Preparing Data Sets for the Data Mining Analysis using the Most Efficient Horizontal Aggregation Method in SQL

Jasna S
MTech Student
TKM College of engineering
Kollam

Manu J Pillai
Assistant Professor
TKM College of engineering
Kollam

ABSTRACT
A huge amount of time is needed for making the dataset for the data mining analysis because data mining practitioners required to write complex SQL queries and many tables are to be joined to get the aggregated result. The traditional SQL aggregations prepare the data sets in vertical layout that is; they return result on one column per aggregated group. But for the data mining project, the data set to be required in horizontal layout. In order to transform the data into suitable form the existing three horizontal aggregation methods are used. The existing method for evaluating horizontal aggregation are SPJ (select, project, join) method, CASE method and PIVOT method. The analysis become more efficient if the dataset obtained is in the horizontal form. The main aim is to identify the most efficient method from these three methods in terms of time and space complexity. So these methods are compared using large tables and identified that the CASE method is more efficient than SPJ and PIVOT method.

General Terms
SQL Queries, Data Base Management, Data Mining

Keywords
SQL Operators, Aggregate functions, Data Set Preparation

1. INTRODUCTION
Data mining refers to the finding of relevant and useful information from databases. A data mining project consists of several phases. The first phase is called the data preparation phase. The second phase involves the analysing of data sets using data mining algorithms. The third phase involves validating of results. The fourth phase involves deploying of statistical results on new data sets. The first phase involves the extracting data from multiple operational databases and from external sources, cleaning of data where unnecessary information is removed, and transforming data into suitable form for the task of data mining. For transforming the data, the aggregation in SQL is used. In SQL, the aggregation of data is done using the aggregate functions such as minimum, maximum, average, count and sum and the result is obtained in the vertical layout. By using this data set as such, the person that done data miners need to write large SQL queries to convert it into the appropriate form.

Most of the algorithms in the data mining require the dataset in the tabular form. This is the case in clustering, regression, classification etc. So the new set of functions called horizontal aggregation is used to get the dataset in horizontal layout. The three methods for evaluating horizontal aggregation are SPJ method, CASE method and PIVOT method.

As discussed so far, it is difficult for preparing data set for data mining purposes. To convert the dataset into suitable form the data practitioner has to write the complex SQL queries. The two main operations that are used in such SQL queries are join and aggregation. In this article the main focus is on aggregation. There are several aggregate functions in SQL. Each of these functions returns its result over a group of rows. But these aggregate functions are unable to prepare the suitable dataset for the data mining purposes. So a significant effort is needed for computing aggregation when they are required in horizontal layout. Due to these drawbacks, a new class of aggregation are required to get the suitable layout. This new class of functions are called horizontal aggregation. The existing CASE construct in SQL and the pivot operation are used to obtain the horizontal layout. There are three methods for evaluating the horizontal aggregation. These methods are used for preparing the data set for the data mining algorithms. The main focus of this paper is to find the most efficient method from these three methods in terms of time complexity and space complexity.

This article is organized as follows. Section II introduces definitions and motivating examples. Literature survey is discussed in section III. Section IV introduces horizontal aggregations. Three methods are used to evaluate horizontal aggregation. Section V presents the comparison of three horizontal aggregation methods. These methods are compared using large datasets and analyse its time complexity and space complexity. Section VI contains the conclusions and directions for future work.

2. DEFINITIONS
In this article, R is a table from which the aggregated data is required and whose cardinality is d. R_V (vertical) and R_H (horizontal) are used to denote the tables for vertical and horizontal aggregation.

Suppose there are t+v GROUP BY columns and the aggregate attribute is X. The result of the vertical aggregation contains t+v columns that becomes the primary key and is stored in R_V. The aim of horizontal aggregation is to obtain the result in R_H with n rows and t+p columns, where each of the p columns represents a unique combination of the m grouping columns. For that, a small syntax extension is required to the aggregate function call in a select statement.

2.1 Motivating Examples
Figure 1.shows an example. R is the base table. The result of vertical aggregation is stored in R_V and that of horizontal aggregation is stored in R_H.

SQL query is:

```
SELECT storied, salesday, sum(salesamt)

FROM R

GROUP BY storied, salesday;
```

The table R_V has six rows. But in R_H, it has only three rows. It is necessary to fill the field with null when there is no aggregated result.

Consider the analysis of sales details in various stores. Suppose there are 50 stores and stores are open 7 days a week.

Then the above query will give 350 rows. For these type of analysis the data mining algorithms should require storied as primary key and the remaining columns as non-key. It means that the data mining algorithms require the result in horizontal layout. Also, the data set in horizontal format can be analysed more efficient than data set in vertical format.

R

Storeid	Salesday	Salesamt	Billno
1	Monday	1500	101
1	Tuesday	2500	102
2	Monday	1800	101
2	Tuesday	1200	102
3	Tuesday	1400	101
3	Tuesday	1500	102
1	Monday	1200	103
2	Tuesday	1800	103
3	Wednesday	2000	103

R_V

Storeid	Salesday	Salesamt
1	Monday	2700
1	Tuesday	2500
2	Monday	1800
2	Tuesday	3000
3	Tuesday	2900
3	Wednesday	2000

R_H

Storeid	Salesmon	Salestue	Saleswed
1	2700	2500	Null
2	1800	3000	Null
3	Null	2900	2000

Fig.1: Example of Horizontal Aggregation.

3. LITERATURE SURVEY

The programming of the clustering algorithm with SQL queries is explored in [2], which shows that the horizontal layout of the data set enables simpler SQL queries. The SPJ method proved that the horizontal aggregations can be evaluated with relational algebra operators, exploiting outer joins, showing this work is connected to traditional query optimization [1]. The optimization of SQL queries with outer joins can be possible. The outer join operations are optimized by reordering the operations and using transformation rules [3]. In the context of data cube computations, the importance of producing an aggregation table with a cross-tabulation of aggregated values is identified in [4]. For the classification algorithms, the scalability of algorithms to large databases can be achieved by observing the most algorithms are driven by a set of sufficient statistics that are significantly smaller than the data. To unpivot the table that produce several rows in the vertical layout and the decision trees are computed are proposed in [5]. The unpivot operator produces many rows in the table with attribute-value pairs for each input row and it is an inverse process of horizontal aggregations.

Different SQL primitive operators for transforming data sets for data mining were introduced in [6]; the most similar needed for horizontal aggregation is an operator to transpose a table, based on the chosen column. The TRANSPOSE operator [7] is same as the unpivot operator, that produce several rows for one input row. The important difference between the PIVOT and TRANSPOSE operators is that the TRANSPOSE operator allows two or more columns to be transposed in the same query, and also reducing the number of table scans. Therefore both the UNPIVOT and TRANSPOSE is complementary to each other with respect to horizontal aggregations.

Horizontal aggregations are related to horizontal percentage aggregations in [8]. The differences between both approaches are that percentage aggregations require aggregating at the two grouping levels and also require dividing numbers and need taking care of numerical issues (e.g. dividing by zero). Horizontal percentage aggregations provide a starting point to extend standard aggregations to return results in horizontal form. Optimizing vertical percentage queries with different groupings in each term seems similar to association mining using bottom-up search. Reducing the number of comparisons needed to compute horizontal percentage aggregations may lead to changing the algorithm to parse and evaluate a set of aggregations when they are combined with "case" statements with disjoint conditions. A set of percentage queries on the same table may be efficiently evaluated using shared summaries. Horizontal aggregation is first proposed on the [9], in which only the SPJ method and CASE method are proposed. Also in [10] another method called PIVOT method is proposed in addition to SPJ and CASE. In this, the SQL code for evaluating horizontal aggregation is done after the vertical aggregation is performed. But in this article, the horizontal aggregation is performed directly from the table.

4. HORIZONTAL AGGREGATION

A new class of aggregations called horizontal aggregations are used to obtain result in horizontal layout. Since the existing SQL aggregation produce tables in vertical layout. The SQL code generation for horizontal aggregation is discussed follows.

4.1 SQL Code Generation

To obtain horizontal aggregation, a small syntax extension to aggregate functions called in a select statement. This query will produce a table with y+1 column. The data are grouped based on the unique combination of values $n_1,n_2,\ldots,n_y$, and one aggregated value per group. To execute this query the query optimizer takes three input parameters. 1) Input table, R

2) list of grouping attributes $n_1,n_2,\ldots,n_t$ 3)attribute to aggregate(X). The aim of horizontal aggregation is to transpose aggregate attribute X by a column subset of $n_1,n_2,\ldots,n_t$. Assume that such subset is $m_1,m_2,\ldots\ldots,m_v$, where v<y. Or the GROUP BY list can be partitioned into two sub lists: one list contain $n_1,n_2,\ldots,n_t$ and another list contain $m_1,m_2,\ldots\ldots,m_v$ to transpose aggregated values. So, in a horizontal aggregation there are four input parameters to generate SQL code.

1) Input table F,
2) The list of GROUP BY attribute $n_1,n_2,\ldots,n_t$.
3) The attribute to aggregate(X).
4) The list of transposed columns $m_1,m_2,\ldots\ldots,m_v$

4.2 Horizontal Evaluation Methods

The existing three methods are used to evaluate horizontal aggregations. They are SPJ method, CASE method and PIVOT method. The SPJ method is based on relational operations in SQL. The CASE method uses the CASE construct in SQL. The PIVOT method uses the pivot built-in operator, which transforms rows to columns.

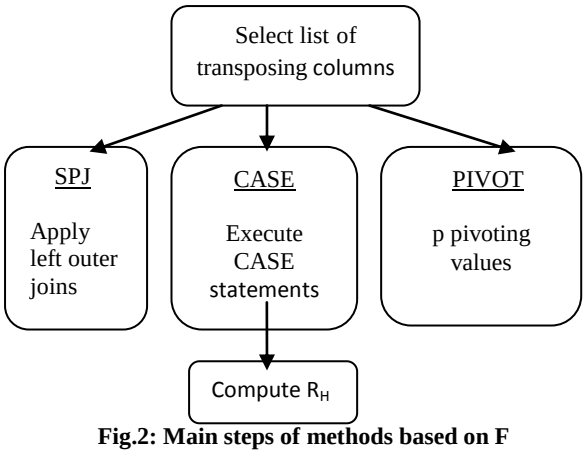

Fig.2: Main steps of methods based on F

4.2.1 SPJ Method

The SPJ method is based on the relational operators such as select, join and project. The idea is to create the aggregated result for each values of the attributes that to be transposed and the result is stored in individual tables say R_i(i=1,…..p),for each distinct value of $m_1,m_2,\ldots\ldots,m_v$. These tables are then left outer joined with the table R_0,that contain the distinct values of $n_1,n_2,\ldots n_t$. Left outer join is used to deal with missing information. This operation is performed by taking all tuples in the left relation that did not match with any tuple in the right relation and set the attributes from the right relation to null. Horizontal aggregation queries can be evaluated by aggregating data directly from R based on the distinct values of grouping attributes. So the distinct values of m1,m2,….m_v are to identified, that define the matching boolean expression for result columns. These results are then transposed to produce R_H. Agg() is the standard vertical aggregation that has aggregate attribute as argument.
The SQL code for generating R_0:

SELECT DISTINCT $n_1,n_2,\ldots,n_t$
FROM R

Table R_0 identifies the number of rows in the final result and it become the primary key.

Tables $R_1,R_2,\ldots R_l$ contain the individual aggregations for each combination of $m_1,m_2,\ldots..,m_V$. { $n_1,n_2,\ldots,n_t$ } become the key of R_I.

SQL code for generating R_I,

SELECT $n_1,n_2,\ldots,n_t$, Agg(X)
FROM R
WHERE $m_1=v_{1I}$ AND……AND m_{vI}
GROUP BY $n_1,n_2,\ldots,n_t$

The final SQL code for SPJ method

SELECT $R_0.n_1,R_0.n_2,\ldots\ldots,R_0.n_t$,

$R_1.X, R_2.X,\ldots\ldots, R_l.X$

FROM R_0

LEFT OUTER JOIN R_1

ON $R_0.n_1 = R_1.n_1$ and…and $R_0.n_t = R_1.n_t$

LEFT OUTER JOIN R_2

ON $R_0.n_1 = R_2.n_1$ and… and $R_0.n_t = R_2.n_t$

…

LEFT OUTER JOIN R_l

ON $R_0.n_1 = R_d.n_1$ and… and $R_0.n_t = R_l.n_t$;

4.2.2 CASE Method

In this method, the CASE construct available in SQL is used. The case statement returns a value selected from a set of values based on Boolean expressions. Horizontal aggregation queries can be evaluated by aggregating data directly from R based on the distinct values of grouping attributes. So the distinct values of $m_1,m_2,\ldots.m_v$ are to identified, that define the matching boolean expression for result columns. These results are then transposed to produce R_H. If there are no qualifying rows for the specific aggregate group then result must set to null as in the case of SPJ method. Agg() in the SQL code is the standard SQL aggregation that contain the case statement as the argument.

SQL Code for getting each distinct values of transposing columns,

SELECT DISTINCT $m_1,m_2,\ldots\ldots,m_v$

FROM R;

SQL Code for getting R_H ,

SELECT $n_1,n_2,\ldots,n_t$

,Agg(CASE WHEN $m_1 = v_{11}$ and …and $m_v = v_{v1}$

THEN X ELSE null END)…….

…..

,Agg(CASE WHEN $m_1 = v_{1l}$ and …and $m_v = v_{vl}$

THEN X ELSE null END)

FROM R

GROUP BY $n_1,n_2,\ldots,n_t$;

4.2.3 PIVOT Method

This method uses the built-in PIVOT operator in a Commercial DBMS, which transforms rows to columns. This operator can perform transposition, so it can be used for evaluating the horizontal aggregation. This method needs to determine how many columns are required to store the transposed table and it is combined with the GROUP BY clause. Horizontal aggregation queries can be evaluated by aggregating data directly from R based on the distinct values of grouping attributes. So the distinct values of $m_1,m_2,\ldots m_v$ are to identified, that define the matching boolean expression for result columns. These results are then transposed to produce R_H. If there are no qualifying rows for the specific aggregate group then result must set to null as in the case of SPJ method.

SQL code for getting the distinct values of transposing columns,

SELECT DISTINCT m_1

FROM R;

SQL code for generating R_H

SELECT $n_1,n_2,\ldots,n_t$

$,v_1, v_2,\ldots,v_l$

INTO R_H

FROM (SELECT $n_1,n_2,\ldots,n_t,m_1$,X FROM R) R_u

PIVOT(

Agg (X) FOR m_1 in ($v_1, v_2,\ldots, v_l$)

) AS P;

4.3 Advantages

The proposed horizontal aggregation has several advantages. By using the horizontal aggregation the manual work in preparation of data is reduced because there is no need to write the long SQL code by the data mining practitioner. Another advantage is, the SQL queries are written by person who has well knowledge about DBMS. So it is well efficient than the queries written by an end user. Next advantage is that the datasets are created in less time. By using the most efficient method that is, the CASE method reduces the time and space needed for processing the SQL query.

5. EXPERIMENTAL EVALUATION

The SQL code generator was programmed in the C#.NET language and the SQL server 2005 was used that provide the build-in operators in SQL. The CASE, PIVOT operators are available in the SQL language implementation by the DBMS. The horizontal aggregation methods are evaluated with a real data set and a synthetic data set. The real data set came from the UCI Machine Learning Repository. This data set contained a collection of records from the US Census. The synthetic data set was generated as follows. The attributes are generated whose cardinalities reflect a typical transaction table from a data warehouse. Analyse SQL queries having only one horizontal aggregation that have different grouping and horizontalization columns. Each method is executed with these datasets and return the average time in milliseconds. Different column combinations are taken to get different d and n.

5.1 Analysis

The time complexity and the space complexity of each method are analysed. Consider that $N=|R|$, $n=|R_H|$ and l is the dataset dimensionality.

In the SPJ method, the p tables are to generated for each distinct values of $m_1,\ldots..,m_v$. These tables are then joined with the table R_0. So the time complexity of SPJ method is high because of generating p tables and the left outer join operation. Also the space is needed for storing p tables and for R_0 table. And as the dimensionality increases, the space complexity also increases.

In the CASE method the time complexity and space complexity is less compared to SPJ method. The v comparisons are needed to get the aggregated result and there is no intermediate tables are needed to get the aggregated result. So the space complexity is less. As the comparisons are done in parallel, the time required for computing the case statement is less. So the time complexity of CASE method is less.

In the PIVOT method, the parallel step does not appear when evaluating the query. So the time required for executing the PIVOT method is more compared to the CASE method. In the PIVOT method the space is needed for the intermediate table that stores the grouping attributes and the aggregate attribute. From this table the values of the transposing columns are taken. Due to this table, the space complexity is more compared to the CASE method.

By analysing the three methods, the SPJ method is the slowest method. Also the CASE method is more efficient than the PIVOT method in terms of time complexity. The time complexity of each method grows as the n grows. Also, the high variation is occurring at the SPJ method. Time taken for executing each method is shown in table 1.

The most important difference between CASE and PIVOT method is that the CASE method has a parallel step which allows evaluating aggregations in parallel. This parallelism does not appear when PIVOT operator evaluates the query from R. SPJ method takes more time than PIVOT and CASE method because it has more operations to be performed. Tables are to be generated for each distinct combination of transposing columns. Also, the table R_0 to be generated that contain distinct values of the grouping columns. Then left outer join is to be performed between these tables. SPJ method also requires more space than CASE and PIVOT. From these observations it is found that the CASE method is more efficient than CASE and PIVOT method.

Table 1. Comparison of three horizontal evaluation methods

Number of records	Dimensionality	Time in milliseconds	Method
250	25	182	CASE
250	25	243	PIVOT

250	25	1350	SPJ
150	12	156	CASE
150	12	202	PIVOT
150	12	851	SPJ
100	7	124	CASE
100	7	140	PIVOT
100	7	355	SPJ

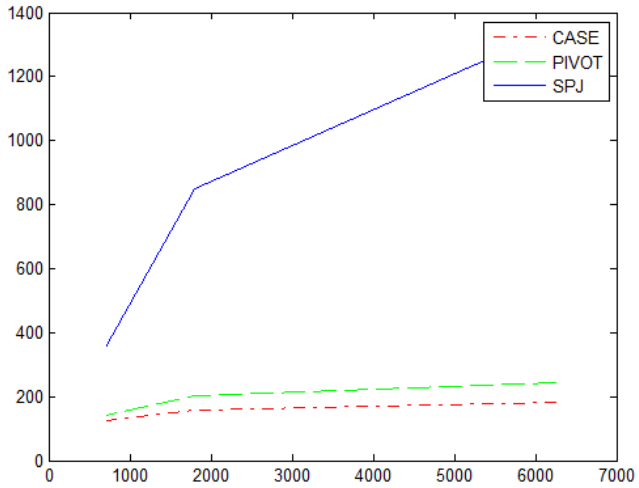

Fig 3: Time complexity of three horizontal aggregation methods

6. CONCLUSION

A new class of aggregate functions in SQL called horizontal aggregation is used that helps for making datasets for the data mining projects. These functions are used for creating data sets in horizontal layout, because most of the data mining algorithms require datasets in horizontal layout. Mainly, the existing SQL aggregations return results in one column per aggregated group. But in horizontal aggregation, it returns a set of numbers instead of a single number for each group. Three query evaluation methods are proposed. The first method focuses on the relational operators in SQL. The second method focuses on the SQL case construct. The third method focuses on the PIVOT built-in operator in a commercial DBMS. These three methods are then analysed using the large dataset and observe the time and space complexity of each method. By analysing, it is observed that the CASE method is the most efficient method.

Horizontal aggregation produces tables with fewer rows but with more columns. So the traditional query optimization techniques are inappropriate for the new class of aggregations. So the next plan is to develop the most appropriate query optimization technique for the horizontal aggregation.

7. REFERENCES

[1] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 1st edition, 2001.

[2] 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.

[3] C. Galindo-Legaria and A. Rosenthal. Outer join simplification and reordering for query optimization. *ACM TODS*, 22(1):43.73, 1997.

[4] 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.

[5] G. Graefe, U. Fayyad, and S. Chaudhuri. On the efficient gathering of sufficient statistics for classification from large SQL databases. In p*roc.ACM KDD Conference*, pages 204.208, 1998.

[6] J. Clear, D. Dunn, B. Harvey, M.L. Heytens, and P. Lohman. Non-stop SQL/MX primitives for knowledge discovery. In *ACM KDD Conference*, pages 425.429, 1999.

[7] C. Ordonez. Vertical and horizontal percentage aggregations. In *Proc.ACM SIGMOD Conference*, pages 866.871, 2004.

[8] C. Cunningham, G. Graefe, and C.A. Galindo-Legaria. PIVOT and UNPIVOT: Optimization and execution strategies in an RDBMS. In *Proc. VLDB Conference*, pages 998.1009, 2004.

[9] C. Ordonez. Horizontal aggregations for building tabular data sets. In *Proc. ACM SIGMOD Data Mining and Knowledge Discovery Workshop*, pages 35.42, 2004.

[10] C. Ordonez, Zhibo Chen. Horizontal aggregations in SQL to prepare Data Sets for Data Mining Analysis. IEEE Transactions on Knowledge and Data Engineering (TKDE), 2012.