The Soul of the Data Warehouse, Part 2: Drilling Across

By Ralph Kimball

The three fundamental themes that make up the soul of every data warehouse are drilling down, drilling across, and handling time. In Part One of "The Soul of the Data Warehouse," I showed that drilling down was nothing more than adding a row header, *any row header*, to an existing query. Although we often grouse about SQL's limitations as a report writer, when it comes to drilling down, SQL gracefully expands an existing query whenever a new row header is added. This simple result led to the powerful realization that when data is organized in a symmetrical, predictable fashion starting at the most atomic level, all queries and applications benefit.

If drilling down is the most fundamental maneuver in a data warehouse, drilling across is a close second. From the perspective of an answer set, drilling across *adds more data to an existing row.* Note that this result isn't what you get from a `UNION` of rows from separate queries. It's better described as the column accretion from separate queries.

Drilling across by adding another measured fact to the `SELECT` list from the existing fact table mentioned in the query is a trivial accomplishment. What's more interesting and important is adding another measured fact from a new fact table.

The issues raised by this simple view of drilling across are at the heart of data warehouse architecture. These issues boil down to an observation and a choice.

Drill-across observation: The new fact table called for in the drill-across operation must share certain dimensions with the fact table in the original query.

Certain dimensions will be named in the original query because they contribute row headers. Remember that these row headers are the basis of the grouping that creates the answer-set row. These dimensions will appear in the `FROM` clause of the SQL code and will be joined to the fact table through the relationship of a foreign key to primary key. The new fact table must also support exactly these same row headers, or the context of the answer-set row is meaningless.

Drill-across choice: Either send a single, simultaneous SQL request to the two fact tables or send two separate requests.

Although sending a single SQL request to the two fact tables seems cleaner, this choice can become a showstopper. Sending a single request means mentioning both fact tables in the `FROM`clause of the SQL code and joining both fact tables in some way to the common dimension tables I just discussed. This commingling of two fact tables in the same SQL statement causes these problems:

- Because the two fact tables will be joined together either directly or through the common dimensions, the query must specify whether the many-to-many relationship between the two fact tables is handled with inner or outer joins. This fundamental challenge arises from the relational model. It's effectively impossible to get this right, even if you're an expert SQL programmer. Depending on the relative cardinality of the two fact tables, your aggregated numeric totals can either be too low or too high, or both! Even if you don't believe me, you have to deal with the next bullet point.

- The vast majority of queries the relational database receives are generated by powerful query tools and report writers, and you have no direct control over the SQL they emit. You don't *want* control over the SQL. Some of these tools generate mind-boggling reams of SQL and you can't effectively intervene.
- Emitting a single SQL statement precludes you from requesting data from separate table spaces, separate machines, or separate database vendors. You're stuck in the same table space on the same machine talking to one database vendor. If you can easily avoid this problem, why take on these restrictions?
- Finally, if you emit a single SQL statement involving both fact tables, you'll almost certainly be unable to use any of the powerful query-rewrite tools that perform *aggregate navigation*.

Implementing Drill-Across

If you've followed the logic of the observation and the choice, the architecture to support drill-across begins to emerge. I use some modern data warehouse words to describe the two key aspects of this architecture:

1. All fact tables in a drill-across query must use *conformed dimensions*.
2. The actual drill-across query consists of a *multi-pass* set of separate requests to the target fact tables followed by a simple *sort/merge* on the identical row headers returned from each request.

The simplest definition of conformed dimensions is that two instances of a conformed dimension are identical. So if two fact tables have a "Customer" dimension, then "Customer" is conformed if the two dimensions are exactly the same. But this definition is unnecessarily restrictive. Here's the precise definition of conformed dimensions: *Two dimensions are conformed if the fields that you use as common row headers have the same domain.*

When you bring two separate queries together in a drill-across operation, both queries must have the same number of row headers, arranged from left to right in the same order. All the rest of the columns (the computed facts) in the two queries, by definition, are not row headers. In other words, an independent examination of both queries shows that neither query has rows that duplicate the same row headers. To put it another way, the row headers form a unique key for each row of the answer set.

To sort/merge the two queries, you must sort them the same way. At this point, it becomes possible to merge the rows of the two queries together in a single pass. The resulting merged answer set has a single set of row headers plus the combined set of computed facts returned from both queries. Because traditional sort/merge is the same as an outer join, it is possible for a row in the final merged answer set to have nulls for either the first set of computed facts or the second set, but not both!

Once you've visualized the sort/merge step in drilling across, you really understand conformed dimensions. With conformed dimensions, the only thing you care about is matching row headers. If the contents of the respective fields you're using for the sort/merge are drawn from the same domains, then the match makes sense. If you try to match row headers from two dimensions that aren't conformed, you're guaranteed to get garbage. The sort/merge will fail, and the SQL engine will post the results from the two queries on separate lines — and probably in separate sorting locations in the merged answer set.

Amazing Magic

In my classes, I sometimes describe conformed dimensions as either dimensions that are exactly equal (the trivial case) or dimensions where "one is a subset of the other." For example, a brand dimension may be a subset of a more detailed product dimension. In this case, you can drill across two fact tables, one at the brand level with a brand dimension (such as a forecast), and the other at a detailed product level with a product dimension (such as sales

transactions). Assume that the product dimension is a nice flat table containing the low cardinality brand attribute.

If the row header of the two queries is simply “brand,” then some amazing magic takes place. The engine automatically aggregates both data sets to the brand level, which is exactly the right level for the drill-across. If the names of the brands are drawn from the same domain, you can complete the drill-across (of forecast vs. actual) by confidently merging the rows with the same brand names. Many commercial query and report-writing tools perform this drill-across operation.

You can see that it’s possible to conform two dimensions even when they have some incompatible fields. You just need to be careful to avoid using these incompatible fields as row headers in drill-across queries. Not avoiding it lets a dimension contain some private fields that are meaningful only to a local user group.

Centralizing vs. Conforming

One final thought: In a recent column, my colleague Margy Ross pointed out that the decision to physically centralize a data warehouse has very little to do with conformed dimensions. Her point was that if you combine two data sets in a drill-across operation, you have to label them the same way, which is true whether the data sets are tightly administered on a single piece of hardware by one DBA or are loosely administered by remote IT organizations that merely agree to create a set of overlapping labels.