



Kimball Design Tip #14: Reporting Balances On Arbitrarily Chosen Days In A Transaction Fact Table

By Ralph Kimball

Design Tip #13 showed how to attach a slowly changing dimension table (the account table in a bank) to a fast moving transaction grained fact table (the account transactions). We saw how the slowly changing dimension was itself something like a fact table because it was the target of a set of transactions that modified the account profiles.

In this Design Tip, we switch our focus to the big fact table that records all the transactions against the bank account. Let's boil this fact table down to a really simple design for discussion purposes:

- Date Key (FK)
- Account Key (FK)
- Transaction Type Key (FK)
- Transaction Sequence Number
- Final Flag
- Amount
- Balance

Here's what the fields contain: Date Key = surrogate key pointing to daily grain calendar dimension; Account Key = surrogate key pointing to account table; Transaction Type Key = surrogate key pointing to a small table of allowed transaction types; Transaction Sequence Number = continuously increasing numeric sequence number running for the lifetime of the account; Final Flag = TRUE if this is the last transaction on a given day, FALSE otherwise; Amount = amount of this transaction; Balance = resulting account balance after the transaction.

Like all transaction grained tables, this one only has a record in it if a transaction was performed. If an account was quiet for two weeks, say October 1 through 14, there will be ZERO records in the fact table for this account in this time span. But suppose we want to ask what all the account balances were on October 5?

In this case we need to look for the most recent previous fact record for each account on or before our requested date.

Here's some tested SQL that does the trick:

```
SELECT a.acctnum, f.balance
FROM fact f, account a
WHERE f.account_key = a.account_key
AND f.final_flag = 'True'
AND f.date_key =
  (SELECT MAX(g.date_key)
   FROM fact g
   WHERE g.account_key = f.account_key
   AND g.date_key IN
     (SELECT t.date_key
```

```
FROM time t
WHERE t.fulldate <= 'October 5, 2000'))
```

If you study this SQL you probably have a couple of questions!

Question 1. RALPH, how could you use a time surrogate key as the basis for a constraint??? The whole point of surrogate keys is that they have no semantics.

Answer 1. Yes, except for the time surrogate key which is a set of integers running from 1 to N. For a completely separate reason, we have already placed a predictable ordering on the time surrogate key. We need the time surrogate key to be ordered so we can impose physical partitioning on this large fact table based on this key. This neatly segments the physical table so we can perform discrete administrative actions on certain ranges of times, like moving to off-line storage, or dropping and rebuilding indexes. This time dimension is the ONLY dimension that has any logic to the surrogate keys and is the only one we dare place application constraints on. We use this ordering to advantage in the above SQL when we find the most recent prior end-of-day transaction.

Question 2. Why did you go to the trouble of linking off to the time table when you could have just constrained a conventional time stamp in the fact table? Then we wouldn't need a surrogate key (seems like a lot of trouble...).

Answer 2. Putting in a time stamp instead of the surrogate key introduces a whole set of problems we solved with surrogate keys, including null dates, inapplicable dates, and hasn't-happened dates. We have discussed all of this in other places. But more important, a naked date stamp in the fact table doesn't let us do realistic complex time constraints. What if instead of October 5, the request had been for the balances on "3rd quarter Federal Reserve reporting date" whatever that is. In this case, the above SQL is barely altered. Just replace the last line with the appropriate constraint in the time dimension table for this special calendar event.

Question 3. Isn't this design very sensitive to backdated transactions being inserted into the fact table?

Answer 3. That is a good question. This fact table must be COMPLETE and ACCURATE. Every transaction against the account must appear in this table or else the running balance cannot be computed. Certainly a late arriving transaction record would require sweeping forward from the point of insertion in that account and incrementing all the balances and all the transaction sequence numbers. Note that we haven't explicitly used the transaction sequence number in this discussion, although something is needed in this design to reliably reconstruct the true sequence of transactions and to provide the basis of a key for the fact table (account_key + date_key + sequence number). I like the sequence number rather than a time-of-day stamp because differences between the sequence numbers are a valid measure of account activity.

I have a number of additional astounding insights about this design but you have to come to Maui to hear what they are. Just kidding. Write to me with comments.