

Design Tip #85 Using Smart Date Keys to Partition Large Fact Tables

By Warren Thornthwaite

I've recently had two people ask if it's OK to use a meaningful key for the Date dimension; an integer of the form YYYYMMDD. In one case, my recommendation was no; in the other it was yes. In the no case, the user's goal was directly to provide users and applications with a key in the fact table they could recognize and query directly thereby bypassing the Date dimension. We describe why this actually reduces usability in [Design Tip #51](#). This approach could also hurt performance in some database platforms.

The yes case involved partitioned fact tables. Partitioning allows you to create a table that is segmented into smaller tables under the covers, usually by date. Thus you can load data into the current partition and re-index it without having to touch the whole table. Partitioning dramatically improves load times, backups, archiving old data, and even query performance. It makes it possible to build terabyte data warehouses.

So why does partitioning lead us to consider a smart surrogate key in the Date dimension? It turns out updating and managing partitions is a fairly tedious, repetitive task that can be done programmatically. These programs are much easier to write if the table is partitioned on date and the date key is an ordered integer. If the date key follows a date-style pattern you can also take advantage of date functions in the code.

In SQL Server 2005, for example, the initial definition of a simple partitioned table that holds the first three months of sales data for 2006 might look like this:

```
CREATE PARTITION FUNCTION SalesPartFunc (INT) AS RANGE RIGHT
FOR VALUES (10000000, 20060101, 20060201, 20060301, 20060401)
```

The values are breakpoints and define six partitions, including a partition to hold the non-date entries in the table (DateKey<10000000), and empty partitions on either side of the data to make it easier to add, drop and swap partitions in the future.

Prior to loading the fourth month of 2006, we would add another partition to hold that month's data. This hard coded version splits the empty partition by putting in the next breakpoint, in this case, 20060501:

```
ALTER PARTITION FUNCTION PFMonthly () SPLIT RANGE (20060501)
```

The following Transact SQL code automatically generates the command using a variable called @CurMonth. It is a bit convoluted because it converts the integer into a date type for the DATEADD function, then converts it to VARCHAR(8) to concatenate it into the SQL statement string. Finally, the EXEC command executes the string.

```
DECLARE
    @CurMonth INT, @DateStr Varchar(8), @SqlStmt VARCHAR(1000)
SET @CurMonth = 20060401
SET @DateStr = CONVERT(VARCHAR(8),DATEADD(Month, 1, CONVERT(datetime,
    CAST(@CurMonth AS varchar(20)), 112)),112)
```

```
SET @SqlStmt = 'ALTER PARTITION FUNCTION PFMonthly () SPLIT RANGE (' + _  
    @DateStr + ')'  
EXEC (@SqlStmt)
```

So, by using a smart YYYYMMDD key, you still get the benefits of the surrogate key and the advantage of easier partition management. We recommend using your front end metadata to hide the foreign keys in the fact table to discourage your users from directly querying them. If you would like a working example of this code for SQL Server 2005 or Oracle 10g, please send me a note.

As a final note, remember that this logic surrounding the date key cannot be replicated for any other dimension such as Customer or Product. Only the Date dimension can be completely specified in advance before the database is created. Calendar dates are perfectly stable: they are never created or deleted!