Query Optimization using SQL Approach for Data Mining Analysis

M.Stella Inba Mary
PG Scholar
Department of CSE (PG)
National Engineering College, Kovilpatti

V.Kalaivani
Associate Professor
Department of CSE (PG)
National Engineering College, Kovilpatti

ABSTRACT

Relational databases are acceptable repository for structured data; integrating data mining algorithms with a relational DBMS is an essential research issue for database programmers. In a relational database, a significant effort is required to prepare a summary data set that can be used as input for the data mining process. It requires many complex SQL queries, joining tables and aggregating columns. This paper realizes the research on extending SQL code for data mining processing and related work on query optimization. Also the paper proposes the following approaches, transposition, pivoting and cross tabulation. The approaches exhibit efficient optimizations with SQL extensions using aggregated Queries.

Keywords

Relational DBMS, SQL, Aggregation, Query Optimization.

1. INTRODUCTION

The integration of data mining algorithms with a relational Data Base Management System (DBMS) is an important and challenging problem; a considerable effort is required to prepare an abstract of data set that can be used as input for a data mining models. Implementations of pivoting (horizontal layout) functionality already exist for the purpose of data presentation [8], but these operations are usually performed either outside the RDBMS or as a simple post-processing operation outside of query processing.

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. 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.

One of the primary goals of business intelligence is to transform raw data into meaningful information, by combining its sources, discovering its dependencies and patterns, and using them to predict future trends. The Data Mining Query task provides a means for capturing their outcome into an arbitrary table. Query execution leverages data mining models, which apply specifically crafted algorithms to data exposed via a mining structure.

This paper introduces a new set of aggregate functions that can be used to build data sets in a horizontal layout automating SQL query writing and extending SQL capabilities. In data mining, statistical or machine learning algorithms generally require aggregated data in summarized form [3]. Based on current available functions and clauses in SQL, a significant effort is required to compute aggregations when they are desired in a cross tabular (horizontal) form, suitable to be used by a data mining algorithm. This paper explains how to evaluate and optimize horizontal aggregations generating standard SQL code.

Figure 1 gives an example showing the input table, and a horizontal aggregated table. To compute queries like ”summarize sales for each store by each day of the week”; ”compute the total number of items sold by department for each store”. These queries can be answered with standard SQL, but additional code needs to be written or generated to return results in tabular form. Aggregations can be used to generate SQL code from a data mining tool to build data sets for data mining analysis.

Table 1.Input Table

N	S1	S2	SUM
1	3	A	9
2	2	B	6
3	1	B	10
4	2	B	0
5	2	A	1
6	1	A	Null
7	3	A	8
8	2	A	7

Table 2.Aggregated column for Input Table

S1	S2_A	S2_B
1	Null	10
2	8	6
3	17	Null

2. RELATED WORK

To prepare summarized format for data mining algorithm, many methods are introduced by researchers in the past. Carlos Ordonez [3] introduced three SQL implementations of the popular K-means clustering algorithm to integrate it with a relational DBMS. Xiaoxin Yin [14] proposed a new approach, called CrossMine, which includes a set of novel and powerful methods for multirelational classification.

Carlos Ordonez [2] focused on programming Bayesian classifiers in SQL. Carrasco [6] defined a new type of object dmFSQL consists of a series of operations on the object project (create, alter, drop…). The DML of dmFSQL executes the true DM process.

Elena Baralis[9] presented the IMine indx, a general and compact structure which provides tight integration of item set extraction in a relational DBMS. Charu C. Aggarwal[7] provided a survey of uncertain data mining and management applications. Sally McClean[11] considered the problem of aggregation using an imprecise probability data model. Conor Cunningham [8] described PIVOT and UNPIVOT, two operators on tabular data that exchange rows and columns.

Haixun Wang [14] implemented ATLaS, to develop complete data-intensive applications in SQL—by writing new aggregates and table functions in SQL, it includes query rewriting, optimization techniques and the data stream management module.Carlos Ordonez [1] 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

There exist many proposals that have extended SQL syntax. Programming three methods with SQL queries is explored in [5], which shows a horizontal layout of the data set enables easier and simpler SQL queries. Alternative SQL extensions to perform spreadsheet-like operations were introduced in [16]. 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.

The closest data mining problem associated to OLAP processing is association rule mining [17]. SQL extensions to define aggregate functions for association rule mining are introduced, In this case the goal is to efficiently compute itemset support.

3. PROPOSED WORK

3.1 Motivation:

The proposed work provides the small syntax extension to the SELECT statement, which allows understanding the proposal in an intuitive manner. The proposed extension represents non-standard SQL because the columns in the output table are not known when the query is parsed. The input table does not change while the aggregation is evaluated because new values may create new result columns. The new approach extends standard SQL aggregate functions with a "transposing" BY clause followed by a list of columns to produce a horizontal set of numbers instead of one number.

3.2 Extended SQL syntax:

```
SELECT L1….Lj , H(A BY R1…..Rk)
FROM F
GROUP BY L1…….Lj ;
```

- R1…..Rk - should be a parameter associated to the aggregation itself. That is they appear inside the parenthesis as arguments,
- H( ) represents some SQL aggregation (e.g. sum(), count(), min(), max(), avg()).

  The function H () must have at least one argument represented by A, followed by a list of columns.
- L1….Lj - The result rows are determined by these columns in the GROUP BY clause if present. Result columns are also determined by all potential combinations of columns R1; : : : ;Rk,

The proposal has the following rules.

- The GROUP BY clause is optional
- When the clause GROUP BY is present there should not be a HAVING clause.
- The transposing BY clause is optional.
- Horizontal aggregations can be combined with vertical aggregations or other horizontal aggregations.
- The argument to aggregate represented by A is required; A can be a column name or an arithmetic expression. In the particular case of count () A can be the .DISTINCT keyword.

3.3 Aggregated Table Definition:

```
CREATE TABLE FH ( L1 int …,Lj int
,X1 real…Xd real) PRIMARY KEY(L1……..Lj );
```

- Table FH that has {L1….Lj} as primary key and d columns corresponding to each distinct subgroup.
- FH has d columns for data mining analysis and j + d columns in total, where each Xj corresponds to one aggregated value based on a speci_c R1; : : : ;Rk values combination.

3.4 Discussion

In a data mining project most of the effort is spent in preparing and cleaning a data set. This effort involves deriving metrics and coding categorical attributes from the data set and storing them in a tabular form for analysis so that they can be used by a data mining algorithm. To get a consistent query evaluation, the SQL extension to use locking concepts.

The main reasons are that any insertion into table during evaluation may cause inconsistencies:

(1) it can create extra columns in output table, for a new combination of R1….Rk;
(2) it may change the number of rows of that table,
for a new combination of L1…..Lj ;
(3) it also may change actual aggregation values .
In order to return consistent answers, to use table-level locks on input and output tables, acquired before the first statement starts and released after table has been populated.

In other words, the entire set of SQL statements becomes a long transaction. Hence to use the highest SQL isolation level: SERIALIZABLE, an alternative simpler solution would be to use a static (read-only) copy during query evaluation. That is, aggregations can operate on a read-only database without consistency issues.

For all proposed methods to evaluate horizontal aggregations, to summarize common requirements,

(1) All methods require grouping rows using one or several queries.

(2) All methods must initially get all distinct combinations to know the number and names of result columns. Each combination will match an input row with a result column.

This step makes query optimization difficult by standard query optimization methods because such columns cannot be known when a horizontal aggregation query is parsed and optimized.

3.5 Proposed Methods

The main goal is to define a template to generate SQL code by combining aggregation and transposition. The proposal has two perspectives such as to evaluate efficient aggregations and perform query optimization. The first one includes the following approaches, pivoting, transposition and cross-tabulation.

Pivoting approach is a built-in method in a commercial DBMS. It can help evaluating an aggregated tabular format for summarized data set.

It perform the following steps,

The pivoting method is used to write cross-tabulation queries that rotate rows into columns, aggregating data in the process of the rotation. The output of a pivot operation typically includes more columns and fewer rows than the starting data set.

The pivot computes the aggregation functions specified at the beginning of the clause. Aggregation functions must specify a GROUP BY clause to return multiple values; the pivot performs an implicit GROUP BY.

New columns corresponding to values in the pivot, each aggregated value is transposed to the appropriate new column in the cross-tabulation.

The subclauses of the pivot have the following semantics:

expr - specify an expression that evaluates to a constant value of a pivot column.

Subquery – to specify a subquery, all values found by the subquery are used for pivoting. The subquery must return a list of unique values at the execution time of the pivot query.

ANY - The ANY keyword is used only in conjunction with the XML keyword. The ANY keyword acts as a wildcard and is similar in effect to subquery. The output is not the same cross-tabular format returned by non-XML pivot queries.

CUBE -The CUBE operation in the simple_grouping_clause groups the selected rows based on the values of all possible combinations of expressions in the specification. It returns a single row of summary information for each group. You can use the CUBE operation to produce cross-tabulation values.

The Transposition method is producing several rows for one input row. An important difference is that, compared to PIVOT, TRANSPOSE allows two or more columns to be transposed in the same query, reducing the number of table scans.

Cross-tabulations also called as crosstabs, are statistical reports that group data by one field, creating one column for each distinct value of another field. In colloquial terms, this way of representing data is called "breaking down the data by X and Y," where X and Y are the names of two columns in the dataset. In SQL crosstab produces a SQL query to cross-examine a database and generate a cross-tabulation report. The amount of parameters needed to achieve the result is kept to a minimum. In this approach, to indicate which columns and rows to cross and from which table(s) they should be taken. Compared to spreadsheet based cross-tabulations, SQL crosstab has two distinct advantages, i.e. it keeps the query in the database work space, fully exploiting the engine capabilities, and does not limit the data extraction to one table.

In order to evaluate the query optimization using the above approaches, the query optimizer takes three input parameters:

(1) The input table F,

(2) The list of grouping columns L1…, Lm;

(3) The column to aggregate (A).

The basic goal of a efficient aggregation is to transpose the aggregated column A by a column subset of L1, . . . , Lm;

A SQL statement can be executed in many different ways, such as full table scans, index scans, nested loops, and hash joins. The query optimizer determines the most efficient way to execute a SQL statement after considering many factors related to the objects referenced and the conditions specified in the query. This determination is an important step in the processing of any SQL statement and can greatly affect execution time.

The optimizer first evaluates expressions and conditions containing constants as fully as possible. The optimizer determines the goal of optimization. For a join statement that joins more than two tables, the optimizer chooses which pair of tables is joined first, and then which table is joined to the result.

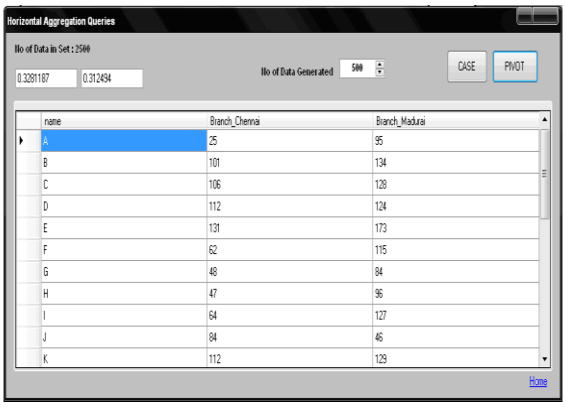
The SQL Server Query Optimizer is a cost-based optimizer. It analyzes all methods for a given query, Therefore, it is the SQL Server component that has the biggest impact on the performance of any real time databases applications. After all, selecting the right execution method could mean the difference between a query execution time of milliseconds, and one of minutes or even hours. Naturally, a better understanding of how the Query Optimizer works can help both database administrators and developers to write better queries and to provide the Query Optimizer with the information it needs to produce efficient execution plans.

4. EXPERIMENTAL RESULTS

The proposed methods are implemented in the commercial data base applications. To execute the efficient aggregated queries for these large data sets and to evaluate optimization strategies for aggregation queries with synthetic data sets, provide the results as follows.

The analyzed queries have aggregation with different grouping and cross tabulations. Finally, to evaluate query optimizations, compare the query evaluation methods with time complexity with dimensionality.

Fig 2a show the summarized sales information for the products in two branches. The SQL query with Pivoting and Transposition produces the same results but they has the different time complexity.


Horizontal Aggregation Queries

No of Data in Set : 2500

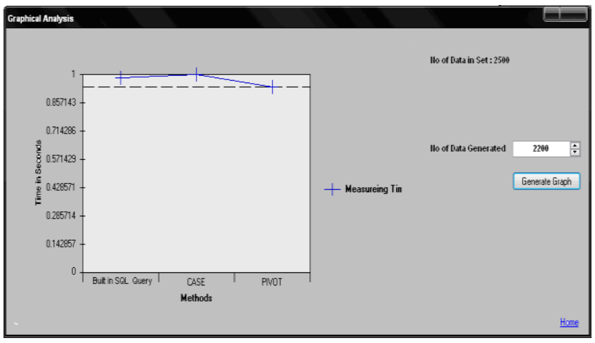
0.3281187 0.312494

No of Data Generated 500

name	Branch_Chennai	Branch_Madurai
A	25	95
B	101	134
C	106	128
D	112	124
E	131	173
F	62	115
G	48	84
H	47	96
I	64	127
J	84	46
K	112	129

Fig 2 a) Query Evaluation for Data Sets

The main task of optimization is to assess the acceleration obtained by precomputing a cube This optimization provides a different gain, depending on the methods. Fig 2b shows the time optimization between built in SQL query and the proposed approaches. The Select and Join optimization is best for small n, for pivoting for large n and for CASE there is rather a less dramatic improvement all across n.


No of Data in Set : 2500

No of Data Generated 2200

Fig 2b) Query Optimization

Based on the time complexity, time grows as n grows for all methods. Hence n is the main performance factor for PIVOTING and Transpositions methods, d is the data set dimensionality (number of cross-tabulated aggregations). It is used to evaluate the query. This analysis considers every method precomputes FV.

In existing system, there exist two DBMS limitations with horizontal aggregations: reaching the maximum number of columns in one table and reaching the maximum column name length when columns are automatically named .A horizontal aggregation can return a table that goes beyond the maximum number of columns in the DBMS when the set of columns {R1, . . .,Rk} has a large number of distinct combinations of values, or when there are multiple horizontal aggregations in the same query. On the other hand, the second important issue is automatically generating unique column names. If there are many subgrouping columns R1, . . .,Rk or columns are of string data types, this may lead to generate very long column names, which may exceed DBMS limits. it will be difficult or impossible to compute a data mining model.

In the new approach, the large column name length can be solved. The problem of d going beyond the maximum number of columns can be solved by vertically partitioning FH so that each partition table does not exceed the maximum number of columns allowed by the DBMS. Evidently, each partition table must have L1, . . . , Lj as its primary key. Alternatively, the column name length issue can be solved by generating column identifiers with integers and creating a "dimension" description table that maps identifiers to full descriptions, but the meaning of each dimension is lost. An alternative is the use of abbreviations, which may require manual input.

5. CONCLUSION

The proposed approaches implements an abstract but minimal extension to SQL standard aggregate functions to compute efficient summarized data set which just requires specifying sub grouping columns inside the aggregation function call. From a query optimization perspective,

The proposed system describes the possibility of extending SQL OLAP aggregations with horizontal layout capabilities. Horizontal aggregations produce tables with fewer rows, but with more columns. The aggregated tables are useful to create data sets with a horizontal layout, as commonly required by data mining algorithms and OLAP cross-tabulation.

The output of a query optimization can immediately be applied back to the data gathering, transformation, and analysis processes. Anomalous data can be detected in existing data sets, and new data entry can be validated in real time, based on the existing data. SQL Server Data Mining contains multiple algorithms that can perform churn analysis based on historical data. Each of these algorithms will provide a probability.

In future, research issues is proposed on extending SQL code for data mining processing. The related work on query optimization is proposed and compared to horizontal aggregations with alternative proposals to perform transposition or pivoting.

It includes to develop more complete I/O cost models for cost-based query optimization and to study optimization of horizontal aggregations processed in parallel in a shared-nothing DBMS architecture.

5. REFERENCES

[1] Carlos Ordonez," Statistical model computation with UDFs", IEEE Transactions on Knowledge and Data Engineering, vol. 22, no.22, pp. 1752 - 1765, Dec. 2010.

[2] Carlos Ordonez, Pitchaimalai. S.K, "Bayesian Classifiers Programmed in SQL", IEEE Trans. Knowledge and Data Eng, vol. 22, no. 1, pp.909-921, Jan. 2010.

[3] Carlos Ordonez, "Integrating K-means clustering with a relational DBMS using SQL" IEEE Trans. Knowledge and Data Eng, vol. 18 no. 2, pp.181-201, Feb. 2006

[4] Carlos Ordonez, Omiecinski. E, “Efficient Disk-Based K-Means Clustering for Relational Databases”, IEEE Trans. Knowledge and Data Eng., vol. 16, no. 8, pp.909-921, Aug. 2004.

[5] Carlos Ordonez, Zhibo Chen, “Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining Analysis”, IEEE Trans. Knowledge and Data Eng., vol. PP, no. 99, Jan. 2011.

[6] Carrasco, R.A.; Vila, M.A.; Araque, F.,” dmFSQL: a Language for Data Mining”, DEXA '06. 17th International Workshop on 2006, pp-440-444, 2006

[7] Charu C. Aggarwal, Philip S. Yu. “A Survey of Uncertain Data Algorithms and Applications”, IEEE Transactions on Knowledge and Data Engineering, Vol. 21, No. 5. pp. 609-623, May 2009.

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

[9] Elena Baralis, Tania Cerquitelli, Silvia Chiusano, "IMine: Index Support for Item Set Mining," IEEE Transactions on Knowledge and Data Engineering, vol. 21, no.4, pp 493-506, April 2009

[10] Hendrik Decker, “Inconsistency – Tolerant Integrity Checking”, IEEE Transactions on Knowledge and Data Engineering, Vol. 23, No. 2., pp- 218 – 234, Feb 2011

[11] McClean, S. Scotney, B. Shapcott, M. “Aggregation of Imprecise and Uncertain nformation in Databases”, Knowledge and Data Engineering, IEEE Transactions , Vol. 13, No. 6, pp 902 – 912, Nov/Dec 2001 .

[12] Netz, A, Chaudhuri. S, Fayyad. U, Bernhardt. J, "Integrating Data Mining with SQL Databases: OLE DB for Data Mining",17th International Conference on 2001, pp.379-387, 2001

[13] Pitchaimalai, S., Ordonez, C., Garcia-Alvarado, C., “Efficient Distance computation Using SQL Queries and UDFs”, IEEE HPDM (High Performance Data Mining Workshop, at ICDM), 2008.

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

[15] Yin, X. Han, J. Yang, J. Yu, P.S. “Efficient Classification across Multiple Database Relations: A Cross Mine Approach” IEEE Trans. Knowledge and Data Eng., vol. 18, no. 6, pp. 770-783, Jun. 2006.

[16] A. Witkowski, S. Bellamkonda, T. Bozkaya, G.Dorman, N. Folkert, A. Gupta, L. Sheng, and S.bramanian. “Spreadsheets in RDBMS for OLAP” In Proc. ACM SIGMOD Conference, pages 52–63, 2003.

[17] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: lternatives and implications.’ In Proc. ACM SIGMOD Conference, pages 343–354, 1998