www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 1 January 2015, Page No. 9784-9788

Proficient Techniques For The Enhancement Of Horizontal Aggregations In SQL

V Prasanti Parimala[1], *R Praveen Kumar*[2]

[1]**Student in Dept. of CSE (M.Tech)**
,Srinivasa Institute of Engineering and Technology
cheyyeru village,AP
prasantiparimala@gmail.com

[2]**Asst.Professor ,Dept.of CSE,**
Srinivasa Institute of Engineering and Technology
cheyyeru village,AP
sripraveen4u@gmail.com

Abstract- **The importance of datasets in data mining for knowledge discovery is universally recognized. Raw data sets that are available are not directly amenable for data mining analysis. Vertical and horizontal aggregations have been extensively used to transform the original data set for this purpose. However, vertical aggregations contribute only minimally for preparation of data sets in data mining analysis. Horizontally aggregated data sets are extensively used. Horizontal data sets are generated directly by using simple, yet powerful methods such as CASE, PIVOT, and SPJ. Basic SQL aggregations limitations to return one column per aggregated group using group functions is overcome by these three methods to generate aggregated columns in a horizontal tabular layout that are suitable for data mining analysis. Of these three methods SPJ employs relational operators to realize horizontal aggregations which are a better strategy in comparison to case and pivot. However SPJ's is weak in performance. In order to enhance the performance of SPJ method which improves horizontal aggregation performance in parallel, we proposed technique to improve SPJ methodology by using Join Enumeration strategies which includes a query tree generation with quantifier's algorithm. Then horizontal aggregations performance improvement is attempted using secondary indexes on common grouping columns. In conclusion we found that the above two changes improved Horizontal aggregations performance significantly and produce efficient dataset in horizontal format.**

Index Terms -- **Aggregations, SQL, Non inner joins, data set generations.**

I. INTRODUCTION

In data mining data sets play an important role in knowledge discovery. Usually data in real-time is not in well-organized form. So we have to prepare data sets suitable for data mining from that unorganized data. This data is stored and retrieved through front end applications

which use SQL to interact with relational databases. Producing datasets needs identification of similar data and then normalizing the tables. The aggregate functions supported by SQL are SUM, MIN, MAX, COUNT and AVG. These functions produce single value output. These are known as vertical aggregations where each function operates on the values of a domain vertically and produces a single value result. However, they can't be directly used in data mining analysis. Summary data sets can be prepared and they can be used further in data mining

978-1-4799-3064-7/14/$31.00 ©2014 IEEE

operations and in statistical algorithms [4][9]. Most of the data mining algorithms expect a data set with horizontal layout with many rows and one variable per column. Mining Algorithms like regression, classification, PCA and clustering require data set in summarized format [4]. Horizontal aggregations are capable of producing data sets that can be used for real world data mining activities, by using horizontal aggregate functions SPJ, PIVOT and CASE. The empirical results revealed that these operations are able to produce data sets with horizontal layout that is suitable for OLAP data cube [8] operations or data mining operations. Experimental results in [2] reveal that Performance of CASE, PIVOT was considered decent in terms of speed and scalability where as SPJ lags in speed and scalability metrics. So, it is important to enhance the functioning of existing native RDBMS methods such as SPJ rather than build new ones to incorporate mining and there is need to enhance the overall performance of horizontal aggregations. In this paper we proposed top down join enumeration algorithm (ES) which enhance performance of SPJ method and secondary indexing in order to improve performance of horizontal aggregations.

A. Paper Organization

This paper is organized as follows: Section 2 introduces horizontal aggregations which use three methods to evaluate Horizontal aggregations using existing SQL constructs. Section 3 presents proposed techniques for performance improvement of horizontal aggregations .Section 4 presents experimental results .Section 5 draws conclusions and directions for future work.

II. HORIZONTAL AGGREGATIONS

Horizontal aggregations are new class of aggregate functions that aggregate numeric expressions and transpose results to produce a data set with a horizontal layout. Horizontal aggregations are extension to traditional SQL aggregations, which return multiple values in a horizontal layout (similar to a multi-dimensional vector), instead of a single value per tuple. Horizontal aggregations produce a data set with a horizontal layout which is in summarized form (de normalized) that is suitable for many data mining algorithms. Horizontal aggregations were first proposed in paper [10] and later extended by using pivot operator in [2].

Tables that will be used to explain SQL queries are defined here. F is a input table as shown in table 1.

TABEL 1: Input Table, F

k	Store Id	product	A
1	3	Milk	9
2	4	Eggs	4
3	5	Milk	2
4	2	Bread	6
5	1	Bread	10
6	1	Bread	0
7	5	Eggs	11
8	4	Eggs	Null
9	2	Milk	8
10	1	Milk	7
11	3	Milk	6
12	2	Milk	5
13	2	Eggs	14

TABLE 2: Vertical Table, FV

Store Id	product	A
1	Bread	10
1	Milk	7
2	Bread	6
2	Milk	13
2	Eggs	14
3	Milk	15
4	Eggs	4
5	Eggs	11
5	Milk	2

TABLE 3: Horizontal Table, FH

Store Id	Count Bread	Count Milk	Count Eggs
1	10	7	Null
2	6	13	14
3	Null	15	Null
4	Null	4	Null
5	Null	2	11

Here F is a fact table which is in unorganized form that is similar to company's unorganized data. Fv is vertical table where the data is in vertical format which is not suitable for data mining algorithms like clustering classification and regression algorithms etc…,so in order to produce data in summarized form we go for horizontal aggregations which is suitable for data mining analysis. Table2 is in vertical format which returns single value per each row for respective column. Whereas Table3 has returned multiple values per each row for particular column which is somewhat similar to multidimensional vector .

There are three fundamental methods to compute horizontal aggregations.

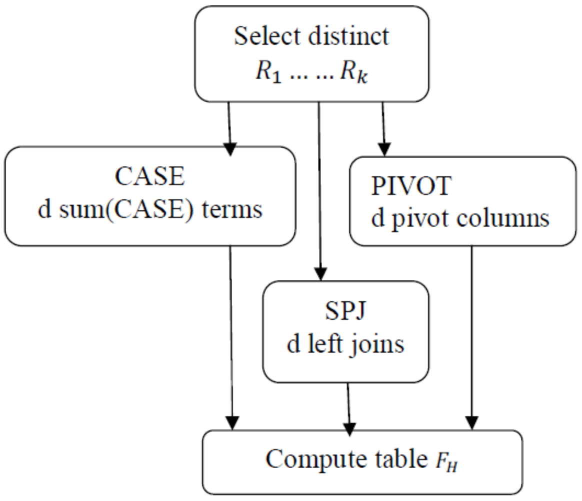


Fig 1 - Main steps of methods based on F.

A. SPJ method:

It is based on standard relational algebra to select project join. In SPJ we create one table with a vertical aggregation for each result column, and then join all those tables to produce F_H by using left outer joins. Then all such tables are joined in order to generate a table containing horizontal aggregations. The projected tables with Select-Project-Join-Aggregation queries are aggregated from input table F. It is necessary to introduce an additional table F0 that will be outer joined with projected tables to get a complete result data set.

The SPJ method example code to produce table 3 (F_H) is shown here (computed from F):

```
 /* SPJ method */
INSERT INTO F1
SELECT  Store Id,count(A) AS A
FROM F
WHERE product='Bread'
GROUP BY Store Id;
INSERT INTO F2
SELECT Store Id, count(A) AS A
FROM F
WHERE product='Milk'
GROUP BY Store Id;
 INSERT INTO F3
SELECT Store Id, count(A) AS A
FROM F
WHERE product='Eggs'
GROUP BY Store Id ;
INSERT INTO F_H
SELECT F0.Store Id,F1.A AS product Count Bread ,F2.A
AS product Count milk , F3.A AS product Count Eggs.

FROM F0 LEFT OUTER JOIN F1 on F0.Store Id=F1. Store
Id LEFT OUTER JOIN F2 on F0. Store Id =F2. Store Id;
```

B. CASE method

In this method the "case" programming construct which is available in SQL is used. The case statement returns a value selected from a set of values based on Boolean expressions. Horizontal aggregation queries can be evaluated by directly aggregating from F and transposing rows at the same time to produce F_H. First, we get the unique combinations of R1…. Rk, those define the matching Boolean expression for result columns.

The CASE method example code to produce table3 (F_H) is shown here (computed from F):

```
/* CASE method */
INSERT INTO FH
SELECT
Store Id
,COUNT(CASE WHEN product ='Bread ' THEN A
ELSE null END) as product count Bread
,COUNT(CASE WHEN product ='Milk' THEN A
ELSE null END) as product count Milk
,COUNT(CASE WHEN product ='Eggs' THEN A
ELSE null END) as product count Eggs

FROM F
GROUP BY Store Id;
```

C. PIVOT method

This aggregation is based on the PIVOT operator [7] available in RDBMS. Pivot transforms a series of rows into a series of fewer numbers of rows with additional columns. As it can provide transpositions, it will be used to evaluate horizontal aggregations. PIVOT operator determines how many columns are required transpose and it can be combined with GROUP BY clause and it rotates a table valued expression by turning the unique values from one column in the expression into multiple columns in the output. i.e. it rotates a rows to columns and aggregations where they are required on any remaining column values that are wanted in the final output.

Finally, the PIVOT method example code in SQL to produce table3 (F_H)is shown here (computed from F):

```
/* PIVOT method */
INSERT INTO FH
SELECT  Store Id
,[Bread] as product count Bread
,[Milk] as product count Milk
,[Eggs] as product count Eggs
FROM ( SELECT Store Id, Product, A FROM F
) as p  PIVOT (
COUNT(A)
FOR product  IN ([Bread], [Milk],[Eggs])
) as pvt;
```

III . PROPOSED TECHNIQUES

A. Top-Down Join Enumeration:

As discussed above SPJ has less speed and scalability compared to CASE and PIVOT .In SPJ we use left outer joins to compute horizontal aggregations. To improve the performance of SPJ we propose Join Enumeration strategies to perform left outer joins which increase the performance of horizontal aggregations .The algorithm includes a query tree generation with quantifiers algorithm, which has both relations referenced by the join predicate that are used to associate each join predicate and additional relations needed by a predicate to preserve the semantics of the original query.

a). Algorithm Based On Top-Down Enumeration

In a bottom-up optimizer dynamic programming smaller quantifier sets must be dealt before processing larger quantifier sets which is different from top-down optimizing. So according to the property of top-down optimizer an algorithm satisfying the top-down optimizing is shown in algorithm 1

Algorithm 1 the construction of ES

```
GetES
Input: original query tree G = (V,E) with quantifiers q={q1,…, qN }
Output:  a set contains the NS and ES  of each join predicate
1  Loop            //for each join predicate p
2    if ○p is inner join or antijoin
3      for each relation r ref(p)
4        s=s+ set_outerjoin(r);
5      for each member of s
6        set the set_outerjoin of the member to be s
7    if ○p is outerjoin
8      for each relation r in ref(nullproducting(p))
9        v=v+set_outerjoin(r);
10     set the ES of p to be v
11     for each relation r in ref(preserving(p))
12         u=u+ set_antijoin(r);
13     for each relation r ref(nullproducting(p))
14         add all the tables in u to the set_antijoin(r);
15   if ○p is antijoin
16      for each r in ref(preserving(p))
17        w=w+ set_antijoin(r);
18      set the ES of p to be w
```

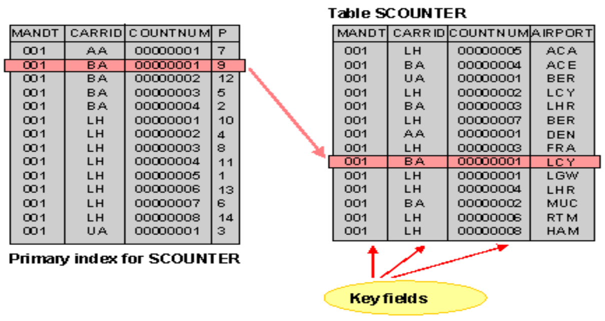
For convenience we use the following notations, NS includes relations referenced by the join predicate, which is used to associate each join predicate. An ES includes additional relations needed by predicate to preserve the semantics of the original query. We denote by the ref(preserving(p)) and ref(nullproducting(p)) are the set of table reference by the join predicate p in the preserving side and null producing side. The ref(p) denotes the set of tables referenced by the join predicate p. Anset_antijoin includes the relations that are linked to each relation through outer joins pointing to each relation. An set_outerjoin contains all the relations associated by inner join or anti join predicates. After initializing ES with NS for every join predicate p and an set_outerjoin and an set_antijoin for each table in the join to include only the table itself. Algorithm 1 sets the ES of outer join, anti join, inner join predicates.The construction of ES used various mechanisms, we propose to use query tree generation with quantifiers algorithm for constructing ES resulting in an optimized SPJ query results. This strategy Improves performance of SPJ along the lines of CASE, PIVOT.

B. Secondary Indexes:

An index whose search key specifies an order different from the sequential order of the file is secondary indexing also called non-clustering index or we can say secondary index provides a secondary means of accessing a file for which some primary access already exists. One may want to find all the records whose values in a certain field frequently (which is not the search-key of the primary index) satisfy some condition. So in order to satisfy that purpose we can have secondary indexes with an index record for each search-key value.

Index can be considered as a copy of a database table which is always in sorted form. This Sorting provides faster access to the data records of the table. The fields not contained in the index can also be read because index also contain corresponding record pointer of the actual table. Primary index is different from the secondary indexes of a table and it also contains the key fields of the table and a pointer to the non-key fields of the table and primary index will be automatically created automatically when the table is created in the database. In order to improve performance we have to use secondary indices. Consider an example where Table SCOUNTER of flight model contains the assignment of the airport carrier counters. The primary index on this table consists of pointer to the original data records and the key fields of the table as shown in fig 2.

We can also create more indexes on a table which are called as secondary indexes. This is required if the table is frequently accessed such that it does not take advantage of the sorting of the primary index for the access In this example all the carrier counters at a particular airport are often searched for bookings of flight. The airport ID is used to search for counters for such an access. Primary index sorting is of no use in speeding up this access. Table SCOUNTER has a large number of entries in order to support access effectively using the airport ID ,a secondary index on the field AIRPORT (ID of the airport) must be created as shown in fig 3.



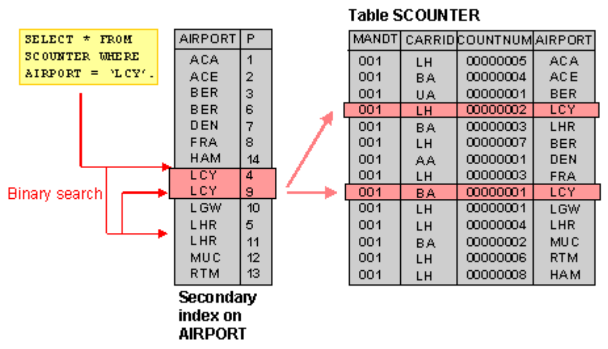
MANDT	CARRID	COUNTNUM	P
001	AA	00000001	7
001	BA	00000001	9
001	BA	00000002	12
001	BA	00000003	5
001	BA	00000004	2
001	LH	00000001	10
001	LH	00000002	4
001	LH	00000003	8
001	LH	00000004	11
001	LH	00000005	1
001	LH	00000006	13
001	LH	00000007	6
001	LH	00000008	14
001	UA	00000001	3

Primary index for SCOUNTER

Table SCOUNTER

MANDT	CARRID	COUNTNUM	AIRPORT
001	LH	00000005	ACA
001	BA	00000004	ACE
001	UA	00000001	BER
001	LH	00000002	LCY
001	BA	00000003	LHR
001	LH	00000007	BER
001	AA	00000001	DEN
001	LH	00000003	FRA
001	BA	00000001	LCY
001	LH	00000001	LGW
001	LH	00000004	LHR
001	BA	00000002	MUC
001	LH	00000006	RTM
001	LH	00000008	HAM

Fig .2. primary indexing



```
SELECT * FROM
SCOUNTER WHERE
AIRPORT = 'LCY'.
```

AIRPORT	P
ACA	1
ACE	2
BER	3
BER	6
DEN	7
FRA	8
HAM	14
LCY	4
LCY	9
LGW	10
LHR	5
LHR	11
MUC	12
RTM	13

Secondary index on AIRPORT

Table SCOUNTER

MANDT	CARRID	COUNTNUM	AIRPORT
001	LH	00000005	ACA
001	BA	00000004	ACE
001	UA	00000001	BER
001	LH	00000002	LCY
001	BA	00000003	LHR
001	LH	00000007	BER
001	AA	00000001	DEN
001	LH	00000003	FRA
001	BA	00000001	LCY
001	LH	00000001	LGW
001	LH	00000004	LHR
001	BA	00000002	MUC
001	LH	00000006	RTM
001	LH	00000008	HAM

Fig . 3. secondary indexing

So by using this secondary indexing technique we can improve overall performance of Horizontal aggregations.

IV.EXPERIMENTAL RESULTS

For the experimental studies the environment used for the development of prototype web based application that demonstrates the techniques to improve performance of horizontal aggregations includes MySQL v5 running on DBMS server , Core 2 duo processor, 2 GB of RAM and 250 GB on hard disk. The SQL code generator was programmed in the Java language and connected to the server via JDBC.

We used large real time data sets that are taken from wall mart ecommerce application. We analyzed queries having only one horizontal aggregation, with different grouping and horizontal columns. To implement three methodologies and top-down join enumeration algorithm we use java Programming language which is an object oriented high level programming language and java script and html is used to developed web based application to provide better GUI to user. The results showing the performance comparison is shown in figure 4 .

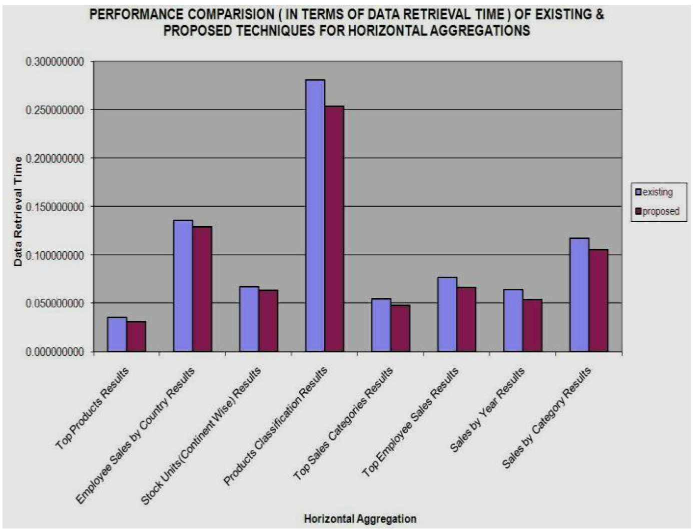


Fig . 4 .Graphical Comparison

Based on requirement of user we select various attributes like total product results, top sales results, product classification results etc..Results produced by existing methodology to produce horizontal aggregation and our proposed algorithm and secondary indexes are graphed. The results reveal that the top down join enumeration algorithm has improved performance of horizontal aggregations in aspect of data retrieval time.

V. CONCLUSIONS AND FUTURE WORK

Horizontal aggregations are used efficiently in decision making and providing a macroscopic view of entire business. These aggregations use a simple, yet powerful, mining methods (CASE, PIVOT, and SPJ) of RDBMS to generate aggregated columns in a horizontal tabular layout. Horizontal aggregations evaluated with the CASE method have similar performance to the built-in PIVOT operator but both CASE and PIVOT evaluation methods are significantly faster than the SPJ method. It is important to enhance the functioning of existing native RDBMS methods such as SPJ .So we propose Join Enumeration strategies and secondary indexing. With the help of join enumeration technique we perform left outer join in SPJ by following ES algorithm , which includes relations referenced by the join predicate that are used to associate each join predicate and also considering additional relations needed by a predicate to preserve the semantics of the original query.
We will apply Secondary indexes on common grouping columns that accelerate computation and performance in producing horizontal datasets that are suitable for efficient data mining analysis.

There are several research issues. The Aggregated results produced by a horizontal aggregation functions can be applied to data cube exploration for multi-dimensional view of data .Generated horizontal datasets can be applied to clustering algorithms like k-means and optimize the performance of K-means clustering algorithm which is most challenging task.

REFERENCES

1. WanliZuo,Yongheng Chen, Fenglin He and Kerui Chen,” Optimization Strategy of Top-Down Join Enumeration on Modern Multi-Core CPUs,” Journal of computers , Vol. 6, NO. 10, October 2011.
2. CarlosOrdonez, ZhiboChen,“Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining Analysis”, IEEE Trans. Knowledge and Data Eng, Vol. 24, No. 4, April 2012.

3 .Venky Harinarayan ,Ashish Guptay,”Generalized projections – A powerful query optimization technique”.

C. Ordonez, “Data Set Preprocessing and Transformation in a Database System,” Intelligent Data Analysis, vol. 15, no. 4, pp. 613-631, 2011.

4 .E.F. Codd, “Extending the Database Relational Model to Capture More Meaning,” ACM Trans. Database Systems, vol. 4, no. 4, pp. 397-434, 1979.

5. C.Galindo-Legaria and A. Rosenthal, “Outer Join Simplification and Reordering for Query Optimization,” ACM Trans. Database Systems, vol. 22, no. 1, pp. 43-73, 1997.
6. C.Cunningham, G. Graefe, and C.A. Galindo-Legaria, “PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS,” Proc. 13th Int’l Conf. Very Large Data Bases (VLDB ’04), pp. 998-1009, 2004.
7. J. Gray, A. Bosworth, A. Layman, and H. Pirahesh, “Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross- Tab and Sub-Total,” Proc. Int’l Conf. Data Eng., pp. 152-159, 1996.
8. C.Ordonez, “Statistical Model Computation with UDFs,” IEEE Trans Knowledge and Data Eng., vol. 22, no. 12, pp. 1752 - 1765,Dec. 2010. [10] C. Ordonez, “Integrating k- Means Clustering with a RelationalDBMS Using SQL,” IEEE Trans. Knowledge and Data Eng., vol. 18,no. 2, pp. 188-201, Feb. 2006..
9. C. Ordonez, “Horizontal Aggregations for Building Tabular Data Sets,” Proc. Ninth ACM SIGMOD Workshop Data Mining and Knowledge Discovery (DMKD ’04), pp. 35-42, 2004.

10 .Han and M. Kamber. Data Mining: Concepts and Techniques.MorganKaufmann,SanFrancisco,1stedition,2001.