Summary of "Data Cube: A Relational Aggregation Operator Generalizing GROUP BY, Cross-Tab and Sub-Totals"

2011-03-02

Gray et al. introduce the CUBE operator as an N-dimensional generalization of the GROUP BY operator and SQL aggregation functions.

The GROUP BY is a relational aggregation operator that partitions a relation into disjoint sets of tuples. The sets are then aggregated using a function. In the SQL Standard five aggregate functions are provided: COUNT(), SUM(), MIN(), MAX(), and AVG(). Many SQL systems are also extended to provide support for other aggregate functions, such as median, standard deviation and variance. Some systems like Informix Illustra allow users to implement their own aggregation functions by implementing the call-back functions Init(&handle), Iter(&handle, value), and value=Final(&handle). The Init function allocates and initializes a new handle for the aggregation. The Iter function aggregates the next value with the current value. The Final function computes the resulting aggregate with the data saved in the handle and deallocates the handle.

Certain common forms of data analysis are difficult using the GROUP BY operator. Using the GROUP BY operator poses three different problems:

1. There is no support for the GROUP BY operator using grouping functions as opposed to dimensions. The authors refer to grouping functions as histograms.

2. Rolling-up totals or subtotals and drilling down into the data create representations with empty cells that are not allowed in relational databases. These problems can be solved with different relation representations, such as using cross tabulations. However, many of them tremendously increase the dimensionality of the tables. Also, expressing roll-up and cross-tabulation operations with conventional SQL are difficult tasks.

3. GROUP BYs are also inadequate when used for all possible groupings because the resulting aggregation is too complex to analyze for optimization. For example, for six columns, a 64-way union of different GROUP BY operators would be necessary to build the desired representation. On most SQL systems the union of GROUP BYs would result in a long wait of 64 scans of the data and 64 sorts or hashes.

The CUBE operator is equivalent to the union of all possible group-by operations that can be performed on a set of attributes. To do this, the power set of the columns participating in the CUBE operations is generated. For example, for columns A, B, and C, the power set would be , A, B, C, A, B, B, C, A, C, A, B, C. The CUBE operator returns a table with padding values named 'all' for the dimensions whose values are not grouped. The value 'all' represents the set of all possible values for a given dimension. The resulting CUBE is represented as a relation of (the product of $C_i + 1$) rows. The cardinality of each dimension, C_i, is incremented by 1 because of the new extra value 'ALL' that is added to pad cells which represent the entire set of elements in a dimension. Alternatively, the cells having the entire set of elements of a dimension can be padded with the 'NULL' value, but this may create ambiguity if the dimension allows NULLs.

Gray et al. also explore the ROLLUP operator, which works especially well for running sums or running averages, as it is naturally sequential. In contrast, the CUBE operator is naturally non-linear (multi-dimensional). The ROLLUP operator is the union of the groupings that can be formed with the different prefixes of the dimensions specified. For example: A, B, C, A, B, A, .

The authors extend SQL to support combinations of GROUP BY, ROLLUP and CUBE. These are organized in a compound order where the "most powerful" operator, CUBE, is at the core, followed by ROLLUP and GROUP BY as shown: GROUP BY ¡select list¿ ROLLUP ¡select list¿ CUBE ¡select list¿ Gray et al. also extend the SQL syntax to support decorations. Decorations are attributes that are functionally dependent on the attribute that is being aggregated. The selection of decorations is not allowed in Standard SQL. The simplest and slowest approach for computing the cube is the 2^n-algorithm which makes $T \times 2^n$ calls to a function. Here, T is the cardinality of the base table, and n is the number of dimensions. It is often faster to compute the CUBE operation from other sub-aggregations. However, this possibility depends on the classification of the aggregate function used. The aggregate functions are classified as follows:

1. Distributive functions: These functions can be decomposed in sub-aggregate functions and their computation does not depend on the entire set of the input. Some example algebraic functions include: COUNT(), MIN(), MAX() and SUM().

2. Algebraic functions: These functions can be decomposed in sub-aggregate functions but one needs to keep track of two or more components of the aggregation. An example is the average; where for each sub-aggregate the sum and count are kept. Some example algebraic functions include: Standard Deviation and Center of Mass.

3. Holistic functions: These functions cannot be decomposed into sub-aggregate functions and their computation depends on the entire set of the input. Some example holistic functions include: Median(), Mode(), and Rank().

More efficient algorithms than the 2^n-algorithm exist when using distributive and algebraic functions. However, for holistic aggregation functions, the authors state that they do not know of more efficient algorithms than the 2^n algorithm.