Dynamic Multidimensional Data Cubes for Interactive Analysis of Massive Datasets[1]

Mirek Riedewald
Cornell University, USA

Divyakant Agrawal
University of California, Santa Barbara, USA

Amr El Abbadi
University of California, Santa Barbara, USA

INTRODUCTION: DATA WAREHOUSING, OLAP, AND DATA CUBES

Rapidly improving computing and networking technology enables enterprises to collect data from virtually all its business units. The main challenge today is to extract useful information from an overwhelmingly large amount of raw data. To support complex analysis queries, data warehouses were introduced. They manage data, which is extracted from the different operational databases and from external data sources, and they are optimized for fast query processing. For modern data warehouses, it is common to manage Terabytes of data. According to a recent survey by the Winter Corporation (2003), for instance, the decision support database of SBC reached a size of almost 25 Terabytes, up from 10.5 Terabytes in 2001 (Winter Corporation, 2001).

Human analysts cannot "digest" such large amounts of information at a detailed level. Instead they rely on the system to provide a summarized and task-specific view of selected data. Consequently, *efficient summarization and aggregation of large amounts of data* play a crucial role in the analysis process. The goal of Online Analytical Processing (OLAP) is to support this style of analysis at interactive response times for massive data collections. OLAP applications often maintain aggregate information in data cubes. Pre-computed aggregate values that speed up query execution for group-bys, cross-tabs, and subtotals can be easily included in the model (e.g., CUBE operator, which is discussed later).

WHY MAKE DATA CUBES DYNAMIC

The primary goal of data warehouses is to support data analysis. Update costs were often not considered to be important, and hence systems are typically oriented towards batch updates, which are applied to the warehouse during times of low system load. The corresponding data cubes would be batch-updated as well, or even be recomputed from scratch. There are several arguments why incrementally maintainable *dynamic* data cubes are preferable.

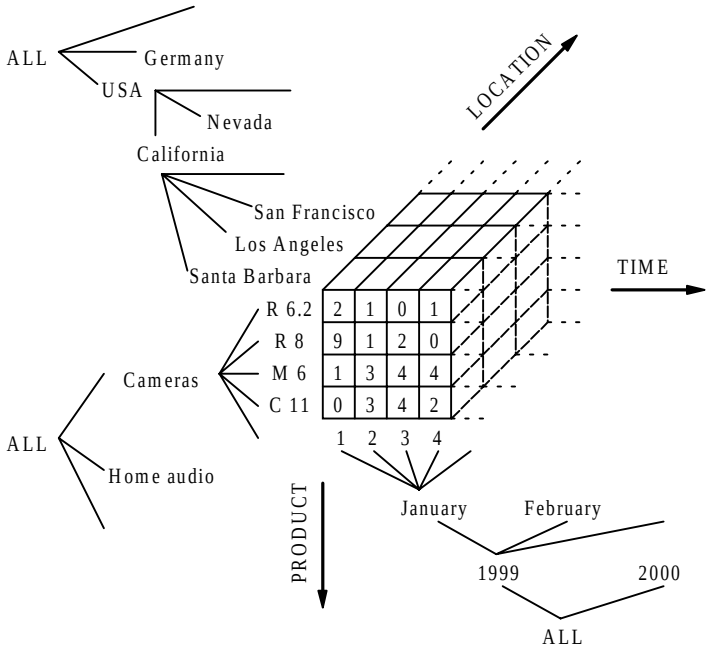
First, for very large data collections, re-computing a data cube from scratch is very costly, even if updates are batched. Second, some data structures that are heavily optimized towards query processing (e.g., the PS technique as discussed below) support only very limited update rates even if the updates are applied in batches. Third, OLAP is inherently an *interactive* information exploration process that includes what-if scenarios. What-if analysis requires real-time and hypothetical data to be integrated with historical data for the purpose of instantaneous analysis and subsequent action.

This article surveys dynamic solutions that offer different tradeoffs between query, update, and storage costs, allowing an administrator to find the matching approach for each application. It will be shown that efficient queries and updates can both be supported for multidimensional data cubes.

BACKGROUND

The terms "cube" and "data cube" are not used consistently in the data warehousing research literature. In this article they refer to a conceptual model similar to a multidimensional array. A simple example is a sales data cube SalesCube with dimensions product, time, and location, and the measure attribute amount (value of sales transaction). Figure 1 shows a possible instance with dimension hierarchies indicated (ALL represents the whole domain of an attribute). Note that we do not advocate any

Figure 1. Cube example



particular cube implementation (Roussopoulos, Kotidis & Roussopoulos, 1997; Sarawagi & Stonebraker, 1994) or system architecture like MOLAP or ROLAP.

Dominant aggregate query types on data cubes are roll-up/drill-down and slice-and-dice (Chaudhuri & Dayal, 1997). These queries are typically *range queries* or sequences of related range queries like hierarchical range queries (Koudas, Muthukrishnan & Srivastava, 2000). Hence we are mainly concerned with support for *aggregate range queries*. For the cube in Figure 1, an example is: "Compute the total sales of cameras in California for the first half of January 1999." The corresponding SQL query is

```
SELECT SUM(amount)
FROM SalesCube
WHERE time>='1-Jan-1999' AND time<='15-Jan-1999' AND
location.state='California' AND
product.category='Cameras'.
```

Note that location is a *categorical dimension* for which arbitrary range selections like "San Francisco < town < San Diego" are not meaningful. On the other hand it is very common to aggregate according to the dimension hierarchy. In the example, California corresponds to a range of selected cities (by making sure the dimension values are ordered according to the hierarchy). The same holds for other categorical dimension attributes like product.

For the following discussion we assume that the data cube has N cells in each dimension, that is, it has a total of N^d cells. Note that all techniques also work for cubes with different dimension sizes, but since our goal is to present the main tradeoffs of the techniques, we try to avoid notational clutter as much as possible. For the same reason we also omit ceiling and floor brackets from the formulas.

DATA CUBES FOR INVERTIBLE AGGREGATE OPERATORS

For invertible aggregate operators like SUM, COUNT, and AVG (when expressing it with SUM and COUNT), elegant techniques have been developed that do not require additional storage compared to materializing the original (dense) MOLAP cube. In the following we will survey techniques for the operator SUM. Other invertible operators are handled in a similar way.

Let A denote the original data cube. The cost of an operation is measured in terms of accessed cube cells, for example, updating a single cell in A results in a cost of 1. Answering an aggregate range query using A can be very