International Refereed Journal of Engineering and Science (IRJES)
ISSN (Online) 2319-183X, (Print) 2319-1821
Volume 1, Issue 3 (November 2012), PP.28-33
www.irjes.com

Improving Analysis Of Data Mining By Creating Dataset Using Sql Aggregations

* J.Mahesh , **M.Antony Kumar
** Dean[CSE/IT] P.M.R. Engineering College Chennai-India*
Asst.Professor, IT Department, P.M.R. Engineering College Chennai-India

Abstract : *In Data mining, an important goal is to generate efficient data. Efficiency and scalability have always been important con-cerns in the field of data mining. The increased complexity of the task calls for algorithms that are inherently more expensive. 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. PIVOT method is much faster method and offers much scalability. Partitioning large set of data, obtained from the result of horizontal aggregation, in to homogeneous cluster is important task in this system. K-means algorithm using SQL is best suited for implementing this operation.*

Index terms : *PIVOT, SQL, Data Mining, Aggregation*

I. INTRODUCTION

Existing SQL aggregate functions present important limitations to compute percentages. This article proposes two SQL aggregate functions to compute percentages addressing such limitations. The first function returns one row for each percentage in vertical form like standard SQL aggregations. The second function returns each set of percentages adding 100% on the same row in horizontal form. These novel aggregate functions are used as a framework to introduce the concept of percentage queries and to generate efficient SQL code. Experiments study different percentage query optimization strategies and compare evaluation time of percentage queries taking advantage of our proposed aggregations against queries using available OLAP extensions. The proposed percentage aggregations are easy to use, have wide applicability and can be efficiently evaluated.

II. RELATED WORK

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 Cunningalam [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 Sub-Totals. 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 ***Volume 2, Issue 5, May 2012 www.ijarcsse.com © 2012, IJARCSSE All Rights Reserved Page | 205***
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.

III. EXISTING SYSTEM

Existing system

In existing work, 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. Standard aggregations are hard to interpret when there are many result rows, especially when grouping attributes have high cardinalities. There exist many aggregation functions and operators in SQL. Unfortunately, all these aggregations have limitations to build data sets for data mining purposes.

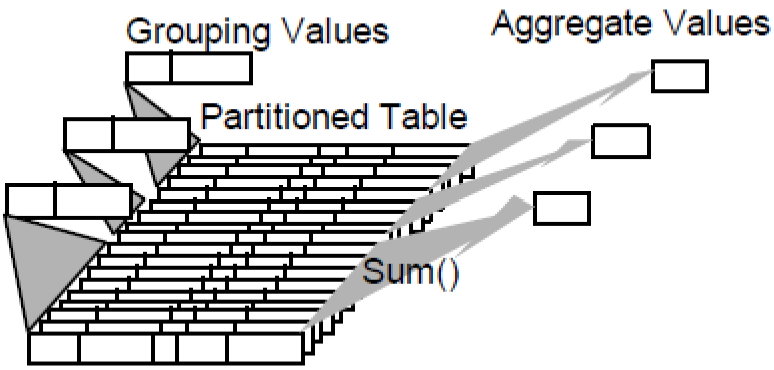
IV. PROPOSED SYSTEM

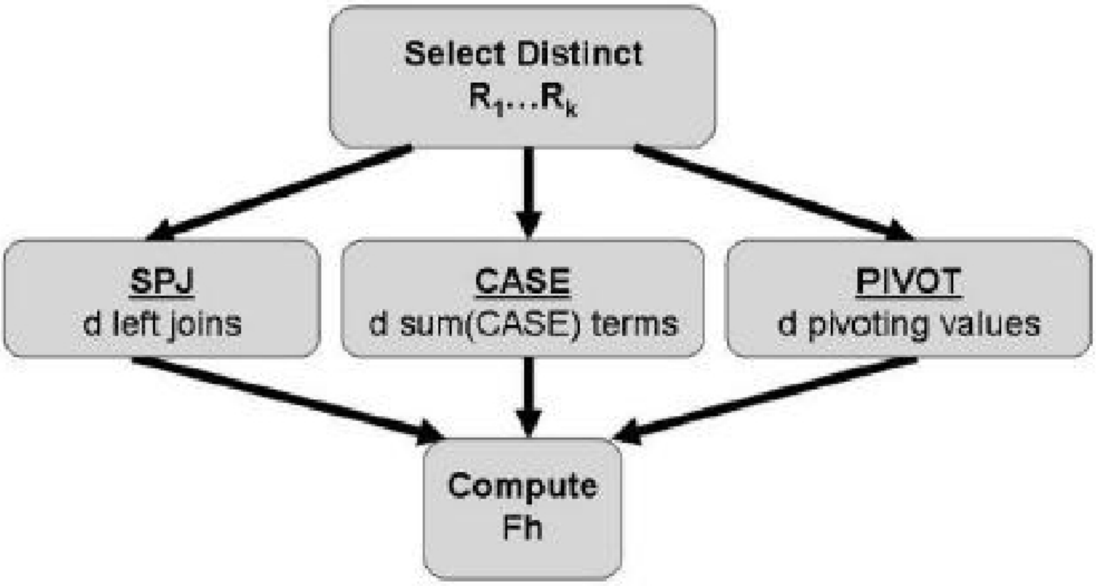
Proposed System

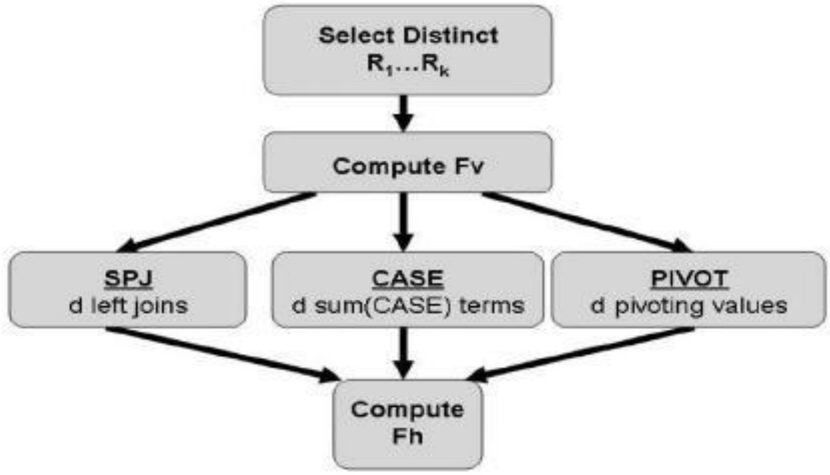
Our proposed horizontal aggregations provide several unique features and advantages. First, they represent a template to generate SQL code from a data mining tool to build data sets for data mining analysis. Such SQL code automates writing SQL queries, optimizing them and testing them for correctness. Horizontal aggregations represent an extended form of traditional SQL aggregations, which return a set of values in a horizontal layout, instead of a single value per row. Horizontal aggregations preserve evaluation semantics of standard SQL aggregations. The main difference will be returning a table with a horizontal layout, possibly having extra nulls

.

V. FUNCTIONAL DIAGRAM







VI. EXECUTION STRATEGIES IN AGGREGATION

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 modelled 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: SELECT L1, .., Lm, sum (A) FROM F1,F2 GROUP BY L1, Lm; In a horizontal aggregation there are four input parameters to generate SQL code: 1) The input table F1,F2……,Fn 2) The list of GROUP BY columns L1, . . ., Lj , 3) The column to aggregate (A), 4) The list of transposing columns R1, . . .,Rk. This aggregation query will produce a wide table with m+1 columns (automatically determined), with one group for each unique combination of values L1, . . . , Lm and one aggregated value per group (i.e., sum(A)). In order to evaluate this query the query optimizer takes three input parameters. First parameter is the input table F. Second parameter is the list of grouping columns L1, . . . , Lm. And the final parameter is the column to aggregate (A). Example

In the Fig.1 there is a common field K in F1 and F2.In F2, D2 consist of only two distinct values X and Y and is used to transpose the table. The aggregate operation is used in this is sum (). The values within D1 are repeated, 1 appears 3 times, for row 3, 4 and, and for row 3 & 4 value of D2 is X & Y. So D2X and D2Y is newly generated columns in FH.

K	D_1	D_2
1	3	X
2	2	Y
3	1	Y
4	1	Y
5	2	X
6	1	X
7	3	X
8	2	X

F_1

K	A
1	9
2	6
3	10
4	0
5	1
6	Null
7	8
8	7

F_2

D_1	D_2X	D_2Y
1	Null	10
2	8	6
3	17	null

F_H

Fig 1.An example of Horizontal aggregation

Commonly using Query Evaluation methods in Horizontal aggregation functions [12] are

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,RI1,......,RIj then FH can be computed from FV.
To get FH system need *n* left outer join with *n*+1 tables 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.

```
INSERT INTO FH
SELECT F0 . LE1 , F0. LE2 ,..., F0. LEj,
F1.A, F2 .A,......, Fn .A
FROM F0
LEFT OUTER JOIN F1
ON F0. LE1= F1. LE1 and. . . and F0. LEj= F1. LEj
LEFT OUTER JOIN F2
ON F0. LE1= F2. LE1 and. . . and F0. LEj= F2. LEj
. . . .
LEFT OUTER JOIN Fn
ON F0. LE1= Fn. LE1 and. . . and F0. LEj= Fn. LEj
```

It is easy to see that left outer join is based on same columns. This strategy basically needs twice I/O operations by doing updates rather than insertion.

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.

```
SELECT DISTINCT RI1
FROM F;
INSERT INTO FH SELECT LE1,LE2,....,LEj,
V(CASE WHEN RI1=v11 and . . . Rk=vk1 THEN A ELSE null END)
..
, V(CASE WHEN RI1=v1n and . . . Rk=vkn THEN A ELSE null END)
FROM F
GROUP BY LE1,LE2,. . .,LEj
```

PIVOT method

Pivot transforms a series of rows into a series of fewer numbers of rows with additional columns Data in one source column is used to determine the new column for a row, and another source column is used as the data for that new column. The wide form can be considered as a matrix of column values, while the narrow form is a natural encoding of a sparse matrix In current implementation PIVOT operator is used to calculate the aggregations. One method to express pivoting uses scalar sub queries. Each pivoted is created through a separate sub query. PIVOT operator provides a technique to allow rows to columns dynamically at the time of query compilation and execution.

```
SELECT *FROM (Bill Table PIVOT (SUM (Amount) for Month in („Jan ,  Feb  ,  Mar  )
```

This query generate table with jan,feb and mar as column attribute and the sum of the amount of particular customer that are stored inside the Bill Table. The pivot method is more efficient method than other two methods. Because the pivot operator internally calculates the aggregation operation and no need to create extra tables. So operation performed within this method is less compared to other methods.

VII. CONCLUSION

This article proposed two aggregate functions to compute percentages. The first function returns one row for each Computed percentage and it is called a vertical percentage aggregation. The second function returns each set of percentages adding 100% on the same row in horizontal form and it is called a horizontal percentage aggregation. The proposed aggregations are used as a framework to study percentage queries. Two practical issues when computing vertical percentage queries were identified: missing rows and division by zero. We discussed alternatives to tackle them. Horizontal percentages do not present the missing row issue. We studied how to efficiently evaluate percentage queries with several optimizations including indexing, computation from partial aggregates, using either row insertion or update to produce the result table, and reusing vertical percentages to get horizontal percentages. Experiments study percentage query optimization strategies and compare our proposed percentage aggregations against queries using OLAP aggregations. Both proposed aggregations are significantly faster than existing OLAP aggregate functions showing about an order of magnitude improvement.

REFERENCES

[1.] C. Cunningham, G. Graefe, and C.A. GalindoLegaria. PIVOT and UNPIVOT: Optimization and execution strategies in an RDBMS. In Proc. VLDB Conference, pages 998–1009, 2004.

[2.] H. Wang, C. Zaniolo, a006Ed C.R. Luo. ATLaS: A small but complete SQL extension for data mining and data streams. In Proc. VLDB Conference, pages 1113–1116, 2003.

[3.] 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.

[4.] C. Ordonez. Integrating K-means clustering with a relational DBMS using SQL. IEEE Transactions on Knowledge and Data Engineering (TKDE), 18(2):188–201, 2006.

[5.] G. Graefe, U. Fayyad, and S. Chaudhuri. On the efficient gathering of sufficient statistics for classification from large SQL databases. In Proc. ACM KDD Conference, pages 204–208, 1998.

[6.] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and subtotal. In ICDE Conference, pages 152–159,1996.

[7.] G. Luo, J.F. Naughton, C.J. Ellmann, and M. Watzke. Locking protocols for materialized aggregate join views. IEEE Transactions on Knowledge and Data engineering (TKDE), 17(6):796–807, 2005.

[8.] C. Ordonez and S. Pitchaimalai. Bayesian classifiers programmed in SQL. IEEE Transactions on Knowledge and Data Engineering (TKDE), 22(1):139–144, 2010.

[9.] Xiang Lian, Student Member, IEEE, and Lei Chen, General Cost Models for Evaluating Dimensionality Reduction in High-Dimensional Spaces. IEEE Transactions on Knowledge and Data Engineering (TKDE), 22(1):139–144, 2010.

[10.] C. Ordonez. Statistical model computation with UDFs. IEEE Transactions on Knowledge and Data Engineering (TKDE), 22, 2010.

[11.] C. Ordonez. Vertical and horizontal percentage aggregations. In Proc. ACM IGMOD Conference, pages 866–871, 2004.

[12.] C. C. Ordonez and Zhibo Chen. Horizontal Aggregation in SQL to prepare Data Sets forData Mining Analysis. . IEEE Transactions on Knowledge and Data Engineering (TKDE), 1041- 4347/11/$26.00 ,2011