Volume 4, Issue 2, February 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

# A Review on Techniques for Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining

**Rekha S. Nyaykhor**
*M.Tech*
*Dept. of CSE, Priyadarshini Bhagwati College of Engineering*
*Nagpur, India*

**Nilesh.T.Deotale**
*Asst. Professor*
*Dept. of CSE, Priyadarshini Bhagwati College of Engineering*
*Nagpur, India*

***Abstract — Preparing a data set for analysis is the most time consuming task in a data mining project, which requires many complex SQL queries, joining tables and aggregated columns. Existing SQL aggregations have limitations to prepare data sets as they return one column per aggregated group. Manual effort is required to build data sets, where a horizontal layout is required. Horizontal aggregations function is proposed which is an extension to SQL aggregate functions to produce aggregations in tabular form, returning a set of numbers instead of one number per row. In this survey paper, we discuss about the various techniques which were used to get the data sets in a tabular form and returns a set of numbers instead of one value per row. Horizontal aggregations build data sets with a horizontal layout, which is the standard layout required by most data mining algorithms.***

***Keywords— Horizontal Aggregations, SQL, data sets, data mining, horizontal layout.***

## I. INTRODUCTION

Preparing a database needs identification of relevant data and then normalizing the tables. Building the relevant data for data mining, is a most time consuming task. This task generally requires writing long SQL statements or customizing SQL code is needed if it is automatically generated by some tool. There are two main ingredients in such SQL code namely join and aggregation. The most widely-known aggregation is the sum of a column over group of rows. There are many aggregation functions and operators in SQL. Unfortunately, all these aggregations have limitations to build data sets for data mining purposes. Based on current available functions and clauses in SQL, a significant effort is required to compute aggregations. Such effort is due to the size and complexity of SQL code which needs to be written, optimized and tested. In general aggregations are hard to interpret when there are many result rows. In order to perform the analysis of exported tables in spreadsheets it will be more convenient to have aggregations of the same group in one row. With such limitations in mind, we propose a new class of aggregate functions that aggregates numeric expressions and transpose results to produce a data set with a horizontal layout.

Functions belonging to this class are called horizontal aggregations. Horizontal aggregations represent an extended form of traditional SQL aggregations, which return a set of values in a horizontal layout instead of a single value per row. Horizontal aggregations provide several unique features and advantages. First, they represent a template to generate SQL code from a data mining tool. This SQL code reduces manual work in the data preparation phase in a data mining project. Second, the automatically generated SQL code is more efficient than SQL code written by an end user. Third, the data set can be created entirely inside the DBMS. Horizontal aggregations just require a small syntax extension to aggregate functions called in a SELECT statement. Alternatively, horizontal aggregations can be used to generate SQL code from a data mining tool to build data sets for data mining analysis.

Horizontal aggregation is evaluated by the following approach. Case: This method uses the “case” programming construct available in SQL. The case statement returns a value selected from a set of values based on Boolean expressions. SPJ: This method is based on relational operators only. The basic idea is to create one table with vertical aggregation for each result column, and then join all those tables to produce another table. [9] Pivot: It is a built-in operator offered by some DBMS, which transforms row to columns. This method internally needs to determine how many columns are needed to store the transposed table and it can be combined with the GROUP BY clause.

Horizontal aggregations propose a new class of functions that aggregate numeric expressions and the result are transposed to produce data sets with a horizontal layout. The operation is needed in a number of data mining tasks, such as unsupervised classification and data summation, as well as segmentation of large mixed data sets into smaller uniform subsets that can be easily manage, separately model and analyzed. To create datasets for data mining related works, efficient and summary of data are needed. Database as their nature contains large amount of data. To extract information from the database Structured Query Languages are used. SQL commonly used for the aggregation of large volumes of

 

data. With the help of aggregation details in one table can be aggregated with details in another table. Aggregation functions play a major in the summarization of tables. Normal SQL aggregation functions are sum ( ), avg ( ), min ( ), max ( ) and count ( ).

Vertical aggregations: Vertical aggregation is similar to standard SQL aggregations. This produces results in a vertical format and contains more rows. There are some approaches which produce results in vertically aggregated form.

Horizontal aggregations: Horizontal aggregations are also similar to standard SQL aggregations but this can produce results in horizontal tabular format. Here data sets for all the operations are produced from some data mining tool and apply the aggregation operations on that dataset. To produce results in horizontal layout small syntax extensions to normal SQL syntax is needed.

There are some advantages for horizontal aggregations:

1) SQL code reduces manual work in the data preparation phase in a data mining project.
2) The SQL code is automatically generated it is likely to be more efficient than SQL code written by an end user.
3) The data sets can be created in less time.
4) The data set can be created entirely inside the DBMS.

## II. RELATED WORK

SQL extensions to define aggregate functions for association rule mining. Their optimizations have the purpose of avoiding joins to express cell formulas, but are not optimized to perform partial transposition for each group of result rows. C. Cunningham [2] proposed an optimization and execution strategies in an RDBMS which uses two operators i.e., PIVOT operator on tabular data that exchange rows and columns, enable data transformations useful in data modelling, data analysis, and data presentation. They can quite easily be implemented inside a query processor system, much like select, project, and join operator. Such a design provides opportunities for better performance, both during query optimization and query execution. Pivot is an extension of Group By with unique restrictions and optimization opportunities, and this makes it very easy to introduce incrementally on top of existing grouping implementations.

H Wang, C. Zaniolo [3] proposed a small but complete SQL Extension for data mining and data Streams. This technique is a powerful database language and system that enables users to develop complete data-intensive applications in SQL by writing new aggregates and table functions in SQL, rather than in procedural languages as in current Object-Relational systems. The ATLaS system consist of applications including various data mining functions, that have been coded in ATLaS SQL, and execute with a modest (20–40%) performance overhead with respect to the same applications written in C/C++. This system can handle continuous queries using the schema and queries in Query Repository.

C. Ordonez (2004) [4] introduced two aggregation functions. The first function returns one row for each percentage in vertical form like standard SQL aggregations. The second function returns each set of percentages adding 100% on the same row in horizontal form. These novel aggregate functions are used as a framework to introduce the concept of percentage queries and to generate efficient SQL code. Experiments study different percentage query optimization strategies and compare evaluation time of percentage queries. The advantage is that horizontal aggregation reduces the number of rows and columns. Disadvantage is that vertical aggregation increase the number of rows and columns. Thus increases the complexity.

G. Luo and J.F. Naughton (2005) [5] developed the immediate materialized view maintenance with transaction consistency where is enforced by generic concurrency control mechanism. A latch pool for aggregate join view is introduced. The latches in the latch pool guarantee that for each aggregate group, at most one tuple corresponding to this group exists in the aggregate join view. The main advantage is that deadlock problem is solved. The main disadvantage is many join operations are used.

J. Gray, A. Bosworth, A. Layman, and H. Pirahesh [6] proposed a relational aggregation operator that generalizing Group-By, Cross-Tab, and Sub-Totals. The cube operator generalizes the histogram, cross tabulation, roll-up, drill-down, and sub-total constructs. The cube operator can be imbedded in more complex non-procedural data analysis programs and data mining. The cube operator treats each of the N aggregation attributes as a dimension of N-space. The aggregate of a particular set of attribute values is a point in this space and the set of points forms an N-dimensional cube. Super-aggregates are computed by aggregating the N-cube to lower dimensional spaces. Creating a data cube requires generating the power set (set of all subsets) of the aggregation columns. Since the CUBE is an aggregation operation, it makes sense to externalize it by overloading the SQL GROUP BY operator.

C. Ordonez [13] proposes an Integration of K-means clustering with a relational DBMS using SQL. This technique consists of three SQL implementations. First step is a straightforward translation of K-means computations into SQL, and an optimized version based on improved data organization, efficient indexing, sufficient statistics, and rewritten queries, and an incremental version that uses the optimized version as a building block with fast convergence and automated reseeding. The first implementation is a straightforward translation of K-means computations into SQL, which serves as a framework to build a second optimized version with superior performance. The optimized version is then used as a building block to introduce an incremental K-means implementation with fast convergence and automated reseeding.

## III. HORIZONTAL AGGREGATION

Horizontal aggregations are the operations that perform horizontal summary of data in tabular format or horizontal layout. The base tables used to describe the proof of concept are presented in fig. 1. These tables are used for describing operations of SPJ, PIVOT and CASE.

 

| 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 |

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

| $D_1$ | $D_2$X | $D_2$Y |
|---|---|---|
| 1 | null | 10 |
| 2 | 8 | 6 |
| 3 | 17 | null |

Fig. 1 – Input table F (a), traditional vertical aggregation $F_V$ (b), and horizontal aggregation $F_H$(c)

As seen in Fig. 1, sample data is given in input table. Vertical aggregation result is presented in (b). In fact the result generated by SUM function of SQL is presented in (b). Horizontal aggregation results are presented in (c). There is a common field K in F and $F_V$. In $F_V$, D2 consist of only two distinct values X and Y and is used to transpose the table. The aggregate operation is used in this is sum ( ). The values within $D_1$ are repeated, 1 appears 3 times, for row 3, 4 and, and for row 3 & 4 value of $D_2$ is X & Y. So $D_2$X and $D_2$Y is newly generated columns in $F_H$.

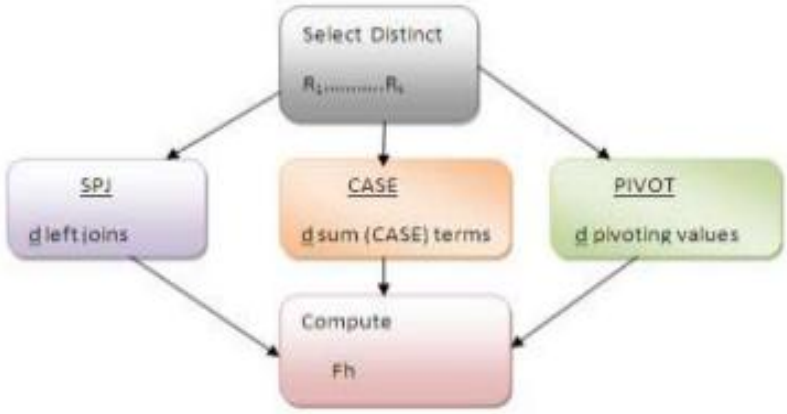


Fig. 2- shows steps on all methods based on input table

As seen in Fig. 2, for all aggregations such as PIVOT, CASE and SPJ certain steps are carried out. However, the first step of all operations starts with SELECT query. It directly aggregate from F. Then based on the operation other activities are performed for computing horizontal aggregations.

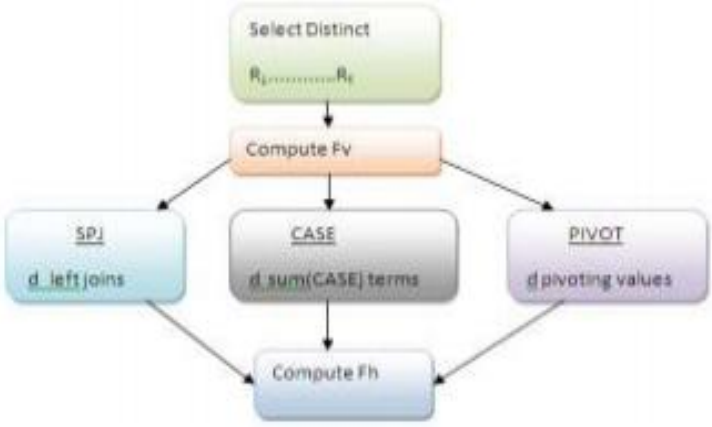


Fig. 3- shows steps on all methods based on table containing results of vertical aggregations

As seen in fig. 3, for all aggregations such as PIVOT, CASE and SPJ certain steps are carried out. It computes the equivalent vertical aggregation in a temporary table Fv. Then horizontal aggregations can be instead computed from Fv, which is a compressed version of F.

SPJ Method: Vertical operations are used in SPJ method. For every column one table is generated in this model. Afterwards, the tables generated are joined in order to obtain final horizontal aggregations. The procedure followed is as given in [9].

PIVOT Method: RDBMS has built in PIVOT operation. This is used by the PIVOT operation we proposed in this paper. This construct can provide transpositions. Therefore for evaluating horizontal aggregations it can be used.

 

```
SELECT L1,..,Lj
,sum(CASE WHEN R1= v11 and .. and Rk = vk1
THEN A ELSE null END)
..
SELECT DISTINCT R1
FROM F; /* produces v1; . . . ; vd */
SELECT
L1; L2; . . . ; Lj
,v1; v2; . . . ; vd
INTO Ft
FROM F
SELECT L1; L2; . . . ; Lj,R1, A
FROM F) Ft
PIVOT(
V (A) FOR R1 in (v1; v2; . . . ; vd)
) AS P;
```

Listing 1 – Shows optimized instructions for PIVOT construct

As seen in listing 1, the queries have been optimized by choosing only the columns that are required by horizontal aggregations.

CASE Method: This operation is based on the CASE structure provided by SQL. It has many built in Boolean expressions. Out of them one of the expressions is returned. Projection or aggregation is similar to this from relational query point of view. It is achieved internally by using many conditions with conjunctions. In this case horizontal aggregations exhibit two strategies. The first one does the computations directly from the table given as input while the second one performs vertical aggregation and the results are sent to an arbitrary table. This table is used again in horizontal aggregation generation. The procedure used here is as presented in [9].

## IV. COMPARATIVE STUDY

Various approaches for performing horizontal aggregation are taken for comparison is as follows:

A. *Grouping combinations*

This operator is developed to handle the aggregation and grouping of high dimensional data. This operator can solve limitations of normal GROUPING operator. The operators like GROUPING SET, ROLLUP, and CUBE [6] can also perform aggregation and can produce tabular results. But these are difficult to use when the available input dataset is very large. When the available input dataset is large GROUPING SET operator require long complex SQL queries. The ROLL UP operator can perform aggregation on smaller datasets and produce tabular results vertical format. But the vertical format is not efficient for many data mining approaches. The CUBE operator can perform aggregations on large datasets. But the CUBE operator eliminates some of the details when aggregation is performed. Because of these limitations GROUPING COMBINATION operator is developed. But the GROUPING COMBINATION operator can implemented only with the help of complex algorithms. So its performance is low in the case of execution.

B. *Atlas*

It is a database language developed to solve the limitations of SQL operator. ATLaS [3] can perform aggregations that are not possible with standard SQL. Standard SQL can support only basic aggregation operations. This language use aggregations and table functions in SQL. To perform operations in ATLAS entire SQL statement is divided into three functions INTIALIZE, ITERATE, TERMINATE. Declarations are given in the INITIALIZE section. The major operation is specified in the ITERATIVE section. The final statement to execute is specified in the TERMINATE sections. The major advantage of ATLaS is that it can support online aggregations. In online aggregation user evaluate aggregation query in an online fashion execution. But the execution of ATLaS operator consumes more space than executing with normal SQL [3]. Also it cannot results in horizontal tabular format.

C. *Horizontal And Vertical percentage*

This aggregations help to calculate percentages for operations using vertical and horizontal aggregations. Vertical percentage aggregation returns one row for percentage in vertical format. Horizontal percentage aggregation returns entire 100% of results on the same row. This percentage aggregation used only for computing percentages in vertical or horizontal format. These aggregations are similar to normal vertical and horizontal aggregation except that it can compute results only in percentage format. So there may be extra work in the percentage conversion when other computations are required on the dataset.

D. *Interpreted storage format*

This is developed to handle null values in horizontal and vertical layouts. Interpreted format can handle all the sparse data management complexities. Horizontal aggregation requires more space due to large number of null values. Vertical

 

aggregations have small number of null values. Interpreted storage format store nothing for null attributes. When the tuple has value for an attribute in the table, attribute identifier (attribute_id), a length field, value appears in the tuple. This stored along with particular head. The major problem here is that the value stored in this format is not easily accessible for operations.

Table I. COMPARISON of AGGREGATION APPROACHES

| Method Discussed | Types of Aggregations | Features |
|---|---|---|
| Grouping Combination Operator | Horizontal And Vertical | Implemented with complex algorithms |
| ATLAS | Vertical | Solve limitation of normal SQL |
| Vertical And Horizontal Percentage Aggregation | Vertical And Horizontal | Can only operate on percentages |
| Interpreted Storage Format | Vertical And Horizontal | Data Retrieval is difficult |
| UNPIVOT operator | Horizontal | Use small syntax extension in select statement |

E. *UNPIVOT operator*

UNPIVOT operator is opposite of PIVOT operator that is they transform columns into rows. This creates additional rows from columns to produce a big table. Because of these vertical layout it cannot use for most of the mining algorithms which require horizontal table as input. UNPIVOT operator [2] is commonly used for the statistical computation of some data mining approaches. The normal syntax is given below.

## V. CONCLUSION

This paper presents techniques to support horizontal aggregations through SQL queries. The result of the queries is the data which is suitable for data mining operations. It does mean that this paper achieves horizontal aggregations through some constructs that includes SQL queries as well. The methods we use in this paper include CASE, SPJ and PIVOT.

We have explained it is not possible to evaluate horizontal aggregations using standard SQL without either joins or ”case” constructs using standard SQL operators. Our proposed horizontal aggregations can be used as a database method to automatically generate efficient SQL queries.

We have tried to introduce a new class of extended aggregate functions, called horizontal aggregations which help preparing data sets for data mining and OLAP cube exploration. Specifically horizontal aggregations are useful to create data sets with a horizontal layout, as commonly required by data mining algorithms and OLAP cross-tabulation. Basically, a horizontal aggregation returns a set of numbers instead of a single number for each group, resembling a multi-dimensional vector. We proposed an abstract, but minimal, extension to SQL standard aggregate functions to compute horizontal aggregations which just requires specifying subgrouping columns inside the aggregation function call. From a query optimization perspective, we proposed three query evaluation methods. The first one (SPJ) relies on standard relational operators. The second one (CASE) relies on the SQL CASE construct. The third (PIVOT) uses a built-in operator in a commercial DBMS that is not widely available.

## REFERENCES

[1] M. Madhavi and S. Kavitha,"*Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining Analysis*", Madhavi et al. / IJEA, Vol. 1 Issue 6 ISSN: 2320-0804, PP 1-7, 2013.

[2] C. Cunningham, G. Graefe, and C.A. Galindo-Legeria, “*PIVOT AND UNPIVOT: Optimization and Execution Strategies in an RDBMS*,”Proc: 13th Int’l Conf. Very Large Data Bases (VLDS’04), pp.998-1009, 2004.

[3] H. Wang, C. Zaniolo, and C.R. Luo.”*ATLaS: A small but complete SQL extension for data mining and data streams*”. In Proc. VLDB Conference, pages 1113–1116, 2003.

[4] C. Ordonez, “*Vertical and Horizontal Percentage Aggregations*,” Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD’04), pp. 866-871,2004.

[5] G. Luo, J.F. Naughton, C.J. Ellmann, and M. Watzke, “*Locking Protocols for Materialized Aggregation Join Views*,” IEEE Trans. Knowledge and Data Eng., vol. 17, no.6, pp. 796-807, June 2005.

[6] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. “*Data cube: A relational aggregation operator generalizing group-by, cross-tab and subtotal*”. In ICDE Conference, pages 152–159,1996

[7] J. Han and M. Kamber, “*Data Mining: Concepts and Techniques*”, first ed. Morgan Kaufmann, 2001.

[8] C. Ordonez, “*Horizontal Aggregations for Building Tabular Data Sets*,” IEEE Trans. Knowledge and Data Eng,” VOL. 24, NO. 4, pp. April 2012.

 

[9] Carlos Ordonez and Zhibo Chen," *Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining Analysis*", IEEE transactions on knowledge and data engineering, vol. 24, NO. 4, pp 1-14, APRIL 2012.

[10] Mr. Ranjith Kumar K and Mrs. Krishna Veni," *Prepare datasets for data mining analysis by using horizontal aggregation in SQL*", Ranjith Kumar K et al, Int.J. Computer Technology & Applications, Vol 3(6), 1945-1949 IJCTA, pp. 1-5, Nov-Dec 2012

[11] P. Goel, and B.R. Iyer, "*Hyper graph Based Reordering of Outer Join Queries with Complex Predicates,*" Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '95) , pp. 304-315, 1995.

[12] C. Galindo-Legaria and A. Rosentahl, "*Outer Join Simplification and Reordering for Query Optimization,*" ACM Trans. Database Systems, vol.22, no.1, pp.43-73, 1997.

[13] C. Ordonez. "*Integrating K-means clustering with a relational DBMS using SQL*," IEEE Transactions on Knowledge and Data Engineering (TKDE), 18(2):188–201, 2006.

[14] C. Ordonez, "*Data Set Preprocessing and Transformation in a Database System,*" Intelligent Data Analysis, vol. 15, no. 4, pp. 613-631, 2011.

[15] Nisha.S and B.Lakshmipathi, "*Optimization of Horizontal Aggregation in SQL by Using K-Means Clustering*", International Journal of Advanced Research in Computer Science and Software Engineering , Volume 2, Issue 5, ISSN: 2277 128X, PP. 1-6, May 2012.

[16] Rajesh Reddy Muley, Sravani Achanta and Prof.S.V.Achutha Rao, "*Query Optimization Approach in SQL to prepare Data Sets for Data Mining Analysis*", International Journal of Computer Trends and Technology (IJCTT) – volume4Issue8,pp 1-5,August 2013.

[17] Durka.C and Kerana Hanirex.D, '' *An Efficient Approach for Building Dataset in Data Mining*", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 3,pp 1-5, March 2013.

[18] Pradeep Kumar and Dr. R. V. Krishna, "*Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining Analysis*," IOSR Journal of Computer Engineering (IOSRJCE) ISSN: 2278-0661, ISBN: 2278-8727 Volume 6, Issue 5, PP. 36-41, (Nov. - Dec. 2012).

[19] K. Anusha, P. Radhakrishna and P. Sirisha," *Horizontal Aggregation using SPJ Method and Equivalence of Methods*", IJCST, Vol. 3, Issue 1, Spl. 5, pp 1-4, Jan. - March 2012 .

 