# SQL Data Sets Pivoting For Data Mining Analysis

**Mrs. N.Pooja[1], Dr.R.Bulli Babu[2]**
[1]Dept of CSE
[2]Professor, Dept of CSE
[1, 2] St.Marys Group of Institutions Guntur, AP

***Abstract-*** *Now a days many complex queries are required to prepare data sets for data mining analysis. They require more time and much effort is need for joining tables and aggregate columns. Existing SQL methods have limitations to prepare data sets because they return one column per aggregated group. A data mining project, requires many SQL queries, joining tables and aggregating columns. Conventional RDBMS usually manage tables with vertical form. Aggregated columns in a horizontal tabular layout returns set of numbers, instead of one number per row. The system uses one parent table and different child tables, operations are then performed on the data loaded from multiple tables. In general, a significant manual effort is required to build data sets, where a horizontal layout is required. The system use specific 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 de normalized layout which is the standard layout required by most data mining algorithms. The system propose three fundamental methods to generate data sets for mining analysis.*

***Keywords***- Pivot, CASE, Aggregation, OLTP

**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.

## I. INTRODUCTION

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. In general, a significant manual effort 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.

## II. WHAT IS DATA MINING

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, and regression. Each research discipline uses different terminology to describe the data set. In data mining the common terms are point-dimension.

In This paper introduce 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. The proposed system show evaluating horizontal aggregations is a challenging and interesting problem and we introduced alternative methods and optimizations for their efficient evaluation**.**

**Horizontal aggregations:**

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.

**CASE, SPJ and PIVOT METHODS**

This paper propose three fundamental methods to evaluate horizontal aggregations:

- CASE (Exploiting the programming CASE on struct)
- SPJ( Based on standard relational algebra operators (SPJ queries));
- PIVOT(Using the PIVOT operator, which is offered by some DBMS)

Experiments with large tables compare the pro posed query evaluation methods. The 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.

This paper proposes 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. This article explains how to evaluate and optimize horizontal aggregations generating standard SQL code.

**Relation B/W CASE,SPJ and PIVOT METHODS and Data Mining:**

The proposed horizontal aggregations provide several unique features. 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.

**Importance of CASE, SPJ and PIVOT Methods :**

This paper is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company.

$F$

| $K$ | $D_1$ | $D_2$ | $A$ |
|---|---|---|---|
| 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 |

$F_V$

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

$F_H$

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

**Fig 1. Example of F, FV and FH**

**Explanation:**

This section defines the table that will be used to explain SQL queries throughout this work. In order to present definitions and concepts in an intuitive manner, we present our definitions in OLAP terms. 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, . . . , Dp,A).

Our definitions can be easily generalized to multiple numeric attributes. In OLAP terms, F is a fact table with one column used as primary key, p dimensions and one measure column passed to standard SQL aggregations. That is, table F will be manipulated as a cube with p dimensions [9]. Subsets of dimension columns are used to group rows to aggregate the measure column. 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 (not to be confused with n, the size of the answer set). That is, |F| = N. Table F represents a temporary table or a view based on a “star join” query on several tables.

The sytem now explain tables FV (vertical) and FH (horizontal) that are used throughout the article. Consider a standard SQL aggregation (e.g. sum()) with the GROUP BY clause, which returns results in a vertical layout. Assume there are j + k GROUP BY columns and the aggregated attribute is A. The results are stored on table FV having j + k columns making up the primary key and A as a non-key attribute.

Table FV has a vertical layout. The goal of a horizontal aggregation is to transform FV into a table FH with a horizontal layout having n rows and j+d columns, where each of the columns represents a unique combination of the k grouping columns. Table FV may be more efficient than FH to

handle sparse matrices (having many zeroes), but some DBMSs like SQL Server [2] can handle sparse columns in a horizontal layout. The n rows represent records for analysis and the d columns represent dimensions or features for analysis. Therefore, n is data set size and d is dimensionality. In other words, each aggregated column represents a numeric variable as defined in statistics research or a numeric feature as typically defined in machine learning research.

**Pattern of producing SQL Queries:**

We now show actual SQL code for our small example. This SQL code produces FH in Figure 1. Notice the three methods can compute from either F or FV , but we use F to make code more compact.

**/* 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;

INSERT INTO FH SELECT F0.D1,F1.A AS D2_X,F2.A AS D2_Y FROM F0 LEFT OUTER JOIN F1 on F0.D1=F1.D1 LEFT OUTER JOIN F2 on F0.D1=F2.D1;

**/* 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_YFROM F GROUP BY D1;

**/* 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])

**Time Complexity and I/O Cost for Each Method:**

We now analyze time complexity for each method. Recall that N = |F|, n = |FH| and d is the data set dimensionality (number of cross-tabulated aggregations).We consider one I/O to read/write one row as the basic unit to analyze the cost to evaluate the query. This analysis considers every method pre computes FV .

**SPJ:** We assume hash or sort-merge joins are available. Thus a join between two tables of size O(n) can be evaluated in time O(n) on average. Otherwise, joins take time O(n log2n). Computing the sort in the initial query "SELECT DISTINCT.." takes O(N log2(N)). If the right key produces a high d (say $d \geq 10$ and a uniform distribution of values).

Then each σ query will have a high selectivity predicate. Each $|Fi| \leq n$. Therefore, we can expect $|Fi| < N$. There are d σ queries with different selectivity with a conjunction of k terms O(kn + N) each. Then total time for all selection queries is O(dkn +dN). There are d GROUP-BY operations with L1, . . ., Lj producing a table O(n) each.

Therefore, the d GROUP-BYs take time O(dn) with I/O cost 2dn (to read and write). Finally, there are d outer joins taking O(n) or O(nlog2(n)) each, giving a total time O(dn) or O(d nlog2(n)). In short, time is O(Nlog2(N)+dkn+dN) and I/O cost is Nlog2(N)+3dn+dN with hash joins. Otherwise, time is O(Nlog2(N) + dknlog2(n) + dN) and I/O cost is Nlog2(N) + 2dn + dnlog2(n) + dN with sort-merge joins. Time depends on number of distinct values, their combination and probabilistic distribution of values.

**CASE:** Computing the sort in the initial query "SELECT DISTINCT.." takes O(N log2(N)). There are O(dkN) comparisons; notice this is
fixed. There is one GROUP-BY with L1, . . ., Lj in time O(dkn) producing table O(dn). Evaluation time depends on the number of distinct value combinations, but not on their probabilistic distribution. In short, time is O(Nlog2(N)+dkn+N) and I/O cost is Nlog2(N)+n+N. As we can see, time complexity is the same, but I/O cost is significantly smaller compared to SPJ.

**PIVOT:**We consider the optimized version which trims F from irrelevant columns and k = 1. Like the SPJ and CASE methods, PIVOT depends on selecting the distinct values from the right keys R1, . . . , Rk. It avoids joins and saves I/O when it receives as input the trimmed version of F. Then it has similartime complexity to CASE. Also, time depends on number of distinct values, their combination and probabilistic distribution of values.

**REFERENCES**

[1] J. Han and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco, 1st edition, 2001.

[2] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: alternatives and implications. In Proc. ACM SIGMOD Conference, pages 343–354, 1998.

[3] C. Ordonez. Data set preprocessing and transformation in a database system. Intelligent Data Analysis (IDA), 15(4), 2011.
[4] C. Galindo-Legaria and A. Rosenthal. Outer join simplification and reordering for query optimization. ACM TODS, 22(1):43–73, 1997.