International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347- 5552, Volume-1, Issue-1, September- 2013

Itemset Mining over Large Transactional Tables on the Relational Databases

Arun Pratap Srivastava, Prof.(Dr) Mohd. Hussain

Abstract- **Most of the itemset mining approaches are memory-like and run outside of the database. On the other hand, when we deal with data warehouse the size of tables is extremely huge for memory copy. In addition, using a pure SQL-like approach is quite inefficient. Actually, those implementations rarely take advantages of database programming. Furthermore, RDBMS vendors offer a lot of features for taking control and management of the data. We purpose a pattern growth mining approach by means of database programming for finding all frequent itemsets. The main idea is to avoid one-at-a-time record retrieval from the database, saving both the copying and process context switching, expensive joins, and table reconstruction. The empirical evaluation of our approach shows that runs competitively with the most known itemset mining implementations based on SQL. Our performance evaluation was made with SQL Server 2000 (v.8) and T-SQL, throughout several synthetical datasets.**

Index Terms- **SQL, RDBMS, Mining, Itemset, OLAP**

I. INTRODUCTION

The problem of finding all frequent itemsets [2] given a Dataset D with a minimum support threshold S is the most time consuming task on association rule mining. In order to solve this problem, two ways are likely to be chosen: one using algorithms that employed sophisticated in memory data structures, where the data is stored into and retrieved from flat files; and another using algorithms that are based on SQL statements and extensions to query and update a database. The former is very efficient when it is compared with the later. On the other hand, when we deal with data warehouse the size of tables is extremely huge for memory copy. Nevertheless, it becomes important for Relational Database Management Systems (RDBMS) to offer new analytic functionalities to support business intelligence applications.

There are a few implementations based on SQL [12, 13, 14, 15, 16], but they have performance issues concentrated in two central points: candidate-set generation and test (Apriori-bottleneck); and table reconstruction of conditional pattern trees (FP- Growth-bottleneck).

In this work we do not intend to compare the effectiveness of itemset mining based on database programming with the memory ones. Instead, we purpose a solution for bringing the itemset mining process to the RDBMS server side, which generates the following contributions:

1. A procedural schema for itemset mining, so the process can be run as a batch script on the RDBMS server side.
2. A cascading approach working with single tables, and also avoiding the complexity of mining frequent itemsets from several multi-tables joins.
3. A pattern growth mining which doesn't surfer of several tables reconstruction (of conditional pattern trees)

Manuscript received September 10, 2013.
Arun Pratap Srivastava, Ph.D. Student, NIMS University, Jaipur, India, (e-mail: arun019@yahoo.com).
Prof.(Dr). Mohammad Husain, Director, MG Institute of Management & Technology, Lucknow, India, (e-mail: mohd.husain90@gmail.com).

II. FREQUENT PATTERN MINING

The frequent pattern mining problem can be defined as follow: Given a set of items I, a transaction database D over I, and a minimal support threshold S, find all itemsets F(D,S). Indeed, we are not only interested in the set of itemsets (F), but also in the actual supports of these itemsets.

The most known implementation of frequent pattern mining algorithm is Apriori [3]. Several A p r i o r i - based algorithms have been purposed for getting better performance and I/O costs [9, 10, 11]. Recently, an FP-tree based frequent pattern mining method, called FP-growth, developed by Han et al. [8], achieved high efficiency, when compared with the Apriori-like approaches. Basically, the FP-growth method adopts the divide-and- conquer strategy. It uses only two full I/O scans of the database, and avoids iterative candidate generation. In general terms, the mining process consists of making available the FP-tree data structure, and then FP-growth is applied over a FP-tree for getting frequent itemsets.

There are implementations that suggest enhancements into the frequent pattern mining in order to make the process interactive, constrained, and incremental [6, 7]. Those aspects will not be discussed in this work. Instead, we focus on the first issue, i.e., finding frequent itemsets closer to RDBMS. The above implementations cannot be applied directly on the main problem of this work, since they need to copy tables out from the database for proper execution. Besides, after its execution, the results must be load again to the database for getting suitable analysis.

A. Pattern Growth Mining

Pattern Growth Mining can be viewed as first mining

frequent 1-itemset and then progressively growing each such itemset by mining its conditional pattern base, which \implies first mining its frequent 1-itemset and then progressively growing each such itemset by mining its conditional pattern base, etc [8]. Thus, a frequent k-itemset mining problem can be transformed into a sequence of k frequent 1-itemset mining problems via a set of conditional pattern bases. The main aspects of the algorithm can be summarized as follows:

1. For each node in the FP-tree construct its conditional pattern base, which is a "sub-database" constructed with the prefix sub-path set co-occurring with the suffix pattern in the FP-tree. FP-growth traverses nodes in the FP-tree from the least frequent item in I.
2. From each conditional pattern base construct its conditional FP-tree.
3. Finally, if the conditional FP-tree has a single path, simply enumerate all patterns, on the contrary run pattern growth mining recursively over the conditional FP-tree.

B. SQL Based

There are a few SQL-based implementations that can be used to mine frequent patterns over large transactional tables [1, 12, 13, 15, 16]. Even so, all of them are based on nature of Apriori-like approach. There is another approach that uses FP- Growth [14] in RDBMS. Nevertheless, the process of reconstructing conditional FP tables for large datasets may pose performance issues. Therefore, we must avoid the previous mentioned bottlenecks: candidate set generation and test; and table reconstruction. Moreover, we have designed a procedural schema for mining all patterns on the RDBMS server side.

We examined those assumptions, and purpose a new approach for pattern growth mining using database programming facilities. By using a pattern growth approach we are able to manage the first bottleneck. However, the current SQL implementation of FP-Growth [14] cannot handle the second issue. Consequently, we need to provide a solution for pattern growth mining which must have the ability to: work in RDBMS server side, prevent multi-table joins and table reconstruction of conditional pattern trees.

III. DISCOVERING FREQUENT ITEMSETS ON LARGE TRANSACTIONAL TABLES

In order to provide itemset mining over large transactional tables on the RDBMS server side, we present a procedural schema by means of using several database facilities such as stored procedures, SQL-cursors, and UDF functions. We also call this approach as a Pattern Growth mining with SQL-Extensions (PGS). The whole procedural schema cannot, for reasons of space, be presented here, but can be found in [4].

The whole process can be summarized into two main steps: one for generating the pattern tree and another one for mining all patterns.

Table1 shows the frequent 1-itemsets extracted from a transactional table (columns TID and Items). A new transactional table (column Freq. 1-itemsets) containing only records with frequent 1-itemsets is created, and thus its related pattern tree is also built. Finally, Table 2 presents the pattern growth method applied over the pattern tree. For instance, giving that only items (column Item) are frequents, from its pattern tree, we work with only a subset (column SUBFP) for reaching its conditional pattern tree (column CONFP) and then enumerating all frequent patterns.

Table 1: A transaction database with a support= 3.

TID	Items	Freq. 1-itemsets
1	1, 3, 5, 6, 7	3, 5, 7, 1, 6
2	2, 3, 5, 6, 7	3, 5, 7, 6
3	1, 2, 3, 4, 5	3, 5, 1
4	3, 6, 7	3, 7, 6
5	1, 4, 5, 7	5, 7, 1

Table 2. Extracting all patterns (R=root).

Item	SUBFP	CONFP	PATTERNS
3	Null	Null	Null
5	R=1, R:3=3	3:3	3%5:3
7	R:3=1, R:3:5=2, R:5=1	3:3, 5:3	3%7:3, 5%7:3
1	R:3:5=1, R:3:5:7=1, R:5:7=1	5:3	5%1:3
6	R:3:5:7=1, R:3:5:7:1=1, R:3:7=1	3:3,7:3	3%6:3, 7%6:3, 3%7%6:3

A. Step 1: Creating pattern-tree Table

Several tables are manipulating during the process of generating all itemsets (see Fig.1 and Fig 2). They are built just one time. In the recursive part, where pattern growth is applied, other structures are required, but they are created dynamically by using UDF functions and database cursors. In fact, they provide SUBFP (sub-path) tables for extracting single and not-single patterns.

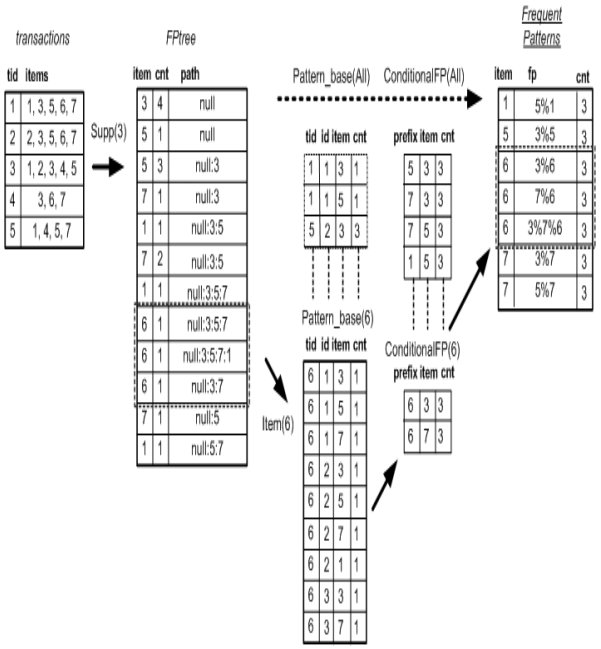

Fig. 1: An overall picture of the tables involved in the process of generating all itemsets

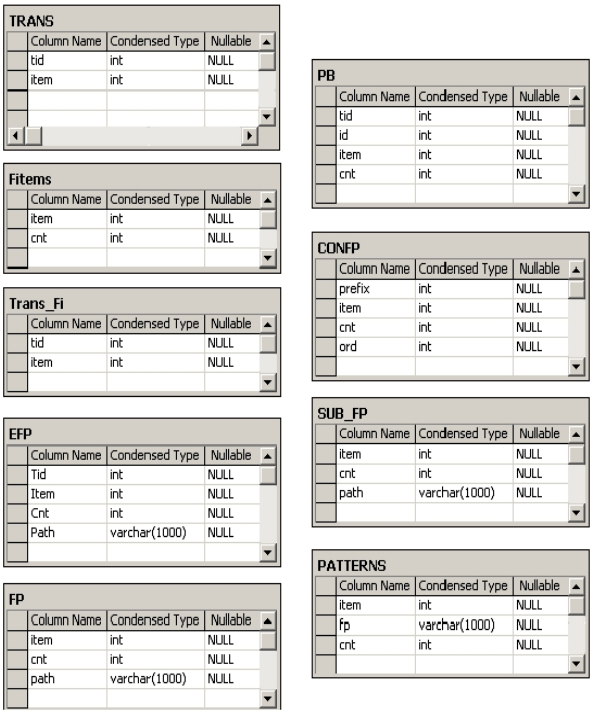

Fig. 2: PGS database table schema

We mean single patterns for those which are enumerated directly from its conditional FP table (CONFP) (for instance "3%5"[1]), meaning a co-occurrence of item 3 with item 5, without handling sub-path tables. A pattern such as "3%7%6" is extracted by combine those items that co-occurs added with it is respective sub-path tables [8].

As a pattern growth approach the first step requires a pattern-tree structure also called FP-tree. Even though FP-tree is a compact structure, it is unlikely to build such structure in memory for large databases. Consequently, using RDBMS capabilities like buffer management, query processor or SQL-Extensions, it is possible to take advantage of those mechanisms avoiding size considerations of data, in this particular case, FP (pattern-tree) tables.

The construction of FP table is set up on the following steps:

(a) Based on a given support threshold (s), frequent 1-itemsets are selected from the transactional table TRANS.

(b) A new transaction table TRANSFI is created based on transactions which contains those frequent 1-itemsets.

(c) From the TRANSFI table, an EFP table which stands for Extended FP is built as a preprocessing step for reaching an FP table.

(d) Finally, the FP table is created by means of an SQL expression, with proper aggregate function over EFP table.

The EFP table is an interesting approach for getting FP table, since it avoids for each frequent item to be tested if it should be or not inserted into FP table [14].

```
# a piece of the Pattern TREE (FP) source code #
PROCEDURE EFP
DO with (EXISTS TRANSFI)
CREATE TABLE EFP (item, cnt, path)
CREATE TABLE FP (item, cnt, path)
 DECLARE
    BEGIN
     count=1
     curpath = null
      c_transfi CURSOR for TRANSFI
          FOR each row    in c_transfi BEGIN
        curpath = curpath + ':' + c_transfi.item
        INSERT INTO EFP
          values(c_transfi.item, count, curpath)
     END
     SELECT item, sum(cnt) as cnt, path
     INTO FP
     FROM EFP
     GROUP BY item, path
    END
```

B. Step 2: Mining pattern-tree Table

For mining FP table it is necessary to build two more auxiliary tables which are the pattern base (PB) and conditional FP table (CONFP). We present an approach where CONFP table is built based on simple SQL with proper aggregate functions over PB table. On the other hand, SQL-based FP-Growth [14] demands several reconstruction processes for those tables. It is almost unrealistic to create those tables several times. Therefore, we use an approach for getting sub-paths by means of UDF functions and database cursors with its respective support threshold over the SUBFP table (a SUBset of FP table).

SUBFP is a table that contains only rows from FP

table which have itemsets enclosed in CONFP. By doing so, we can reduce the search space for getting sub-paths directly from all items in FP table, and also, avoid several reconstruction of PB, FP and CONFP tables. The size of FP is reduced significantly by using SUBFP mainly when dealing with low support thresholds on large datasets. The following steps are required for mining pattern tree table [4]:

(a) Taking as input the same support threshold defined in 3.1, creates the related tables PB, CONFP and SUBFP.

(b) Update the column (pos) in CONF which keeps the position of each item. This is useful for getting sub-path databases in such way that it preserves the order of the items on SUBFP table. This is important for using UDF (table-valued functions).

(c) Extract single patterns by enumerating the prefix-item stored on CONFP table.This also creates the PATTERNS table.

(d) Extract not-single patterns by applying pattern growth over CONFP.

(e) In Fragment Growth step, each prefix-item is extracted from SUBFP table and verified by two UDF functions. One for generating the sub-path databases (function getTable_pb) and other for getting the node support associated to each prefix-item sub-path (function getNodeSupp). Those functions coupled with the SUBFP table play an important role for extracting all frequent patterns, and also avoid the re-construction of PB, FP and CONFP table for each prefix-item sub- path.

```
# a piece of the Fragment Growth source code  #
    DECLARE pg_subPath CURSOR for
    SELECT * FROM getTable_pb(@v_prefix,@v_item)
    order by ord
    SELECT list_pg_item  = pg_subPath.item
          FOR each row in pg_subPath
BEGIN
SELECT node_path = pg_subPath.item+'%'+
       c_confp.item
SELECT  node_supp=
      getNodeSupp(pg_subPath.prefix, node_path)
SELECT pat_item = pg_subPath.item
SELECT pat_fp = node_path+'%'+
                     pg_subPath.prefix
SELECT pat_cnt =  node_supp
SELECT exist_pat = (
SELECT count(*)  FROM
                       PATERNS
WHERE item=pat_item and fp=pat_fp)
 INSERT INTO PATTERNS (item,fp,cnt)
       VALUES (pat_item, pat_fp, pat_cnt)
 SELECT list_pg_item = list_pg_item
         +'%'+ pg_subPath.item
END

UDF FUNCTION getNodeSupp   (@item, @path)
RETURNS @node_supp ## ->   node support
BEGIN
DECLARE @supp int
SELECT @supp = (SELECT sum(cnt)FROM   SUB_FP
         WHERE item=@item and
         path LIKE  '%' +@path+'%')
RETURN (@supp)
UDF FUNCTION getTable_pb (@prefix, @item)
RETURNS TABLE ## -> sub-path databases
AS  RETURN
SELECT prefix,item,cnt,ord  FROM   CONFP
WHERE  prefix=@prefix and
  item<>@item ##-> criteria for sub-path mining
```

IV. PGS EVALUATION ON RDBMS SERVER SIDE

In order to evaluate PGS we compare our results with an Apriori (K-way join) and improved SQL-based FP-growth (EFP). Those algorithms were chosen in sense that they present the basis on the most known itemset mining implementations based on SQL [1, 12, 13, 14, 15, 16]. Both algorithms were implemented to the best of our knowledge based on the published reports on the same machine and compared in the same running environment. The former implementation uses a candidate k-table Ck, which is a slow process for generating all joins and tables. So, when dealing with long patterns and large datasets the K-way join seems to be not efficient. The EFP avoids candidate-set generation been more competitive in low support scenario. However, it demands several tables reconstruction.

Our approach takes the other way around, beyond pure SQL to SQL-Extensions. Also getting sub-paths databases, and restricting the search space for finding frequent itemsets by means of using an SUBFP table coupled with UDF functions. Consequently, we don't need to materialize PB, FP and CONFP tables several times. PGS also has been used in [5] for the extraction and analysis of inter-transactional patterns. The method consists in the combination of association and sequence mining.

A. Datasets

We use the synthetic transaction data generation described in [3] for generating transactional tables. The nomenclature of these data sets is of the form TxxIyyDzzzK. Where xx denotes the average number of items present per transaction, yy denotes the average support of each item in the data set, and zzzK the total number of transactions in K (1000's). Table 3 summarizes those datasets.

Datasets	Dist. Items	Nof. Rows	Avg.1-it. sup	Max.1-it. sup
T5I51K (1)	775	5.112	6	41
T5I5D10K (2)	873	49.257	56	399
T25I10D10K (3)	947	245.933	259	1468
T25I20D100K (4)	981	2.478,55	2526	13.917

Table 3: More information of the transactional datasets

B. Comparative Study

We describe our approach PGS, comparing it with K-Way-join and EFP. Our experiments were performed with Microsoft SQL Server 2000 (v.8.0). The machine was a mobile AMD Athlon ™ 2000+ 645MHZ, 224 MB RAM. The performance measure was the execution time 'the logarithm of the execution time (log (milliseconds))' of the algorithm applied over the four datasets with different support thresholds. We took that log scale in order to get a better view of the performance comparison among all approaches, since PGS has good response time. Fig. 3 shows the total time taken by the all approaches.

From those graphs we can make the following observation: PGS can get competitive performance out of FP and K-way-join. K-way-join has low performance when dealing with large datasets. Besides, when the support decrease the length of frequent itemsets increase causing expensive joins with the transactional tables. Therefore, the other two approaches perform better than K-way-join.

The results of PGS and EFP answered our second issue. The former doesn't use table reconstruction, getting good response time. On the other hand, the latter suffers considerably by working with several table materialization processes. By those results we have accomplished our main goals.

The store procedures which deal with the construction of tables FP and CONFP are the most time-consuming tasks. They respectively took, for each dataset, 35%, 50%, 75% and 85% of the total execution time. Nevertheless, the time for the whole process was quite competitive and those tables are built only once. In order to speed up even more the whole process, we also have been applied two clustered indexes on tables CONFP and SUBFP.

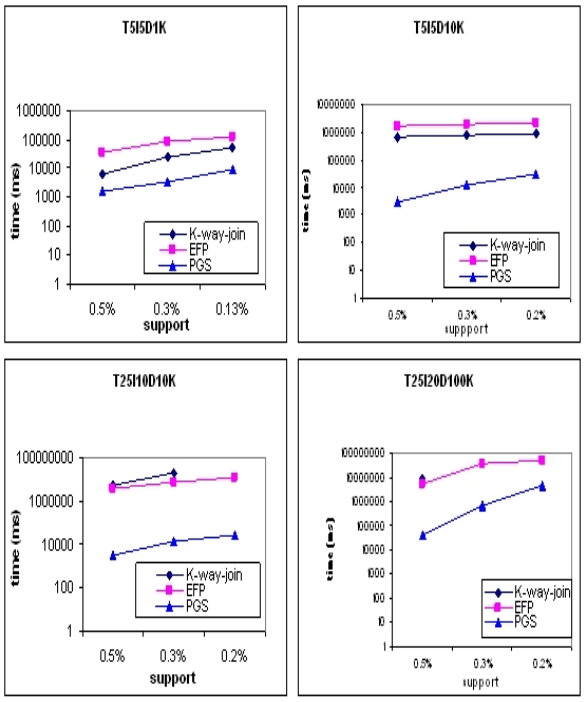

Fig. 3: Time performance of the three approaches. PGS runs competitively in sparse and dense datasets

C. Food Mart Warehouse

FoodMart Warehouse is a sample database provided by Microsoft SQL Server 2000. One can use Analysis Services for applying OLAP and Data Mining techniques over data warehouses. However, only Clustering and Decision Tree methods are available. In order to support itemset mining over FoodMart database we may use PGS. It works only in first step of association rules, which means generating frequent itemsets. For getting all the rules, one can program a store procedure using the pseudo-code in [2]. We choose this last example for presenting some results with a "real" database. Although we know that its size is smaller than the largest one showed in section 4.2, we also can reach interesting itemsets. Therefore we omit the performance comparison among all approaches. It was used the fact table (sales_fact) as the transactional table. This table has 164.558 tuples with five dimensions (product, time, customer, promotion and store). Before using PGS, we must define which dimensions in the fact table will be used as the transaction identifier (tid) and the set of items. Thus, the tid was set to the customer dimension and items to the product dimension. Furthermore, there are 1.559 distinct products distributed along 7.824 customers. The most frequent product was 277 "Great English Muffins" (143) and the less one was 1559 "CDR Apple Preserves" (43). It was executed several supports from (0.05%) to (0.01%). The most interesting itemset was "282%232" means "Best Choice Salsa Dip" and "Great Wheat Bread" in low level hierarchy. Given that the fact table was so sparse, the itemsets was selected only with very low support.

V. CONCLUSION

In this paper, it was purposed a pattern growth mining implementation which takes advantage of SQL-Extensions. Most of the commercial RDBMS vendors have implemented some features for SQL-Extensions. Integrating data mining in RDBMS is a quite promising area. Frequent pattern mining is the basic task in data mining. There are several memory-approaches to mine all patterns. However, some few efforts have been made on database perspective, in those cases, only pure-SQL. Given the large size of database like data warehouse, it is interesting to take advantage of those databases capabilities in order to manage and analyze large tables. We work in this direction purposing an approach with SQL-Extensions. By doing so, we can achieve competitive results and also avoid classical bottlenecks: candidate set generation and test (several expensive joins), and table reconstruction. The Store Procedures that deals with the construction of the tables, FP and CONFP, are the most time-consuming tasks, taking 35%, 50%, 75% and 85% respectively for each synthetic dataset (1, 2, 3, 4) from the small one to the large one. Nevertheless, the time for the whole process was quite competitive. Moreover, it was sed the FoodMart Warehouse with several supports in order to find interesting itemsets. One issue that is controversial is code portability, in sense that PGS is tightly dependent of the database programming language with SQL-Extensions. On the other hand for huge databases, it makes more sense take all the advantages offered by the RDBMS.

As future work, we are working on enhancements for making the process more interactive, constrained and incremental. We also intend to improve the whole performance of PGS by using Table Variables (TV), which allows mimicking an array, instead of using database cursors.

REFERENCES

1. Agarwal, R., Shim., R.: Developing tightly-coupled data mining application on a relational database system. In Proc.of the 2nd Int. Conf. on Knowledge Discovery in Database and Data Mining, Portland, Oregon (1996)
2. Agrawal, R., Imielinski, T., Swami, A..: Mining association rules between sets of items in large databases. In Proc. of the ACM SIGMOD Intl. Conference on Management of Data (1993) 207–216
3. Agrawal, R., Srikant., R.: Fast algorithms for mining association rules. In Proc. of the 20thVery Large Data Base Conference (1994) 487–499
4. Alves, R., Belo, O.: Integrating Pattern Growth Mining on SQL-Server RDBMS. Technical Report-003, University of Minho, Department of Informatics, May (2005) http://alfa.di.uminho.pt/~ronnie/files_files/rt/2005-RT3-Ronnie.pdf
5. Alves, R., Gabriel, P., Azevedo, P., Belo, O.: A Hybrid Method to Discover Inter- Transactional Rules. In Proceedings of the JISBD'2005, Granada (2005)
6. Cheung, W., Zaïane, O. R.: Incremental Mining of Frequent Patterns Without Candidate Generation or Support Constraint, Constraint, Seventh International Database Engineering and Applications Symposium (IDEAS 2003), Hong Kong, China, July 16-18 (2003) 111-116
7. El-Hajj, M., Zaïane, O.R.: Inverted Matrix: Efficient Discovery of Frequent Items in Large Datasets in the Context of Interactive Mining, in Proc. 2003 Int'l Conf. on Knowledge Discovery and Data Mining (ACM SIGKDD), Washington, DC, USA, August 24-27 (2003) 109-118
8. Han, J., Pei, J., Yin., Y.: Mining frequent patterns without candidate generation. In Proc. of ACM SIGMOD Intl. Conference on Management of Data, (2000) 1–12.
9. Hidber, C.: Online association rule mining. In A. Delis, C. Faloutsos, and S.Ghandeharizadeh, editors, Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, volume 28(2) of SIGMOD Record. ACM Press (1999) 145–156
10. Orlando, S., Palmerini, P., Perego, R.: Enhancing the apriori algorithm for frequent set counting. In Y. Kambayashi, W. Winiwarter, and M. Arikawa, editors, Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery, volume 2114 of Lecture Notes in Computer Science (2001) 71–82
11. Orlando, S., Palmerini, P., Perego, R., Silvestri, F.: Adaptive and resource-aware mining of frequent sets. In V. Kumar, S. Tsumoto, P.S. Yu, and N.Zhong, editors, Proceedings of the 2002 IEEE International Conference on Data Mining. IEEE Computer Society (2002)
12. Rantzau, R.: Processing frequent itemset discovery queries by division and set containment join operators. In DMKD03: 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (2003)
13. Sarawagi, S., Thomas, S., Agrawal, R.: Integrating mining with relational database systems: alternatives and implications. In Proc. of the ACM SIGMOD Conference on Management of data, Seattle, Washington, USA (1998)
14. Shang, X., Sattler, K., Geist, I.: Sql based frequent pattern mining without candidate generation. In SAC'04 Data Mining, Nicosia, Cyprus (2004)
15. Wang, H., Zaniolo, C.: Using SQL to build new aggregates and extenders for Object- Relational systems. In Proc. Of the 26th Int. Conf. on Very Large Databases, Cairo, Egypt (2000)
16. Yoshizawa, T., Pramudiono, I., Kitsuregawa, M.: Sql based association rule mining using commercial rdbms (ibm db2 udb eee). In In Proc. DaWaK, London, UK (2000)