Abstract

Preparing Reports and Dataset are difficult task in data mining. Our 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 denormalized 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, we 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. We propose three fundamental methods to evaluate horizontal aggregations: CASE: Exploiting the programming CASE construct; SPJ: Based on standard relational algebra operators (SPJ queries); PIVOT: Using the PIVOT operator, which is offered by some DBMSs. Experiments with large tables compare the proposed query evaluation methods. Our CASE method has similar speed to the PIVOT operator and it is much faster than the SPJ method. In general, the CASE and PIVOT methods exhibit linear scalability, whereas the SPJ method does not.

Keywords

Aggregation, Data Preparation, Pivoting

I. Introduction

Data mining involves the use of sophisticated data analysis tools to discover previously unknown, valid patterns and relationships in large data sets [1]. These tools can include statistical models, mathematical algorithms, and machine learning methods (algorithms that improve their performance automatically through experience, such as neural networks or decision trees). Consequently, data mining consists of more than collecting and managing data, it also includes analysis and prediction. Data mining can be performed on data represented in quantitative, textual, or multimedia forms. Data mining applications can use a variety of parameters to examine the data. They include association (patterns where one event is connected to another event, such as purchasing a pen and purchasing paper), sequence or path analysis (patterns where one event leads to another event, such as the birth of a child and purchasing diapers), classification (identification of new patterns, such as coincidences between duct tape purchases and plastic sheeting purchases), clustering (finding and visually documenting groups of previously unknown facts, such as geographic location and brand preferences), and forecasting (discovering patterns from which one can make reasonable predictions regarding future activities, such as the prediction that people who join an athletic club may take exercise classes.

While data mining products can be very powerful tools, they are not self-sufficient applications. To be successful, data mining requires skilled technical and analytical specialists who can structure the analysis and interpret the output that is created. Consequently, the limitations of data mining are primarily data or personnel related, rather than technology-related.

Although data mining can help reveal patterns and relationships, it does not tell the user the value or significance of these patterns. These types of determinations must be made by the user. Similarly, the validity of the patterns discovered is dependent on how they compare to “real world” circumstances. For example, to assess the validity of a data mining application designed to identify potential terrorist suspects in a large pool of individuals, the user may test the model using data that includes information about known terrorists. However, while possibly reaffirming a particular profile, it does not necessarily mean that the application will identify a suspect whose behavior significantly deviates from the original model.

Another limitation of data mining is that while it can identify connections between behaviors and/or variables, it does not necessarily identify a causal relationship. For example, an application may identify that a pattern of behavior, such as the propensity to purchase airline tickets just shortly before the flight is scheduled to depart, is related to characteristics such as income, level of education, and Internet use. However, that does not necessarily indicate that the ticket purchasing behavior is caused by one or more of these variables. In fact, the individual’s behavior could be affected by some additional variable(s) such as occupation (the need to make trips on short notice), family status (a sick relative needing care), or a hobby (taking advantage of last minute discounts to visit new destinations).

The need to understand large, complex, information-rich data sets is common to virtually all fields of business, science, and engineering. In the business world, corporate and customer data are becoming recognized as a strategic asset. The ability to extract useful knowledge hidden in these data and to act on that knowledge is becoming increasingly important in today’s competitive world. The entire process of applying a computer-based methodology, including new techniques, for discovering knowledge from data is called data mining.

Horizontal aggregation is new class of function to return aggregated columns in a horizontal layout. Most algorithms require datasets with horizontal layout as input with several records and one variable or dimensions per columns. Managing large data sets without DBMS support can be a difficult task. Trying different subsets of data points and dimensions is more flexible, faster and easier to do inside a relational database with SQL queries than outside with alternative tool. Horizontal aggregation can be performed by using operator, it can easily be implemented.
inside a query processor, much like a select, project and join. PIVOT operator on tabular data that exchange rows, enable data transformations useful in data modeling, data analysis, and data presentation.

There are many existing functions and operators for aggregation in Structured Query Language. The most commonly used aggregation is the sum of a column and other aggregation operators return the average, maximum, minimum or row count over groups of rows. All operations for aggregation have many limitations to build large data sets for data mining purposes. Database schemas are also highly normalized for On-Line Transaction Processing (OLTP) systems where data sets that are stored in a relational database or data warehouse. But data mining, statistical or machine learning algorithms generally require aggregated data in summarized form.

Data mining algorithm requires suitable input in the form of cross tabular (horizontal) form, significant effort is required to compute aggregations for this purpose. Such effort is due to the amount and simplicity of SQL code which needs to be written, optimized and tested.

Data aggregation is a process in which information is gathered and expressed in a summarized form, and which is used for purposes such as statistical analysis. A common aggregation purpose is to get more information about particular groups based on specific variables such as age, name, phone number, address, profession, or income. Most algorithms require input as a data set with a horizontal layout, with several records and one variable or dimension per column. That technique is used with models like clustering, classification, regression and PCA. Dimension used in data mining technique are point dimension.

Preparing a data set for analysis is generally the most time consuming task in a datamining 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. In general, a significant manual efforts is required to build data sets, where a horizontal layout is required. We propose simple, yet powerful methods to generate SQL code to return aggregated columns in a horizontal tabular layout, returning a set of numbers instead of one number per row. This new class of functions is called horizontal aggregations. Horizontal aggregations build data sets with a horizontal denormalized layout (e.g. point-dimension, observation-variable, instance-feature), which is the standard layout required by most data mining algorithms. We propose three fundamental methods to evaluate horizontal aggregations: CASE: Exploiting the programming CASEconstruct; SPJ: Based on standard relational algebra operators (SPJ queries); PIVOT: Using the PIVOT operator, which is offered by some DBMSs. In a relational database, especially with normalized tables, a significant effort is required to prepare a summary data set which can be used as input for a data mining or statistical algorithm [15, 17]. Most algorithms require input as 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 and PCA; consult [10, 15]. Each research discipline uses different terminology to describe the data set. In data mining the common terms are point-dimension.

Statistics literature generally uses observation-variable. Machine learning research uses instance-feature. This article introduces a new class of aggregate functions that can be used to build data sets in an horizontal layout (denormalized with aggregations), automating SQL query writing and extending SQL capabilities. We show evaluating horizontal aggregations is a challenging and interesting problem and we introduce alternative methods and optimizations for their efficient evaluation 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 [16]; we 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 On-Line 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 transposes results (sometimes called PIVOT [5]). Transposition can bemoore efficient if there are mechanisms combining aggregation and transposition together.

There exist many proposals that have extended SQL syntax. The closest data mining problem associated to OLAP processing is association rule mining [18]. SQL extensions to define aggregate functions for association rule mining are introduced in [19]. 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. Programming a clustering algorithm with SQL queries is explored in [14], which shows a horizontal layout of the data set enables easier and simpler SQL queries. Alternative SQL extensions to perform spreadsheet-like operations were introduced in [20]. 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. The PIVOT and CASE methods avoid joins as well.

II. Existing System

1. 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.

2. To 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 parts in such SQL code: joins and aggregations.

3. 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.
4. The main reason is that, in general, data sets that are stored in a relational database (or a data warehouse) come from On-Line 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.

5. 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.

III. Proposed System

We now turn our attention to a small syntax extension to the SELECT statement, which allows understanding our proposal in an intuitive manner. We must point out the proposed extension represents non-standard SQL because the columns in the output table are not known when the query is parsed. We assume F does not change while a horizontal aggregation is evaluated because new values may create new result columns. Conceptually, we extend standard SQL aggregate functionsm with a “transposing” BY clause followed by a list of columns (i.e. R1, . . ., Rk), to produce a horizontal set of numbers instead of one number. Our proposed syntax is as follows.

Assume we want to summarize sales information with one store per row for one year of 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 following query in our extended SELECT syntax provides the desired data set, by calling three horizontal aggregations.

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 heterogeneous data sets into smaller homogeneous subsets that can be easily managed, separately modeled and analyzed. To create datasets for data mining related works, efficient and summary of data are needed. For that this proposed system collect particular needed attributes from the different fact tables and displayed columns in order to create date in horizontal layout. 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. Lm from D1.Dp.

We propose three fundamental methods to evaluate horizontal aggregations: CASE, SPJ, PIVOT.

A. SPJ Method

The SPJ method is based on only relational operators. The basic concept in SPJ method is to build a table with vertical aggregation for each resultant column. To produce Horizontal aggregation FH system must join all those tables. There are two sub-strategies to compute Horizontal aggregation: First strategy includes direct calculation of aggregation from fact table. Second one compute the corresponding vertical aggregation and store it in temporary table FV grouping by LE1... LEi, R11, ..., Rj then FH can be computed from FV. To get FH system need n left outer join with n+1 table so that all individual aggregations are properly assembled as a set of n dimensions for each group. Null should be set as default value for groups with missing combinations.

To get FH system need n left outer join with n+1 table so that all individual aggregations are properly assembled as a set of n dimensions for each group. Null should be set as default value for groups with missing combinations for a particular group.

B. CASE Method

In SQL build-in “case” programming construct are available, it returns a selected value rather from a set of values based on Boolean expression. Queries for FH can be evaluated by performing direct aggregation form fact table F and at the same time rows are transposing to produce the FH. First, we need to get the unique combinations of R1 . . ., Rk 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 [4].

**C. PIVOT Method**

We 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.

The basic syntax to exploit the PIVOT operator to compute a horizontal aggregation assuming one BY column for the right key columns (i.e. k = 1) is as follows:

```
SELECT DISTINCT R
FROM F; /* produces v_1, v_2, ..., v_d */
SELECT L_1, L_2, ..., L_j, v_1, v_2, ..., v_d
INTO F;
FROM F;
PIVOT (V(A) FOR R_1 in (v_1, v_2, ..., v_d)) AS P;
```

The SPJ method code is as follows (computed from F):

```
/* SPJ method */
INSERT INTO F1
SELECT D1, sum(A) AS A
FROM F
WHERE D2='X'
GROUP BY D1;
INSERT INTO F2
SELECT D1, sum(A) AS A
FROM F
WHERE D2='Y'
GROUP BY D1;
```

The CASE Method Code is as Follows (Computed from F):

```
/* CASE method */
INSERT INTO FH
SELECT
D1, SUM(CASE WHEN D2='X' THEN A ELSE null END) as D2_X,
SUM(CASE WHEN D2='Y' THEN A ELSE null END) as D2_Y
FROM F
GROUP BY D1;
```

Finally, the PIVOT Method SQL is as Follows (Computed from F):

```
/* PIVOT method */
INSERT INTO FH
SELECT
D1, [X] as D2_X,
[Y] as D2_Y
FROM (SELECT D1, D2, A FROM F
) as p
PIVOT (
SUM (A)
FOR D2 IN ([X], [Y])
) as pvt;
```

**D. Optimized Performance**

Not only multidimensional issues, but all types of processing can benefit from enhanced aggregation facilities. Transaction processing, financial and manufacturing systems—all of these generate large numbers of production reports needing substantial system resources. Improved efficiency when creating these reports will reduce system load. In fact, any computer process that aggregates data from details to higher levels will benefit from optimized aggregation performance. These extensions provide aggregation features and bring many benefits, including:

- Simplified programming requiring less SQL code for many tasks.
- Quicker and more efficient query processing.
- Reduced client processing loads and network traffic because
aggregation work is shifted to servers.

- Opportunities for caching aggregations because similar queries can leverage existing work.

IV. Experimental Results

Figure shows the impact of increasing the size of the fact table (N). The left plot analyzes a small FH table (n=1k), whereas the right plot presents a much larger FH table (n=128k). The time is O(Nlog²N)+N when N grows and the other sizes (n, d) are fixed (the first term corresponds to SELECT DISTINCT and the second one to the method). The trend indicates all evaluation methods get indeed impacted by N. Times tend to be linear as N grows for the three methods, which shows the method queries have more weight than getting the distinct dimension combinations. The PIVOT and CASE methods show very similar performance. On the other hand, there is a big gap between PIVOT/CASE and SPJ for large n. The trend indicates such gap will not shrink as N grows.

![Figure 2: Time Complexity Varying N (d = 16, Uniform Distribution).](image)

V. Conclusion

In SQL build-in “case” programming construct are available, it returns a selected value rather from a set of values based on Boolean expression. Queries for FH can be evaluated by performing direct aggregation form fact table F and at the same time rows are transposing to produce the FH. First, we need to get the unique combinations of R₁, . . ., Rₖ 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 [4].

References


