International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 6 4295 - 4300

Generic Framework for Gaining Insight Into Data

Shaila Eksambekar
Department of Computer Engineering
PES's Modern College of Engineering
Pune, India
shaila.ek@gmail.com

Prof. S.A. Itkar
Department of Computer Engineering
PES's Modern College of Engineering
Pune, India
suhasini_naik@yahoo.com

Abstract—Efficient data analysis can be made easier with datasets having columns in horizontal tabular layout. Aggregations using standard SQL return one column per aggregated group. So existing SQL aggregations have limitations in preparing datasets. In this paper we have proposed a framework to build dataset using a new class of functions called horizontal aggregations. To speed up the dataset preparation task we have partitioned vertical aggregations on grouping column and optimized SPJ method. Also it is proposed to integrate summary dataset, obtained from the result of horizontal aggregation, into homogeneous cluster using K-means algorithm.

Keywords-*Data preprocessing, Aggregation, clustering, K-means, SQL, SPJ, relational DBMS*

*****

I. INTRODUCTION

It is a complex task to prepare a summary dataset in a relational database. Such summary dataset is very informative and can be used as input for data mining algorithms. It becomes faster and easier managing large data inside DBMS than outside with alternative tool. This paper presents an overview of different techniques using extended SQL functions and relational DBMS features to prepare datasets, their advantages and challenges that needs attention. After extensive survey and thorough study of related work it is found that using horizontal aggregation functions we can build summary dataset easily. These horizontal aggregation functions [7] prepared in SQL provides opportunity for automation and optimization. Proposed system provides a generic framework that can be customized and used to meet the data pre-processing needs of data mining professional with minimal effort and seamless implementation.

We believe the present work will accelerate the state of the art in dataset preparation task by partitioning large vertical aggregations on grouping columns and working on all intermediate vertical aggregations in parallel. We demonstrate our approach in an implementation that achieves respectable performance in dataset preparation task, and exhibits graceful progress in performance with larger, automatically generated datasets with horizontal layout consisting of tens of thousands of records. Section 2.0 discusses related work (Literature survey). Section 3.0 briefs technical details. In section 4.0 experimental evaluations are discussed. Section 5.0 gives conclusions.

II. RELATED WORK

This section gives detailed information about related work in data pre-processing using extended SQL techniques and RDBMS features. This literature survey is essential to analyze the importance of the proposed work. C. Ordonez in his paper discusses about how datasets with a horizontal denormalized layout can be built using Horizontal Aggregations [7]. Migrating data set preprocessing and transformation performed by external programs into a database system exploiting the SQL language. Paper has showed how to compute common sufficient statistics efficiently in SQL that benefit several models, which effectively summarize large data sets. Transforming and scoring data sets is much faster inside the database system. In summary, users can enjoy the major data management features provided by the database system (querying, recovery, security and concurrency control) and data set preprocessing, transformation and analysis are faster inside the database system [1]. A powerful language and system ATLaS implemented by H. Wang facilitates users to develop data-intensive application in SQL completely. He emphasized on writing new aggregates and table functions in SQL, rather than in procedural languages as in current Object-Relational systems. In this case, the goal is to efficiently compute item set support. Unfortunately, there is no notion of transposing results since transactions are given in a vertical layout [2]. Extending SQL with a computational clause which allows us to treat a relation as a multi-dimensional array and specify a set of formulas over it. This not only allows for ease of programming, but also offers the RDBMS an opportunity to perform better optimizations. Proposed optimizations in the paper avoid joins to express cell formulas, but they are not optimized to perform partial transposition for each group of result rows. The PIVOT and CASE methods avoid joins [3]. Like standard aggregation vertical percentage aggregation computes one percentage per row. The horizontal percentage aggregation returns each set of percentages adding 100 percentage as one row. Existing OLAP aggregate functions are slower than both proposed aggregations significantly. Horizontal aggregations are more general, have wider applicability. We can use them as a primitive extended operator to compute percentages [4]. In her paper Sunita Sarawagi has discussed architectural alternatives for mining activities. Her study briefs about coupling mining with database systems [5]. Two operators PIVOT and UNPIVOT on tabular data exchange rows and columns which enable data transformations. This transformed data is helpful in data analysis, data presentation and data modeling [6]. Popular K-means algorithm for clustering is implemented using three SQL methods. These SQL implementations are explained in the paper and showed how it can be integrated with a relational DBMS. K-means implementations work accurately. It can cluster data sets properly with large size as well. K-means algorithm using SQL is best suited for partitioning large set of

ISSN: 2321-8169
4295 - 4300

data, obtained from the result of horizontal aggregation, into homogeneous cluster [8]. Kmeans and Self-Organizing Maps for simultaneously processing Heterogeneous Data Mining (HDM) by Unified Vectorization[9]. Programming Bayesian classifiers (fundamental classification technique) in SQL using K-means clustering are explained in the paper [10].

III. TECHNICAL DETAILS

The architecture described in this paper applies to a wide set of data. The application and experiments presented here make use of SQL features of RDBMS. Many other data preprocessing applications can adapt our approach.

A. Horizontal aggregations

Three fundamental methods are proposed to evaluate horizontal aggregations [7]: SPJ (Select, Project, Join): SPJ queries are based on standard relational algebra operators; CASE: SQL queries are based on the programming CASE construct; PIVOT: SQL queries are based on the PIVOT operator, which is offered by DBMSs. As mentioned in the paper [7], all three query evaluation methods are compared and experiments with large tables state that CASE method is faster than the SPJ method but has similar speed to the PIVOT operator.

In this paper we have proposed a generic technique to auto generate SQL code to have aggregation and transposition together. Out of three evaluation methods we have optimized SPJ method by partitioning vertical aggregations and working on all intermediate results in parallel.

First consider the following standard SQL query that takes a subset $G_1 \ldots G_m$ from $D_1 \ldots D_p$

SELECT $G_1 \ldots G_m$, *sum(AggrColumn)*
From F
GROUP BY $G_1 \ldots G_m$;

Such aggregation query produces a wide table with grouping columns i.e. $m+1$ columns, with one group for each unique combination of values $G_1 \ldots G_m$ and one aggregated value per group (*sum*(AggrColumn) in this case). Evaluation of this query, the query optimizer takes three parameters as input: (1) the input table F, (2) the list of grouping columns $G_1 \ldots G_m$, (3) the column to aggregate (AggrColumn). The basic purpose of a horizontal aggregation is to transpose/pivot the aggregated column AggrColumn by a column subset of $G_1 \ldots G_m$; for simplicity assume such subset is $T_1 \ldots T_k$ where $k<m$. We partition the *GROUP BY* list into lists i.e. grouping columns and transposing columns. One list required to produce each group (j columns $1, \ldots, G_j$) and another list (k columns $T_1 \ldots T_k$) required to transpose aggregated values, where $\{G_1, \ldots, G_j\} \cap \{T_1, \ldots, T_k\} = 0$. Each distinct combination of $\{T_1, \ldots, T_k\}$ will automatically produce an output column. Particularly if $k = 1$ then there are $| \pi\ T_1\ (F) |$ columns (i.e. each value in T_1 becomes a column storing one aggregation). We have four input parameters required to generate SQL code to have horizontal aggregation -

- Input table F,
- List of *GROUP BY* columns $G_1, \ldots, G_j$,
- Column to aggregate (*AggrColumn*),
- List of transposing columns (Analysis columns) $T_1, \ldots, T_k$.

The main Fig. 1 gives an example of the input table F, a vertical *sum*() aggregation which we get traditionally stored in F_V, and a summary result in the form of horizontal aggregation stored in F_H.

F

K	D_1	D_2	A
1	3	X	9
2	2	Y	6
3	1	Y	10
4	1	Y	0
5	2	X	1
6	1	X	null
7	3	X	8
8	2	X	7

F_V

D_1	D_2	A
1	X	null
1	Y	10
2	X	8
2	Y	6
3	X	17

F_H

D_1	D_2X	D_2Y
1	null	10
2	8	6
3	17	null

Figure 1. Example of a Horizontal Aggregation [7].

SQL Extension with different Syntax:-

We have to extend *SELECT* statement which allows implementation of horizontal aggregations. This extended SQL represents non-standard SQL because columns in the result table are unknown when parsing of the query is done. F should be unchanged during evaluation of horizontal aggregation as new values may create new result columns. We extend standard SQL aggregate functions with a transposing *BY* clause followed by a column list (i.e. $T_1, \ldots, T_k$). Instead of producing one number for aggregate function, this extension produces a horizontal set of numbers. Suggested syntax is as follows.

SELECT $G_1, \ldots, G_j$, H *(AggrColumn BY* $T_1, \ldots, T_k$*)*
FROM F
GROUP BY $G_1, \ldots, G_j$;

The subgroup columns (i.e. Transposing columns) $T_1, \ldots, T_k$ should be a parameter associated to the aggregation. Therefore they should be inside the parenthesis as arguments. Here *H()* represents some SQL aggregation (e.g. *sum(), count(), min(), max(), avg()*). *H()*function must have at least one argument represented by AggrColumn, followed by a column list. Columns $G_1, \ldots, G_j$ in the *GROUP BY* clause (if present) determines result rows. Result columns are determined by all combinations of columns $T_1, \ldots, T_k$, where $k=1$ is the default. Also, $\{G_1, \ldots, G_j\} \cap \{T_1, \ldots, T_k\} = 0$. We also have to develop sound and efficient evaluation mechanisms.

Fig. 2 and Fig. 3 show an overview of the main steps required to prepare dataset in Horizontal Layout.

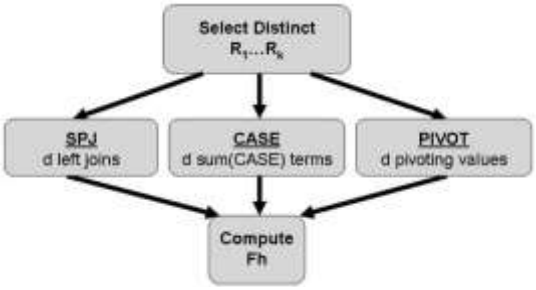

Figure 2. Horizontal Aggregation Steps based on F (un-optimized) [7]

 ISSN: 2321-8169
4295 - 4300

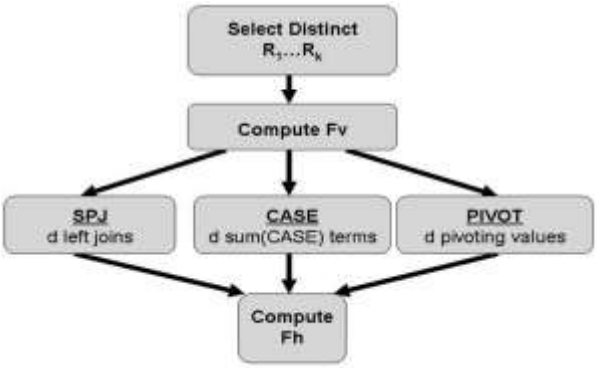

Figure 3. Horizontal Aggregation Steps based on F_V (optimized) [7]

B. SPJ Method

The SPJ method explained by Ordonez [7] creates an intermediate table with vertical aggregation for each result column. All these intermediate tables are then joined to produce result table F_H. Proposed method aggregates from fact table F into d projected tables with d Select-Project-Join-Aggregation queries. Each intermediate table F_I corresponds to one subgrouping combination for selected analysis columns and has $\{L_1, \ldots, L_j\}$ as primary key and an aggregation on A as the only column which is non-key. It is required to create an additional table F_0 that will be outer joined with all intermediate projected tables to get a complete horizontal layout result dataset. Ordonez has proposed two basic methods to compute horizontal table F_H. The first method directly aggregates from fact table F. In the second method equivalent vertical aggregations are computed in a temporary table F_V grouping by $L_1, \ldots, L_j, R_1, \ldots, R_k$. Then horizontal aggregations are computed from F_V. As standard aggregations are distributive, F_V is a compressed version of F. Here we consider the aggregation computed indirectly based on the intermediate temporary table F_V. Considering Fv is a table containing the vertical aggregation, based on $L_1, \ldots, L_j, R_1, \ldots, R_k$. Let $V()$ represent the corresponding vertical aggregation for $H()$. The statement to compute Fv gets a cube:

INSERT INTO F_V
SELECT $L_1, \ldots, L_j, R_1, \ldots, R_k$, $V(A)$
FROM F
GROUP BY $L_1, \ldots, L_j, R_1, \ldots, R_k$;

Table F_0 defines the number of rows in result horizontal layout table. It builds the primary key. F_0 is contains every existing combination of $L_1, \ldots, L_j$. Table F_0 has $\{ L_1, \ldots, L_j \}$ as primary key and it does not have any non-key column.

INSERT INTO F_0
SELECT DISTINCT $L_1, \ldots, L_j$
FROM $\{F \mid F_V\}$;

We have to get all distinct combinations of subgrouping columns $R_1, \ldots, R_k$, to auto-generate the name of dimension columns, to get d, the number of dimensions (d number of intermediate tables), and to generate the *WHERE* clause boolean expressions. Each *WHERE* clause consists of a conjunction of k equalities based on $R_1, \ldots, R_k$.

SELECT DISTINCT $R_1, \ldots, R_k$
FROM $\{F \mid F_V\}$;

Tables $F_1, \ldots, F_d$ contain individual aggregations for each combination of $R_1, \ldots, R_k$ analysis columns . The primary key of table F_I is $\{L_1, \ldots, L_j\}$.

INSERT INTO FI SELECT $L1, \ldots, Lj$; $V(A)$
FROM $\{F \mid F_V\}$;
WHERE $R_1 = v1I$ *AND ... AND* $R_k = vkI$
GROUP BY $L_1, \ldots, L_j$;

Each table F_I then aggregates only those rows that corresponds to the I^{th} unique combination of $R_1, \ldots, R_k$, given by the *WHERE* clause. In SPJ method optimization is possible by synchronizing table scans to compute the d tables in one pass. Finally, to get F_H we need d left outer joins with the $d+1$ tables so that all individual aggregations are properly assembled as a set of d dimensions for each group. Outer joins set result columns to null for missing combinations for the given group. In general, nulls should be the default value for groups with missing combinations. It would be incorrect to set the result to zero or some other number by default if there are no qualifying rows. Such approach should be considered on a per case basis.

INSERT INTO F_H
SELECT
$F_0.L_1, F_0.L_2 \ldots F_0.L_j,$
$F_1.A, F_2.A, \ldots, Fd.A$
FROM F_0
LEFT OUTER JOIN F_1
ON $F_0.L_1 = F_1.L_1$ *and…* $F_0.L_j = F_1.L_j$
LEFT OUTER JOIN F_2
ON $F_0.L_1 = F2.L_1$ *and …* $F_0.L_j = F_2.L_j$
…
LEFT OUTER JOIN F_d
ON $F_0.L_1 = F_d.L_1$ *and…* $F_0.L_j = F_d.Lj$;

C. Optimized SPJ

We have optimized SPJ method to evaluate temporary table F_V by partitioning whole vertical aggregation $V(A)$ on grouping columns. We have evaluated intermediate temporary tables $F_{v1}, F_{v2} \ldots$ instead of evaluating whole F_V. Number of intermediate vertical aggregation table depends upon distinct values $R_1, \ldots R_k$.

INSERT INTO Fvi
SELECT $L_1, \ldots, L_j, R_1, \ldots, R_k$, $V(A)$
FROM F
GROUP BY $L_1, \ldots, L_j, R_1, \ldots, R_k$

Using these intermediate Fvi we have evaluated Horizontal aggregations F_{Hi}. Number of F_H equals to number of Fv. All these F_{Hi} can be merged to prepare single F_H.
Fig. 4 shows an overview of the main steps required to prepare dataset in Horizontal Layout using optimized SPJ method.
Table I and II gives example of multidimensional dataset which can be created using horizontal aggregations.

Advantages of Partitioned/Parallel optimized SPJ evaluation method are listed:

- Speed: Parallel SPJ evaluation method generates horizontal layout data set in less time.
- Scalability: As dataset creation task is distributed over multiple threads the system is scalable to huge datasets.

 ISSN: 2321-8169
4295 - 4300

TABLE I. DATASET IN HORIZONTAL LAYOUT, SUITABLE FOR DATA MINING

CityID	Scooty_Male	Scooty_Female	Activa_Male	Activa_Female	Vespa_Male	Vespa_Female
1	50	80	200	90	25	30
2	45	75	190	85	27	28
3	52	85	220	80	12	20
4	47	77	192	87	29	30
5	60	90	210	100	35	40

TABLE II. A MULTIDIMENSIONAL DATASET IN HORIZONTAL LAYOUT, SUITABLE FOR DATA MINING

	SalesAmt				TransactionCount				VehicleCount				TotatalSale
CityID	Sun	Mon	…	Sat	Jan	Feb	…	Dec	Activa	Scooty	…	Vespa	AmtInLacs
1	10	3	…	9	50	56	…	56	400	200	…	50	400
2	12	4	…	10	60	66	…	45	450	300	…	20	600
3	15	5	…	12	55	54	…	45	350	200	…	35	500
4	10	4	…	11	60	45	…	60	300	150	…	20	450
5	13	3	…	10	45	50	…	50	400	300	…	10	500

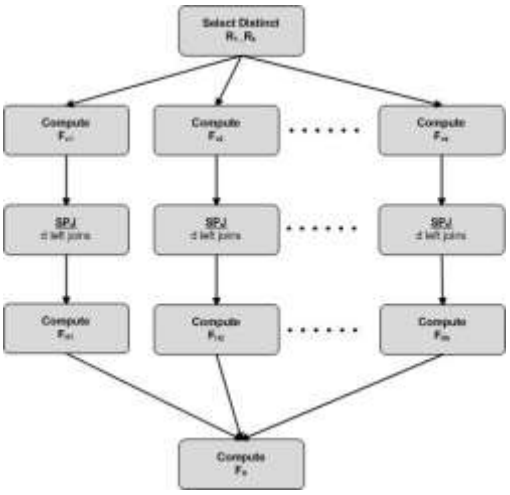

Figure 4. Horizontal Aggregation Steps based on F_V (Optimized SPJ)

D. *Integrating K-Means Algorithm with Horizontal Aggregation [8]*

F_H or F_{Hi} generated using SPJ method is used as input to K-Means clustering algorithm. These clusters can be useful in further data mining analysis.

Advantages of integrating clustering algorithm with Horizontal Aggregations are listed:

- Horizontal layout data set has less number of rows compared to vertically aggregated or unprocessed dataset. Eventually time required to generate clusters is less.
- Horizontal layout data set has multiple columns so that one can create clusters by selecting columns by choice.
- Clusters generated for horizontal layout dataset are clearer than to clusters generated for unprocessed dataset.

E. *DBMS Limitations*

There exist two DBMS limitations with horizontal aggregations:

- Reaching the maximum number of columns in one table.
- Reaching the maximum column name length when columns are automatically named

When the set of transposing columns { $R_1, \ldots, R_k$ } has a large number of distinct combinations of values, Horizontal aggregation can return a table that goes beyond the maximum number of columns in the DBMS. Secondly, the important issue is automatically generating column names uniquely. Long column names may get generated if there are many analysis columns (subgrouping) $R_1, \ldots, R_k$ or columns are of string data types. These very long column names may exceed DBMS limits. But there is solution to these limitations. F_H can be vertically partitioned to solve the problem of maximum number of columns. Each partitioned table should be such that it does not exceed the maximum number of columns allowed by the DBMS. Each partitioned table must have $L_1, \ldots, L_j$ as its primary key. Second problem of column name length can be solved by generating identifiers for columns using integers. To map column identifiers to full descriptions one mapping dimension description table is required.

IV. EXPERIMENTAL RESULTS

This section, presents our experimental evaluation on a commercial DBMS. We evaluate query optimizations, compare the normal SPJ and optimized method. We have also evaluated impact on cluster formation when input is provided as horizontal layout of dataset against raw dataset.

A. *Setup: Computer Configuration and Datasets*

We used MS SQL Server 2012, running on a Desktop machine with 1.70 GHz running capacity, Dual Core

ISSN: 2321-8169
4295 - 4300

processor, RAM- 4 GB and 10 GB on disk. The C# language is used to programme SQL code generator and connected to the database server via SQLClient. We used UCI machine learning datasets like Adult, Bank, Automobile, Tumor. We analyzed queries having one as well as multidimensional horizontal aggregation, with different grouping and horizontalization columns. Each experiment was repeated three times and the average time in milliseconds is considered. We have selected several column combinations to get results. To get meaningful data sets we have selected high selectivity columns for the grouping columns (left key) and low selectivity columns for the transposing columns (right key).

B. Query Optimizations

Fig. 5 analyzes our query optimization, applied to SPJ method. Our purpose is to assess the acceleration obtained by partitioning whole vertical aggregation *V(A)* on grouping columns. We can see this optimization accelerates SPJ method. We can see the impact is significant and accelerated evaluation time considerably.

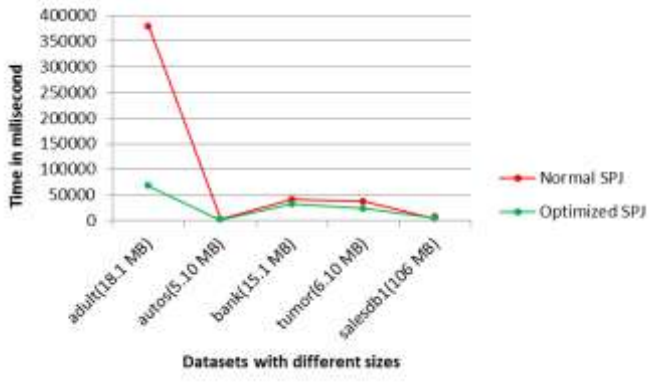

Figure 5. Normal SPJ vs. Optimized SPJ

C. Cluster Generation

Fig. 6 analyzes performance achieved in cluster generation if horizontal layout dataset is provided as input to K-means clustering algorithm. This acceleration is obtained as horizontal layout data set has less number of rows compared to vertically aggregated or unprocessed dataset. We can see the impact is significant and accelerated clustering time considerably.

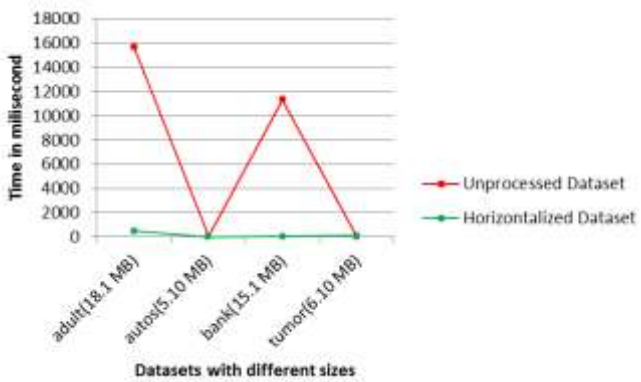

Figure 6. Unprocessed Dataset vs. Horizontal lyout Dataset

V. CONCLUSIONS

We have proposed a framework to implement horizontal aggregations. These horizontal aggregations helps preparing data sets and gaining insight into data. This summary dataset is useful in data mining and OLAP cube exploration. Horizontal aggregations create data sets with a horizontal layout. Data mining algorithms commonly require such transformed dataset. OLAP cross-tabulation also needs such dataset. Instead of a single number per group horizontal aggregation basically returns a set of numbers resembling a multidimensional vector. We proposed a generic framework to automate dataset preparation in horizontal layout form. From a query optimization perspective, we proposed optimized SPJ method in which partitioning of large dataset is done as per analysis columns (transposing columns) and parallel operations are implemented to generate summary dataset in lesser time. As SPJ method is based on select, project and joins queries it is important from a theoretical point of view. We proved the performance of optimized SPJ method over normal SPJ method. Our proposed framework for preparing horizontal aggregations can be used as a database method to generate efficient SQL queries automatically. This method requires three sets of parameters: grouping columns (Primary columns), subgrouping columns (Analysis columns/Transposing columns), and aggregated column. In our experiment we have generated results for multiple aggregated columns but for same set of grouping columns. It is also proposed integrating horizontal layout dataset with K-means clustering algorithm for faster and meaningful cluster generation.

REFERENCES

[1] Carlos Ordonez, "Data Set Preprocessing and Transformation in a Database System", Intelligent Data Analysis, pp. 613-631, Vol. 15, No. 4, 2011.

[2] H. Wang, C. Zaniolo, and C.R. Luo, "ATLAS: A Small But Complete SQL Extension for Data Mining and Data Streams",Proc. 29th International Conf. Very Large Data Bases (VLDB 2003), pp. 1113-1116, 2003.

[3] A.Witkowski, S. Bellamkonda, T. Bozkaya, G. Dorman, N. Folkert, A. Gupta, L. Sheng, and S. Subramanian, "Spreadsheets in RDBMS for OLAP ", Proc. ACM SIGMOD International Conf. Management of Data (SIGMOD 2003), pp. 52-63, 2003.

[4] Ordonez, "Vertical and Horizontal Percentage Aggregations", Proc. ACM SIGMOD International Conf. Management of Data (SIGMOD 2004), pp. 866-871, 2004.

[5] Sunita Sarawagi, Shiby Thomas, Rakesh Agrawal, "Integrating Association Rule Mining with Relational database systems: Alternative and Implications", ACM SIGMOD 1998, 1998.

[6] Conor Cunningham, Cesar A. Galindo-Legaria , Goetz Graefe, "PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS",VLDB Conference 2004, 2004.

[7] Carlos Ordonez and Zhibo Chen, "Horizontal Aggregations in SQL to prepare Data Sets for Data Mining Analysis", IEEE Transactions on Knowledge and Engineering, pp. 678-691,Vol. 24, No. 4, April 2012.

[8] Carlos Ordonez,"Integrating K-Means Clustering with a Relational DBMS Using SQL" IEEE Transactions on Knowledge and Data Engineering, pp-188-201,Vol. 18, No. 2, February 2006.

[9] Farid Bourennani, Mouhcine Guennoun , Ying Zhu, "Clustering Relational Database Entities using K-

Means",Second International Conference on Advances in Databases , Knowledge and Data Applications, 2010.

[10] Carlos Ordonez, Sasi K. Pitchaimalai, Ying Zhu, "Bayesian Classifiers Programmed in SQL", IEEE Transactions on Knowledge and Engineering, Vol 22, No. 1, January 2010, pp. 139-144.