Fistful of Flaws

By Margy
Ross

People often engage us to conduct dimensional model design reviews. In this column, I'll provide a laundry list of common design flaws to scout for when performing a review. I encourage you to use this list to critically review your own draft schemas in search of potential improvements.

What's the Grain?

When a data warehouse team proudly unrolls its draft dimensional modeling masterpiece, one of our first questions is "What's the grain of the fact table?" We need to know the specific level of detail captured in the fact table. Surprisingly, we often get inconsistent answers to this inquiry. Declaring a clear and concise definition of the fact table grain is critical to a productive modeling effort. Without agreement, the design team and business liaisons will spin in circles.

For maximum flexibility and extensibility, you should build your fact table at the lowest level of granularity possible. You can always roll up the granular details. On the other hand, there's no way to drill down into the details if you've only loaded preaggregated, summary information. Obviously, the lowest level of granularity depends on the business process being modeled.

Mixed-Grain or Textual Facts?

Once you have established the fact table granularity, identify the facts that are consistent with this grain. If some facts are line-item level metrics, while others exist only at the header level, you must either allocate the header facts to the line item grain or create a separate fact table with the granularity of one row per header.

Fact tables typically consist of foreign keys plus numeric counts and amounts measuring business performance. Optimally, the facts are additive, meaning they can be summed across any dimension. In an effort to improve performance or reduce query complexity, aggregated facts such as year-to-date totals sometimes sneak into the fact row. These totals are dangerous because they aren't perfectly additive. A year-to-date total may reduce the complexity and run time of a few queries, but having it in the fact table invites other queries to double count the year-to-date column (or worse) when more than one date is included.

You should also prohibit text fields, including cryptic indicators and flags, from entering the fact table. They almost always take up more space in the fact table than a surrogate key. More important, business users generally want to query, constrain, and report against these text fields. You can provide quicker responses and more flexible access by handling these textual values in a dimension table, along with additional descriptive roll-up attributes often associated with the indicators or flags.

Dimension Descriptors and Decodes?

Any identifiers and codes in the dimension tables should be accompanied by descriptive decodes. It's time for us to dismiss the misperception that business users prefer to work with codes. If you need to convince yourself, just stroll down to their offices to see the decode listings filling their bulletin boards or lining their monitors. Adding descriptive names makes the data more legible to business users. If required by the business, operational codes can accompany the descriptors as dimension attributes, but they shouldn’t be the dimension table primary keys.

Design teams sometimes opt to embed complex filtering or labeling logic in the data access application rather than supporting it via a dimension table. Although query and reporting tools may let you decode within the application, we recommend that decodes be stored as data elements instead. Applications should be data-driven in order to minimize the impact of decode additions and changes. Placing decodes in the dimensions ensures greater report labeling consistency.

Handling of Hierarchies?

Each dimension associated with a fact table should take on a single value with each fact row. Similarly, each dimension attribute should take on one value for each dimension row. If the attributes have a one-to-many relationship, then this hierarchical relationship can be represented within a single dimension. You generally should look for opportunities to collapse dimension hierarchies whenever possible, except in the case of really large dimensions with highly volatile attribute changes. It isn’t uncommon to represent multiple hierarchies in a single dimension.

Designers sometimes attempt to deal with the dimension hierarchies within the fact table. For example, rather than having a single foreign key to the product dimension, they include fact table foreign keys for the key elements of the product hierarchy, such as brand, category, and department. Before you know it, a compact fact table turns into an unruly monster, joining to dozens of dimension tables. This example is a severe case of having “too many dimensions.” If the fact table has more than 20 foreign keys, you should look for opportunities to combine or collapse them into dimensions.

In general, we discourage snowflaking, or normalizing, dimension tables. Snowflaking may reduce the disk space needed for dimension tables, but the savings are usually insignificant when compared with the entire data warehouse and seldom offset the disadvantages in ease of use or query performance.

Outriggers are a variation of the snowflake theme. Rather than normalizing the entire dimension, a cluster of relatively low-cardinality or frequently reused attributes is placed in an outrigger joined to the dimension. Instead, dimensions should be a single join away from the fact table in most cases. Be careful to avoid abusing the outrigger technique; outriggers should be the exception rather than the rule. Similarly, if your design is riddled with bridge tables to capture many-valued dimension relationships, you need to go back to the drawing board. Chances are that you have an issue with the fact table’s granularity.

Explicit Date Dimension?

Every fact table should have at least one foreign key to an explicit date dimension. The SQL date function doesn’t support date attributes such as fiscal periods, seasons, and holidays. Rather than trying to determine these nonstandard calendar calculations in a query, you should store them in a date dimension table.

Designers sometimes avoid a date dimension altogether by representing a series of monthly buckets of facts on a single fact table row. These fixed time slots often become an access and maintenance nightmare. The recurring time buckets should be presented as separate rows in the fact table instead.

Control Numbers as Degenerate Dimensions?

In transaction-oriented fact tables, treat the operational control numbers (such as the purchase order or invoice number) as degenerate dimensions. They reside as dimension keys on the fact table, but don’t join to a corresponding dimension table.

Teams are sometimes tempted to create a dimension table with information from the operational header record, such as the transaction number, transaction date, transaction type, or transaction terms. In this case, you'd end up with a dimension table that has nearly as many rows as the fact table. A dimension table growing at nearly the same pace as the fact table is a warning sign that a degenerate dimension may be lurking within it.

Surrogate Keys?

Instead of relying on operational keys or identifiers, you should use meaningless surrogate keys for all the dimension primary keys and fact table foreign keys. The administrative overhead to manage surrogate keys is minor, while the benefits are multifold. They isolate the warehouse from operational changes (such as recycling closed account numbers), while letting the warehouse handle "not applicable" or "to be determined" conditions. Because surrogate keys are typically four-byte integers, performance is improved due to the smaller fact keys, smaller fact tables, and smaller indices.

Surrogate keys let you integrate data with multiple operational keys from multiple sources. They are also required to support the dominant technique for tracking changes to dimension table attributes.

Slowly Changing Dimension Strategies?

Your dimensional design isn't complete until you have identified a slowly changing dimension strategy for each dimension attribute. You may opt to overwrite the column, add a new row, add a new column, or even add a new dimension to track changes.

It's important that the strategy, or combination of strategies, is well thought out before development occurs.

Well-Understood Business Requirements?

Not to sound like a broken record, but there's no way to effectively conduct a design review without first having a solid understanding of your business requirements. You need to be keenly aware of both the business requirements and data realities to review a dimensional model with any sense of confidence. Business subject matter experts or liaisons are typically excellent guides along this path; you can't expect to take any shortcuts.

This column describes our primary areas of focus during a dimensional design review. We strongly encourage you to use this list of questions to assess your own dimensional models. Here's hoping you'll pass with flying colors!