Data Mining Analysis using PIVOT Method in Horizontal Aggregation

V. Vijayadeepa
Head of the Department / Academy of Computer Science
Muthayammal College of Arts and Science.

S. Bharathi
M. Phil Scholar,
Muthayammal College of Arts and Science.

ABSTRACT

To analyze data efficiently, Data mining systems are widely using datasets with columns in horizontal tabular layout. Preparing a data set is more complex task in 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. PIVOT operator, offered by RDBMS is used to calculate aggregate operations. Relational databases are acceptable repository for structured data; integrating data mining algorithms with a relational DBMS is an essential research issue for database programmers. In a relational database, a significant effort is required to prepare a summary data set that can be used as input for the data mining process.

Keywords: Aggregation, Data Mining, Structured query language (SQL), PIVOT, Lattice.

I. INTRODUCTION

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 performing 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 modelling, 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 complexity 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 summary 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. There are several advantages for horizontal aggregation. First one is horizontal aggregation represent a template to generate SQL code from a data mining tool. This SQL code reduces manual work in the data preparation phase in data mining related project. Second is automatically generated code, which is more efficient than end user written SQL code. Thus datasets for the data mining projects can be created in less time. Third advantage is the data sets can be created entirely inside the DBMS.

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 start by explaining how to automatically generate SQL code.

3. Background Work


2. Horizontal Aggregations

We introduce a new class of aggregations that have similar behavior 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.
PIVOT and UNPIVOT, two operators on tabular data that exchange rows and columns. Haixun Wang [14] implemented ATLaS, to develop complete data-intensive applications in SQL—by writing new aggregates and table functions in SQL, it includes query rewriting, optimization techniques and the data stream management module. Carlos Ordonez [1] introduced techniques to efficiently compute fundamental statistical models inside a DBMS exploiting User-Defined Functions (UDFs).

Two summary matrices on the data set are mathematically shown to be essential for all models. There exist many proposals that have extended SQL syntax. Programming three methods with SQL queries is explored in [5], 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 [16]. 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. SQL extensions to define aggregate functions for association rule mining. 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. Conor Cunninghamalam [1] proposed an optimization and Execution strategies in an RDBMS which uses two operators i.e., PIVOT operator on tabular data that exchange rows and columns, enable data transformations useful in data modelling, data analysis, and data presentation. They can quite easily be implemented inside a query processor system, much like select, project, and join operator. Such a design provides opportunities for better performance, both during query optimization and query execution. Pivot is an extension of Group By with unique restrictions and optimization opportunities, and this makes it very easy to introduce incrementally on top of existing grouping implementations.

H Wang.C.Zaniolo [2] proposed a small but Complete SQL Extension for data Mining and Data Streams. This technique is a powerful database language and system that enables users to develop complete data-intensive applications in SQL by writing new aggregates and table functions in SQL, rather than in procedural languages as in current Object-Relational systems.

The ATLaS system consist of applications including various data mining functions, that have been coded in ATLaS" SQL, and execute with a modest (20–40%)
performance overhead with respect to the same applications written in C/C++. This system can handle continuous queries using the schema and queries in Query Repository. Sarawagi, S. Thomas, and R. Agrawal [3] proposed integrating association rule mining with relational database systems. Integrating Association rule mining include several method. Loose - coupling through a SQL cursor interface is an encapsulation of a mining algorithm in a stored procedure.

Second method is caching the data to a file system on-the-fly and mining tight-coupling using primarily user-defined functions and SQL implementations for processing in the DBMS. Loose-coupling and Stored-procedure architectures: For the loose-coupling and Stored-procedure architectures, can use the implementation of the Apriori algorithm for finding association rules.C. Ordonez [4] proposes an Integration of K-means clustering with a relational DBMS using SQL. This technique consist of three SQL implementations. First step is a straightforward translation of K-means computations into SQL, and an optimized version based on improved data organization, efficient indexing, sufficient statistics, and rewritten queries, and an incremental version that uses the optimized version as a building block with fast convergence and automated reseeding. The first implementation is a straightforward translation of K-means computations into SQL, which serves as a framework to build a second optimized version with superior performance. The optimized version is then used as a building block to introduce an incremental K-means implementation with fast convergence and automated reseeding. G. Graefe, U. Fayyad, and S. Chaudhuri [5] introduced efficient gathering of sufficient statistics for classification from large SQL Databases. This technique use a SQL operator (Unpivot) that enables efficient gathering of statistics with minimal changes to the SQL backend. Need a set of counts for the number of co-occurrences of each attribute value with each class variable. In classification the number of attribute values is not large (in the hundreds) the size of the counts table is fairly small. Continuous-valued attributes are discretized into a set of intervals. The most familiar selection measures used in classification do not require the entire data set, but only sufficient statistics of the data. A straightforward implementation for deriving the sufficient statistics on a SQL database results in unacceptably poor performance. The problem of optimizing queries with outer joins is not new. Optimizing joins by
reordering operations and using transformation rules is studied. This work does not consider optimizing a complex query that contains several outer joins on primary keys only, which is fundamental to prepare data sets for data mining. Traditional query optimizers use a tree based execution plan, but the use of hyper-graphs to provide a more comprehensive set of potential plans. J. Gray, A. Bosworth, A. Layman, and H. Pirahesh [6] proposed a relational aggregation operator that generalizing Group-By, Cross-Tab, and SubTotals. The cube operator generalizes the histogram, cross tabulation, roll-up, drill-down, and sub-total constructs. The cube operator can be imbedded in more complex non-procedural data analysis programs and data mining. The cube operator treats each of the N aggregation attributes as a dimension of N-space. The aggregate of a particular set of attribute values is a point in this space and the set of points forms an N-dimensional cube. Super-aggregates are computed by aggregating the N-cube to lower dimensional spaces. Creating a data cube requires generating the power set (set of all subsets) of the aggregation columns.

Since the CUBE is an aggregation operation, it makes sense to externalize it by overloading the SQL GROUP BY operator. G. Luo, J.F. Naughton, C.J. Ellmann, and M. Watzke [7] proposed Immediate materialized view introduces many lock conflicts or deadlocks. System results in low level of concurrency and high level of deadlocks. To solve the materialized view update problem V-locks (View locks) augment with a “value-based” latch pool. Direct Propagate Updates propagate updates on base relations directly to the materialized view without computing any join operator. Granularity and the No-Lock Locking Protocol locks have some interesting properties with respect to granularity and concurrency. Finer granularity locking results in higher concurrency. In the no-lock locking protocol, like the V locking protocol, updaters of the materialized view must get X locks on the tuples in the base relations they update and S locks on the tuples in the other base relations mentioned in the view. Xiang Lian and Lei Chen [9] analyzed cost models for evaluating dimensionality reduction in high-dimensional Spaces. This model is general cost models for evaluating the query performance over the reduced data sets by GDR, LDR, and ADR, in light of which we introduce a novel (A) LDR method, Partitioning based on Randomized Search (RANS). Formal cost models to evaluate the
effectiveness and efficiency of GDR, LDR, and ADR for range queries. Furthermore, we present a novel partitioning based (A) LDR approach, PRANS, which is based on our cost model and can achieve good query performance in terms of the pruning power. Extensive experiments have verified the correctness of our cost models and indicated that compared to the existing LDR method, can result in partitions with a lower query cost .C. Ordonez [10] introduced techniques to efficiently compute fundamental statistical models inside a DBMS exploiting User-Defined Functions (UDFs).

Two summary matrices on the data set are mathematically shown to be essential for all models: the linear sum of points and the quadratic sum of cross products of points. Introduce efficient SQL queries to compute summary matrices and score the data set. Based on the SQL framework, introduce UDFs that work in a single table scan.

Aggregate UDFs to compute summary matrices for all models and a set of primitive scalar UDFs are used to score data sets. C. Ordonez [11] proposed two SQL aggregate functions to compute percentages addressing many limitations. The first function returns one row for each percentage in vertical form and the second function returns each set of percentages adding 100% on the same row in horizontal form. These novel aggregate functions are used as to introduce the concept of percentage queries and to generate efficient SQL code in data mining related works. Queries using percentage aggregations are called percentage queries. Two practical issues were identified when computing vertical percentage queries. First issue is missing rows and second issue is division by zero.

Algorithm: (Lai )

Detection System ti:
collect raw alerts ri locally
// LAi : correlate – and – filter (ri,ti)
LAI \leftarrow\text{correlate and filter}(ri,1)
for each pij \in LAi do
// look up destination node for pij
dt=lookup(srcIP of pij)
subscribe(pij,nij,dj) on dt
end for
end for

4. Proposed Methods

The main goal is to define a template to generate SQL code by combining aggregation and transposition. The proposal has two perspectives such as to evaluate efficient aggregations and perform query optimization. The first one includes the
following approaches, pivoting, transposition and cross-tabulation. Pivoting approach is a built-in method in a commercial DBMS. It can help evaluating an aggregated tabular format for summarized data set. It perform the following steps, The pivoting method is used to write cross-tabulation queries that rotate rows into columns, aggregating data in the process of the rotation. The output of a pivot operation typically includes more columns and fewer rows than the starting data set. The pivot computes the aggregation functions specified at the beginning of the clause. Aggregation functions must specify a GROUP BY clause to return multiple values; the pivot performs an implicit GROUP BY. New columns corresponding to values in the pivot, each aggregated value is transposed to the appropriate new column in the cross-tabulation. The subclauses of the pivot have the following semantics: expr - specify an expression that evaluates to a constant value of a pivot column. Subquery – to specify a subquery, all values found by the subquery are used for pivoting. The subquery must return a list of unique values at the execution time of the pivot query.

5. Result Analysis
Comparing Evaluation Methods

On the other hand, the second important issue is automatically generating unique column names. If there are many sub grouping columns \( R_1; \ldots; R_k \) or columns are of string data types, this may lead to generate very long column names, which may exceed DBMS limits. However, these are not important limitations because if there are many dimensions that is likely to correspond to a sparse matrix (having many zeroes or nulls) on which it will be difficult or impossible to compute a data mining model. On the other hand, the large column name length can be solved as explained below. The problem of \( d \) going beyond the maximum number of columns can be solved by vertically partitioning \( FH \) so that each partition table does not exceed the maximum number of columns allowed by the DBMS. Evidently, each partition table must have \( L_1; \ldots; L_j \) as its primary key. Alternatively, the column name length issue can be solved by generating column identifiers with integers and creating a “dimension” description table that maps identifiers to full descriptions, but the meaning of each dimension is lost. An alternative is the use of abbreviations, which may require manual input.
Query Optimizations

Our first query optimization, applied to three methods. Our goal is to assess the acceleration obtained by precomputing a cube and storing it on FV. We can see this optimization uniformly accelerates all methods. This optimization provides a different gain, depending on the method: for SPJ the optimization is best for small n, for PIVOT for large n and for CASE there is rather a less dramatic improvement all across n. It is noteworthy PIVOT is accelerated by our optimization, despite the fact it is handled by the query optimizer. Since this optimization produces significant acceleration for the three methods (at least 2 faster) we will use it by default. Notice that precomputing FV takes the same time within each method. Therefore, comparisons are fair. We now evaluate an optimization specific to the PIVOT operator. This PIVOT optimization is well known, as we learned from SQL Server DBMS users groups. shows the impact of removing (trimming) columns not needed by PIVOT. That is, removing columns that will not appear in FH. We can see the impact is significant, accelerating evaluation time from three to five times. All our experiments incorporate this optimization by default.

<table>
<thead>
<tr>
<th>File Size</th>
<th>SPJ</th>
<th>CASE</th>
<th>PIVOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>58</td>
<td>54</td>
<td>49</td>
</tr>
<tr>
<td>19</td>
<td>72</td>
<td>71</td>
<td>66</td>
</tr>
<tr>
<td>25</td>
<td>86</td>
<td>83</td>
<td>83</td>
</tr>
</tbody>
</table>

Table.1 Comparing Query Evaluation Methods(1kb) All with Optimization Computing FV.

<table>
<thead>
<tr>
<th>File Size</th>
<th>SPJ</th>
<th>CASE</th>
<th>PIVOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>88</td>
<td>90</td>
<td>84</td>
</tr>
<tr>
<td>19</td>
<td>96</td>
<td>94</td>
<td>95</td>
</tr>
<tr>
<td>25</td>
<td>92</td>
<td>95</td>
<td>87</td>
</tr>
</tbody>
</table>

Table.2 Variability of Mean Time (100kb, One Standard Deviation, Percentage of Mean Time)

<table>
<thead>
<tr>
<th>File Size</th>
<th>SPJ</th>
<th>CASE</th>
<th>PIVOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>99</td>
<td>98</td>
<td>84</td>
</tr>
<tr>
<td>19</td>
<td>102</td>
<td>101</td>
<td>99</td>
</tr>
<tr>
<td>25</td>
<td>103</td>
<td>99</td>
<td>92</td>
</tr>
</tbody>
</table>
Variability of Mean Time (1.5 mb, One Standard Deviation, Percentage of Mean Time)

Time complexity varying d (1.5 mb, uniform distribution).

**Time Complexity**

We now verify the time complexity analysis given in We plot time complexity keeping varying one parameter and the remaining parameters fixed. In these experiments, we generated synthetic data sets similar to the fact table of TPC-H of different sizes with grouping columns of varying selectivities (number of distinct values). We consider two basic probabilistic distribution of values: uniform (unskewed) and zipf (skewed). The uniform distribution is the distribution used by default.

**6. Conclusion**

The proposed approaches implements an abstract but minimal extension to SQL standard aggregate functions to compute efficient summarized data set which just requires specifying sub grouping columns inside the aggregation function call. From a query optimization perspective, The proposed system describes the possibility of extending SQL OLAP aggregations with horizontal layout capabilities. Horizontal aggregations produce tables with fewer rows, but with more columns. The aggregated tables are useful to create data sets with a horizontal layout, as commonly required by data mining algorithms and OLAP cross-tabulation. The output of a query optimization can immediately be applied back to the data gathering, transformation, and analysis processes. Anomalous data can be detected in existing data sets, and new data entry can be validated in real time, based on the existing data. SQL Server Data Mining contains multiple algorithms that can perform churn analysis based on historical data. Each of these algorithms will provide a probability. In future, research issues is proposed on extending SQL code for data mining processing. The related work on query optimization is proposed and compared to horizontal aggregations with alternative proposals to perform transposition or pivoting. It includes to develop more complete I/O cost models for cost-based query optimization and to study optimization of horizontal aggregations processed in parallel in a shared-nothing DBMS architecture.

**References**


[2] Carlos Ordonez, Pitchaimalai. S.K, —Bayesian Classifiers Programmed in


V.Vijayadeepa received her B.Sc degree from university of Madras and M.Sc degree and MCA degree from Periyar University. She has completed her M.Phil at Bharathidasan University. She is having 10 years of experience in collegiate teaching and She is the Associate Professor and head of department of computer science and applications in Muthayammal college of Arts and Science,Rasipuram affiliated by Periyar University. Her main research interests include personalized Web search, Web information retrieval, data mining, data warehousing and information systems.

S.Bharathi received her B.Sc., degree in Muthayammal College of Arts and Science from Periyar University, Salem (2005-2008)[Tamil Nadu(India)].Then,She did her MCA degree in Vivekanandha Institute of Engineering and Technology for Women from Anna University of Technology, Coimbatore (2008-2011).She is the M.Phil Research Scholar of Muthayammal College of Arts and Science. Her Area of interest is Datamining and Warehousing.