Materialized Data Mining Views*

Tadeusz Morzy, Marek Wojciechowski, Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
tel. +48 61 6652378, fax +48 61 8771525
{morzy,marek,mzakrz}@cs.put.poznan.pl

Abstract. Data mining is a useful decision support technique, which can be used to find trends and regularities in warehouses of corporate data. A serious problem of its practical applications is long processing time required by data mining algorithms. Current systems consume minutes or hours to answer simple queries. In this paper we present the concept of materialized data mining views. Materialized data mining views store selected patterns discovered in a portion of a database, and are used for query rewriting, which transforms a data mining query into a query accessing a materialized view. Since the transformation is transparent to a user, materialized data mining views can be created and used like indexes.

1 Introduction

Data mining, also referred to as database mining or knowledge discovery in databases (KDD), aims at discovery of useful patterns from large databases or warehouses. Currently we are observing the evolution of data mining environments from specialized tools to multi-purpose data mining systems offering some level of integration with existing database management systems. From a user's point of view data mining can be seen as advanced querying: a user specifies the source data set and the requested class of patterns, the system chooses the right data mining algorithm and returns discovered patterns to the user. The most serious problem concerning data mining queries is a long response time. Current systems consume minutes or hours to answer simple queries.

Another important feature of data mining is that it is an iterative and interactive process. Users very often periodically perform the same data mining tasks to get the up-to-date information. We claim that data mining systems should provide support for such repetitive queries. It is desirable to store the results of a data mining query that will be repeated after some changes to the database because there are known algorithms for incremental data mining. In this paper we propose using periodically refreshed *materialized data mining views* (MDMVs) for repetitive data mining

* This work was partially supported by the grant no. KBN 43-1309 from the State Committee for Scientific Research (KBN), Poland.

queries in the same manner as materialized views are used in relational database management systems to store results of complex and time consuming queries.

Benefits of using MDMVs to answer data mining queries where the query to be answered is the same as the query defining an existing MDMV are obvious. The question we try to answer in this paper is: can we use MDMVs to efficiently answer a data mining query that is not equal but only similar to the query defining some MDMV? We consider two data mining queries similar, if they have the same schema of source datasets and resulting patterns, and differ in selection predicates applied to the query on the source dataset and/or constraints concerning statistical strength and contents of patterns.

In this paper we present the concept of MDMVs and their application in the discovery of frequent itemsets and association rules. Since it is straightforward to generate association rules from frequent itemsets, we focus on the frequent itemsets only. We illustrate our optimization rules with many examples expressed in *MineSQL*, which is a declarative, multi-purpose *SQL*-like language for interactive and iterative data mining in relational databases, developed by us over the last couple of years [8][9].

1.1 Basic Definitions

Frequent itemsets. 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. The *support* of the itemset X is the percentage of T in D that support X. The problem of mining frequent itemsets in D consists in discovering all itemsets whose support is above a user-defined support threshold.

Association rules. 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 s% of itemsets in D support $X \cup Y$. The rule $X \rightarrow Y$ has confidence c if c% of itemsets in D that support X also support Y. The problem of mining association rules in D consists in discovering all associations rules whose support and confidence are above user-defined support thresholds for support and confidence respectively.

1.2 Data Mining Queries

We have proposed a declarative language, called *MineSQL*, for expressing data mining problems by means of *data mining queries*. *MineSQL* is a *SQL*-based interface between a client application and a data mining system. It plays similar role to data mining applications as *SQL* does to traditional 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 data mining methods. In this section we present elements of *MineSQL* that are used later in the paper, where MDMVs and their usage are discussed. The detailed syntax of *MineSQL* can be found in [8].

MineSQL language defines a set of new *SQL* data types, which are used to store and manage association rules and itemsets. The *SET OF* data types family (*SET OF NUMBER, SET OF CHAR*, etc.) is used to represent sets of items, e.g. a shopping cart contents. In order to convert single item values into a *SET OF* value, we use a new *SQL* group function called *SET*.

The *ITEMSET OF* data types family is used to represent frequent itemsets. For an itemset its support is stored together with the set of items it contains. We define a set of *SQL* functions and operators that operate on rules: `SIZE(x)` returns the number of items in the itemset *x*, `s CONTAINS q` returns *TRUE* if the itemset *s* contains the set *q*, `SUPPORT(x)` returns support of the itemset *x*.

The *RULE OF* data types family is used to represent association rules, containing body, head, support and confidence values. We define a set of *SQL* functions and operators that operate on rules: `BODY(x)` returns the SET OF value representing the body of the rule *x*, `HEAD(x)` returns the SET OF value representing the head of the rule *x*, `SUPPORT(x)` returns support of the rule *x*, `CONFIDENCE(x)` returns confidence of the rule *x*.

The central statement of the *MineSQL* language is *MINE*. *MINE* is used to discover frequent itemsets or association rules from the database. *MINE* also specifies a set of predicates to be satisfied by the returned rules or patterns.

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

```
MINE ITEMSET, SUPPORT(ITEMSET)
FOR X FROM (SELECT SET(ITEM) AS X
            FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.1;
```

1.3 Related Work

The problem of mining association rules was first introduced in [1] and an algorithm called *AIS* was proposed. In [2], two new algorithms were presented, called *Apriori* and *AprioriTid* that are fundamentally different from the previous ones. The algorithms achieved significant improvements over *AIS* and became the core of many new algorithms for mining association rules. *Apriori* and its variants first generate all frequent itemsets (sets of items appearing together in a number of database records meeting the user-specified support threshold) and then use them to generate rules. *Apriori* and its variants rely on the property that an itemset can only be frequent if all of its subsets are frequent. It leads to a level-wise procedure. First, all possible 1-itemsets (itemsets containing 1 item) are counted in the database to determine *frequent 1-itemsets*. Then, frequent 1-itemsets are combined to form potentially frequent 2-itemsets, called *candidate 2-itemsets*. Candidate 2-itemsets are counted in

the database to determine *frequent 2-itemsets*. The procedure is continued by combining the frequent 2-itemsets to form *candidate 3-itemsets* and so forth. A disadvantage of the algorithm is that it requires *K* or *K+1* passes over the database to discover all frequent itemsets, where *K* is the size of the greatest frequent itemset found.

In [4], an algorithm called *FUP* (Fast Update Algorithm) was proposed for finding the frequent itemsets in the expanded database using the old frequent itemsets. The major idea of *FUP* algorithm is to reuse the information of the old frequent itemsets and to integrate the support information of the new frequent itemsets in order to reduce the pool of candidate itemsets to be re-examined. Another approach to incremental mining of frequent itemsets was presented in [11]. The algorithm introduced there required only one database pass and was applicable not only for expanded but also for reduced database. Along with the itemsets, a *negative border* [12] was maintained.

In [10] the issue of interactive mining of association rules was addressed and the concept of *knowledge cache* was introduced. The cache was designed to hold frequent itemsets that were discovered while processing other queries. Several cache management schemas were proposed and their integration with the *Apriori* algorithm was analyzed. An important contribution was an algorithm which used itemsets discovered for higher support thresholds in the discovery process for the same task, but with a lower support threshold.

The idea of precomputing frequent itemsets in a partitioned database and using them while discovering association rules in the whole database or parts of it was discussed in [13]. The itemsets were materialized and store in a compact form. The proposed method exploited the property that an itemset can be frequent in the union of a number of partitions if and only if it is frequent in at least one of the partitions. Thus itemsets that were frequent in at least one of the partitions of the mined dataset, formed the set of candidates for one verifying database pass.

The notion of data mining queries (or *KDD* queries) was introduced in [6]. The need for Knowledge and Data Management Systems (KDDMS) as second generation data mining tools was expressed. The ideas of application programming interfaces and data mining query optimizers were also mentioned. Several data mining query languages that are extensions of *SQL* were proposed [3][5][7][8][9].

2 Data Mining 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 user 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 data mining view presenting the results of the data mining task discussed earlier.

```
CREATE VIEW BASKET_ITEMSETS
AS MINE ITEMSET
FOR X FROM (SELECT SET(ITEM) AS X FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.1;
```

In the defining statement of a data mining view, there are two classes of constraints: database constraints and mining constraints. *Database constraints* are located within the *SELECT* statement in the *FROM* clause of the *MINE* statement. Database constraints are used to apply selection conditions on the source dataset that is being mined. *Mining constraints* are located in the *WHERE* clause of the *MINE* statement and are used to specify selection conditions on the set of patterns to be discovered.

An important advantage of data mining views is separation of applications processing results of data mining queries from predicates defining parameters of data mining algorithms. If applications access frequent patterns by means of data mining views, they do not have to be changed when only selection predicates (database or mining predicates) are changed in a data mining query. In such case only views have to be modified.

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.

3 Materialized Data Mining Views

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. In fact, in the area of data mining, changes to the source database should not be considered to be a serious problem because data mining tasks are usually performed on data warehouses rather than on operational databases. In data warehouses, changes are applied in bulks and materialized data mining views should be refreshed only after a series of changes, together with other views existing in the data warehouse.

We introduce materialized data mining views (MDMVs) with the option of automatic periodic refreshing. A materialized data mining view is a database object containing patterns (association rules or frequent itemsets) discovered as a result of a data mining query. It contains rules and patterns that were valid at a certain point of time. MDMVs can be used for further selective analysis of discovered patterns with no need to re-run mining algorithms. 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 itemsets, whose specification does not change in time, but he or she always wants to have access to relatively recent information.

The following statement creates a MDMV containing all frequent itemsets with support greater than 0.1, discovered in the set of transactions from *PURCHASED_ITEMS* table. The view is to be refreshed once a week.

```
CREATE MATERIALIZED VIEW BASKET_ITEMSETS
REFRESH 7 AS MINE ITEMSET
FOR X FROM (SELECT SET(ITEM) AS X FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.1;
```

In most cases when a MDMV is being refreshed, it can be refreshed efficiently with one of the algorithms for incremental mining. Moreover, it is desirable to store information about the time of last changes applied to the source objects, in order to detect situations when refreshing is not necessary, since the source dataset has not changed.

4 Data Mining Query Rewriting with Materialized Data Mining Views

MDMVs can be also used to reduce execution time of data mining queries, which are not identical to those, on which the views were built. Consider the following example: we are given a MDMV defined over the following data mining query.

```
CREATE MATERIALIZED VIEW V1
AS MINE ITEMSET
FOR ITEMS FROM (SELECT SET(ITEM) AS ITEMS
                FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.2;
```

Assume that a user wants to discover frequent itemsets with the following data mining query.

```
MINE ITEMSET
FOR ITEMS FROM (SELECT SET(ITEM) AS ITEMS
                FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.2
AND ITEMSET CONTAINS TO_SET('A,D');
```

Notice that in order to execute the query, we can simply filter the actual contents of the materialized data mining view *V1*, without running a data mining algorithm. Thus, MDMVs can play a similar role to data mining queries, as indexes or materialized views do to database queries. Application developers can create MDMVs to transparently decrease execution times of their applications' data mining queries.

We need formal methods for determining data mining query execution plans, which use MDMVs to reduce time complexity. First, we define four relations, which may occur between two data mining queries, DMQ_1 and DMQ_2. We say that:

1. DMQ_1 *extends database constraints* of DMQ_2, if DMQ_1 does one of the following:
- appends a *WHERE* or *HAVING* clause of database constraints of DMQ_2
- appends an additional *ANDed* condition to a *WHERE* or *HAVING* clause of database constraints of DMQ_2

- removes an *ORed* condition from a *WHERE* or *HAVING* clause of database constraints of DMQ_2

Example. The following data mining query DMQ_1 extends database constraints of the data mining query DMQ_2.

DMQ_1:

```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
   FROM PURCHASED_ITEMS
   WHERE ITEM!='D' AND T_ID>100
   GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.2;
```

DMQ_2:

```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    WHERE ITEM!='D'
    GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.2;
```

Intuitively, extension of database constraints means narrowing the mined data set.

2. *DMQ_1 reduces database constraints* of DMQ_2, if DMQ_1 does one of the following:
- removes a *WHERE* or *HAVING* clause of database constraints of DMQ_2
- appends an additional *ORed* condition to a *WHERE* or *HAVING* clause of database constraints of DMQ_2
- removes an *ANDed* condition from a *WHERE* or *HAVING* clause of database constraints of DMQ_2

Example. The following data mining query DMQ_1 reduces database constraints of the data mining query DMQ_2.

DMQ_1:

```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.2;
```

DMQ_2:

```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    GROUP BY T_ID
    HAVING COUNT(*)>10)
WHERE SUPPORT(ITEMSET)>0.2;
```

Intuitively, reduction of database constraints means extending the mined data set.

3. *DMQ_1 extends mining constraints* of DMQ_2, if DMQ_1 does one of the following:
- replaces `SUPPORT(ITEMSET)>x` with `SUPPORT(ITEMSET)>y` in DMQ_2, where $x<y$
- replaces `SUPPORT(ITEMSET)<x` with `SUPPORT(ITEMSET)<y` in DMQ_2, where $x>y$
- replaces `ITEMSET CONTAINS X` with `ITEMSET CONTAINS Y` in DMQ_2, where $X \subset Y$
- replaces `ITEMSET NOT CONTAINS X` with `ITEMSET NOT CONTAINS Y` in DMQ_2, where $Y \subset X$
- replaces `SIZE(ITEMSET)>x` with `SIZE(ITEMSET)>y` in DMQ_2, where $x<y$
- replaces `SIZE(ITEMSET)<x` with `SIZE(ITEMSET)<y` in DMQ_2, where $x>y$
- appends a *WHERE* or *HAVING* clause of mining predicates of DMQ_2
- appends an additional *ANDed* condition to a *WHERE* or *HAVING* clause of mining constraints of DMQ_2
- removes an *ORed* condition from a *WHERE* or *HAVING* clause of mining constraints of DMQ_2

Example. The following data mining query DMQ_1 extends mining constraints of the data mining query DMQ_2.

DMQ_1:
```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.4;
```

DMQ_2:
```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.2;
```

Intuitively, extension of mining constraints means narrowing the resulting set of discovered patterns.

4. *DMQ_1 reduces mining constraints* of DMQ_2, if DMQ_1 does one of the following:
- replaces `SUPPORT(ITEMSET)>x` with `SUPPORT(ITEMSET)>y` in DMQ_2, where $x>y$
- replaces `SUPPORT(ITEMSET)<x` with `SUPPORT(ITEMSET)<y` in DMQ_2, where $x<y$
- replaces `ITEMSET CONTAINS X` with `ITEMSET CONTAINS Y` in DMQ_2, where $Y \subset X$
- replaces `ITEMSET NOT CONTAINS X` with `ITEMSET NOT CONTAINS Y` in DMQ_2, where $X \subset Y$
- replaces `SIZE(ITEMSET)>x` with `SIZE(ITEMSET)>y` in DMQ_2, where $x>y$
- replaces `SIZE(ITEMSET)<x` with `SIZE(ITEMSET)<y` in DMQ_2, where $x<y$
- removes a *WHERE* or *HAVING* clause from mining constraints of DMQ_2
- appends an additional *ORed* condition to a *WHERE* or *HAVING* clause of mining constraints of DMQ_2
- removes an *ANDed* condition from a *WHERE* or *HAVING* clause of mining constraints of DMQ_2

Example. The following data mining query DMQ_1 reduces mining constraints of the data mining query DMQ_2.

DMQ_1:
```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.4;
```

DMQ_2:
```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.2;
```

Intuitively, reduction of mining constraints means expanding the resulting set of discovered patterns.

We also define four different mining methods, which will be used to execute data mining queries over MDMVs: full mining, incremental mining, complementary mining, and verifying mining. *Full mining* (FM) refers to executing a complete algorithm for discovering frequent itemsets (e.g. [2]). This method is used if MDMV contents is unusable to execute the data mining query. *Incremental mining* (IM) refers to discovering frequent itemsets in an incremented data set (e.g. [4]). It can be used for data mining queries which reduce database constraints. *Complementary mining*

(CM) refers to discovering frequent itemsets based on currently materialized itemsets, which will remain frequent (e.g. [10]). This method can be used for data mining queries which reduce mining constraints. Finally, we have *verifying mining* (VM), that simply consists in pruning those materialized itemsets, which do not satisfy mining constraints. It is used for data mining queries, which extend mining constraints.

If two relations occur between a data mining query and a data mining query on which a MDMVis based, then we use the compatibility table (see Table 1) to decide what mining method to use.

Table 1. Compatibility table for using materialized data mining views

	reduction of database constraints	extension of database constraints	-
reduction of mining constraints	CM, IM	FM	CM
extension of mining constraints	VM, IM	FM	VM
-	IM	FM	-

Example. We are given the following data mining query DMQ_1 and the materialized data mining view $MDMV_1$.

DMQ_1:
```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.4;
```

$MDMV_1$:
```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    GROUP BY T_ID
    HAVING COUNT(*)>10)
WHERE SUPPORT(ITEMSET)>0.2;
```

Since DMQ_1 extends mining constraints (higher minimum support) and reduces database constraints (removed HAVING clause) of the data mining query of $MDMV_1$, we perform verifying mining (*VM*), and then incremental mining (*IM*). The verifying mining prunes all materialized itemsets, whose support value is not above 0.4, while the incremental mining discovers frequent itemsets using the information on frequent itemsets discovered in a subset of the mined data set. It was proven in the literature that the execution time of the above mining algorithms will be shorter than when performing full mining.

Example. We are given the following data mining query DMQ_1 and the materialized data mining view $MDMV_1$.

DMQ_1:
```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.3
AND ITEMSET CONTAINS TO_SET('A,B');
```

$MDMV_1$:
```
MINE ITEMSET
FOR ITEMS
FROM (SELECT SET(ITEM) AS ITEMS
    FROM PURCHASED_ITEMS
    GROUP BY T_ID)
WHERE SUPPORT(ITEMSET)>0.2;
```

Since DMQ_1 extends mining constraints (higher minimum support as well as additional *AND*ed condition) of the data mining query of $MDMV_1$, we perform verifying mining (*VM*). The verifying mining prunes all materialized itemsets, whose support value is not above 0.3 or which do not contain the subset {A,B}.

5 Conclusions and Future Work

In this paper we have presented the concept of materialized data mining views. We have proposed several rules for optimization of data mining queries in environments, where MDMVs, containing results of other data mining queries are available. These rules can serve as a basis for rule-based data mining query optimizers. An important advantage of the solutions we propose is that the algorithms required to implement our optimization framework have already been introduced and verified.

In the future we plan to address the problem of cost-based data mining query optimization, especially concentrating on situations when there are several MDMVs that can be used to optimize the processing of a given data mining query.

Another topic that we plan to discuss is concurrent refreshing of several MDMVs. We believe that in such case, sometimes it might be desirable to combine mining tasks associated with several MDMVs to optimize the global performance of the refresh operation.

In the paper we focused on discovery of frequent itemsets and association rules. In the future we plan to analyze possibilities of using data mining views for optimizing queries concerning other data mining tasks such as discovery of sequential patterns.

References

1. Agrawal R., Imielinski T., Swami A.: Mining Association Rules Between Sets of Items in Large Databases. Proc. of the 1993 ACM SIGMOD Conf. on Management of Data (1993)
2. Agrawal R., Srikant R.: Fast Algorithms for Mining Association Rules. Proc. of the 20th Int'l Conf. on Very Large Data Bases (1994)
3. 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)
4. Cheung D.W., Han J., Ng V., Wong C.Y.: Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. Proc. of the 12th ICDE (1996)
5. 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)
6. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
7. Imielinski T., Virmani A., Abdulghani A.: Datamine: Application programming interface and query language for data mining. Proc. of the 2nd KDD Conference (1996)
8. Morzy T., Wojciechowski M., Zakrzewicz M.: Data Mining Support in Database Management Systems. Proc. of the 2nd DaWaK Conference (2000)
9. Morzy T., Zakrzewicz M.: SQL-like Language for Database Mining. ADBIS'97 Symposium (1997)
10. Nag B., Deshpande P.M., DeWitt D.J.: Using a Knowledge Cache for Interactive Discovery of Association Rules. Proc. of the 5th KDD Conference (1999)
11. Thomas S., Bodagala S., Alsabti K., Ranka S.: An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases. Proc. of the 3rd KDD Conference (1997)
12. Toivonen H.: Sampling Large Databases for Association Rules. Proc. of the 22nd Int'l Conference on Very Large Data Bases (1996)
13. Wojciechowski M., Zakrzewicz M.: Itemset Materializing for Fast Mining of Association Rules. Proc. of the 2nd ADBIS Conference (1998)