

Design Tip #84 Reader Suggestions for Fact Table Surrogate Keys

By Ralph Kimball and Bob Becker

In July, Bob Becker wrote in Design Tip #81 about fact table surrogate keys. To refresh your memory, he described the circumstances where it makes sense to create a single field in a fact table that is a classic surrogate key, in other words, a simple integer field assigned sequentially as fact table records are created. Such a surrogate key has no internal structure or obvious meaning. Bob pointed out that these surrogate keys had a number of useful purposes, including disambiguating otherwise identical fact records that could arise under unusual business rules, as well as providing a way to confidently deal with suspended load jobs in the ETL back room.

Since July, a number of readers have written to us suggesting additional clever ways in which fact table surrogate keys can be exploited. The reader should note that these concepts primarily provide for behind-the-scene improvements in query performance or ETL support.

Reader Larry pointed out that in Oracle his experience was that a fact table surrogate key can make the database optimizer happier in a situation where declaring the key (the combination of fields that guarantee a row is unique) of the fact table otherwise would require enumerating a large number of foreign keys to dimensions. Some optimizers complain when a large number of B-Tree indexes are defined. With a surrogate key, a single B-Tree index could be placed on that field, and separate bitmap indexes placed on all the dimension foreign keys. Another reader, Eric, similarly reports that in Microsoft SQL Server "Another reason to use a surrogate key with a clustered index on a fact table is to make the primary key of each fact table row smaller, so that the nonclustered indexes defined on the fact table (which contain these primary key values as row identifiers) will be smaller."

Larry also reports that "If I have a user who complains about a report or query being wrong, I can often simply add in the surrogate key column to the report or query, forcing the results to show EVERY row that contributes to it. This has proven useful for debugging." In a similar vein, he also uses the surrogate key as an efficient and precise way to identify a specific fact record that he may wish to point out to the ETL development team as an example of a problem.

And finally Larry reports that "In healthcare (payers), it seems that there is a lot of late arriving dimension data, or changes to dimension data. This always seems to happen on a type two dimension. Often this has caused me to need to update the dimension rows, creating new rows for a prior period forward. A simple query of the fact table can return a list of the surrogate keys for affected rows. Then this can be used to limit the retroactive update to fact rows to only those that need to be touched. I have seen this technique improve this kind of update substantially."

Reader Norman offers an interesting query application depending on fact table surrogate keys. He writes "I wanted to offer up an additional reason for having surrogate keys in the fact tables. This is in cases where it is necessary to join two fact records together in a single query for the purpose of performing calculations across related rows. The preference would be to do these calculations in the ETL and then store them in the fact records, but I have had some requirements where the possible number of stored calculations could have been immense (and thus would have greatly increased the size of the fact tables) and/or the calculations were important yet infrequently performed by the query application, thus needing to be supported with relatively high query speeds, but not at the expense of calculating and storing the calculations. Storing "next record" surrogate keys in the fact table supports these types of requirements." Following Norman's interesting insight, we would assume a

fact table row contains a field called NextSurrogateKey which contains the surrogate key of the desired companion record. Constraining on this value immensely simplifies the SQL which otherwise would have to repeat all the dimension constraints of the first record. You would just have an embedded SELECT statement which you would use in a computation as if it were a variable, like (SELECT additional_fact from fact table b where b.surrogatekey = NextSurrogateKey).

And finally reader Dev writes about a similar applications technique where instead of using a fact table surrogate key to link to an adjacent record in the same fact table, he embeds the surrogate key of a related record that resides in another fact table. His example linked a monthly grained fact table that records health care plan measures to a second fact table that records client group benchmarks. Like Norman's example, embedding the surrogate key of the second fact table in the first allowed the applications to be far simpler. Bob and I recommend that, like Norman's example, the link from one fact table to another be handled with an explicit SELECT clause rather than a direct join between fact tables. All too often, we have seen correct but weird results from SQL when two fact tables are joined directly, because of tricky differences in cardinality between the two tables. The explicit SELECT statement should eliminate this issue.

Clearly, while these last two examples enable joins between fact table rows, we are NOT advocating creating surrogate keys for your fact tables to enable fact table to fact table joins. In most circumstances, delivering results from multiple fact tables should use drill across techniques described in Design Tip #68. Keep in mind that in this design tip we are discussing advanced design concepts to support unique business requirements. These concepts need to be carefully considered and tested before being implemented in your environment.