# Storage Strategies in Data Warehouses

**Xinjian Lu**
*California State University, Hayward, USA*

## INTRODUCTION

A data warehouse stores and manages historical data for on-line analytical processing, rather than for on-line transactional processing. Data warehouses with sizes ranging from gigabytes to terabytes are common, and they are much larger than operational databases. Data warehouse users tend to be more interested in identifying business trends rather than individual values. Queries for identifying business trends are called analytical queries. These queries invariably require data aggregation, usually according to many different groupings. Analytical queries are thus much more complex than transactional ones. The complexity of analytical queries combined with the immense size of data can easily result in unacceptably long response times. Effective approaches to improving query performance are crucial to a proper physical design of data warehouses.

One of the factors that affect response time is whether or not the desired values have been pre-computed and stored on the disk. If not, then the values have to be computed from base data, and the data has to be both retrieved and processed. Otherwise, only data retrieval is needed, resulting in better query performance. Storing pre-computed aggregations is a very valuable approach to reducing response time. With this approach, two physical design questions exist:

- Which aggregations to pre-compute and store?
- How to structure the aggregations when storing them?

When processing queries, the desired data is read from disk into memory. In most cases, the data is spread throughout different parts of the disk, thus requiring multiple disk accesses. Each disk access involves a seek time, a rotational latency, and a transfer time. Both seek time and rotational latency are setup times for an actual retrieval of data. The organization of data on the disk thus has a significant impact on query performance. The following is another important question:

- How to place data on the disk strategically so that queries can be answered efficiently?

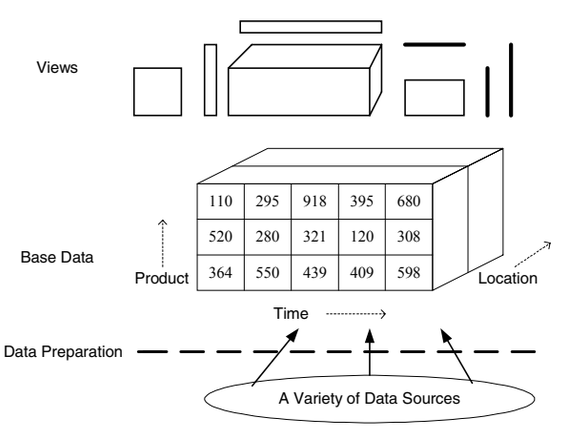
This chapter presents notable answers from existing literature to these questions, and discusses challenges that remain.

## BACKGROUND

In a data warehouse, data is perceived by users and often presented to users as multidimensional cubes (Chaudhuri & Dayal, 1997; Datta & Thomas, 1999; Vassiliadis & Sellis, 1999). Attributes are loosely classified as either independent or dependent. Together the values of independent attributes determine the values of dependent ones (Date, 2000). The independent attributes form the dimensions of the cube by which data is organized. Each dependent attribute is known as a measure. These dimensions can be used as addresses for looking up dependent values, similar to the way coordinates describe objects (points, lines and planes) in a Cartesian coordinate system. Values of a dependent attribute are also called fact values. As an example, we use sales (e.g., dollar amount in thousands) as a measure. Each fact value indicates the sales during a specific time period at a particular store location for a certain type of product. See Figure 1 for a multidimensional presentation of some sample base data. Base data is extracted from various transactional sources.

In addition to the base data, users are also interested in aggregated sales. For example, what are the total (or: average, minimum, and maximum) sales across all loca-

*Figure 1. An Illustration of data sources, base data, and views*



tions in each day, for every product name? With the SQL language, this question can be answered using "Group By Day, ProductName". A large number of groupings exist; and each of them may be requested. The result of a grouping is also known as a view, or summary table (Ramakrishnan and Gehrke, 2003, pp. 870-876). A view can be computed from the base cube when requested, which would involve both reading the base data as well as computing the aggregates. Performance can be improved through pre-computing and physically storing the view. When a view is pre-computed and stored, it is called a materialized view (Harinarayan, Rajaraman, & Ullman, 1996). A view is also perceived as a data cube. A view has a smaller size than that of the base cube, so less time would be needed to read it from disk to memory. Further, if the query can be answered using values in the materialized view without any aggregating, the computing time can be avoided. However, it most cases, the number of possible groupings can be so large that materializing all views becomes infeasible due to limited disk space and/or difficulties in maintaining the materialized views.

It should be emphasized that although data is interpreted as multidimensional cubes, it is not necessarily stored using multidimensional data structures. How data is physically stored also affects the performance of a data warehouse. At one end of the spectrum, both base cube and materialized views are stored in relational databases, which organize data into tables with rows and columns. This is known as ROLAP (relational OLAP). At the other end of the spectrum, both the base cube and materialized views are physically organized as multidimensional arrays, an approach known as MOLAP (multidimensional OLAP). These two approaches differ in many ways, with performance and scalability being the most important ones. Another approach, known as HOLAP (hybrid OLAP), tries to balance between ROLAP and MOLAP, and stores all or some of the base data in a relation database and the rest in a multidimensional database (Date, 2000).

Regardless of how data is physically structured (ROLAP, MOLAP or HOLAP), when answering a query, data has to be read from a disk. In most cases, the desired data for an analytical query is a large amount, and it is spread over noncontiguous spots, requiring many disk accesses for one query. Each disk access involves a significant setup time. Hence, data should be partitioned and placed strategically on the disk so that the expected number of disk accesses is reduced (Lu & Lowenthal, 2004).

## MAIN THRUST

### Which Views to Materialize?

Users may navigate along different dimensions, or explore at different levels in the same dimension (for example date, month, quarter, or year in a Time dimension), trying to discover interesting information. It would be ideal to materialize all possible views. This approach would give the best query performance. For a data cube with $N$ dimensions, if the attribute from each dimension is fixed, there are $2^N - 1$ different views. Using [Date, City, Brand] from the dimensions shown on Figure 1, the following views are the possible groupings:

- Date, City, Brand
- Date, City
- Date, Brand
- City, Brand
- Date
- City
- Brand
- (none)

Moreover, a dimension may have one or more hierarchical structures. Replacing Date with Month, another set of views can be generated; and still another set can be generated when Quarter is picked from the Time dimension. It becomes evident that the number of all possible views can be very large, so will be the disk space needed to store all of them.

In a data warehouse environment, new data is loaded from data sources into the data warehouse periodically (daily, weekly, or monthly). With an updated base cube, the materialized views must also be maintained to keep them consistent with the new base data. Reloading the base cube and maintaining the views take a certain amount of time, during which the data warehouse becomes unavailable. This time period must be kept minimal (Mumick, Quass, & Mumick, 1997; Roussopoulos, Kotidis, & Roussopoulos, 1997).

Due to the space limit and time constraint, materializing all possible views is not a feasible solution. To achieve a reasonable query performance, the set of views to materialize must be chosen carefully. As stated in Ramakrishnan and Gehrke (2003, p. 853), "In current OLAP systems, deciding which summary tables to materialize may well be the most important design decision."

Harinarayan, Rajaraman and Ullman (1996) have examined how to choose a good set of views to materialize based on the relational storage scheme. It is assumed that the time to answer a query is proportional to