RESEARCH ARTICLE OPEN ACCESS

Dataset Preparation in Datamining Analysis Using Horizontal Aggregations

Shine V.
M. Tech Software Engineering Sarabhai Institute of Science & Technology,shineisv736@gmail.com

Abstract
Data mining an interdisciplinary subfield of computer science, is the computational process of discovering patterns in large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics, and database systems. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use. Preparing a data set for analysis is generally the most time consuming task in a data mining project, requiring many complex SQL queries, joining tables, and aggregating columns. Existing SQL aggregations have limitations to prepare data sets because they return one column per aggregated group. Horizontal aggregations build data sets with a horizontal de normalized layout, which is the standard layout required by most data mining algorithms. In this paper proposed three fundamental methods to evaluate horizontal aggregations: 1. CASE: Exploiting the programming CASE construct; 2.SPJ: Based on standard relational algebra operators; 3. PIVOT: Using the PIVOT operator, which is offered by some DBMSs
Keywords: CASE, SPJ, PIVOT

I. INTRODUCTION

In a relational database, especially with normalized tables, a significant effort is required to prepare a summary data set that can be used as input for a data mining or statistical algorithm. Most algorithms require as input a data set with a horizontal layout, with several records and one variable or dimension per column. That is the case with models like clustering, classification, regression. This paper introduces a new class of aggregate functions that can be used to build data sets in a horizontal layout (de normalized with aggregations), automating SQL query writing and extending SQL capabilities. building a suitable data set for data mining purposes is a time-consuming task. This task generally requires writing long SQL statements or customizing SQL code if it is automatically generated by some tool. There are two main ingredients in such SQL code: joins and aggregations; focus on the second one. The most widely known aggregation is the sum of a column over groups of rows. Some other aggregations return the average, maximum, minimum, or row count over groups of rows. There exist many aggregation functions and operators in SQL. Unfortunately, all these aggregations have limitations to build data sets for data mining purposes. The main reason is that, in general, data sets that are stored in a relational database (or a data warehouse) come from Online Transaction Processing (OLTP) systems where database schemas are highly normalized. But data mining, statistical, or machine learning algorithms generally require aggregated data in summarized form. Based on current available functions and clauses in SQL, a significant effort is required to compute aggregations when they are desired in a cross tabular (horizontal) form, suitable to be used by a data mining algorithm. Such effort is due to the amount and complexity of SQL code that needs to be written, optimized, and tested. There are further practical reasons to return aggregation results in a horizontal (cross-tabular) layout. Standard aggregations are hard to interpret when there are many result rows, especially when grouping attributes have high cardinalities. To perform analysis of exported tables into spreadsheets it may be more convenient to have aggregations on the same group in one row (e.g., to produce graphs or to compare data sets with repetitive information). OLAP tools generate SQL code to transpose results Transposition can be more efficient if there are mechanisms combining aggregation and transposition together.

In this paper proposed a new class of aggregate functions that aggregate 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 (somewhat similar to a multidimensional vector), instead of a single value per row.

II. ADVANTAGES

The proposed horizontal aggregations provide several unique features and advantages. First, they represent a template to generate SQL code from a data mining tool. Such SQL code automates writing SQL queries, optimizing them, and testing them for correctness. This SQL code reduces manual work in the data preparation phase in a data mining project. Second, since SQL code is automatically generated it is likely to be more efficient than SQL code written by an end user. For instance, a person who does not know SQL well or someone who is not familiar with the database schema (e.g., a data mining practitioner). Therefore, data sets can be created in less time. Third, the data set can be created entirely inside the DBMS. In modern database environments, it is common to export de normalized data sets to be further cleaned and transformed outside a DBMS in external tools (e.g., statistical packages). Unfortunately, exporting large tables outside a DBMS is slow, creates inconsistent copies of the same data and compromises database security. Therefore, provide a more efficient, better integrated and more secure solution compared to external data mining tools. 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.

III. Horizontal Aggregations

This paper introduced a new class of aggregations that have similar behavior to SQL standard aggregations, but which produce tables with a horizontal layout. In contrast, 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.

IV. SQL CODE GENERATION

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 $L_1 \dots L_m$ from $D_1. \dots .D_p$.

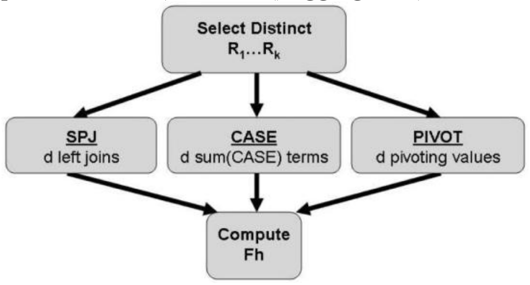
SELECT $L_1 \dots L_m$, sum(A)
FROM F
GROUP BY $L_1 \dots L_m$;

This aggregation query will produce a wide table with m + 1 columns (automatically determined), with one group for each unique combination of values $L_1 \dots L_m$ and one aggregated value per group (sum(A) in this case). In order to evaluate this query the query optimizer takes three input parameters: 1) the input table F, 2) the list of grouping columns $L_1 \dots L_m$, 3) the column to aggregate (A).

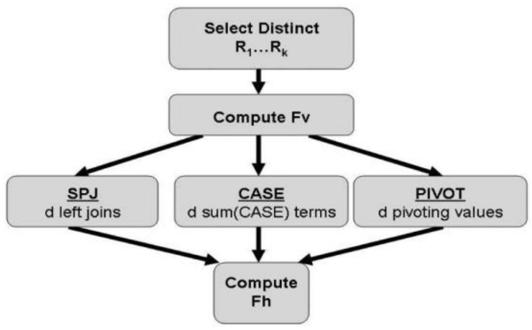
The basic goal of a horizontal aggregation is to transpose (pivot) the aggregated column A by a column subset of $L_1 \dots ; L_m$; for simplicity assume such subset is $R_1 \dots R_k$ where $k < m$. In other words, partition the GROUP BY list into two sublists: one list to produce each group (j columns $L_1 \dots ;L_j$) and another list (k columns $R_1 \dots R_k$) to transpose aggregated values, where $\{L1 \dots L_j\} \cap \{R_1 \dots R_k\} = \varnothing$. Each distinct combination of $\{R_1 \dots R_k\}$ will automatically produce an output column. In particular, if k = 1 then there are $|\pi_{R1}(F)|$columns (i.e., each value in R_1 becomes a column storing one aggregation). Therefore, in a horizontal aggregation there are four input parameters to generate SQL code:1. the input table F, 2. the list of GROUP BY columns $L_1 \dots L_j$, 3. the column to aggregate (A), 4. the list of transposing columns $R_1 \dots R_k$.

V. QUERY EVALUATION METHODS

This paper proposed three methods to evaluate horizontal aggregations. The first method relies only on relational operations. That is, only doing select, project, join, and aggregation queries; call it the SPJ method. The second form relies on the SQL "case" construct; call it the CASE method. Each table has an index on its primary key for efficient join processing. Additional indexing mechanisms to accelerate query evaluation is not considered. The third method uses the built-in PIVOT operator, which transforms rows to columns (e.g., transposing). Figsshow an overview of the main steps to be explained below (for a sum() aggregation).



Main steps of methods based on F (unoptimized



Main steps of methods based on FV (optimized

a) SPJ Method

The SPJ method is interesting from a theoretical point of view because it is based on relational operators only. The basic idea is to create one table with a vertical aggregation for each result column, and then join all those tables to produce F_H. Then aggregate from F into d projected tables with d Select-Project-Join-Aggregation queries (selection, projection, join, aggregation). Each table F_I corresponds to one subgrouping combination and has $\{L_1 \ldots L_j\}$as primary key and an aggregation on A as the only nonkey column. It is necessary to introduce an additional table F_0, that will be outer joined with projected tables to get a complete result set. Proposed two basic sub strategies to compute F_H. The first one directly aggregates from F. The second one computes the equivalent vertical aggregation in a temporary table F_V grouping by $L_1 \ldots L_j;R_1 \ldots R_k$. Then horizontal aggregations can be instead computed from F_V , which is a compressed version of F, since standard aggregations are distributive.

b) CASE Method

For 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. From a relational database theory point of view this is equivalent to doing a simple projection/aggregation query where each nonkey value is given by a function that returns a number based on some conjunction of conditions. Here proposed two basic sub strategies to compute F_H. In a similar manner to SPJ, the first one directly aggregates from F and the second one computes the vertical aggregation in a temporary table F_V and then horizontal aggregations are indirectly computed from $F_{V.}$

Now presents the direct aggregation method. Horizontal aggregation queries can be evaluated by directly aggregating from F and transposing rows at the same time to produce F_H. First,get the unique combinations of $R_1 \ldots R_k$ that define the matching Boolean expression for result columns. The SQL code to compute horizontal aggregations directly from F is as follows: observe V() is a standard (vertical) SQL aggregation that has a "case" statement as argument. Horizontal aggregations need to set the result to null when there are no qualifying rows for the specific horizontal group to be consistent with the SPJ method and also with the extended relational model.

c) PIVOT Method

Consider the PIVOT operator which is a built-in operator in a commercial DBMS. Since this operator can perform transposition it can help evaluating horizontal aggregations. The PIVOT 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.

VI. CONCLUSION

Introduced a new class of extended aggregate functions, called horizontal aggregations which helps in 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 multidimensional vector. Proposed an abstract, but minimal, extension to SQL standard aggregate functions to compute horizontal aggregations which just requires specifying sub grouping columns inside the aggregation function call. From a query optimization perspective, 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. The SPJ method is important from a theoretical point of view because it is based on select, project, and join (SPJ) queries. The CASE method is most important contribution. It is in general the most efficient evaluation method and it has wide applicability since it can be programmed combining GROUP-BY and CASE statements. The three methods produce the same result. It is not possible to evaluate horizontal aggregations using standard SQL without either joins or "case" constructs using standard SQL operators.

REFERENCES

[1] Nisha S.,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, May 2012
[2] G. Bhargava, P. Goel, and B.R. Iyer, "Hypergraph Based Reorderings of Outer

Join Queries with Complex Predicates," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '95), pp. 304-315, 1995.

[3] J.A. Blakeley, V. Rao, I. Kunen, A. Prout, M. Henaire, and C. Kleinerman, ".NET Database Programmability and Extensibility in Microsoft SQL Server," Proc. ACM SIGMOD Int'l Conf.Management of Data (SIGMOD '08), pp. 1087-1098, 2008.

[4] V.Pradeep Kumar, Dr.R.V.Krishnaiah, "Horizontal Aggregations in SQL to Prepare Data Sets for DataMining Analysis", International Journal of Advanced Research in Computer Science and Software Engineering Volume 6, Issue 5 (Nov. - Dec. 2012).

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

[6] G. Graefe, U. Fayyad, and S. Chaudhuri, "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, 1998.