Mining Association Rules Inside a Relational Database – A Case Study

Mirela Danubianu, Stefan Gheorghe Pentiuc
Faculty of Electrical Engineering and Computer Science
"Stefan cel Mare" University of Suceava
Suceava, Romania
mdanub@eed.usv.ro, pentiuc@eed.usv.ro

Iolanda Tobolcea
Faculty of Psychology and Educational Sciences
"A.I.Cuza" University of Iasi
Iasi, Romania
itobolcea@yahoo.com

Abstract - **In the context of the necessity to find new knowledge in data, last decade, data mining has become an area of great interest. Although most data mining systems work with data stored in flat files, sometimes it is beneficial to implement data mining algorithms within a DBMS, in order to use SQL or other facilities provided, to discover patterns in data. In this paper we consider a way to discover association rules from data stored into a relational database. We make also a comparative study of performances obtained by applying the following methods: stored procedures in database or candidate and frequent itemsets generated in SQL using a k-way join and a subquery-based algorithm. This study is used to choose the best solution to implement in the particular case of building a dedicated data mining system for personalized therapy of speech disorders optimization.**

Keywords-data mining method, association rules, relational database, SQL , stored-procedures

I. INTRODUCTION

The most common way to store data collected in various areas is in relational databases. Information and Communication Technology development has lead to a huge volume of data stored and to the inability to extract useful information and knowledge from this data by using the traditional methods. For this reason, data mining has developed as a specific field. Mining association rules is one of the commonly used methods in data mining. Association rules model dependencies between items in transactional data. Most data mining systems work with data stored in flat files. However, it has been shown it is beneficial to implement data mining algorithms within a DBMS, and using of SQL to discover patterns in data can bring certain advantages.

There is some research focused on issues regarding the integration of data mining with databases. There have been proposed language extensions of SQL to support mining operations. For instance, in [1], DMQL extends SQL with a series of operators for generation of characteristic rules, discriminant rules and classification rules.

This paper aims to present some aspects of coupling data mining algorithms with database management systems.

Section II shows the reasons to try to mine data directly in databases and a possible architecture for such data mining system. In Section III, there are defined association rules, presents the methods and some algorithms that find association rules in data. In Section IV, we focus on the possibility to use SQL in order to generate the frequent itemsets. Section V contains a case study.

II. PERFORMING DATA MINING INTO A DATABASE MANAGEMENT SYSTEM

As we have noted above, sometimes it is useful to implement data mining algorithms in database management systems (DBMS).

First, one can use the database indexing capability and query processing and optimization facilities provided by a DBMS. Second, for long running mining applications it can be useful the support for checkpointing and last, but not least one can exploit the possibility of SQL parallelization, especially for a SMP environment.

This involves the development of data mining applications tightly–coupled with a relational database management system. In [2], a methodology to achieve this goal is presented.

Accordingly to this methodology, the records of a database are not fetched into the application, but modules of the application program that perform various operations on the retrieved records, are pushed in the database system.

Although in this case the core of data mining is found in the database, no changes were made on database management software, and appropriate functionalities were provided by user-defined procedures and functions stored in the database.

A possible architecture for such data mining approach is presented in Figure 1.

The graphical interface allows users to formulate the data mining problem and to establish parameters, such as: minimum thresholds for support and confidence for association rules, or minimum value for accuracy for classification.

Preprocessing module aims to translate the mining problem in the corresponding SQL instruction set. We consider the Oracle 9i SQL dialect because it contains object relational capabilities and it allows user-defined function and functions table.

Finally, the processing results are converted and presented to the user in an intelligible form through the graphical interface.

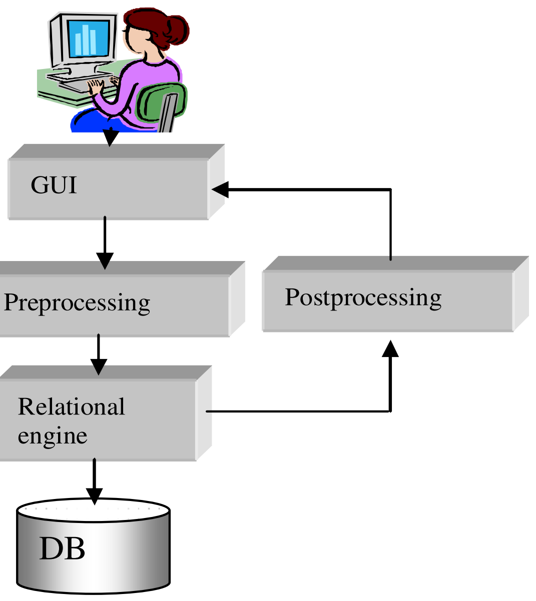

Figure 1. A proposed architecture for data mining in a DBMS

III. Association Rules: definition, methodology, algorithms

Association rules aims to discover the dependencies between the items in transactional databases.

Briefly, one can define association rules as follows: if we have a set of transactions, where each transaction is a set of items, an association rule [3] is an implication A->B, where A and B are disjoint sets of items. It means that if a transaction contains the items in A, it tends also to contain the items in B. A is called the antecedent of rule and B is called its consequent.

In terms of association rules, a set of k items is called *k-itemset*. In a transactional database, for an itemset A we can define a measure called support, that represents the fraction of transaction that contains A. If we note the database with D, the expression for the support for A is:

$$\mathrm{sup}(A) = |A| / |D| \quad (1)$$

where |A| is the number of transaction containing A, and |D| is the cardinality of the database.

The support for the rule A-> B is defined as:

$$\mathrm{sup}(A \rightarrow B) = \mathrm{sup}(A \cup B) \quad (2)$$

and represent the percentage of all transactions that contain both A and B.

The rule holds in the transaction database with confidence calculated with the following expression:

$$\mathrm{conf}(A \rightarrow B) = \frac{\mathrm{sup}(A \cup B)}{\mathrm{sup}(A)} \quad (3)$$

The problem of finding association rules is to generate all association rules that have support and confidence greater than two user–specified thresholds called *minsupp* and *minconf.*

The mining process for association rules can be decomposed in two phases: first find all combination of items whose support is greater than *minsupp*, called frequent itemsets and second, use these frequent itemsets to generate the rules.

Since the generation of frequent itemsets is the most expensive part in terms of resources and time consuming, a lot of algorithms for this task were developed. Most algorithms use a method that build candidate itemsets, which are sets of potential frequent itemsets, and then test them. Support for these candidates is determined by taking into account the whole database D. The process of generating candidate itemsets considers the information regarding the frequence of all candidates already checked. So, the procedure is the following: the closure of frequent itemsets assumes that all subsets of a frequent itemset are also frequent. This allows remove those sets that contain at least one set of items that is not frequent, from candidate itemsets. After generating, the appearance of each candidate in the database is counted, in order to retain only those having the support greater than *minsup*.

Then we can move to the next iteration. The whole process ends when there are no potential frequent itemsets.

The most known algorithm, which uses the above mechanism, is Apriori [4]. On this basis some variants such as Apriori Tid, Apriori All, Apriori Some or Apriori Hibrid were developed. Figure 2 presents the Apriori algorithm. We use the following notation:

D- the transaction database (transaction table)
t- tuples in *D*
k-itemset-set of *k* items
F_k- frequents k-itemsets
C_k- k-itemsets candidates (potential frequents)
c.count – number of transactions containing each c candidate set of items

```
1 $F_1$ = [frequent 1-itemsets]
2 k=2
3 while $F_{k-1} \neq \Phi$ do
4   $C_k$=gen_apriori($F_{k-1}$)    &&generating new candidates
5   for each $t \in D$ do
6     $C_t=\{C_k | C_k \subset t)$         &&candidates in t
7     for each $c \in C_t$ do
8       c.count=c.count+1
9   end
10  $F_k = \{c \in C_k \mid c.count \geq minsup\}$
11  k=k+1
12 end
13 end
```

Figure 2. The Apriori algorithm

Once found all frequent k-itemsets one can generate rules having minimum confidence *minconf*. In order to do that, we consider all non-empty subsets of every frequent itemset *f*. Then, for each subset *x*, we find the confidence of the rule x-> *(f-x)*, and if it is equal or greater than *minconf*, we output the rule.

Figure 3 presents in pseudo code the procedure for finding the association rules. We use the following notation:
F_k- frequents k-itemsets
A_i– a subset of j items from F_i (i<k,j<i)
B–the remaining subset of (i-j) items from F_i

```
1  F2 = [frequents 2-itemsets]
2  k>2
3  for i=2,k do
4    for j=1,i-1 do
5          $A_{i-1}=\{a_j | a_j \in F_i\}$
6          $B=\{b | b \in F_i \wedge b \neq a_j\}$
7      $conf(A_{i-1} \rightarrow B)=supp(A_{i-1} \cup B)/suppA_{i-1}$
8      $Rij=\{A_{i-1} \rightarrow B | conf(A_{i-1} \rightarrow B)>minconf\}$
9    end
10 end
```

Figure 3. The algorithm for finding association rules

As we can see above, the rules are generated in an iterative way. In each iteration *j* we generate rules with consequent of size *j*. Then we consider the following property: for a frequent itemset, if a rule with consequent *b* holds, then rules with consequents that are subsets of *b* holds also. We use this property to generate rules in iteration j based on rules with consequents of length (j-1) found in the previous iteration.

IV. Using SQL for frequent k-itemsets generation

In most implementation Apriori works with data stored in flat files but, data is ordinary stored in databases, so, for moving it in a flat file there are necessary some additional preprocessing operations.

Here we describe a way to find the frequent k-itemsets using SQL and manipulating data directly in a database. First of all we assume that the transaction table D has two columns: transaction identifier and item. As we don't know the number of items per transaction, this structure is more practical that alternatives presented in [5] where each item of a transaction is placed in a different column.

A. *Stored procedure approach*

In the first step of the algorithm one have to generate the frequent 1-itemset, by finding the support for each item and by removing those items that have support lesser that *minsup*.

The SQL query that performs this task, corresponding to the first line in the pseudo-code described above is:

```
insert into F1
     select item, count(*)                          (4)
        from D
           group by item
           having count(*)>minsup;
```

Further, each step k of the algorithm, first generate a candidate k-itemset C_k from which we will find the frequent k-itemsets F_k. In order to do that, we have to execute the following:

- to generate C_2, which will be stored in a table with two columns (one for each item in the combination) we use:

```
insert into C2
     select a.item1, b.item2                        (5)
      from F1 a, F1 b
     where a.item<b.item
     order by a.item;
```

- to find F_2 we must count the support of all 2-itemsets from C_2, and insert into F_2 only those who have the support greater than minsup. F_2 is a table with three columns (two columns for the two items and one column for support). We use a stored procedure that contain a sequence such as:

```
              ….
select count(id) into vsup
      from D
      where D.item=vc1 and exists
          (select tid
             from D d1                              (6)
             where d1.item=vc2 and D.tid=d1.tid);
if vsup>vminsup then
      insert into F2
      values(vc1, vc2, vsup);
end if;
                 ……….
```

To generalize, in order to find a frequent k-itemset F_k we will use the following SQL statements:

```
insert into Ck
  select a.item1,a.item2,…, a.item(k-1), b.item(k-1)
    from Fk-1 a, Fk-1 b
     where a.item1=b.item1 and                      (7)
         a.item2=b.item2  and
              ……
         a.item(k-2)=b.item(k-2)and
         a.item(k-1)<b.item(k-1)
     order by a.item1, a.item2,…, a.item(k-1);
```

for generating C_k and a stored procedure including:

```
  select count(id) into vsup
    from D
     where D.item=vc1 and exists                    (8)
    (select tid
     from D $d_1$
     where $d_1$.item=$vc_2$ and D.tid=$D_1$.tid and exists
          (…………
               (select tid
                from D $d_{k-1}$
                where $d_{k-1}$.item=$vc_k$
                and $d_{k-2}$.tid=$d_{k-1}$.tid)
                                              …);
```

in order to find the support of candidate k-itemsets.

One can see that for counting support for a k-itemset we need a select SQL statement that have k-1 subqueries, so, the number of nested selects is direct related with the size of the itemset to be analyzed.

B. The SQL Approach

To take advantage of the query optimizer, which is present in every relational DBMS, it is possible to implement Apriori algorithm only in SQL. In this case the generation of candidates Ck is made in the same manner as above.

In fact, the expression (7) generates an extensive set of candidate k-itemsets. If we consider that to be frequent, a k-itemset must contain only subsets of frequent (k-1) itemsets, it could be necessary to eliminate candidates that contain subsets that are not frequent from those generated by (7). This operation is the so-called pruning step, and is very useful because, after that, the remaining candidates could fit into memory, and the counting for support could be made pipeline, without materializing the candidates.

In [6] it is shown that we can perform the prune step in the same time with the join of F_{k-1} and F_{k-1}. In order to combine these tasks we could use the following SQL statement:

```
insert into $C_k$
 select $I_1$.$item_1$,$I_1$.$item_2$,…,$I_1$.$item_{(k-1)}$, $I_2$.$item_{(k-1)}$
from $F_{k-1}$ $I_1$, $F_{k-1}$ $I_2$, $F_{k-1}$ $I_3$,…,$F_{k-1}$ $I_k$
where
       $I_1$.$item_1$=$I_2$.$item_1$ and                              (9)
       $I_1$.$item_2$=$I_2$.$item_2$ and
       ……
       $I_1$.$item_{(k-2)}$=$I_2$.$item_{(k-2)}$and
       $I_1$.$item_{(k-1)}$<$I_2$.$item_{(k-1)}$and
            $I_1$.$item_2$=$I_3$.$item_1$ and                 && skip $item_1$
                   ……
            $I_1$.$item_{(k-1)}$=$I_3$.$item_{(k-2)}$and
            $I_2$.$item_{(k-1)}$=$I_3$.$item_{(k-1)}$and
                   ……
            $I_1$.$item_1$=$I_k$.$item_1$ and              &&   skip   $item_{(k-2)}$
                   ……
            $I_1$.$item_{(k-1)}$=$I_k$.$item_{(k-2)}$and
            $I_2$.$item_{(k-1)}$=$I_k$.$item_{(k-1)}$
order by $I_1$.$item_1$, $I_1$.$item_2$,…, $I_1$.$item_{(k-1)}$;
```

The expression (9) is a **k-way join**. This method is used since for any k-itemset there are k subsets of size k-1, which must be member of the frequent (k-1)-itemsets (F_{k-1}).

In the above expression after joining I_1 and I_2, we obtain the following k-itemset (I_1.$item_1$, I_1.$item_2$,…,I_1.$item_{(k-1)}$, I_2.$item_{(k-1)}$). It contains two (k-1)-itemsets, which are frequent since they are member of F_{k-1}. These are (I_1.$item_1$,I_1.$item_2$, …,I_1.$item_{(k-1)}$) and (I_1.$item_1$, I_1.$item_2$,…,I_1.$item_{(k-2)}$, I_2.$item_{(k-1)}$). The rest of (k-2) subsets must be checked, and in order to do that we use additional joins whose selection predicates are build by skipping one item at a time from the k-itemset.

Concrete, first we skip $item_1$ and check if the (k-1) itemset (I_1.$item_2$, …, I_1.$item_{(k-1)}$, I_2.$item_{(k-1)}$) belong to F_{k-1}. This is done by the join with I_3. Second we skip $item_2$ to see if (I_1.$item_1$, I_1.$item_3$, …, I_1.$item_{(k-1)}$, I_2.$item_{(k-1)}$) belong also to F_{k-1}.

In general we perform the join with I_n and, in the predicate we skip the item (n-2) from the k-itemset to check if the subset build by deleting the $(n-2)^{th}$ item from the original k-itemset, belong to F_{k-1}.

Further it is necessary to count candidates' support to find frequent itemsets. In order to do that we use the candidate itemsets C_k and the transaction table D. We consider the two approaches, which were found to be the best ones in [5]. There are: the K-way joins and the subquery-based implementation.

In k-way joins the candidate itemset C_k is joined with k transaction tables D and the support is counted using a group by clause on all k items. The general expression for finding a frequent k-itemset F_k is:

```
insert into $F_k$
     select $item_1$, $item_2$, …, $item_k$, count(*)
       from $C_k$, D $d_1$, …, D $d_k$
       where    $d_1$.item=$C_k$.$item_1$ and
                $d_2$.item= $C_k$.$item_2$ and                    (10)
                ……..
                $d_k$.item=$C_k$.$item_k$ and
                $d_1$.tid=$d_2$.tid and
                $d_2$.tid = $d_3$.tid and
                ….
                $d_{k-1}$.tid=$d_k$.tid
       group by $item_1$, $item_2$, ….,$item_k$
       having count(*)>:minsup;
```

The subquery-based approach tries to reduce the amount of work during the support counting by using the common prefixes between the items in C_k. To do this, the support counting phase is split into a cascade of k subqueries.

The n^{th} (n=1..k) subquery is build based on the result of $(n-1)^{th}$ subquery, which is joined with the transaction table D, and the distinct itemsets consisting of the first n+1 columns of C_k. To obtain the final output we make a group-by on the k items to calculate the support and we remove those rows that have the calculated support less than minsup.

The statement for generating F_k is:

```
insert into $F_k$
   select $item_1$, $item_2$, …, $item_k$, count(*)
   from ($SQ_k$)                                          (11)
   group by $item_1$, $item_2$, ….,$item_k$
   having count(*)>:minsup;
```

where SQ_k is the kth subquery . Subquery Q_n (n=1..k) is

Select $item_1$, $item_2$, …, $item_n$, tid
From D d_n,(Subquery Q_{n-1}) SQ_{n-1},(select distinct $item_1$,…, $item_n$ from C_k) c_n
where $SQ_{n-1}.item_1 = c_n.item_1$ and (12)
… and
$SQ_{n-1}.item_{n-1} = c_n.item_{n-1}$ and
$SQ_{n-1}.tid = d_n.tid$ and
$d_n.item = c_n.item_n$;

V. CASE STUDY

The Center for Computer Research in the University "Stefan cel Mare" of Suceava has implemented the TERAPERS project [7]. TERAPERS is a system, which is able to assist teachers in their speech therapy of dislalya and to follow how the patients respond to various personalized therapy programs. This system is currently used by the therapists from Regional Speech Therapy Center of Suceava.

In the context of the need for more efficient activities, it was showed that data mining methods, applied to data collected in TERAPERS, can provide useful knowledge for personalized therapy optimization. So, the idea about Logo-DM system has started. This is a data mining system, which aims to use the data from TERAPERS database in order to answer the questions such as: what is the predicted final state for a child or what will be his/her state at the end of various stages of therapy, which the best exercises are for each case and how they can focus their effort to effectively solve these exercises or how the family receptivity, which is an important factor in the success of the therapy - is associated with other aspects of family and personal anamnesis [8][9].

Specifically, association rules could reveal interesting relationships between patients' anamnesis, their diagnosis and the results obtained at the end of various stages of therapy.

To find the best way to implement the association rules we have used a real dataset from TERAPERS, on which we have applied the three methods of determining the frequent itemsets discussed above.

Data considered is stored in a relational table with 96 columns (attributes) and 300 cases. A step in the preparation process of this table for applying Apriori algorithm is to change the structure of the initial table in order to obtain the following structure: *id, item.*

After this transformation we have obtained a data set that can be assimilated to a transactional one. As each patient has associated a set of features, we can consider the patient identifier as a transaction identifier and the set of features as a set of items. The transactional dataset obtained as result of pre-processing operations contains 300 cases (transactions), but the average of items per transaction is 65.

We have applied on this data the three methods for finding the frequent itemsets discussed above. All of them have provided as output the same number of frequent itemsets and the same maxim length of the itemsets, but major differences were recorded on response times. To enable the query optimizer to choose the best execution plan, we have constructed some indexes. The source table for data used by Apriori algorithm was indexed on both columns (id, item), and C_k and F_k were indexed on columns ($item_1$, $item_2$, …, $item_k$).

The results obtained are presented in figures below.

Figure 4 and Figure 5 show the number of itemsets obtained and the maximum length of these itemsets. We note that, in this case, taking into account the data characteristics, it should be imposed a minimum value for the threshold *minsup* equal to 0,5. For this value is obtained a considerable number of itemsets (approximately 4000000) with a maximum length of 32 items.

Figure 6 presents a summary of execution times obtained when, in order to finding frequent itemsets, we use the following methods: stored procedures, k-way joins and subquery-based joins. It may be noted that for the last two ways, execution times are relatively closed and they are lower than for stored procedures.

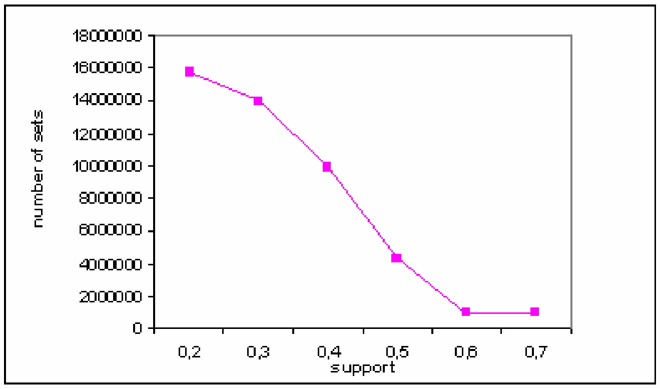

Figure 4. Frequent itemsets found for different values for *minsup*

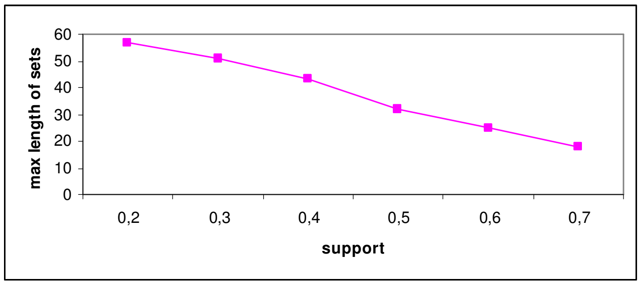

Figure 5. Maximum length of itemsets for different values for *minsup*

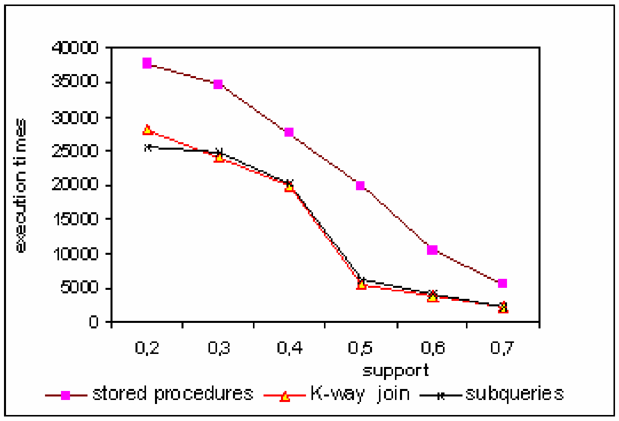

Figure 6. A comparation of execution times

VI. Conclusion

Last time, data mining has become an area of special importance. Based on the results obtained using the patterns provided by data mining in fields such as business or medicine, we have tried to extend the implementation of these methods in other areas. This paper presents a part of the efforts made in order to achieve and to early evaluate the performances of finding the association rules for a system that aims to optimize the personalized therapy of speech disorder.

To take advantage of the facilities offered by the query optimizer, it is proposed to incorporate data mining algorithms in the database that stores data to be analyzed.

First, it was noted that in this context the data pre-processing operations are relatively simple to be achieved. Then it was observed that, due to the data set properties, the largest number of candidate itemsets was not obtained in the second step, and for about half of iterations the number of candidates and of frequent itemsets increases, then this number decreases. The data source contains a lot of items that are found in a large percentage of transactions. As a result it is indicated for the support's threshold to be established to a value of least 0,5. Even in this case the number and the maxim length of frequent itemsets are high. As a result it will get a large number of association rules that should be filtered in order to preserve only the most interesting ones.

Acknowledgment

This paper was supported by the project "Progress and development through post-doctoral research and innovation in engineering and applied sciences– PRiDE - Contract no. POSDRU/89/1.5/S/57083", project co-funded from European Social Fund through Sectorial Operational Program Human Resources 2007-2013.

References

[1] J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A data mining query language for relational datbases. In Proc. of the 1996 SIGMOD workshop on research issues on data mining and knowledge discovery, Montreal, Canada, May 1996.

[2] R. Agrawal and K. Shim. Developing tightly-coupled data mining applications on a relational database system. In Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining, Portland, Oregon, August 1996.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data (ACM SIGMOD '93), 1993

[4] R. Agrawal and R. Srikant. Fast algorithms pentru mining association rules. In Proceedings of the 20th International Conference on Very Large Databases (VLDB '94), Santiago, Chile, June, 1994

[5] K. Rajamani, B. Iyer, and A. Chaddha. Using DB/2's object relational extensions for mining associations rules. Technical Report TR 03,690., Santa Teresa Laboratory, IBM Corporation, sept 1997.

[6] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. In Proc. of the ACM SIGMOD Conference, Seattle, Washington, June 1998.

[7] M. Danubianu, S.G. Pentiuc, O. Schipor, M. Nestor, I. , Ungurean. Distributed Intelligent System for Personalized Therapy of Speech Disorders, Proceedings of ICCGI08, 2008, Atena

[8] M. Danubianu, S.G. Pentiuc, I. Tobolcea, O.A. Schipor. Advanced Information Technology - Support of Improved Personalized Therapy of Speech Disorders, International Journal of Computers Communications & Control, ISSN 1841-9836, 5(5): 684-692, 2010.

[9] S. G. Pentiuc, I. Tobolcea, O. A. Schipor, M. Danubianu, D. M. Schipor, "Translation of the Speech Therapy Programs in the Logomon Assisted Therapy System," Advances in Electrical and Computer Engineering, vol. 10, no. 2, pp. 48-52, 2010, http://dx.doi.org/10.4316/AECE.2010.02008