Design Tip #171 Unclogging the Fact Table Surrogate Key Pipeline

By Joy Mundy

We characterize the ETL system as a back room activity that users should never see nor touch. Even so, the ETL system design must be driven from user requirements. This *Design Tip* looks at the design of one bit of ETL plumbing – the fact table surrogate key pipeline – from the perspective of business users' needs.

The surrogate key pipeline is #14 of the 34 ETL subsystems described by Ralph Kimball in *The Data Warehouse Toolkit, Third Edition*. There's an overview of the ETL subsystems on our website, along with an article by Bob Becker. The surrogate key pipeline is usually the last step of the fact table processing in which the ETL system exchanges source system keys for data warehouse surrogate keys.

For the most part, implementation of the pipeline is a detail safely left to the ETL team. They will decide whether to stage the fact data first, and whether to use an ETL tool widget, lookups, or database joins to do the work of exchanging source system keys for surrogate keys. But the ETL design team needs input from the business user community about how to handle problematic rows described in the following scenarios.

Missing Source System Key

The simplest situation is when an incoming fact row is completely missing a key for one of the dimensions. Sometimes a dimension just doesn't apply to some of the fact rows. For example, consider the scenario of a fact table that describes the sale of products in stores. This dimensional model may include a promotion dimension that describes the marketing promotions and coupons associated with the sale of an item. But many sales are not associated with any promotion at all: the source system promotion code is blank or null.

It is absolutely imperative that each fact table key has a corresponding row in every dimension table. It would be a mistake to handle this missing key problem by inserting a fact row with a null promotion key. A null foreign key in the fact table would automatically violate referential integrity, and no solution involving outer joins would make business sense.

The missing source system key scenario has a simple solution: put a missing row in the dimension table. It's common to use a surrogate key of -1, and then fill in the other dimension columns with some attractive version of missing, unknown, or not available. Make sure you ask the business users how they want the user-visible attributes in the missing member dimension row to look.

Bad Source System Key

A harder design problem is to handle an incoming source system key that the data warehouse has never seen before. Dimensions should be processed before facts, so under normal circumstances there should be a row in each dimension table for every key in a fact row. But the ETL process must account for the unexpected. We have seen many solutions, none of which is perfect.

Option 1: Throw away the bad fact row
There are few scenarios in which it would make sense to the business users for the ETL system to throw away

incoming fact rows. Throwing away the row would distort overall totals and counts, and there may be salvageable content in the row.

Option 2: Log the bad fact row to an error table
This is probably the most common solution. It's easy to implement and keeps the fact table and dimension tables clean. However, if your ETL system simply writes the bad rows to an error table and never reviews them, it's functionally equivalent to throwing away the bad fact row. The correct system design should include two processes:

- An automated process that sends the error table rows through the surrogate key pipeline to see if the dimension member has shown up.
- A person-based process to evaluate aged errors and communicate with the operational systems for correction or resolution.

The biggest downside of this design solution is that the fact data does not get into the fact table. This may be fine for some scenarios, but highly problematic for others, such as finance subject areas which need numbers to balance.

Option 3: Map all bad fact rows to one dimension row
This solution is very similar to the suggestion for handling a missing source system key: all the bad fact rows get mapped to a single dimension row, let's call it the -2 row. Like the missing or -1 dimension row, this unknown dimension row has attractive default values for all attributes, clarifying the row's purpose.

There are two advantages to this approach:

- It's extremely simple to implement.
- It gets the bad fact row into the fact table. In reports, the bad facts all show up associated with the unknown dimension member.

However, there are two significant problems with this approach: 1) several bad fact rows may get lumped into a single super-row which makes no business sense; and 2) it is difficult to fix the data if tomorrow's ETL receives the dimension member's details. If it's possible to receive late arriving information about the dimension, the business users may want the ETL to fix up the bad fact row to point to the correct dimension member. But all the bad fact rows have been mapped to -2, no matter what the incoming source system dimension identifier.

If the business users want the fact rows re-mapped, you will need to capture the source system dimension identifier somewhere. You can either:

- Put the source system identifiers in the fact table, which may add significant width to the fact table and hence degrade query performance.
- Put a surrogate primary key on the fact table, and log the error condition in a separate table which contains both the source system identifiers and the fact table primary key. Use the fact table primary key to find the correct fact rows to update when you learn about a late-arriving dimension member.

In either case, you will need to issue an UPDATE statement against the appropriate fact rows, remapping them from -2 to the correct new dimension key. The primary argument for this approach is simplicity of implementation; but if you need to correct the fact data, this approach isn't simple.

Option 4: Put a unique placeholder row in the dimension (recommended technique)
The final solution to the problem of a bad source system key is to create a placeholder row in the dimension. If the ETL encounters a bad source system key during the fact table processing, it should pause, run over to the dimension table, create a row, get its surrogate key, and hand that key back to the fact table surrogate key pipeline.

These placeholder rows are similar to missing and unknown dimension rows, but they are not exactly the same. We

do know one attribute: the source system key.

With this approach, the ETL system is in a great position to fix up the data when it learns about the dimension member. The standard dimension processing will see the dimension attributes change from unknown to their true values. The fact table won't require any updating because the fact rows for this dimension member already had their own surrogate key.

The risk with the placeholder row approach is if there's a high noise-to-signal ratio: many bad fact rows each of which create a new dimension member, but only a few of those dimension members actually get fixed up in future loads. This is one of the reasons that in real-time ETL systems, where dirty data is arriving every few minutes or seconds, the real time data is often replaced by a conventional batch load at the end of the day which may have better data quality including complete transaction detail.

Business Requirement Implications
Your design decision about which technique to use for handling problematic fact rows should be driven by the following business user needs:

- Do the business users want the "bad" fact rows in the fact table so the numbers balance? If not, then redirect the problem rows to an error table. Make sure you build processes for getting them out of jail.
- Do the business users want to associate fact rows with late arriving corrections to dimension members? If not, mapping all bad facts to the unknown (-2) member is a possible choice.
- If the business users want the problematic facts in the fact table, and they want to update the dimension members as soon as better information arrives, then the placeholder row approach is a better bet.
- In all cases, someone from the business user community should weigh in on how the placeholder and dummy rows look and behave. Consider sort order too – usually the business users want these dummy rows to show up at the bottom of any list.

This detailed example about the surrogate key pipeline is indicative of how the best ETL system designs follow the Kimball mantra: *It's all about the business.*