Horizontal Aggregations In SQL To Generate Data Sets For Data Mining Analysis In An Optimized Manner

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
Data mining is the domain which has utility in real world applications. Data sets are prepared from regular transactional databases for the purpose of data mining. However, preparing datasets manually is time consuming and tedious in nature as it involves aggregations, sub queries and joins. Moreover the traditional SQL Structured Query Language) aggregations such as MAX, MIN etc. can generate single row output which is not useful in generating datasets. Therefore it is essential to build horizontal aggregations that can generate datasets in horizontal layout. These data sets can be used further for data mining in the real world applications. This paper focuses on building user-defined horizontal aggregations such as PIVOT, SPJ(SELECT PROJECT JOIN) and CASE whose underlying logic uses SQL queries.

Keywords-
Data Mining, Horizontal Aggregations, PIVOT, SQL, Data

1. Introduction

Horizontal aggregation is new class of function to return aggregated columns in a horizontal tabular layout. So many algorithms are required datasets with horizontal layout as input with several records and one variable per column. Managed large data sets without DBMS support can be a more difficult task. Different subsets of data points and dimensions are more flexible, easier and faster to do inside a relational database with SQL queries than outside with alternative tool. Horizontal aggregations can be performed by using operator; it is easily implemented inside a query processor, like a select, project and join operations. PIVOT operator on tabular data that exchange rows, enable data transformations useful in data modelling, data analysis, and data presentation. There are many existing functions and operators for aggregation in SQL.

In our horizontal aggregation provides a interface to generate SQL code from a data mining tools. This SQL code is further used to generate SQL queries, optimizing them and testing them for correctness. This SQL code reduces manual work in creating data sets for data mining project. Since SQL code is automatically generated by horizontal aggregation, it is easy and likely to be more efficient than SQL code written by human effort. A person who does not know SQL well or someone who is not familiar with the database schema (e.g. a data mining practitioner) can easily generate the SQL queries. Hence, data sets can be created in less time. The data set can be created inside the DBMS itself. In modern database environments, they used to export de-normalized data sets to cleaned and transformed outside a DBMS by using external tools (e.g. statistical packages). But sending large tables outside a DBMS is time taking, creates inconsistent copies of the same data and it will cause the compromise of database security. So, we are proposing a more effective, better migrated and more secure solution than external data mining tools. A horizontal aggregation needs just small syntax extension to existing SQL aggregate functions. Alternatively, horizontal aggregations can be used to generate SQL code from a data mining tool to build data sets for data mining analysis.

Manuscript received March 5, 2015
Manuscript revised March 20, 2015

2. RELATED WORK

SQL is the de facto standard to interact with relational databases. It is widely used in all kinds of applications where connectivity to database containing valuable business data is required. SQL provides commands of various categories such as DML, DDL, and DCL. Using SELECT query it is possible to use aggregations, sub queries and joins. The vertical aggregations supported by SQL include COUNT, MIN, AVG, MAX and SUM. These are known as aggregate functions as they produce summary of data [5]. The output of these functions is in the form of single row values. These values can’t be directly used for data mining. Therefore it is essential to use some data mining procedures in order to generate data sets.

Association rule mining [6] is used in OLAP applications as they can generate trends in the data [7]. In this paper we extend the SQL aggregate functions in order to build new constructs namely PIVOT, SPJ and CASE. SQL queries are used in clustering algorithms also as explored in [5]. Spreadsheet like operations as extensions to SQL queries are proposed in [8]. The paper also discussed optimizations for joins and other operations. However, it is known that CASE and PIVOT can be used to avoid joins. New class of aggregations can be generated by using algebra that has been used traditionally [9]. In fact this paper focuses on generating new class of aggregations known as horizontal aggregations which will optimize the joins as presented in

[10]. For optimizing queries tree-based plans are used traditionally [11].

On aggregations also there is lot of research found in the literature. Literature also includes cube queries and cross tabulations [12]. Relational tables can unpivoted as presented din [13]. Transformations are available that can be used for horizontal aggregations [14]. Unpivot and TRANSPOSE operators are similar. When compared with PIVOT transpose can reduce the number of operations required. They have inverse relationship between them. They can produce vertical aggregations and decisions tree required by data mining. Both operations are available in SQL Server [15].

Horizontal aggregations are also presented by researchers in [16] and [17] with known limitations. The limitation is that the resultant data cannot be directly used for data mining. In this paper we proposed new operators that are best used for horizontal aggregations. The results of these operations can be used for data mining purposes further. The proposed operations include SPJ, PIVOT and CASE.

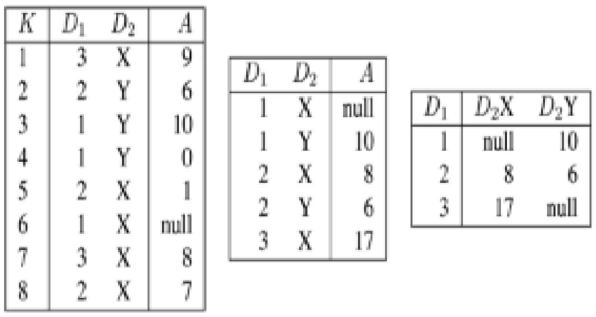
3. DEFINITIONS

Let F be a table having a simple primary key K represented by an integer, p discrete attributes and one numeric attribute: F(K;D1;D2;... …..;Dp;A). In OLAP terms, F is a fact table with one column used as primary key, p represents distinct columns and one measure column passed to standard SQL aggregations. F is assumed to have a star schema to simplify exposition. Column K will not be used to compute aggregations. Dimension lookup tables will be based on simple foreign keys. That is, one dimension column Dj will be a foreign key linked to a lookup table that has Dj as primary key. Input table F size is called N. That is, |F| = N. Table F represents a temporary table or a view based on a, star join, query on several tables. Other two main tables used in our proposed method are Vertical Table (F_V) and Horizontal Table (F_H).

3.1 Example

Fig. 1 gives an example showing the input table F, a traditional vertical sum() aggregation stored in F_V , and a horizontal aggregation stored in F_H. The basic SQL aggregation query is:

```
SELECT D1, D2, sum(A)
FROM F
GROUP BY D1, D2
ORDER BY D1, D2;
```



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_2X	D_2Y
1	null	10
2	8	6
3	17	null

Fig. 1 Example of F, F_V, and F_H.

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). .In F_V, D_2 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_2X and D_2Y are newly generated columns in F_H.

4. HORIZONTAL AGGREGATIONS

We introduce a new class of aggregations that have similar behaviour to SQL standard aggregations, but which produce tables with a horizontal layout. In contrast, we call standard SQL aggregations vertical aggregations since they produce tables with a vertical layout. 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.

4.1 Existing Method

Our main goal is to define a template to generate SQL code combining aggregation and transposition (pivoting). A second goal is to extend the SELECT statement with a clause that combines transposition with aggregation. Consider the following GROUP BY query in standard SQL that takes a subset L1, L2,.....Lm from D1, D2,...., Dp:

```
SELECT L1, L2,......,Lm, sum(A)
FROM F
GROUP BY L1, L2,,......,Lm;
```

This aggregation query will produce a wide table with m + 1 columns (automatically determined), with one group for each unique combination of values L1, L2,......, Lm and one aggregated value per group (sum(A) in this case). In order to evaluate this query the query optimizer takes three input parameters:

- The input table F,

- The list of grouping columns L1, L2,......,Lm;
- The column to aggregate (A).

4.2 Proposed Syntax in SQL

Here we are explaining SQL aggregate functions with a extension of BY clause followed with a list of columns to produce horizontal set of numbers.

```
SELECT L1;L2;.......Lm, H(A BY R1;R2.......;Rk)
FROM F
GROUP BY L1;L2;.......Lm;
```

The sub group columns R1;R2;……Rk should be a parameter of aggregation. Here H( ) represents SQL aggregation. It contains at least one argument represented by A. The result rows are represented by L1;L2;.......;Lm in group by clause. (L1;L2;.......Lm)∩ (R1;R2; …...Rk) = ∅. We have tried to save SQL evolution semantics as possible. And also we have to make efficient evolution mechanisms. So we are proposing some rules.

- The GROUP BY clause is optional.
- If GROUP clause is present, there should not be a HAVING clause.
- The transposing BY clause is optional.
- When BY clause is included, horizontal aggregation reduces the vertical aggregation.
- Horizontal aggregation may combine with vertical aggregation or other horizontal aggregation on the same query.
- Till F does not change, horizontal aggregation can be freely combined.

4.3 SQL Code Generation

In this section, we discuss how to automatically generate efficient SQL code to evaluate horizontal aggregations. We start by discussing the structure of the result table and then query optimization methods to populate it. We will prove the three proposed evaluation methods produce the same result table F_H.

4.3.1 Locking and Table Definition

- **Locking**

In order to get a consistent query evaluation it is necessary to use locking [9], [19]. The main reasons are that any insertion into F during evaluation may cause inconsistencies:

- It can create extra columns in F_H, for a new combination of R1;R2; ...;Rk;
- It may change the number of rows of F_H, for a new combination of L1;L2...;L m;
- It may change actual aggregation values in F_H.

In other words the SQL statement becomes long transaction. Horizontal aggregation can operate on static database without consistency problem.

- **Result Table Definition**

Let the result table be F_H. As mentioned F_H has *d* aggregation columns, plus its primary key. The horizontal aggregation function H( ) returns not a single value, but a set of values for each group L1;L2;...;Lm. Therefore, the result table F_H must have as primary key, the set of grouping columns {L1;L2;...;Lm} and as non key columns all existing combinations of values R1;R2;...;Rk. We get the distinct value combinations of R1;R2;...;Rk using the following statement.

```
SELECT DISTINCT R1;R2;......;Rk
 FROM F;
```

Assume this statement returns a table with *d* distinct rows. Then each row is used to define one column to store an aggregation for one specific combination of dimension values. Table FH that has {L1;L2;...;Lm} as primary key and *d* columns corresponding to each distinct subgroup. Therefore, FH has *d* columns for data mining analysis and j + d columns in total, where each X_j corresponds to one aggregated value based on a specific R1;R2;...;Rk values combination.

4.4 Example

We are using the above some rules and created horizontal table. Assume we want to summarize sales information with one store per row for one year sales. In more detail, we need the sales amount broken down by day of the week, the number of transactions by store per month, the number of items sold by department and total sales. The result is shown in table 2.

5. QUERY EVALUATION mETHODS

Horizontal aggregation is evaluated by the following methods as defined:

5.1 SPJ Method

It is based on standard relational algebra operators (SPJ queries). The basic idea is to create one table with a vertical aggregation for each result column, and then join all those tables to produce another table. It is necessary to introduce an additional table F0 that will be outer joined with projected tables to get a complete result set. The optimized SPJ method code is follows:

```
INSERT INTO FH
SELECT F0.L1, F0.L2,....,F0.Lm,
            F1.A, F2 .A,......,Fn .A
```

```
FROM F0
LEFT OUTER JOIN F1
ON F0. L1= F1. L1 and. . . and F0. Lm= F1. Lm

LEFT OUTER JOIN F2
ON F0. L1= F2. L1 and. . . and F0. m= F2. Lm
 . . . .
LEFT OUTER JOIN Fn
ON F0. L1= Fn. L1 and. . . and F0. Lm= Fn. Lm
```

5.2 PIVOT Method

The pivot operator is a built-in operator which transforms row to columns. It internally needs to determine how many columns are needed to store the transposed table and it can be combined with the GROUP BY clause. Since this operator can perform transposition it can help in evaluating horizontal aggregation. The optimized PIVOT method SQL is as follows:

```
SELECT DISTINCT R1
FROM F; /*produces v1,………,vd*/
SELECT
        L1,L2,…,Lm
        ,v1,v2,….,vd
INTO FH
FROM (
        SELECT L1,L2,…..,Lm, R1,A
        FROM F) Ft
        PIVOT(
                V(A) FOR R1 in (v1,v2,…,vd)
 ) AS P;
```

Table2
A Multidimensional Data Set in Horizontal Layout, Suitable for Data Mining

StoreId	SalesAmt			CountTransactions			Columns			sales
	Mon	Tue …	Sun	Jan	Feb ..	Dec	Dairy	meat	product	
10	125	141	140	2011	1807	4200	54	87	112	25054
30	80	98	88	802	912	1632	42	35	174	13876
.										
.										
.										

5.3 CASE Method

It can be used in any statement or clause that allows a valid expression. The case statement returns a value selected from a set of values based on Boolean expression. The Boolean expression for each case statement has a conjunction of K equality comparisons. Query evaluation needs to combine the desired aggregation with "case" statement for each distinct combination of values of R1;R2……..,Rk. The optimized case method code is as follows:

```
SELECT DISTINCT R1,…..,Rk
 FROM Fv;
 INSERT INTO FH
 SELECT L1,L2,....,Lm
       ,sum(CASE WHEN R1=v11 and . . . Rk=vk1
                    THEN A ELSE null END)
 ....
      ,sum(CASE WHEN R1=v1n and . . . Rk=vkn
                    THEN A ELSE null END)
 FROM Fv
 GROUP BY L1,L2,. . .,Lm ;
```

6. Conclusion AND future work

In this paper we extended three aggregate functions such as CASE, SPJ and PIVOT. These are known as horizontal aggregations. We have achieved it by writing underlying constructs for each operator. When they are used, internally the corresponding construct gets executed and the resultant data set is meant for OLAP (Online Analytical Processing). In order to prepare real world datasets that are very much suitable for data mining operations, we explored horizontal aggregations by developing constructs in the form of operators such as CASE, SPJ and PIVOT. Instead of single value, the horizontal aggregations return a set of values in the form of a row. The result resembles a multidimensional vector. We have implemented SPJ using standard relational query operations. The CASE construct is developed extending SQL CASE. The PIVOT makes use of built in operator provided by RDBMS for pivoting data.
In future, this work can be extended to develop a more formal model of evaluation methods to achieve better results. Then we can also develop more complete I/O cost models.

REFERENCES

[1] Gray, B., et al., (1996), "Data cube: A relational aggregation operator generalizing group-by, cross-tab and subtota"l. In ICDE Conference, pages 152.159.

[2] E.F. Codd,(1979), "Extending the database relational model to capture more meaning". ACM TODS, 4(4):397.434.

[3] Muley, S., et al., (2013), "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.

[4] Blakeley, R., et al., (2008), ".NET database programmability and extensibility in Microsoft SQL Server". In Proc. ACM SIGMOD Conference, pages 1087.1098.

[5] C. Ordonez, (2006), "Integrating K-Means Clustering with a Relational DBMS Using SQL," IEEE Trans. Knowledge and Data Eng., vol. 18, no. 2, pp. 188-201.

[6] Wang, Z., et al., (2003), "ATLAS: A Small But Complete SQL Extension for Data Mining and Data Streams," Proc. 29th Int'l Conf. Very Large Data Bases (VLDB '03), pp. 1113- 1116.

[7] Sarawagi, S., et al', (1998), "Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '98), pp. 343-354.

[8] Witkowski, S.,et al., (2003), "Spreadsheets in RDBMS for OLAP," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '03), pp. 52-63.

[9] Garcia-Molina, J., et al., (2001) "Database Systems: The Complete Book", first ed. Prentice Hall.

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

[11] Bhargava, P., et al., (1995), "Hypergraph Based Reorderings of Outer Join Queries with Complex Predicates," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '95), pp. 304-315.

[12] Gray, A., et al., (1996), "Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross- Tab and Sub-Total," Proc. Int'l Conf. Data Eng., pp. 152-159.

[13] Graefe, U., et al., (1998), "On the Efficient Gathering of Sufficient Statistics for Classification from Large SQL Databases," Proc. ACM Conf. Knowledge Discovery and Data Mining (KDD '98), pp. 204-208.

[14] Clear, D., et al., (1999), "Non- Stop SQL/MX Primitives for Knowledge Discovery," Proc. ACM SIGKDD Fifth Int'l Conf. Knowledge Discovery and Data Mining (KDD '99), pp. 425-429.

[15] Cunningham, G., et al., (2004), "PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS," Proc. 13th Int'l Conf. Very Large Data Bases (VLDB '04), pp. 998- 1009.

[16] C. Ordonez, (2004), "Horizontal Aggregations for Building Tabular Data Sets," Proc. Ninth ACM SIGMOD Workshop Data Mining and Knowledge Discovery (DMKD '04), pp. 35-42.

[17] C. Ordonez, (2004), "Vertical and Horizontal Percentage Aggregations," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '04), pp. 866-871.

[18] Carlos Ordonez and Zhibo Chen, (2012), "Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining Analysis", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 4.

[19] Luo, J, et al., (2005), "Locking Protocols for Materialized Aggregate Join Views," IEEE Trans. Knowledge and Data Eng., vol. 17, no. 6, pp. 796-807.