The Soul of the Data Warehouse, Part 1: Drilling Down

By Ralph Kimball

Although data warehouses come in many shapes and sizes and deal with many different subject areas, every data warehouse must embody a few fundamental themes. The three most important are drilling down, drilling across, and handling time. Modern data warehouses so deeply embed these three themes that I think an "if-and-only-if" relationship has developed between them and a real data warehouse. If a system supports drilling down, drilling across, and handling time, then as long as it's easy to use and runs fast, it automatically qualifies as a data warehouse. But as simple as these three themes might seem, they give rise to a set of detailed and powerful architectural guidelines that should not be compromised.

Drilling down, drilling across, and handling time are so important that I'll devote a separate column in this Fundamentals series to each. In this column, I drill down into drilling down, starting with a precise operational definition. Then, as a good engineer should, I lay out practical guidelines for building systems that do a good job of drilling down.

Drilling Down

Drilling down in a relational database means "adding a row header" to an existing `SELECT`statement. For instance, if you're analyzing the sales of products at a manufacturer level, the select list of the query reads `SELECT MANUFACTURER, SUM(SALES)`. Of course the rest of the query contains join specifications and constraints on other parts of the database, such as time and geography. If you wish to drill down on the list of manufacturers to show the brands sold, you add the `BRAND` row header: `SELECT MANUFACTURER, BRAND, SUM(SALES)`. Now each manufacturer row expands into multiple rows listing all the brands sold. This is the essence of drilling down.

Incidentally, we often call a row header a "grouping column" because everything in the select list that's not aggregated with an operator such as `SUM` must be mentioned in the SQL `GROUP BY`clause. So the `GROUP BY` clause in the second query reads `GROUP BY MANUFACTURER, BRAND`. Row headers and grouping columns are the same thing.

This example is particularly simple because it's very likely that, in a dimensional star schema, both the manufacturer attribute and the brand attribute exist in the same product dimension table. So, after running the first query at the manufacturer level, you look at the list of attributes in the product dimension and opportunistically drag the brand attribute into the query. Then you run it again, thereby drilling down in an ad hoc way. If the brand attribute is indeed in the same dimension table as the manufacturer attribute, then the only adjustments to the original SQL are to add `BRAND` to the select list and the `GROUP BY` clause.

You could just as well have selected the color attribute for drilling down rather than the brand attribute. In fact, if you substitute the word `COLOR` for `BRAND` in the preceding paragraphs, they would be just as valid. This exercise powerfully illustrates the fact that drilling down has nothing to do with descending a predetermined hierarchy. In fact, once you understand this concept, you see that you can drill down using any attribute drawn from any dimension! You could just as well have drilled down on the weekday from the time dimension; the preceding discussion of the select

list and the `GROUP BY` clause would still be identical.

The idea that you can expand any report row to show more detail simply by adding a new row header is one of the powerful ideas that form the soul of a data warehouse. A good data warehouse designer should always be thinking of additional drill-down paths to add to an existing environment. An example of this out-of-the-box thinking is to add an audit dimension to a fact table. The audit dimension contains indicators of data quality in the fact table, such as “data element out of bounds.” But this audit dimension can be part of the drill-down process! Now you can devise a standard report to drill down to issues of data quality, including the proportion of questionable data. By drilling down or data quality, each row of the original report would appear as multiple rows, each with a different data quality indicator. Most of the report results should cluster under the “normal” row headers.

Finally, it’s even possible to drill down with a calculation, as long as you’re careful not to use an aggregate operator such as `SUM` in the calculated quantity. You could drill down on the price point of the manufacturer’s sales by adding `SALES/QUANTITY` to the select list, where this price point calculation contains no aggregate operator. `SALES` and `QUANTITY` are both numeric facts in the fact table. The select list now reads `SELECT MANUFACTURER, SALES/QUANTITY, SUM(SALES)`. You must also add `SALES/QUANTITY` to the `GROUP BY` clause. This expression replaces each original manufacturer row with the sales of each manufacturer’s products at each observed price in the marketplace. Each row shows the price point as well as the total sales at that price point.

Building a System That Works

I can now make some precise technical comments about drilling down:

1. Drilling down is the most basic end-user maneuver in the data warehouse, which must support it in as general and flexible a manner as possible because there’s no way to predict the user’s drill-down path. In other words, every drill-down path must be available and supported with the same user interface gestures because the users see little conceptual difference between the various forms of drilling down described in the preceding examples.

2. The data warehouse must therefore support drilling down at the user interface level, at all times, with the most atomic data possible because the most atomic data is the most dimensional. The most atomic data is the most expressive; more dimensions are attached to atomic data than to any form of aggregated or rolled-up data.

3. Combining the first two points means that for all practical purposes, the atomic data must be in the same schema format as any aggregated form of the data: The atomic data must be a smoothly accessible target for drill-down paths using standard ad hoc query tools. Failure in this area is the showstopper for a weird architecture you’ll occasionally hear about, one in which atomic data is hidden in the back room in an entity-relation physical format, and somehow is accessed after “drilling through” aggregated data marts in dimensional formats. The proponents of this architecture have never explained how this magic occurs and you could conclude they’ve never implemented a system that used commercial query tools to drill down from aggregated data to atomic data. Fortunately, this crisis evaporates if you use the same data structures at all levels of aggregation including the atomic level. See the next point.

4. To build a practical system for drilling down, you want standard ad hoc query tools to present the drill-down choices without special schema-dependent programming, and you want these tools to emit the correct resulting SQL without schema-dependent programming. Schema-dependent programming is the kiss of death for a data warehouse shop, because it means that each schema requires custom-built applications. This problem was a crisis on the 1980s, and there’s no excuse for it to remain a problem now. Avoiding schema-dependent programming means choosing a standard schema methodology for all end-user-facing data sets. I call these end-user-facing data sets the presentation layer of the data warehouse.

5. Only one standard schema methodology exists that’s capable of expressing data in a single, uniform format that looks the same at the atomic layers as in all aggregated layers, and at the same time requires no schema-dependent programming: the star schema, otherwise known as the dimensional model. Star schemas support all forms of drilling

down described in this column. All possible many-to-one and many-to-many data relationships are capable of representation in a star schema, thus the star schema is the ideal platform for ad hoc querying.

6. The star schema design in the presentation layer smoothly supports prebuilt aggregation tables and snowflake designs. Aggregations and snowflakes are related. An aggregated fact table is a mechanically derived table of summary records. The most common reason to build aggregated fact tables is that they offer immense performance advantages compared to using the large, atomic fact tables. But you get this performance boost only when the user asks for an aggregated result! The first example query asking for the manufacturer sales of products was a good example of an aggregated result.

A modern data warehouse environment uses a query-rewrite facility called an aggregate navigator to choose a prebuilt aggregate table whenever possible. Each time the end user asks for a new drill-down path, the aggregate navigator decides in real time which aggregate fact table will support the query most efficiently. Whenever the user asks for a sufficiently precise and unexpected drill down, the aggregate navigator gracefully defaults to the atomic data layer.

Drilling down is probably the most basic capability that a data warehouse needs to support. Drilling down most directly addresses the natural end-user need to see more detail in an interesting result.