. 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 generated rules. These constraints are then converted into the form which is directly applied in generating the rules. We showed that utilizing the query constraints knowledge by the core of the data mining algorithm may reduce the processing time as well as temporary storage requirements.

MineSQL allows logical storing of the generated rules in database tables. Physically, the rules managed by *KDDMS* may be stored in relational database tables as well as in external repositories, such as object databases or even plain files. As an example, in Figure 4 we present the data model (*ER* model) that can be used to physically store rules.

To generate generalized and multiple-level rules, concept hierarchies should be stored in the database. The conceptual hierarchies (TAXONOMY) can be internally stored by means of database tables. As an example, in Figure 5 we present the relational structure that can be used to internally store the conceptual hierarchies.

5 Conclusions

We presented the new SQL-like language for data mining from relational databases, called MineSQL, developed within the scope of the data mining project led at Poznań University of Technology. MineSQL is the extension of industral standard SQL. It allows users to express different types of rule queries for discovering association, characteristic, classification and discriminant rules. Moreover, MineSQL allows specifying the generalized, multiple-level and multiattribute asociation rules as well as specifying rule queries over user-defined attributes. The main features of the language, its syntax and semantics are illustrated by practical examples.

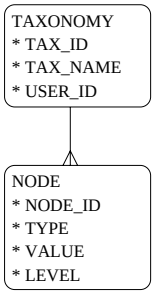

Figure 5: Data model for taxonomy storage

In the paper we focused our attention on the rule query language. Besides the problem of the language for expressing rule queries there are a lot of problems that should be solved and that we have looked into: generation of rules on-demand, storing and indexing rules, handling attributes with continuous values, etc. Moreover, in the same way as today's business applications are currently supported using SQL based API, the data mining applications needs to be provided an application development support. The development of rule querying tools for data mining is one of the big challenges to the database and machine learning communities.

References

[1] Imielinski T., Manilla H., *A Database Perspective on Knowledge Discovery*, Communications of the ACM, Vol. 39, No. 11, Nov. 1996

[2] Agrawal R., Imielinski T., Swami A., *Database Mining: A Performance Perspective*, IEEE Transactions on Knowledge and Data Engineering", Vol. 5, No. 6, Dec. 1993

[3] Agrawal R., Imielinski T., Swami A., *Mining Association Rules Between Sets of Items in Large Databases*, Proc. of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, 1993,

[4] Agrawal R., Srikant R., *Fast Algorithms for Mining Association Rules*, Proc. of the 20th VLDB Conference, Santiago, Chile, 1994,

[5] Fayyad U., Piatetsky-Shapiro G., Smyth P., *The KDD Process for Extracting Useful Knowledge from Volumes of Data*, Communications of the ACM, Vol. 39, No. 11, Nov. 1996

[6] Ceri S., Meo R., Psaila G., *A New SQL-like Operator for Mining Association Rules*, Proceedings of the 22nd VLDB Conference, Bombay, India, 1996

[7] Han J., Fu Y., *Discovery of Multiple-Level Association Rules from Large Databases*, Proc. of the 21st VLDB Conf., Swizerland, 1995,

[8] Han J., Fu Y., Wang W., Chiang J., Gong W., Koperski K., Li D., Lu Y., Rajan A., Stefanovic N., Xia B., Zaiane O.R., *DBMiner: A System for Mining Knowledge in Large Relational Databases*, Proc. Int'l Conf. Data Mining and Knowledge Discovery, Portland, Oregon, August 1996

[9] Imielinski T., Virmani A., Abdulghani A., *Discovery board application programming interface and query language for database mining*, Proc. of KDD96, Portland, Oregon, August 1996

[10] Klemettinen M., Manilla H., Ronkaien P., Toivonen H., Verkamo A. I., *Finding Interesting Rules from Large Sets of Discovered Association Rules*, Proc. of the Third Int'l Conf. on Information and Knowledge Management, Maryland, 1994

[11] Morzy T., Zakrzewicz M., *Constraints-Driven Algorithm for Mining Association Rules On Demand*, Technical Report, Poznan University of Technology, 1997

[12] Shen W.M., Ong K., Mitbander B., Zaniolo C., *Metaqueries for Data Mining, Advances in Knowledge Discovery and Data Mining*, 1996

[13] Savasere A., Omiecinski E., Navathe S., *An Efficient Algorithm for Mining Association Rules in Large Databases*, Proc. of the 21st VLDB Conference, Zurich, Swizerland, 1995,

[14] Srikant R., Agrawal R., *Mining Generalized Association Rules*, Proc. of the 21st VLDB Conf., Swizerland, 1995,

[15] Toivonnen H., *Sampling Large Databases for Association Rules*, Proc. of the 22nd VLDB Conf., India, 1996,