International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT217263

Workload Optimization by Horizontal Aggregation in SQL for Data Mining Analysis

Prasanna M. Rathod[1], Prof. Dr. Anjali B. Raut[2]
[1]M.E. (CSE), H.V.P.M. COET, SGB Amravati University, Maharashtra, India
[2]Professor and Head of Department, H.V.P.M. COET, SGB Amravati University, Maharashtra, India

Article Info
Volume 7, Issue 2
Page Number: 304-309

Publication Issue :
March-April-2021

Article History
Accepted : 12 April 2021
Published : 17 April 2021

ABSTRACT

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. 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 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. For query optimization the distance computation and nearest cluster in the k-means are based on SQL.

Workload balancing is the assignment of work to processors in a way that maximizes application performance. The process of load balancing can be generalized into four basic steps:

1. Monitoring processor load and state;
2. Exchanging workload and state information between processors;
3. Decision making;
4. Data migration.

The decision phase is triggered when the load imbalance is detected to calculate

optimal data redistribution. In the fourth and last phase, data migrates from overloaded processors to under-loaded ones.

Keywords : SQL, CASE, SPJ, PIVOT, Horizontal Aggregation

I. INTRODUCTION

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, regression and PCA consult. 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. We introduce a new class of aggregate functions that can be used to build data sets in a 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 introduce alternative methods and optimizations for their efficient evaluation.

Workload balancing schemes classify into three major classes:

1. Static versus dynamic load balancing;
2. Centralized versus distributed load balancing;
3. Application-level versus system-level load balancing.

Static load balancing can be used in applications with constant workloads, as a pre-processor to the computation. Other applications require dynamic load balancers that adjust the decomposition as the computation proceeds. This is due to their nature which is characterized by workloads that are unpredictable and change during execution. Data mining uses one of these applications.

II. METHODS AND MATERIAL

A. Introduction to Data Mining

We are in an age often referred to as the information age. In this information age, because we believe that information leads to power and success, and thanks to sophisticated technologies such as computers, satellites, etc., we have been collecting tremendous amounts of information. Initially, with the advent of computers and means for mass digital storage, we started collecting and storing all sorts of data, counting on the power of computers to help sort through this amalgam of information. Unfortunately, these massive collections of data stored on disparate structures very rapidly became overwhelming. This initial chaos has led to the creation of structured databases and database management systems (DBMS). The efficient database management systems have been very important assets for management of a large corpus of data and especially for effective and efficient retrieval of particular information from a large collection whenever needed. The proliferation of database management systems has also contributed to recent massive gathering of all sorts of information. Today, we have far more information than we can handle: from business transactions and scientific data, to satellite pictures, text reports and military intelligence. Information retrieval is simply not enough anymore for decision-making. Confronted with huge collections of data, we have now created new needs to help us make better managerial choices. These needs are automatic summarization of data, extraction of the “essence” of information stored, and the discovery of patterns in raw data.

B. SPJ method

The SPJ method is interesting from a theoretical point of view because it is based on relational

operators only. The basic idea is to create one table with a vertical aggregation for each result column, and then join all those tables to produce FH. We aggregate from F into d projected tables with d Select- Project-Join-Aggregation queries (selection, projection, join, aggregation). Each table FI corresponds to one subgrouping combination and has {L1, . . ., Lj} as primary key and an aggregation on A as the only non-key column. It is necessary to introduce an additional table F0, that will be outer joined with projected tables to get a complete result set. We propose two basic sub-strategies to compute FH. The first one directly aggregates from F. The second one computes the equivalent vertical aggregation in a temporary table FV grouping by L1, . . ., Lj, R1, . . .,Rk. Then horizontal aggregations can be instead computed from FV, which is a compressed version of F, since standard aggregations are distributive

C. CASE method

For this method we use the"case" programming construct available in SQL. The case statement returns a value selected from a set of values based on boolean expressions. From a relational database theory point of view this is equivalent to doing a simple projection/aggregation query where each monkey value is given by a function that returns a number based on some conjunction of conditions. We propose two basic sub-strategies to compute FH. In a similar manner to SPJ, the first one directly aggregates from F and the second one computes the vertical aggregation in a temporary table FV and then horizontal aggregations are indirectly computed from FV.

D. 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.

E. 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.

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_2X	D_2Y
1	null	10
2	8	6
3	17	null

F. The Workload Balancing Algorithms

The two main advantages of the workload balancing strategy are:

1. The priority is given to local workload balancing (i.e. intra-cluster). The objective of that is to privilege local communications (LAN network) in order to reduce the overhead caused by the transfer of work or data.
2. The strategy is totally distributed but the decision is taken locally.

Following is the dynamic load balancing algorithms:

A. Module node:

Loop:

- Receives a group of candidates from the coordinator of the cluster.
- Calculates their supports.
- Sends local supports to cluster's coordinator which performs the global supports reduction.
- Every n steps :
  - Updates NSV (Node State Vector)
  - Sends NSV to the Cluster

B. Module cluster coordinator

1. Init_Execution: Receive Partition of the DB.
2. Normal Execution:

Loop:

- Distributes candidate itemsets between nodes according to their capacities. Candidates are distributed by their (k-1) common prefix.

- Performs the global reduction of supports to obtain global frequencies.
- Constructs frequent itemsets (Lk step).
- Constructs candidates itemsets of the following iteration (Ck+1 step).
- Every n steps :
  - Saves the local state;
  - Updates if necessary C_{k+1} step.

3. Monitoring :
   - ❖ Every n steps:
   - ➢ Receives NSV (Node State Vector)
   - ➢ Checks if any overload in nodes
   - ➢ Updates CSV (Cluster State Vector)
   - ➢ Sends CSV
   - ➢ Checks if an Overload is detected in some nodes
     a. Load Balancing :
   - ❖ If an Overload is detected:
   - ➢ Checks (CSV)
   - ➢ Searches_Candidate, in nodes, to balance the load
   - ➢ If Find then Start_load balance (intra-cluster)
   - ➢ Else asks Site to Start_load Balance (intra-Site)

C. Module site coordinator

Loop :

- Updates the global state vector of the site: average(ch_i)).
- Finds the Max overloaded cluster and the max under-loaded cluster:
  - $Cl_{ijmax} \rightarrow \overset{max}{j} (ch_{ij}) > average(ch_i)$
  - $Cl_{ijmin} \rightarrow \overset{min}{j} (ch_{ij}) < average(ch_i)$
- Finds the Max x_c (with the same prefix) on cl_{ijmax} (intra-Site):
  1. $Ch_{ijmin} + x_c \cdot \omega_{ijmin} \leq average(ch_i)$
     //To find the best number of candidates to migrate in order to not overload the destination cluster

     AND
  2. $x_c \cdot \omega_{ijmax} - (x_c \cdot \omega_{ijmin} + long(x_c).\zeta_{ijmaxjmin}) > Seuil_{mc}$
- **If** x_c exists **Then** informs the overloaded c_{lijmax} and the underloaded c_{lijmin} and updates (ch_i).
  - **Asks** from the overloaded cluster to send the family of candidates having the same prefix.
- **Else** asks other sites to balance the load (inter-Sites).

Where,

$\zeta_{ij'}$	:	Transmission speed between clusters cl_{ij} and $cl_{ij'}$;
ω_{ijk}	:	Cycle time of nd_{ijk};
ch_i	:	Charge of S_i;
ch_{ij}	:	Charge of cl_{ij};
ω_{ij}	:	Average(ω_{ijk});
$seuil_{mc}$	:	Significant time limit to trigger candidate itemsets migration between clusters;
$seuil_{mt}$	:	Significant time limit to trigger task migration between sites;
x_c	:	Number of candidates to migrate from one cluster to another.

All load balancing algorithms are executed in parallel without inducing an overhead in execution time. Computing nodes continue working even during work or data migration.

III. RESULTS AND DISCUSSION

As mentioned, 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 ingredients in such SQL code: joins and aggregations; 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. An existing to 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.

Disadvantage:

1) Existing SQL aggregations have limitations to prepare data sets.
2) To return one column per aggregated group.

Proposed Work and Objectives

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.

Advantage:

1) The SQL code reduces manual work in the data preparation phase in a data mining project.

2) The SQL code is automatically generated it is likely to be more efficient than SQL code written by an end user.
3) The data sets can be created in less time.
4) The data set can be created entirely inside the DBMS.

Module Description:

1. Admin Module
2. User Module
3. View Module
4. Download Module

Module 1: Admin Module

Admin will upload new connection form based on regulations in various states. Admin will be able upload various details regarding user bills like a new connection to a new user, amount paid or payable by user. In case of payment various details regarding payment will be entered and separate username and password will be provided to users in large.

Module 2: 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 he can to view the our bill details in a various ways for instance, The year wise bills, Month wise bills, totally paid to bill in EB. This will reduce the cost of transaction. If user thinks that his password is insecure, he has option to change it. He also can view the registration details and allowed to change or edit and save it.

Module 3: View Module

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

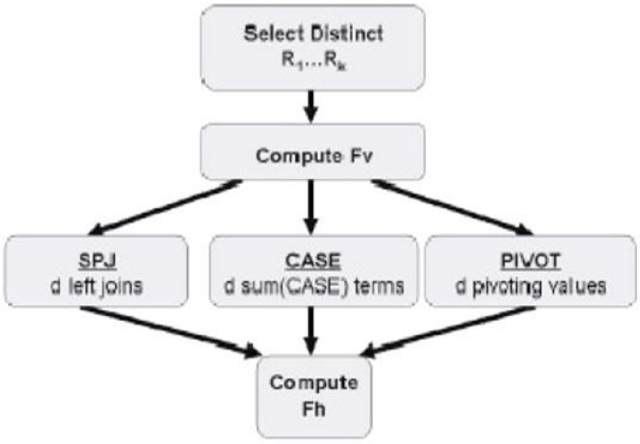
i) SPJ
ii) PIVOT
iii) CASE

i) **SPJ:** While using SPJ the viewing and processing time of user bills is reduced.
ii) **PIVOT:** This is used to draw the user details in a customized table. This table will elaborate us on the various bill details regarding the user on monthly basis.
iii) **CASE:** using CASE query we can customize the present table and column based on the conditions. This will help us to reduce enormous amount of space used by various user bill details. It can be viewed in two difference ways namely Horizontal and Vertical.

In case of vertical the number of rows will be reduced to such an extent it is needed and column will remain the same on other hand the Horizontal will reduce rows as same as vertical and will also increase the columnar format

Module 4: Download Module

User will be able to download the various details regarding bills. If he/she is a new user, he/she can download the new connection form, subscription details etc. then he/she can download his /her previous bill details in hands so as to ensure it.



IV. CONCLUSION

QUERY OPTIMIZATION: PRECOMPUTE VERTICAL AGGREGATION IN F_V (N =12M). TIMES IN SECONDS

n	d	SPJ		PIVOT		CASE	
		F	Fv	F	Fv	F	Fv
1K	7	552	62	121	58	119	59
	12	845	67	122	62	122	64
	25	1619	68	145	61	153	61
100K	7	540	86	131	81	247	81
	12	856	85	136	77	234	80
	25	1633	103	172	89	377	92
1.5M	7	669	230	538	157	242	155
	12	1051	419	751	140	322	141
	25	2776	1086	1240	161	384	150

Above table analyzes our first workload optimization, applied to three methods. Our goal is to assess the acceleration obtained by pre-computing 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 2X faster) we will use it by default. Notice that pre-computing FV takes the same time within each method. Therefore, comparisons are fair. We now evaluate optimization specific to the PIVOT operator. This PIVOT optimization is well-known, as we learned from SQL Server DBMS users groups. Following table 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.

QUERY OPTIMIZATION: REMOVE (TRIM) UNNECESSARY COLUMNS FROM F_V FOR PIVOT (N =12M). TIMES IN SECONDS.

n	d	trim=N	trim=Y
1K	7	282	58
	12	386	62
	25	501	61
1.5M	7	364	157
	12	408	140
	25	522	161

We have try to introduce a new class of extended aggregate functions, called horizontal aggregations which help preparing data sets for data mining and OLAP cube exploration.

- Specifically 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 subgrouping 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.

Using load balancing algorithms and techniques we try to improve the workload optimization

V. REFERENCES

[1]. J.A. Blakeley, V. Rao, I. Kunen, A. Prout, M. Henaire, and C. Kleinerman. .NET database programmability and extensibility in Microsoft SQL Server. In Proc. ACM SIGMOD Conference, pages 1087–1098, 2008.

[2]. 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.

[3]. H. Garcia-Molina, J.D. Ullman, and J. Widom. Database Systems: The Complete Book. Prentice Hall, 1st edition, 2001.

[4]. 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.

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

Cite this article as :

Prasanna M. Rathod, Prof. Dr. Anjali B. Raut, "Workload Optimization by Horizontal Aggregation in SQL for Data Mining Analysis", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 7 Issue 2, pp. 304-309, March-April 2021. Available at doi : https://doi.org/10.32628/CSEIT217263
Journal URL : https://ijsrcseit.com/CSEIT217263