

## **Kimball Design Tip #31: Designing A Real Time Partition**

By Ralph Kimball

Even though the time gap between the production OLTP systems and the data warehouse has shrunk in most cases to 24 hours, the rapacious needs of our marketing users require the data warehouse to fill this gap with real time data.

Most data warehouse designers are skeptical that the existing ETL (extract-transform-load) jobs can simply be sped up from a 24 hour cycle time to a 15 minute cycle time. Data warehouse designers are responding to this crunch by building a real time partition in front of the conventional static data warehouse.

### **Requirements for the Real Time Partition**

To achieve real time reporting we build a special partition that is physically and administratively separated from the conventional static data warehouse tables. The real time partition is actually a separate table subject to special update rules and special query rules.

The real time partition ideally should meet the following tough set of requirements. It must:

- contain all the activity that has occurred since the last update of the static data warehouse.
- link as seamlessly as possible to the grain and content of the static data warehouse fact tables
- be so lightly indexed that incoming data can be continuously "dribbled in"
- support high performance querying

In the dimensional modeling world there are three main types of fact tables: transaction grain, periodic snapshot grain, and accumulating snapshot grain. Our real time partition has a different structure corresponding to each type.

### **Transaction Grain Real Time Partition**

If the static data warehouse fact table has a transaction grain, then it contains exactly one record for each individual transaction in the source system from the beginning of "recorded history". The real time partition has exactly the same dimensional structure as its underlying static fact table. It only contains the transactions that have occurred since midnight, when we loaded the regular data warehouse tables. The real time partition should be almost completely un-indexed, because we need to maintain a continuously "open window" for loading. We avoid building aggregates on this table because we want a minimalist administrative scenario during the day.

We attach the real time partition to our existing applications by drilling across from the static fact table to the real time partition. Time series aggregations (e.g., all sales for the current month) will need to send identical queries to the two fact tables and add them together.

In a relatively large retail environment experiencing 10 million transactions per day, the static fact table would be pretty big. Assuming that each transaction grain record is 40 bytes wide (7 dimensions plus 3 facts, all packed into 4 byte fields), we accumulate 400 MB of data each day. Over a year this would amount to about 150 GB of raw data. Such a fact table would be heavily indexed and supported by aggregates. But the daily tranch of 400 MB (the real time partition) could be pinned

in memory. Forget indexes, except may be a B-Tree index on the primary key to support record inserts! Forget aggregations! Our real time partition can remain biased toward very fast loading performance but at the same time provide speedy query performance.

### **Periodic Snapshot Real Time Partition**

If the static data warehouse fact table has a periodic grain (say, monthly), then the real time partition can be viewed as the current hot rolling month. Suppose we are a big retail bank with 15 million accounts. The static fact table has the grain of account by month. A 36 month time series would result in 540 million fact table records. Again, this table would be extensively indexed and supported by aggregates in order to provide good performance. The real time partition, on the other hand, is just an image of the current developing month, updated and overwritten continuously as the month progresses. Semi-additive balances and fully additive facts are adjusted as frequently as they are reported. In a retail bank, the "core" fact table spanning all account types is likely to be quite narrow, with perhaps 4 dimensions and 4 facts, resulting in a real time partition of 480 MB. The real time partition again can be pinned in memory.

Query applications drilling across from the static fact table to the real time partition have slightly different logic compared to the transaction grain. Although account balances and other measures of intensity can be trended directly across the tables, additive totals accumulated during the current rolling period may need to be scaled upward to the equivalent of a full month to keep the results from looking anomalous.

Finally, on the last day of the month, hopefully the accumulating real time partition can just be loaded onto the static data warehouse as the most current month, and the process can start again with an empty real time partition.

### **Accumulating Snapshot Real Time Partition**

Accumulating snapshots are used for short lived processes like orders and shipments. A record is created for each line item on the order or shipment. In the main fact table, this record is updated repeatedly as activity occurs. We create the record for a line item when the order is first placed, then we update it whenever the item is shipped, delivered to the final destination, paid for, and maybe returned. Accumulating snapshot fact tables have a characteristic set of date foreign keys corresponding to each of these steps.

In this case it is misleading to call the main data warehouse fact table "static" because this is the one fact table type that is deliberately updated, often repeatedly. But let's assume that for query performance reasons, this update occurs only at midnight when the users are off-line. In this case the real time partition will consist of only those line items that have been updated today. At the end of the day, the records in the real time partition will be precisely the new versions of the records that need to be written onto the main fact table either by inserting the records if they are completely new, or overwriting existing records with the same primary keys.

In many order and shipment situations, the number of line items in the real time partition will be significantly smaller than the first two examples. For example, the biggest dog and cat food manufacturer in the United States processes about 60,000 shipment invoices per month. Each invoice may have 20 line items. If an invoice line has a normal lifetime of two months and is updated five times in this interval, then we would see about 7500 line items created or updated on an average working day. Even with the rather wide 80 byte records typical of shipment invoice fact tables, we only have 600KB of data in our real time partition. This will obviously fit in memory. Forget indexes and aggregations on this real time partition.

Queries against an accumulating snapshot with a real time partition need to fetch the appropriate line items from both the main fact table and the partition, and can either drill across the two tables by performing a sort merge (outer join) on the identical row headers, or can perform a union of the rows from the two tables, presenting the static view augmented with occasional supplemental rows in the report representing today's hot activity.

In this column I have made what I hope is a strong case for satisfying the new real time requirement with specially constructed, but nevertheless familiar, extensions of our existing fact tables. If you drop nearly all the indexes and aggregations on these special new tables, and pin them in memory, you should be able to get the combined update and query performance that you need.