Volume : 3 | Issue : 1 | January 2014 • ISSN No 2277 - 8179

Research Paper

Aggregation to Prepare Data Mining Sets for Analysis

Computer Science

KEYWORDS : Horizontal aggregation, Handlers

Sreekar.E Student, Department of CSE, SR Engineering College, Warangal, India

Ramesh. J Assistant professor, Department of CSE, SR Engineering College, Warangal, India

ABSTRACT

Aggregate functions perform a calculation on a set of values and return a single value. Except for COUNT, aggregate functions ignore null values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement. All aggregate functions are deterministic. This means aggregate functions return the same value any time that they are called by using a specific set of input values. The OVER clause may follow all aggregate functions except GROUPING and GROUPING_ID. Aggregate functions can be used as expressions only in the select list of a SELECT statement (either a sub query or an outer query) and A HAVING clause. 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. Managing large datasets except DBMS support can be a difficult job. Trying different subsets of data points and dimensions is more convenient, faster and easier to do inside a relational database with SQL queries than outside with alternative handler. Horizontal aggregation can be performing by using handler, this will be able to simply be alive or implement within a doubt analyzer, a large amount similar to a choose plan and unite.

INTRODUCTION

Data mining is a powerful technology with great potential to help companies focus on important information in the data they have collected about the behavior of their customers. Many researchers have made constant efforts to improve data mining technology .Some of such efforts are [1],[2],[3],[4]. Analysis and Preparation of data sets requires a great effort specially if normalized tables are included [5] for using them as inputs for data mining or statistical algorithms [6],[7].

Database normalization is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. Normalization usually involves dividing large tables

In to smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database using the defined relationships.

The objectives of normalization

- To free the collection of relations from undesirable insertion, update and deletion dependencies.
- To reduce the need for restructuring the collection of relations, as new types of data are introduced, and thus increase the life span of application programs.
- To make the relational model more informative to users.
- To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

Operational, or online transaction processing (OLTP), workloads are characterized by small, interactive transactions that generally require sub-second response times. It is common for OLTP systems to have high concurrency requirements, with a read/write ratio ranging from 60/40 to as low as 98/2. One of the data mining problems associated with OLAP is association rule mapping [8]. Modifications are predominantly singleton statements, and most queries are constrained to simple joins. While limiting joins to as few tables as possible is desirable, a significant number of application systems do join many tables. Standard practices call for indexing strategies in OLTP systems to target an increase in concurrency versus query support; however, more indexes have to be created than is desired to reach acceptable query performance. The lower the proportion of write operations is in the system, the higher the level of indexing that can be tolerated, unless the timing of specific write operations is critical. Database plans generally start with third normal form (3NF) enforced with referential integrity (RI) constraints, and then selectively deviate to second normal form (2NF) when necessary to enhance

Performance .SQL extensions for defining aggregate functions for association rule mining are introduced in [9].

For OLTP systems, the middle tier plays an important role in defining the overall success of the system; while the database is important, it is not the only system component. When a new OLTP system is developed, the initial challenges in scalability and performance are often encountered in the middle tier. It is only after the challenges in the middle tier are addressed that the database tier scalability and performance issues are revealed. We recommend that you refer to the Technical Reference Guides included in the Data Warehouse and App Fabric topics when gathering data points for operational considerations.

PROPOSED WORK

Our proposed horizontal aggregations provide several unique features and advantages. Horizontal aggregations are related to horizontal percentage aggregations [10]. 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).

The SPJ method used by us proved horizontal aggregations can be evaluated with relational algebra exploiting outer joins, showing our work is connected to traditional query optimization [11]. Other methods used by us are CASE and PIVOT which were proposed in [13],[14]. These methods are used to avoid joins as far as possible.

IMPLEMENTATION

A . Admin Module

Admin will upload new connection form based on regulations in various states.

B. User Module

User will be able to view his bill details on any date may be after a month or after months or years and also.

C. View Module

Admin has three ways to view the user bill details, the 3 ways are

SPJ

PIVOT

CASE

/* SPJ method */
Select <attribute list>

From <relation list>

Where <condition list>

Example Filter Query over R(A,B,C):

Select B

From R

Where R.A = "c" Ù R.C > 10

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_Y

FROM F

GROUP BY D1;

It is possible to start pivoting in standard SQL, though the syntax is cumbersome and its performance is generally poor. One method to express pivoting uses scalar sub queries in the projection list. Each pivoted column is created through a separate (but nearly identical) sub query. For database uses that do not support PIVOT, users could employ this technique to perform pivoting operations.

Alas, this approach has limitations that restrict the power of pivoting. Each column has redundant syntax, which is cumbersome as the number of pivoted columns increases. These syntaxes are also potentially tough to optimize. For this syntax, the query optimizer is presented with a number of sub-queries, making it harder to identify that this whole operation represents a "Pivot" on a single table. In practice, this is not an easy operation, making pivot-specific optimizations very difficult. The common problem is that the intent of the query is difficult to infer from the syntax or common relational algebra representation. Therefore, we propose the following syntax for PIVOT as an additional option under the rule of the ANSI SQL grammar. This syntax is easier to read and better captures the intent of the desired operation. Repetition is eliminated, making queries easier to ready, write, and maintain. It shows that this approach also enables additional query optimization techniques.

/* PIVOT method */

Base Case (no pivoted rows):

Inductive Case (one or more pivoted value(s) z):

PIVOT(D,cv, i in W + {z}, Y)

J

PIVOT(D,c v, i in W, Y)

J

LeftOuterJoin(D=D')

GroupBy(D', wi = Y(cv'))

J'

s (p'

CONCLUSION
We introduced a new class of extended aggregate functions, called horizontal aggregations which help preparing data sets for data mining and OLAP cube exploration. Specially, 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 multi-dimensional vector. We 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, we 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 joins (SPJ) queries. The CASE method is our 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. We proved the three methods produce the same result. We have explained it is not possible to evaluate horizontal aggregations using standard SQL without either joins or .case. constructs using standard SQL operators.

REFERENCE

[1] G. Bhargava, P. Goel, and B.R. Iyer. Hypergraph based reordering of outer join queries with complex predicates. | In ACSIGMOD Conference, pages 304.315, 1995. | | [2] J. Clear, D. Dunn, B. Harvey, M.L. Heytens, and P. Lohman. Nonstop SQL/MX primitives for knowledge discovery. In ACM KDD Conference, pages 425.429, 1999. | | [3] E.F. Codd. Extending the database relational model to capture more meaning. | In ACM TODS, 4(4):397.434, 1979. | | [4] C. Galindo-Legaria and A. Rosenthal. Outer join simplification and reordering for query optimization. | ACM TODS, 22(1):43.73, 1997. | | [5] C. Ordonez. Data set preprocessing and transformation in a database system. | Intelligent Data Analysis (IDA), 15(4), 2011. | | [6] C. Ordonez. Statistical model computation with UDFs. IEEE Transactions | on Knowledge and Data Engineering (TKDE), 22, 2010. | | [7] C. Ordonez and S. Pitchaimalai. Bayesian classifiers programmed in | SQL. IEEE Transactions on Knowledge and Data Engineering (TKDE), | 22(1):139.144, 2010. | | | [8] 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.. | | [9] Witkowski, S. Bellamkonda, T. Bozkaya, G. Dorman, N. Folkert, A. Gupta, L. Sheng, and S. Subramanian. Spreadsheets in RDBMS for OLAP. | In Proc. ACM SIGMOD Conference, | pages 52, 63, 2003. | | [10] C. Ordonez. Horizontal aggregations for building tabular data sets. In | Proc. ACM SIGMOD Data Mining and Knowledge Discovery Workshop, | pages 35.42, 2004. | | [11] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A | relational aggregation operator generalizing group-by, cross-tab and subtotal. | In CDE Conference, pages 152.159, 1996. | | [12] H. Garcia-Molina, J.D. Ullman, and | J. Widom. Database Systems: The | Complete Book Prentice Hall, 1st edition, 2001. | | [13] 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. | | [14] G. Graefe, U. Fayyad, and S. Chaudhuri. On the efficient gathering of sufficient statistics for classification from large SQL databases. | In Proc ACMKDD Conference, pages 2 |