Query Optimization Approach in SQL to prepare Data Sets for Data Mining Analysis

Rajesh Reddy Muley[1], Sravani Achanta[2], Prof.S.V.Achutha Rao[3]
[1]pursuing M.Tech(CSE), Vikas College of Engineering and Technology, Nunna, Vijayawada. Affiliated to JNTU-Kakinada, A.P, India

Sravani Achanta is working with Vikas College of Engineering and Technology, Nunna, Vijayawada, India.

[3] S.V. Achuta Rao is working as a Professor & Head,Department of CSE at Vikas College of Engineering and Technology, Nunna, Vijayawada, India.

***Abstract*—Collecting the information from various databases and presenting it to the user is a tough job and presenting the data as per the user requirement is even more tough because the user may need the data to be shown in many different ways. Moreover the data that has to be shown to the may not be present in one single database but may be present in more than one database tables. In normal way if the user is trying to access the details with very few requirements which are in a single table then there is no problem but if the data to be shown to user is present in many tables then the issue is to merge that data by using many queries for retrieval and then arrange as in the way expected by user. To overcome this drawback of presenting the data in much easier way and also reduce the overload on database we have data mining methods to solve the problem. This paper explains us the way to use the data mining methods to show the datasets by mining the data from different tables at the same time. The methods which are suitable for data mining analysis are CASE, SPJ and PIVOT. Coming with CASE we show two possibilities i.e. Vertical view and also the Horizontal View. This paper thus satisfies the main concern i.e. reducing the overload on the databases for retrieval of data.**

***Keywords*—Data base, Data Mining, CASE, PIVOT, SPJ.**

INTRODUCTION

Data Mining is the tool or a software which is used to retrieve the data from the large set of data. The overall goal of Data Mining is to extract information from a dataset and transform it into an understandable structure for further use. The actual data mining task is the automatic or semi-automatic analysis of large quantities of data to extract previously unknown interesting patterns such as groups of data records, unusual records and dependencies. This usually involves using database techniques such as spatial indices. These patterns can then be seen as a kind of summary of the input data, and may be used in further analysis or, for example, in machine learning and predictive analytics. For example, the data mining step might identify multiple groups in the data, which can then be used to obtain more accurate prediction results by a decision support system. Neither the data collection, data preparation, nor result interpretation and reporting are part of the data mining step. Data mining uses information from past data to analyse the result to be shown to the user.

Data Mining in real time is used everywhere, to extract the data. For example, if we have to search for a book in a room which contains some n books then no doubt we can get that book but the factor is how fast we can get that book from that room. For this effective output we need to implement data mining algorithms which help us to retrieve the desired book without wasting the time in searching that book. In the same manner we also have some data mining queries for retrieving the data from the huge database with less time and also reducing the burden on the database. This can be achieved using the methods like CASE, SPJ and PIVOT. All these three methods are useful and reduce the burden on the database and also fetch the result in less time thus making the application safe and flexible. The queries written above are used to fetch the results in smart way but the queries are complex and big. Something which has a benefit also has a drawback, the drawback is not affecting the designed system but is the complex type for writing the query based on the required output.

Creating a data set according to data mining requirements is time consuming and requires long SQL statements. Even though we are using automatically generated tool for data set, we have to customise dataset as per requirement. For this purpose we use joins and aggregations for creating data sets. We are focusing on aggregations. Most commonly used aggregations are sum of a column, average, maximum value, minimum value or row count and many aggregation functions and operations are available in SQL. All these aggregation functions have limitations like they returns scalar values. The

main reason is, datasets stored in database come from On-Line Transaction Processing (OLTP) which is highly normalized. But data mining and machine learning algorithms requires elaborated aggregated form. Significant effort is needed to compute the aggregations on multi table structure. Standard aggregations are difficult to edit when they are performing on multiple tables which are having high cardinalities in many resultant rows. To perform the analysis of mined databases on spreadsheets, it may be more flexible to having the aggregated functions on single group on single row. For example produce graphs, charts or compare datasets with repetitive information. OLAP tools generate SQL cod which transpose results more efficiently on aggregation and transposition mechanisms.

These horizontal aggregations produce added features of traditional SQL aggregations, which return a set of values in a horizontal layout, instead of a scalar values.

PROPOSED METHODOLOGY

Our proposed horizontal aggregation provides effective and easy process for creating datasets in data mining projects. Our aggregate functions creates a interface to generate SQL Code from data mining tools, these SQL code can be used to create SQL queries which are automatically created and used in data set formations.

In this section we are defining some SQL queries which are used in entire our proposed methods. Consider a table F having a simple primary key K represented by an integer, p distinct attributes and one numeric attribute: $F(K, D_1, D_2, D_3, \ldots\ldots, D_p, A)$. In OLAP terms, F is a fact table with one primary key column, p distinct columns and one measure column passed to standard SQL aggregations. That means we are manipulating table F as a cube with p dimensions. These dimensions are used to group the columns for aggregations. Another two important tables used in our proposed method is Vertical Table (F_V) and Horizontal Table (F_H).

Consider a SQL aggregation (e.g. sum()) with the GROUP BY clause, which returns results in a single row format. Let assume there are j + k GROUP BY columns and the aggregated attribute is A. The results are stored on table FV with j + k columns and making A as a non-key attribute. The goal of a horizontal aggregation is to transform FV into a table FH with a horizontal layout having n rows and j + d columns, where each of the d columns represents a unique combination of the k grouping columns. The n rows represent records for analysis and the d columns represent dimensions or features for analysis. Therefore, n is data set size and d is dimensionality. In other words, each aggregated column represents a numeric variable as defined in statistics research or a numeric feature as typically defined in machine learning research.

A. SQL Code Generation

In our proposed methodology key role is SQL Code Generation and transposition. Then we extend the SELECT statements with any of the traditional clause that generates the aggregate functions. Let's take a example an SQL statement that take a subset $L_1 \ldots\ldots L_m$ from$D_1 \ldots D_m$

SELECT $L_1 \ldots\ldots L_m$, sum (A) FROM F GROUP BY $L_1 \ldots\ldots L_m$;

This query will produce a table with m+1 columns with one group of each distinct combination of values and aggregated value. It takes three inputs table, grouping columns, aggregate column. We partition the GROUP BY list into two sub lists: one list to produce each group (m columns $L_1 \ldots\ldots L_m$) and another list (k columns$R_1 \ldots\ldots R_k$) to transpose aggregated values, where $\{L_1 \ldots\ldots L_m\}$ and $R_1 \ldots\ldots R_k$ doesn't have common values.Each distinct combination.

Horizontal aggregation maintains standards of SQL aggregation functionalities. The main difference is returning dimensions. Standard SQL aggregations return vertical dimensions results where our proposed functions returns horizontal dimensions.

B. Proposed Syntax

Here we are elaborating SQL aggregate functions with a extension of BY clause followed with a list of columns to produce horizontal set of numbers.

$$\text{SELECT } L_1, L_2 \ldots\ldots L_j, H(\text{A BY } R_1, \ldots R_k)$$
$$\text{FROM F GROUP BY } L_1 \ldots\ldots L_j$$

The sub group columns $R_1 \ldots\ldots R_k$ should be a parameter of aggregation. Here H() represents SQL aggregation. It contains at least one argument represented by A. The result rows are represented by $L_1 \ldots\ldots L_j$ in group by clause. And $(L_1 \ldots\ldots L_j) \cap (R_1 \ldots\ldots R_k) = \emptyset$.

We have tried to save SQL evolution semantics as possible. And also we have to make efficient evolution mechanisms. So we are proposing some rules

(1) GROUP BY clause is optional,
(2) If GROUP clause is present, there should not be a HAVING clause,
(3) Transposing BY clause is optional.
(4) When BY clause is present, horizontal aggregation reduces the vertical aggregation.
(5) Horizontal aggregation may combined with vertical aggregation or other horizontal aggregation.
(6) As for F does not changes horizontal aggregation can be combined.

C. SQL Code Generation

Here we will discuss on how to generate SQL code to create horizontal aggregation automatically. We discuss the structure of the result table and query optimisation methods. We propose evolution methods produce the same result.

1) Locking

It is mandatory to use locking in creating consistent query evolution. Main reason of inconsistency in query evolution is multiple insertions into tables F.

- It can create the extra columns in F_H
- It changes the number of rows F_H
- It changes aggregation values in F_H

In other words the SQL statements becomes long transaction. Horizontal aggregation can operate on static database without consistency problem.

2) Table Definition

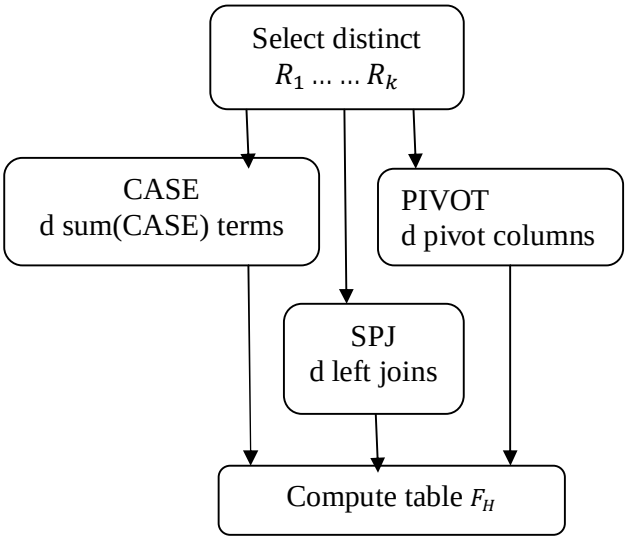
Let assume resultant table as F_H and having d aggregation columns with one primary key. The horizontal aggregation function H() will return set of values for each $L_1 \dots \dots L_j$. Then table $F_H$must contains a primary key column and non key column combination of values$R_1 \dots \dots . R_k$. we will get distinct values of $R_1 \dots \dots . R_k$ with help of

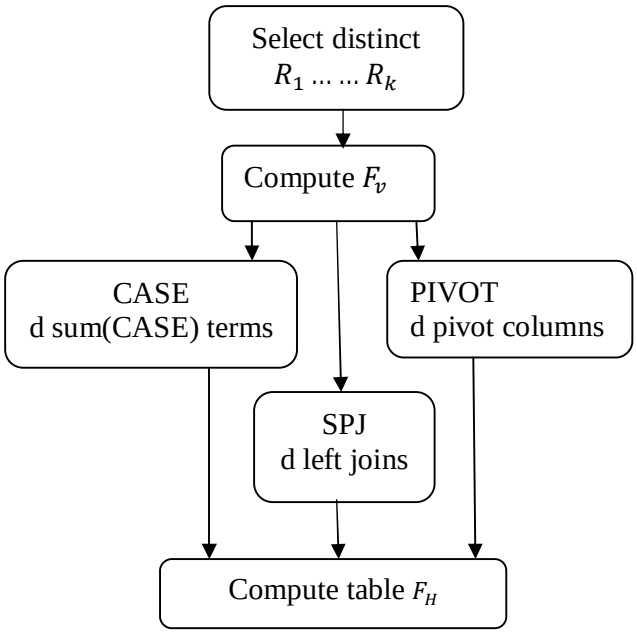
SELECT DISTINCT $R_1 \dots \dots . R_k$ FROM F

This SQL statement returns a table with d distinct rows where each row is used to define one aggregation column. It contains a primary key and j + d columns for data mining analysis.

D. Query Evolution Methods

In this paper we are proposing three evolution methods to horizontal aggregation. First method SPJ, which depends on relational operations that means select, join, retrieving and aggregations. Second is CASE, in this each table will contains primary key column to join process. The third method uses built in PIVOT operator which converts rows to columns.





1) SPJ Method

SPJ method is based on relational operations. In this method we create a table with vertical aggregation columns for each result column and join these tables to produceF_H. We aggregate from F into d projected tables with d. Each table contains primary key and aggregation on as only non-key column. We propose two basic methods to generate F_H.first method directly aggregates from F. Second method joins vertical aggregation in temporary table F_V with grouping columns to computeF_H.

Here we are proposing a indirect aggregation based on intermediate table F_V, which is used in both SPJ and CASE methods.

Assume $F_V$be a table contains vertical aggregations $L_1 \dots\dots L_j$, $R_1 \dots\dots R_K$ and V() be a vertical aggregation for H().

The query to compute F_v as

INSERT INTO F_v
SELECT $L_1 \dots\dots L_j$, $R_1 \dots\dots R_k$, V(A)
FROM F
GROUP BY $L_1 \dots\dots L_j$, $R_1 \dots\dots R_K$;

Table F_0 defined as a number of resultant rows and one primary key. So F_0 is combination of $L_1 \dots\dots L_j$.

INSERT INTO F_0
SELECT DISTINCT $L_1 \dots\dots L_j$
FROM $\{F|F_v\}$

Tables $F_1 \dots\dots F_d$ contains each aggregations of $R_1 \dots\dots R_k$. Primary key of table$L_1 \dots\dots L_j$.

INSERT INTO F_1
SELECT $L_1 \ldots\ldots L_j$,V(A)
FROM $\{F|F_v\}$
WHERE R_1=v_{1I} AND ...AND R_K=v_{kI}
GROUP BY L_1..... L_j.

Then Each table L_I aggregates those rows corresponds to I unique combinations given by WHERE clause. Finally we get $F_H$with help of joining d+1 table with d joins.

INSERT INTO F_H
SELECT
$F_0.L_1, F_0.L_2$ ……$F_0.L_j$
$F_1.A, F_2.A$..... $F_d.A$
FROM F_0.
LEFT OUTER JOIN F_1
ON$F_0.L_1$ =$F_1.L_1$ and……and $F_0.L_j$ =$F_1.L_j$
LEFT OUTER JOIN F_2
ON $F_0.L_1$ =$F_2.L_1$ and……and $F_0.L_j$ =$F_2.L_j$
…….
LEFT OUTER JOIN F_d
ON $F_0.L_1$ =$F_d.L_1$ and……and $F_0.L_j$ =$F_d.L_j$

Query may feel complex but evolution is efficient and effective.

2) CASE Method

The CASE statement returns scalar value which is selected from a set of values based on Boolean expressions. We propose same two methods as in SPJ to evaluate the F_H in CASE method also. First method aggregates directly where second method computes F_H from intermediate tablesF_v.

Here we are explaining direct aggregation process. Aggregation queries can generate directly from F and converts row tables to column tables at the same time to evaluate F_H. We need to find unique combinations of R_1.....R_k which matches the Boolean expression for result columns. Let assume V() is vertical SQL aggregation that has a CASE statement as argument. Here we need to make the result as null when there are no qualifying rows found for horizontal aggregation group.

SELECT DISTINCT R_1.....R_k
FROM F;

INSERT INTO F_H
SELECT L_1.....L_j
V(CASE WHERE R_1=v_{1I} AND ...AND R_K=v_{kI}
THEN A ELSE null END)
...
V(CASE WHEN R_1=v_{1d} AND ...AND R_K=v_{kd}
THEN A ELSE null END)
FROM F
GROUP BY $L_1, L_2, \ldots\ldots L_j$.
This SQL query generates aggregations in a single scan on F.

3) PIVOT Method

We think PIVOT operator is built in operator in a commercial DBMS. The PIVOT method requires to determine how many columns are needed to store the evaluated table and it can be joined to GROUP BY clause. Basic query syntax for PIVOT to compose horizontal aggregation assuming one by column for right key column.

SELECT DINSTINCT R_1
FROM F;

SELECT $L_1, L_{2,\ldots\ldots} L_j$, $v_1, v_2, \ldots\ldots v_d$
INTO F_H
FROM(
SELECT $L_1, L_{2,\ldots\ldots} L_j$,$R_1$,A
FROM F) F_t

PIVOT(V(A) FOR R_1 in ($v_1, v_2, \ldots\ldots v_d$
) AS P;

In this query nested query reduces F from columns that are not later needed.

EXPERIMENTAL EVOLUTIONS

Here we are providing our experimental evolutions on our proposed methods in the aspects of query optimization, time complexity, table size and output data dimensionality.

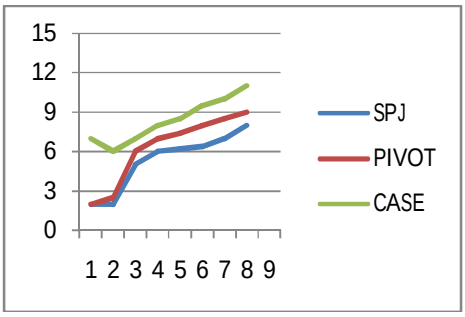
A. Query Optimization

We evaluated optimization strategies for aggregation queries with synthetic data sets generated by TPC-H generator. We used large synthetic data sets to analyse queries with only horizontal aggregation, which having different groups and horizontal columns.

Our goal is improve the accessibility obtained by pre computing a cube and storing it on F_v . we optimising to the PIVOT operator. Table I shows the impact of removing columns not needed by PIVOT.

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

B. Time Complexity



CONCLUSION

We have proposed a work for data mining analysis which helps the user to retrieve the data as their requirement in less amount of time and also reducing the burden on the system whereby accepting multiple inputs from user to perform the search and show the output. Here we proposed three basic methods to elaborate horizontal aggregation, first CASE method; it derives the complete CASE construct. Second SPJ, it is derived on standard relation algebra operations and third PIVOT, using this we can perform some DBMS offered operations. The CASE and PIVOT methods perform linear scalability, where SPJ does not perform. We performed the experimental evolutions on our proposed work with PIVOT and SPJ. We have implemented this concept on a real time client which is Electricity board, here the admin will be performing the data mining using all these queries on a particular user in that locality. Thus after implementation we can conclude that the usage of these queries reduces the burden on the database and processing will be made faster when compared to the normal process.

REFERENCES

G. Bhargava, P. Goel, and B.R. Iyer. Hypergraph based reordering of outer join queries with complex predicates. In *ACM SIGMOD Conference*, pages 304.315, 1995.

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.

J. Clear, D. Dunn, B. Harvey, M.L. Heytens, and P. Lohman. Non-stop SQL/MX primitives for knowledge discovery. In *ACM KDD Conference*, pages 425.429, 1999.

E.F. Codd. Extending the database relational model to capture more meaning. *ACM TODS*, 4(4):397.434, 1979.

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.

C. Galindo-Legaria and A. Rosenthal. Outer join simpli_cation and reordering for query optimization. *ACM TODS*, 22(1):43.73, 1997.

AUTHOR PROFILE

Rajesh Reddy Muley, Pursuing M.Tech(CSE) Vikas College of Engineering and Technology , Nunna, Vijayawada. Affiliated to JNTU-Kakinada, A.P., India

Sravani Achanta, is working with Vikas College of Engineering and Technology, Nunna, Vijayawada, Affiliated to JNTU-Kakinada, A.P., India

Prof S.V.Achutha Rao, is working as an HOD of CSE at Vikas College of Engineering and Technlogy, Nunna, Vijayawada, Affiliated to JNTU-Kakinada, A.P., India