Data Mining Support in Database Management Systems

Tadeusz Morzy, Marek Wojciechowski, Maciej Zakrzewicz

Poznan University of Technology, Poland
{morzy,marek,mzakrz}@cs.put.poznan.pl

Abstract. The most popular data mining techniques consist in searching databases for frequently occurring patterns, e.g. association rules, sequential patterns. We argue that in contrast to today's loosely-coupled tools, data mining should be regarded as advanced database querying and supported by Database Management Systems (DBMSs). In this paper we describe our research prototype system, which logically extends DBMS functionality, offering extensive support for pattern discovery, storage and management. We focus on the system architecture and novel SQL-based data mining query language, which serves as the user interface to the system.

1 Introduction

The primary goal of data mining is to discover frequently occurring, previously unknown, and interesting patterns from large databases [8]. The discovered patterns are usually represented in the form of association rules or sequential patterns. The results of data mining are mostly used to support decisions, observe trends, and plan marketing strategies. For example, the association rule "A & F -> D" states, that the purchase of the products A and F is often associated with the purchase of the product D.

From the conceptual point of view, data mining can be perceived as advanced database querying, since the resulting information in fact exists in the database, but it is difficult to retrieve it. For this reason, there is a very promising idea of integrating data mining methods with DBMSs [14][17], where users specify their problems by means of data mining queries. This leads to the concept of on-line data mining, fully supported by the DBMS architecture (similarly to OLAP).

We have built a research prototype system, called *RD2*, which logically extends DBMS functionality and allows users to mine relational databases. Users or user applications communicate with *RD2* by means of *MineSQL* language, used to express data mining problems. *MineSQL* is a multipurpose, declarative language, based on *SQL*, for discovering, storing and managing association rules and sequential patterns. The novel ideas of data mining views and materialized views are also implemented in *RD2* and supported by *MineSQL*. In this paper we focus on *MineSQL* language and its usage in the area of data mining.

1.1 Basic Definitions

Association rules

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 rule $X \rightarrow Y$ holds in the set D with support s if $100*s\%$ of itemsets in D support $X \cup Y$. The rule $X \rightarrow Y$ has confidence c if $100*c\%$ of itemsets in D that support X also support Y.

Sequential patterns

Let $L = \{l_1, l_2, ..., l_m\}$ be a set of literals called items. An *itemset* is a non-empty set of items. A *sequence* is an ordered list of itemsets and is denoted as $<X_1 X_2 ... X_n>$, where X_i is an itemset ($X_i \subseteq L$). X_i is called an *element* of the sequence. Let D be a set of variable length sequences, where for each sequence $S = <X_1 X_2 ... X_n>$, a timestamp is associated with each X_i.

With no time constraints we say that a sequence $X = <X_1 X_2 ... X_n>$ is *contained* in a sequence $Y = <Y_1 Y_2 ... Y_m>$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i1}$, $X_2 \subseteq Y_{i2}$, ..., $X_n \subseteq Y_{in}$. We call $<Y_{i1}, Y_{i2}, ..., Y_{in}>$ an *occurrence* of X in Y. We consider the following user-specified time constraints while looking for occurrences of a given sequence: minimal and maximal gap allowed between consecutive elements of an occurrence of the sequence (called *min-gap* and *max-gap*), maximal duration (called *time window*) of the occurrence and time *tolerance*, which represents the maximal time distance between two itemsets to treat them as a single one.

A *sequential pattern* is a sequence whose statistical significance in D is above user-specified threshold. We consider two alternative measures of statistical significance for sequential patterns: *support* and *number of occurrences*. The support for the sequential pattern $<X_1 X_2 ... X_n>$ in the set of sequences D is the percentage of sequences in D that contain the sequential pattern. While counting the support it is not important how many times a pattern occurs in a given data sequence. This makes support unsuitable when sequential patterns are mined over a single data sequence ($|D| = 1$). In such case, the number of occurrences is more useful as a statistical measure.

Generalized patterns

In many applications, interesting patterns between items often occur at a relatively high concept level. For example, besides discovering that "40% of customers who purchase *soda_03*, also purchase *potato_chips_12*", it could be informative to also show that "100% of customers who purchase *any of beverages* also purchase *potato_chips_12*". Such association rules or sequential patterns utilize conceptual hierarchy information and are called, respectively, *generalized association rules* and *generalized sequential patterns*.

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 a given *level* in the conceptual hierarchy. Conceptual hierarchies are provided by domain experts or users, or generated automatically.

1.2 Related Work

The problem of mining association rules was first introduced in [1] and an algorithm called *AIS* was proposed. In [3], two new algorithms were presented, called *Apriori* and *AprioriTid* that are fundamentally different from previous ones. In [6] a new algorithm, called Max-Miner, was introduced to efficiently mine long association patterns. In [5] an algorithm exploiting all user-specified statistical constraints (including minimum support and confidence) was presented. In [10] and [15] the problem of finding generalized (also called multiple-level) association rules based on a user-defined taxonomy was addressed.

The problem of mining frequent patterns in a set of data sequences together with a few mining algorithms was first introduced in [4]. The class of patterns considered there, called sequential patterns, had a form of sequences of sets of items. The statistical significance of a pattern (called support) was measured as a percentage of data sequences containing the pattern. In [16], the problem was generalized by adding taxonomy on items and time constraints such as min-gap, max-gap and sliding window (in this paper called tolerance).

Another formulation of the problem of mining frequently occurring patterns in sequential data was given in [13], where discovered patterns (called episodes) could have different type of ordering: full (serial episodes), none (parallel episodes) or partial and had to appear within a user-defined time window. The episodes were mined over a single event sequence and their statistical significance was measured as a percentage of windows containing the episode (frequency) or as a number of occurrences.

In [11], the issue of integrating data mining with current database management systems was addressed and a concept of KDD queries was introduced. Several extensions of *SQL* were presented to handle association rules queries [7][9][12]. In [12], storing of discovered rules in a rulebase was discussed.

In recent years many data mining projects have been developed. Some of them resulted in commercial products. The two particularly interesting data mining systems are *DBMiner* [9] and *IBM Intelligent Miner* (which evolved from the *Quest* project [2]). They are both multi-purpose data mining tools that can be used to discover various frequently occurring patterns in data but they also offer support for high-level tasks such as classification or clustering. Both tools have a graphical user interface that can be used to select the source dataset, the desired data mining method and parameters required by the selected method. In addition, *DBMiner* offers a *SQL*-like data mining query language (*DMQL*) that can be used to specify data mining tasks. Although the systems currently available offer some level of integration with DBMSs, they do not provide mechanisms for storage and management of data mining results.

2 RD2 Architecture

RD2 is a data mining system for relational databases, built initially on top of Oracle database management system. *RD2* discovers association rules and sequential patterns in database tables, according to users' needs expressed in the form of data mining queries. The system uses *RD2* Network Adapter to communicate with user applications and *RD2* Database Adapter to communicate with the database management system. The *RD2* system can work in 2-tier architecture, residing on the database computer, as well as in 3-tier architecture, when residing on a separate computer. Figure 1 gives an overview of *RD2* system architecture.

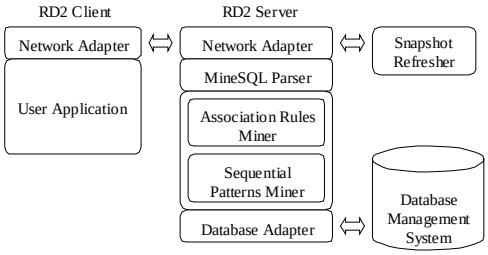


Fig. 1. RD2 Architecture

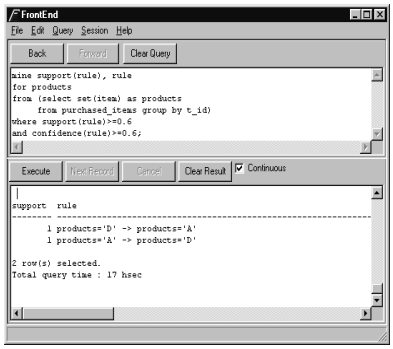


Fig. 2. RD2 FrontEnd application

RD2 Network Adapter is a module for network communication between a client application and *RD2* server. It uses TCP/IP protocol to transmit data mining queries generated by the client application to the server and to transmit the discovered association rules or sequential patterns from the server back to the client application. *RD2* Network Adapter contains the programmer's interface (*RD2 API*), which is used by client applications, cooperating with *RD2*. Advanced users can also use a tool, called *RD2* FrontEnd, which is an *RD2* application for ad-hoc data mining. Users can execute their data mining queries and watch the results on the screen (see Figure 2).

RD2 Database Adapter provides transparent access to various database management systems. Primarily implemented on Oracle DBMS (using Oracle Call Interface), *RD2* architecture is independent on the DBMS vendor. *RD2* Database Adapter translates its API calls into native DBMS functions. The adapter can communicate with both local and remote DBMSs.

The client application requests are expressed in the form of *MineSQL* data mining queries. *RD2* **MineSQL Parser** is used for syntactic and semantic analysis of the queries. It builds the parse tree for a query and then calls the appropriate query processing procedures.

Association Rules Miner is a module for discovering all association rules which satisfy user-specified predicates. The association rules are discovered from the database accessed via Database Adapter while the predicates are extracted from *MineSQL* queries by *MineSQL* Parser. Our own constraints-driven algorithm for association rules discovery is used to perform the task.

Sequential Patterns Miner is a module for discovering all sequential patterns which satisfy user-specified predicates. The sequential patterns are discovered from the database accessed via Database Adapter while the predicates are extracted from *MineSQL* queries by *MineSQL* Parser.

RD2 Snapshot Refresher is an independent module responsible for updating materialized views defined on data mining queries. It re-executes the view queries in regular time intervals and stores the refreshed view contents in the database.

3 MineSQL Data Mining Query Language

MineSQL is a declarative language for expressing data mining problems. It is a *SQL*-based interface between client applications and *RD2* system. *MineSQL* plays similar role to data mining applications as *SQL* does to database applications. *MineSQL* is *declarative* - the client application is separated from the data mining algorithm being used. Any modifications and improvements done to the algorithm do not influence the applications. *MineSQL* follows the syntax philosophy of *SQL* language – data mining queries can be combined with *SQL* queries, i.e. *SQL* results can be mined and *MineSQL* results can be queried. Thus, existing database applications can be easily modified to use *RD2* data mining. *MineSQL* is also *multipurpose* – it can be used for different types of frequent patterns: association rules, sequential patterns, generalized association rules, and generalized sequential patterns. *MineSQL* also supports storage of the discovered patterns in the database – both association rules and sequential patterns can be stored in database tables similarly to alphanumeric data. Repetitive data mining tasks can be optimized by means of data mining views and materialized views.

3.1 New SQL Data Types

MineSQL language defines a set of new *SQL* data types, which are used to store and manage association rules, sequential patterns, itemsets and itemset sequences. The new data types can be used for table column definitions or as program variable types.

The *SET OF* data types family is used to represent sets of items, e.g. a shopping cart contents. *SET OF NUMBER* data type represents a set of numeric items, *SET OF CHAR* represents a set of character items, etc. Below we give an example of a statement, which creates a new table to store basket contents.

```
CREATE TABLE SHOPPING (ID NUMBER, BASKET SET OF CHAR);
```

In order to convert single item values into a *SET OF* value, we use a new *SQL* group function called *SET*. Below we give an example of a *SQL* query, which returns the *SET OF* values from normalized market baskets.

```
SELECT SET(ITEM)
FROM PURCHASED_ITEMS GROUP BY T_ID;
```

SET(ITEM)
A
A,C,D

PURCHASED_ITEMS:

T_ID	ITEM
1	A
2	A
2	C
2	D

We define the following *SQL* functions and operators for the *SET OF* data types: *ITEM(x,i)* - the *i*-th item value of the set *x* (lexicographical ordering presumed), *SIZE(x)* - the number of items in the set *x*, *s CONTAINS q* - *TRUE* if the set *s* contains the set *q*, *s UNION q* - the union of the sets *s* and *q*, *s MINUS q* - the difference of the sets *s* and *q*, *s INTERSECT q* - the intersection of the sets *s* and *q*.

The *RULE OF* data types family is used to represent association rules, containing body, head, support and confidence values. *RULE OF NUMBER* data type represents association rules on numeric values (e.g. "1 & 3 -> 5"), *RULE OF CHAR* data type represents association rules on character values (e.g. "beer & diapers -> chips"), etc. The following statement creates a new table to store association rules.

```
CREATE TABLE MYRULES (ID NUMBER, AR RULE OF CHAR);
```

A user can insert an association rule into the table manually, using a new *SQL* function called *TO_RULE*, e.g.:

```
INSERT INTO MYRULES (ID,AR) VALUES(15, TO_RULE('A,D','B,C',0.8,0.1));
SELECT * FROM MYRULES;
```

ID	AR
15	A & D -> B & C (0.8;0.1)

We also define a set of the following *SQL* functions and operators that operate on rules: *BODY(x)* - the *SET OF* value representing the body of the rule *x*, *HEAD(x)* - the *SET OF* value representing the head of the rule *x*, *SUPPORT(x)* - support of the rule *x*, *CONFIDENCE(x)* - confidence of the rule *x*, *s [NOT] SATISFIES x* - *TRUE* if the set *s* satisfies the rule *x*, *s [NOT] VIOLATES x* - *TRUE* if the set *s* violates the rule *x*.

The following example query displays all sets from *PURCHASED_ITEMS* table, which violate the association rule "A & D -> B & C & G".

```
SELECT SET(ITEM)
FROM PURCHASED_ITEMS GROUP BY T_ID
HAVING SET(ITEM) VIOLATES TO_RULE('A,D','B,C,G',0.8,0.1);
```

The *SEQUENCE OF* data types family is used to represent sequences of sets of items, e.g. histories of customers' purchases. Sequences are ordered collections of (timestamp, value) pairs, where timestamp is usually of date and time type and value can be a set of elements of any type. For example, *SEQUENCE OF CHAR INDEX BY DATE* data type represents a sequence of sets of character items ordered according to timestamps of date type. Below we give an example of a statement, which creates a new table to store purchase histories.

```
CREATE TABLE SHOPPING_HIST
(CUST_ID NUMBER, HIST SEQUENCE OF CHAR INDEX BY DATE);
```

In order to convert a collection of (timestamp, value) pairs into a *SEQUENCE OF* value, we use a new *SQL* group function called *SEQUENCE*. The example below shows a query returning the *SEQUENCE OF* values from normalized market baskets.

```
SELECT SEQUENCE(T_TIME, ITEM)
FROM CUST_TRANSACTIONS GROUP BY C_ID;
```

SEQUENCE(T_TIME, ITEM)
<Feb 20,2000;(A)> <Feb 21,2000;(D)> <Feb 20,2000;(B,D)> <Feb 22,2000;(A)>

CUST_TRANSACTIONS:

T_TIME	C_ID	ITEM
Feb 20, 2000	10	A
Feb 20, 2000	20	B
Feb 20, 2000	20	D
Feb 21, 2000	10	D
Feb 22, 2000	20	A

A user can insert a sequence into the table manually, using a new *SQL* function called *TO_SEQUENCE*. The following example presents the appropriate *INSERT* statement and the resulting contents of *SHOPPING_HIST*.

```
INSERT INTO SHOPPING_HIST (CUST_ID, HIST)
VALUES (51, TO_SEQUENCE('<Feb 20,2000;(A)> <Feb 21,2000;(D,F)>'));
```

We also define a set of the following *SQL* functions and operators that operate on sequences: *ELEMENT(x,i)* - the *i*-th element value of the sequence *x*, *LENGTH(x)* - the number of elements in the sequence *x*, *SIZE(x)* - the number of items in the sequence *x*, *s CONTAINS t* - *TRUE* if the sequence *s* contains the sequence *t*, *s CONTAINS t MAXGAP x* - *TRUE* if the sequence *s* contains the sequence or pattern *t* and the interval between adjacent matching elements in *s* is not greater than x, *s CONTAINS t MINGAP x* - *TRUE* if the sequence *s* contains the sequence or pattern *t* and the interval between adjacent matching elements in *s* is not less than x, *s CONTAINS t WINDOW x* - *TRUE* if the sequence *s* contains the sequence or pattern *t* within the time window of *x*, *s CONTAINS t TOLERANCE x* - *TRUE* if the sequence *s* contains the sequence or pattern *t* with the tolerance of *x*.

The *PATTERN OF* data types family is used to represent sequential patterns and their statistical significance (support or number of occurrences). *PATTERN OF CHAR* data type represents patterns on character values (e.g. "<(TV) (VCR) (DVD)>"), *PATTERN OF NUMBER* data type represents patterns on numeric values (e.g. "<(10 20 30) (40 50)>"), etc. Below we give an example of a statement, which creates a new table to store sequential patterns.

```
CREATE TABLE MYPATTERNS(ID NUMBER,SP PATTERN OF NUMBER);
```

A user can insert a sequential pattern into the table manually, using the new *SQL* function called *TO_PATTERN*. The following example presents the appropriate *INSERT* statement and the resulting contents of *MYPATTERNS*.

```
INSERT INTO MYPATTERNS (ID, SP)
VALUES (15, TO_PATTERN('<(10 20) (30)>',0.3, null));
```

We also define a set of the following *SQL* functions and operators that operate on sequential patterns: *ELEMENT(x,i)* - the *i*-th element of the pattern *x*, *LENGTH(x)* - the number of elements in the pattern *x*, *SIZE(x)* - the number of items in the pattern *x*, *SUPPORT(x)* - support of the pattern *x*, *OCCURRENCES(x)* - number of occurrences of the pattern *x*, *p CONTAINS s* - *TRUE* if the pattern *p* contains the sequence *s*.

The following query displays all patterns from MYPATTERNS, which contain the subsequence <(10)(30)> (notice the dynamic sequence creation in the example).

```
SELECT SP FROM MYPATTERNS
WHERE SP CONTAINS TO_SEQUENCE('<1;(10)><2;(30)>');
```

3.2 Mining Association Rules

The central statement of the *MineSQL* language is *MINE*. *MINE* is used to discover association rules or sequential patterns from the database. *MINE* also specifies a set of predicates to be satisfied by the returned rules or patterns. In order to discover association rules we use the following syntax of *MINE* statement.

```
MINE rule_expression [, rule_expression…] FOR column [, column …]
FROM {table|(query)} WHERE rule_predicate [AND rule_predicate…];
```

where: *rule_expression* is the keyword *RULE* or a function operating on *RULE* (*RULE* represents a single association rule being discovered), *column* is the name of the table column or query column of the type *SET OF*, containing itemsets to be mined; when specifying multiple columns, then the itemsets are combined (columns must be of the same data type), *table* is the name of a table containing the itemsets to be mined, *query* is the *SQL* subquery, returning the itemsets to be mined, *rule_predicate* is a Boolean predicate on a function which operates on *RULE*, to be satisfied by returned association rules.

The following *MINE* statement uses the *PURCHASED_ITEMS* table to discover all association rules, whose support is greater than 0.1 and confidence is greater that 0.3. We display the whole association rules, their bodies and supports.

```
MINE RULE, BODY(RULE), SUPPORT(RULE)
FOR X FROM (SELECT SET(ITEM) AS X
            FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(RULE)>0.1 AND CONFIDENCE(RULE)>0.3;
```

3.3 Mining Sequential Patterns

In order to discover sequential patterns we use the following syntax of *MINE* statement.

```
MINE patt_expression [, patt_expression…]
[WINDOW window][MAXGAP maxgap][MINGAP mingap][TOLERANCE tolerance]
FOR column FROM {table|(query)}
WHERE patt_predicate [AND patt_predicate…];
```

where: *patt_expression* – the keyword *PATTERN* or a function operating on *PATTERN* (*PATTERN* represents a single sequential pattern being discovered), *window* is the time window size, *maxgap* is the maximal gap allowed between consecutive elements of an occurrence of the sequence, *mingap* is the minimal gap allowed between consecutive elements of an occurrence of the sequence, *tolerance* is the time tolerance value for pattern elements, *column* is the name of the table column or query column of the type *SEQUENCE OF*, containing sequences to be mined, *table* is the name of a table containing the sequences to be mined, *query* is the *SQL* subquery, returning the sequences to be mined, *patt_predicate* is a Boolean predicate on a function which operates on *PATTERN*, to be satisfied by returned sequential patterns.

The following *MINE* statement uses the *CUST_TRANSACTIONS* table to discover all sequential patterns, whose support is greater than 0.1. We display the patterns and their supports.

```
MINE PATTERN, SUPPORT(PATTERN)
FOR X FROM (SELECT SEQUENCE(T_TIME, ITEM) AS X
            FROM CUST_TRANSACTIONS GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.1;
```

3.4 Using Views and Materialized Views

Relational databases provide users with a possibility of creating views and materialized views. A view is a virtual table presenting the results of the *SQL* query hidden in the definition of the view. Views are used mainly to simplify access to frequently used data sets that are results of complex queries. When a users selects data from a view, its defining query has to be executed but the user does not have to be familiar with its syntax.

Since data mining tasks are repetitive in nature and the syntax of data mining queries may be complicated, we propose to extend the usage of views to handle both *SQL* queries and *MineSQL* queries. The following statement creates the view presenting the results of one of the data mining tasks discussed earlier.

```
CREATE VIEW BASKET_RULES
AS MINE RULE, BODY(RULE), SUPPORT(RULE)
FOR X FROM (SELECT SET(ITEM) AS X
            FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(RULE)>0.1;
```

Any *SQL* query concerning the view presented above involves performing the data mining task according to the data mining query that defines the view. This guarantees access to up-to-date patterns but leads to long response times, since data mining algorithms are time consuming. In database systems it is possible to create materialized views that materialize the results of the defining query to shorten response times. Of course, data presented by a materialized view may become invalid as the source data changes. One of the solutions minimizing effects of this problem is periodic refreshing of materialized views.

We introduce materialized data mining views with the option of automatic periodic refreshing. A materialized data mining view is a database object containing patterns (association rules or sequential patterns) discovered as a result of a data mining query. It contains rules and patterns that were valid at a certain point of time. Materialized data mining views can be used for further selective analysis of discovered patterns with no need to re-run mining algorithms. Using materialized views is easier than creating a table with columns of type *RULE OF* or *PATTERN OF* and filling it with results of a data mining query. Moreover, materialized views offer additional functionality, because they can be automatically refreshed according to a user-defined time interval. This might be useful when a user is interested in a set of rules or sequential patterns, whose specification does not change in time, but always wants to have access to relatively recent information.

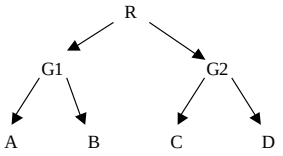
To create a materialized data mining view we use the following syntax.

```
CREATE MATERIALIZED VIEW view_name
[REFRESH time_interval] AS mine_statement
```

In the above syntax *view_name* is the name of a materialized view, *time_interval* denotes the time interval between two consecutive refreshes of the view (in days), and *mine_statement* denotes any variation of the *MINE* statement. The *REFRESH* clause is optional since a user might not want a view to be refreshed automatically.

3.5 Taxonomies and Generalized Patterns

In order to discover generalized association rules and generalized sequential patterns, *MineSQL* allows users to define conceptual hierarchies. A conceptual hierarchy, or a taxonomy, is a persistent database object created by means of a *CREATE TAXONOMY* and *INSERT* statements. The following example illustrates a conceptual hierarchy (called *MY_TAX*) and the statements for its definition.




```
CREATE TAXONOMY MY_TAX OF CHAR;
INSERT INTO MY_TAX NODE 'R';
INSERT INTO MY_TAX NODE 'G1' REFERENCES 'R';
INSERT INTO MY_TAX NODE 'G2' REFERENCES 'R';
INSERT INTO MY_TAX NODE 'A' REFERENCES 'G1';
INSERT INTO MY_TAX NODE 'B' REFERENCES 'G1';
INSERT INTO MY_TAX NODE 'C' REFERENCES 'G2';
INSERT INTO MY_TAX NODE 'D' REFERENCES 'G2';
```

After a conceptual hierarchy has been created, *MINE* statements can be extended to use it. Additional keyword *USING* is used to specify the name of a conceptual hierarchy to be used for the associated attribute. The following example data mining query discovers all generalized association rules between values of the attribute *ITEM* in the database table *PURCHASED_ITEMS*, grouped by the *T_ID* attribute. Values of the attribute *ITEMS* are generalized by means of the conceptual hierarchy called *MY_TAX*.

```
MINE RULE FOR ITEMS USING MY_TAX
FROM (SELECT SET(ITEM) AS ITEMS
      FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(RULE)>0.3
```

Conceptual hierarchies also influence previously presented set, rule, sequence, and sequential pattern operators: *s CONTAINS q USING c - TRUE* if the set *s* contains the set *q* according to the conceptual hierarchy *c*, *s [NOT] SATISFIES x USING c - TRUE* if the set *s* satisfies the rule *x* according to the conceptual hierarchy *c*, *s [NOT] VIOLATES x USING c - TRUE* if the set *s* violates the rule *x* according to the conceptual hierarchy *c*, *s CONTAINS t USING c - TRUE* if the sequence *s* contains the sequence *t* according to the conceptual hierarchy *c*, *p CONTAINS s USING c - TRUE* if the pattern *p* contains the sequence *s* according to the conceptual hierarchy *c*.

4 Concluding Remarks

In the paper we have presented our research prototype system, which logically extends DBMS functionality to mine relational databases. We introduced a data mining query language, called *MineSQL*, which supports pattern discovery, storage and management in relational environment. A number of illustrative examples for various *MineSQL* statements have been presented.

In the future we plan to extend the results of our research along the following lines: 1. DBMS support for data mining in object-oriented databases, focused on using inheritance hierarchies as taxonomies and mining of polymorphic collections, 2. Re-

search on data mining query optimization methods, focused on materialized views group refreshing and physical data structures, 3. Study whether other data mining methods should be supported by *MineSQL*, e.g. data classification, clustering.

Bibliography

1. Agrawal R., Imielinski T., Swami A.: Mining Association Rules Between Sets of Items in Large Databases. Proc. of the ACM SIGMOD Conference on Management of Data (1993)
2. Agrawal R., Mehta M., Shafer J., Srikant R., Arning A., Bollinger T.: The Quest Data Mining System. Proc. of the 2nd KDD Conference (1996)
3. Agrawal R., Srikant R.: Fast Algorithms for Mining Association Rules. Proc. of the 20th Int'l Conf. on Very Large Data Bases (1994)
4. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th Int'l Conference on Data Engineering (1995)
5. Bayardo R. J., Agrawal R., Gunopulos D.: Constraint-Based Rule Mining in Large, Dense Databases. Proc. of the 15th Int'l Conference on Data Engineering (1999)
6. Bayardo R. J.: Efficiently Mining Long Patterns from Databases. Proc. of the ACM SIGMOD International Conference on Management of Data (1998)
7. Ceri S., Meo R., Psaila G.: A New SQL-like Operator for Mining Association Rules. Proc. of the 22nd Int'l Conference on Very Large Data Bases (1996)
8. 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 (1996)
9. 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. of the 2nd KDD Conference (1996)
10. Han J., Fu Y.: Discovery of Multiple-Level Association Rules from Large Databases. Proc. of the 21st Int'l Conf. on Very Large Data Bases (1995)
11. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
12. Imielinski T., Virmani A., Abdulghani A.: Datamine: Application programming interface and query language for data mining. Proc. of the 2nd KDD Conference (1996)
13. Mannila H., Toivonen H., Verkamo A.I.: Discovering frequent episodes in sequences. Proc. of the 1st KDD Conference (1995)
14. Morzy T., Zakrzewicz M.: SQL-like Language for Database Mining. Proc. of the 1st ADBIS Conference (1997)
15. Srikant R., Agrawal R.: Mining Generalized Association Rules. Proc. of the 21st Int'l Conf. on Very Large Data Bases (1995)
16. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th Int'l Conf. on Extending Database Technology (1996)
17. Wojciechowski M.: Mining Various Patterns in Sequential Data in an SQL-like Manner. Proc. of the 3rd ADBIS Conference (1999)