Evolution of Data Cube Computational Approaches

Rebecca Boon-Noi Tan
Monash University, Australia

INTRODUCTION

Aggregation is a commonly used operation in decision support database systems. Users of decision support queries are interested in identifying trends rather than looking at individual records in isolation. Decision support system (DSS) queries consequently make heavy use of aggregations, and the ability to simultaneously aggregate across many sets of dimensions (in SQL terms, this translates to many simultaneous group-bys) is crucial for Online Analytical Processing (OLAP) or multidimensional data analysis applications (Datta, VanderMeer, & Ramamritham, 2002; Dehne, Eavis, Hambrusch, & Rau-Chaplin, 2002; Elmasri & Navathe, 2004; Silberschatz, Korth & Sudarshan, 2002).

BACKGROUND

Although aggregation functions together with another operator, Group-by in SQL terms, have been widely used in business application for the past few decades. Common forms of data analysis include *histograms*, *roll-up totals and sub-totals for drill-downs* and *cross tabulation*. The three common forms are difficult to use with these SQL aggregation constructs (Gray, Bosworth, Lyaman, & Pirahesh, 1996). An explanation of the three common problems: (a) *Histograms*, (b) *Roll-up Totals and Sub-Totals for drill-downs*, and (c) *Cross tabulation* will be presented.

Firstly the problem with *histograms* is that the SQL standard Group-by operator does not allow a direct construction with aggregation over computed categories. Unfortunately, not all SQL systems directly support histograms including the standard. In standard SQL, histograms are computed indirectly from a table-valued expression which is then aggregated. However, the cube-by operator is able to have direct construction. The histogram can be easily obtained from the data cube using the cube-by operator to compute the raw data. The histogram on the right-hand side of Figure 1 shows the sales amount as well as the sub-totals and total of a range of the

Figure 1. An example showing how a histogram is formed from results obtained from a data cube

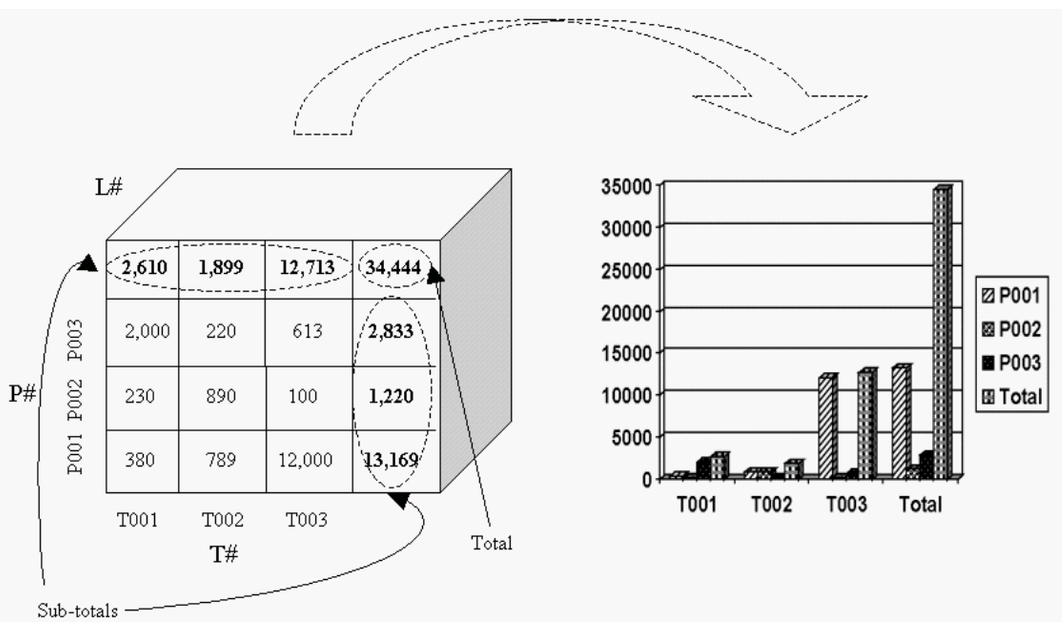
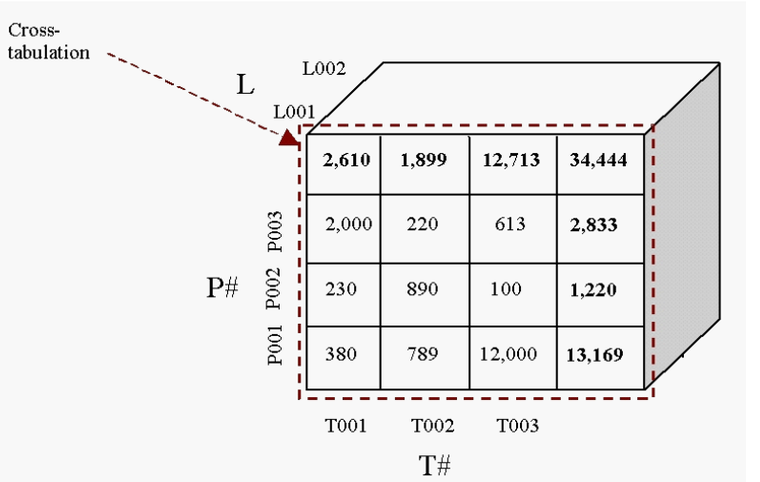


Figure 2. An example of cross-tabulation from a data cube



products at a range of the time-frame on a particular location.

The second problem relates to *roll-ups totals* and *sub-totals for drill-down*. Reports commonly aggregate data initially at a coarse level, and then at successively finer levels. This type of report is difficult with the normal SQL construct. However, the Cube-by operator is able to present the *roll-ups totals* and *sub-totals for drill-down* easily.

The third problem relates to cross-tabulation which is difficult to construct with the current standard SQL. The symmetric aggregation result is cross-tabulation table or cross tab for short (known as a pivot-table in spreadsheets). Using the cube-by operator, cross tab data can be readily obtained which is routinely displayed in the more compact format as shown in Figure 2. This cross tab is a two dimensional aggregation within the red-dotted line. If we add another location such as L002, it becomes a 3D aggregation.

In summary, the problem of representing aggregate data in a relational data model with the standard SQL can be a difficult and daunting task. A six dimensional cross-tab will require a 64 way union of 64 different group-by operators in order to build the underlying representation. This is an important reason why the use of Group-bys is inadequate as the resulting representation of aggregation is too complex for optimal analysis.

MAIN THRUST

Birth of cube-by operator

To overcome the difficulty with these SQL aggregation constructs, (Gray et al., 1996) proposed using data cube operators (also known as cube-by) to conveniently support such aggregates. The data cube is identified as the core operator in data warehousing and OLAP (Lakshmanan, Pei & Zhao, 2003). The cube-by operator computes group-bys corresponding to all possible combinations in a list of attributes. An example of data cube query is as follows:

SELECT Product, Year, City, SUM(amount)
FROM Sales
CUBE BY Product, Year, City

The above query will produce the SUM of amount of all tuples in the database according the 7 group-bys, i.e. (Product, Year, City), (Product, Year), (Product, City), (Year, City), (Product), (Year), (City). Lastly, the 8th group-by denotes as ALL, which contains an empty attribute so as to make all group-bys results union compatible. For example, a cube-by of three attributes (ABC) in data cube query will generate eight or 2^3 group-bys of ([ABC], [AB], [AC], [BC], [A], [B], [C] and [ALL]).

The most straightforward way to execute the data cube query is to rewrite it as a collection of eight group-by queries and execute them separately as shown in Figure 3. This means that the eight group-by queries will need to access the raw data eight times. It is likely to be quite expensive in execution time. If the number of dimension attributes increases, it becomes very expensive to compute the data cube. This is because the required computation cost grows exponentially with the increase of dimension attributes. For instance, 'N' number of dimension attributes of cube-by will form a 2^N number of group-bys. However, there are a number of ways in which this simple solution can be improved.