SQLintersection

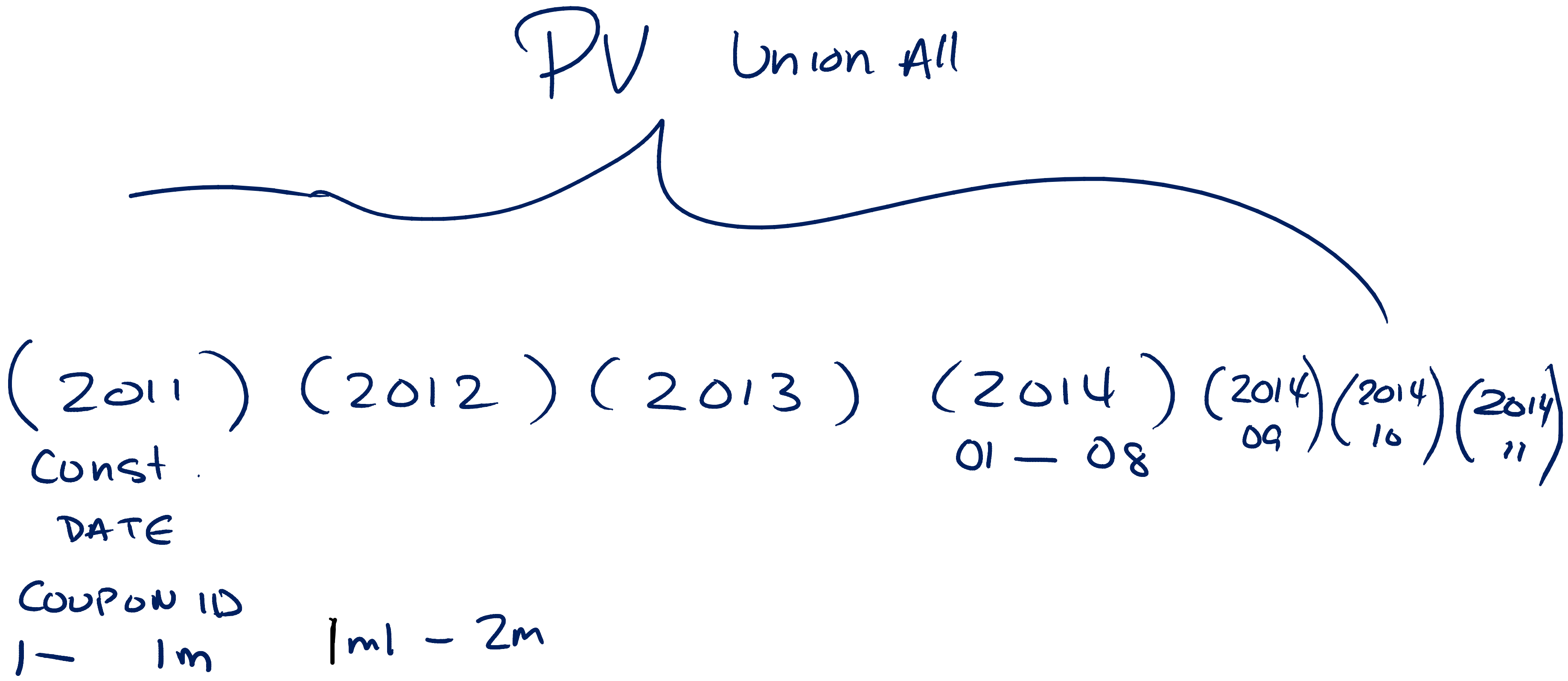
Whiteboard drawing and annotations

Queries Gone Wild 2: Statistics / Cardinality in Versions 2008, 2008R2, 2012, and 2014

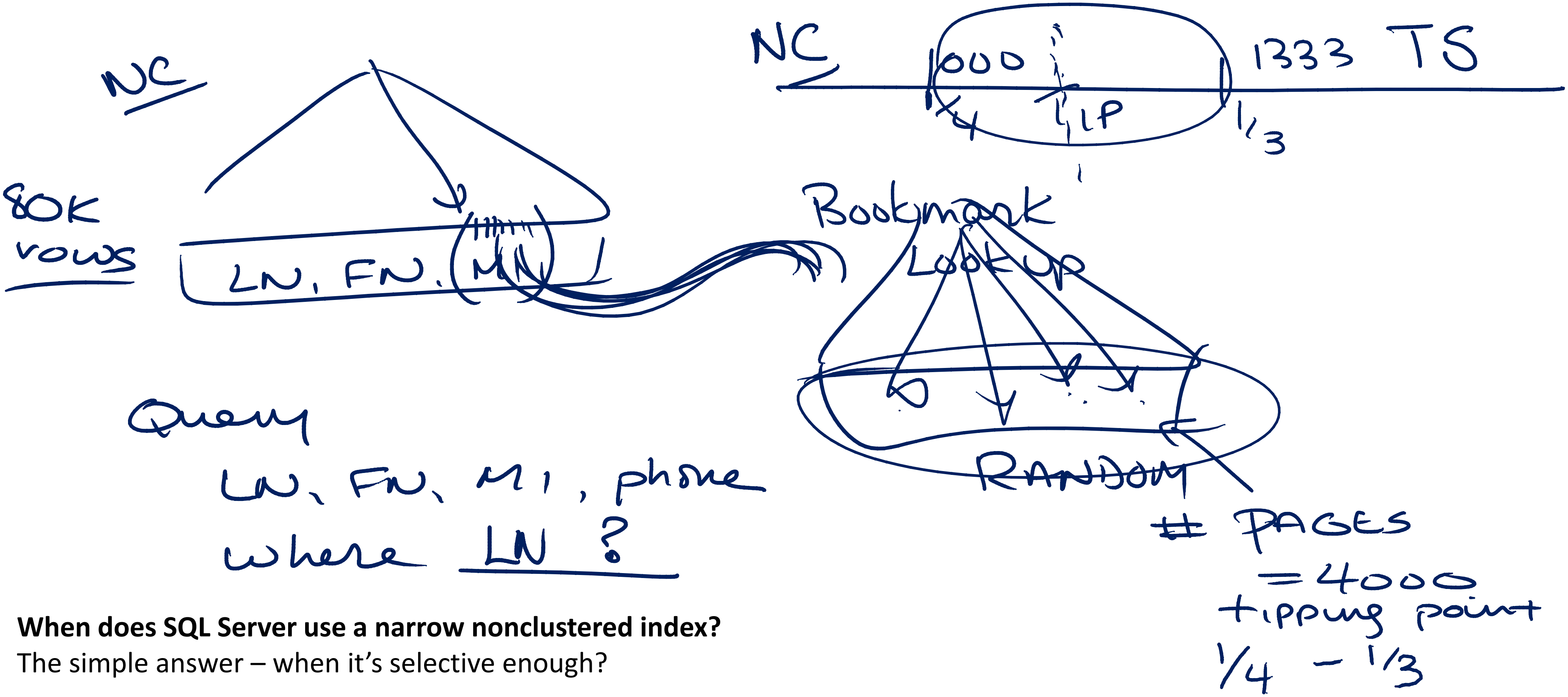
Joe Sack
Joe@SackHQ.com

Kimberly L. Tripp
Kimberly@SQLskills.com

SQL
intersection

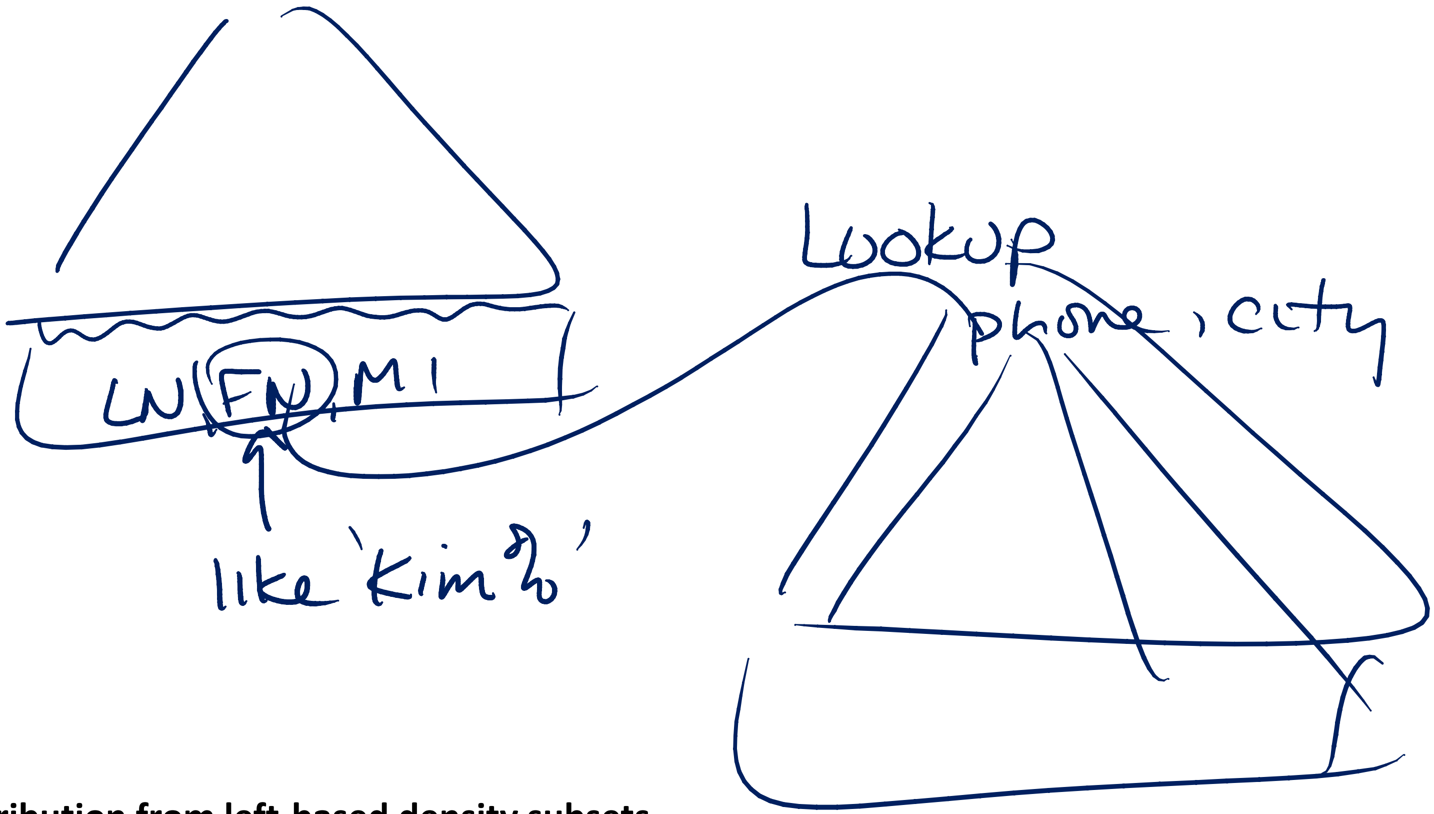


The idea behind this drawing is that a VLT (very large table) can have many maintenance, availability, and performance problems. To resolve many of these your best option is to break this table into smaller tables – union'ing all of the rows in a view (called a partitioned view). The main requirements for this are that you use constraints (one or more). Constraints are validated during optimization. SQL Server is able – when querying through a view – to generate the query tree and then "prune" the tree. This partition elimination removes any of the redundant tables from access. All of this is as long as the constraint is trusted (or, should I say as long as it's NOT untrusted. ☺)



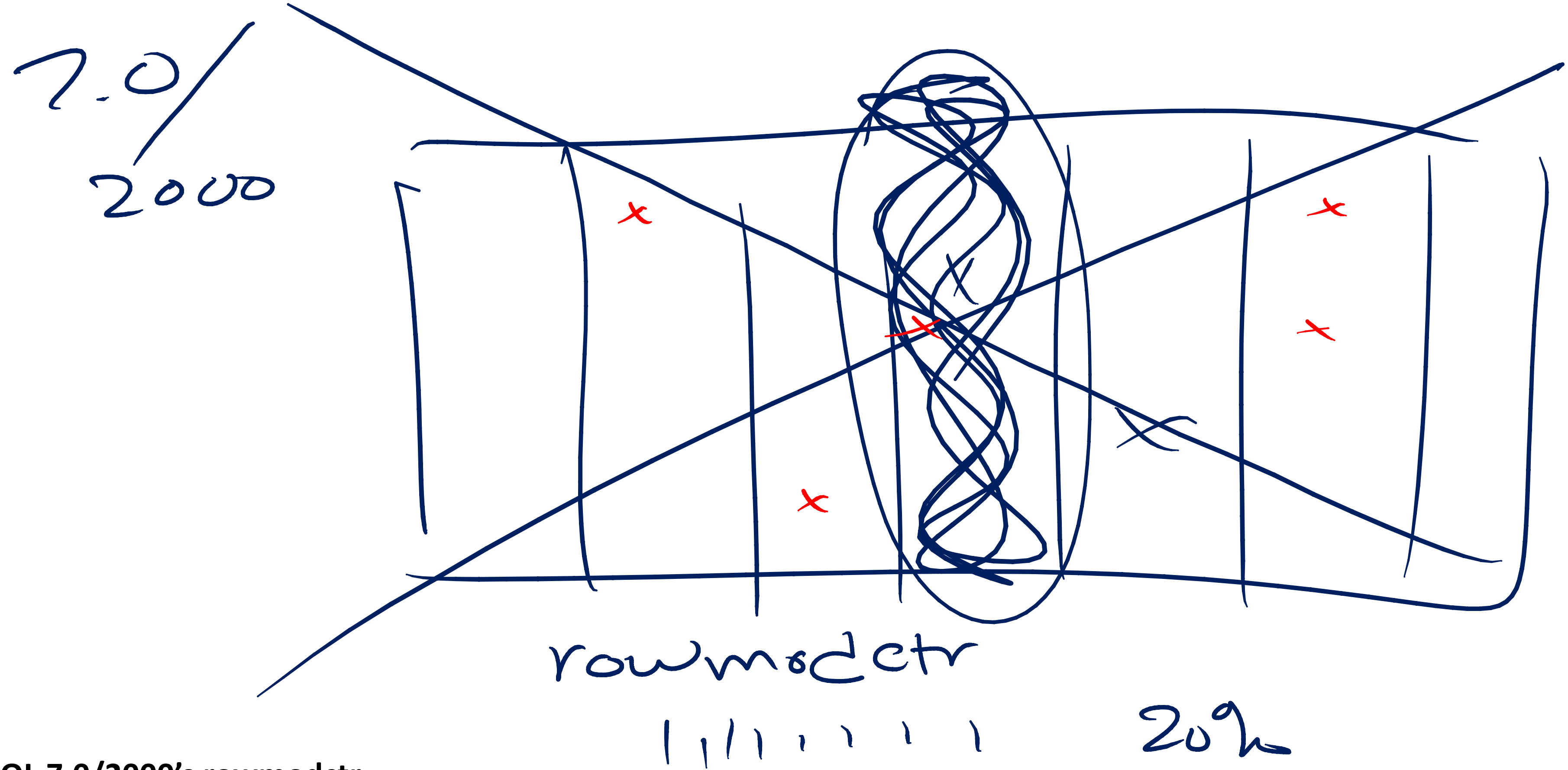
When does SQL Server use a narrow nonclustered index?
The simple answer – when it's selective enough?

When is that? I call it "the tipping point" and I tie it to a relatively simple formula.
Take the number of **pages** in the table and get the $^1/_4$ mark and $^1/_3$ mark. For queries that return **fewer rows** than the number calculated at the $^1/_4$ mark *then the query is probably selective enough*. For queries that return **MORE rows** than the number calculated at the $^1/_3$ mark *then the query is NOT selective enough*; SQL Server will do a table scan.



Column-level distribution from left-based density subsets…

This relates to the discussion around using a nonclustered to scan – even if we have to do "lookups" because that nonclustered index doesn't cover. What we know from existing stats (from the left-based density values) is that last names are horribly non-unique and the combination of last name & first name IS *almost* unique – that doesn't imply anything about first names alone. I could have created a data set of firstnames of Kima, Kimb, Kimc and then multiplied that with last names (Tripp, Randal, Smith) and I would have had similar statistics for last name alone and for the combination of last name, first name. SQL Server does NOT gamble on this – **SQL Server creates column level statistics on first name.**

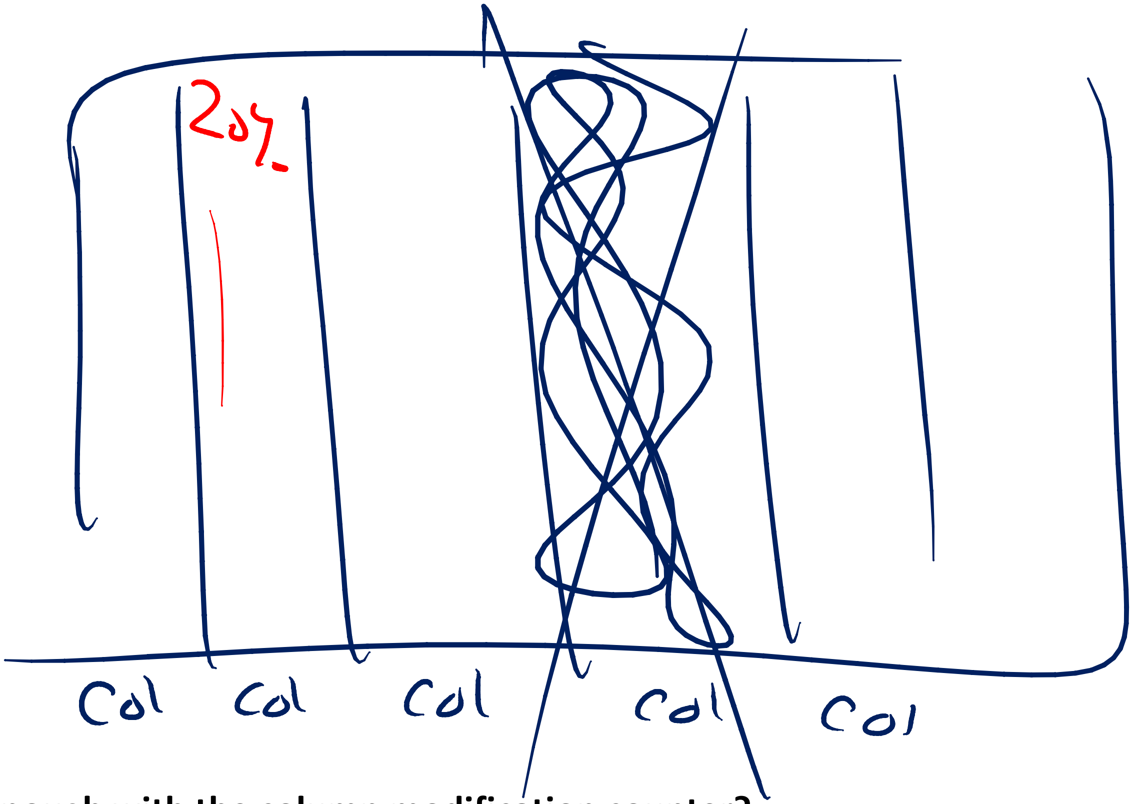


SQL 7.0/2000’s rowmodctr

This diagram shows the pros/cons of the methods for statistics invalidation. SQL Server 7.0 and 2000 used a rowmodctr (sysindexes.rowmodctr) to do invalidation. Even though SQL Server doesn’t use this anymore – **you can** (through sys.dm_db_stats_properties). Internally, in 2005+ they moved to a colmodctr:

- The pro is that you don’t invalidate too soon (when you have a highly volatile column)
- The con is that you might not invalidate soon enough if your modifications are reasonably distributed across columns.

2005

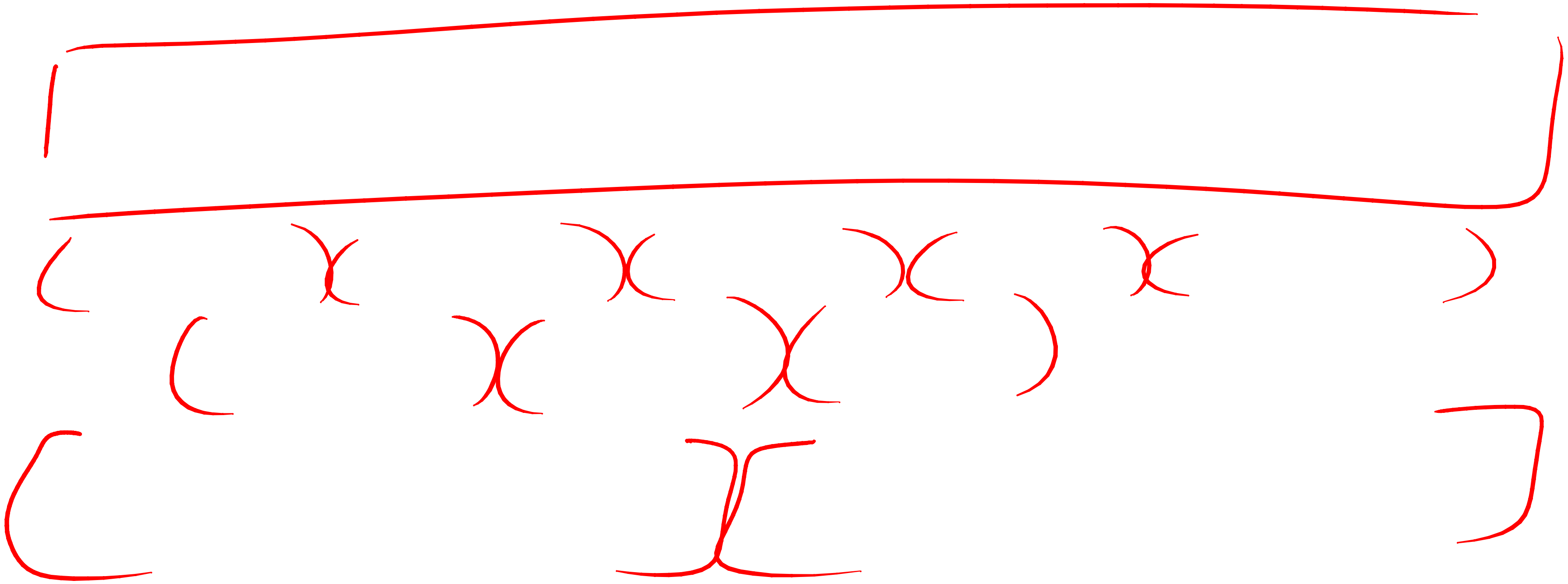


Are we NOT invalidating soon enough with the column modification counter?

This shows how relatively distributed modifications effect each column with a small percentage of the modifications but when they add up to 20% ALL columns become invalidated (this was the PRE-2005 way of doing it):

- The pro was that each column had a reasonable (but lower) percentage of rows modified and the stats were invalidated
- The con is that a single overly volatile column would cause ALL statistics to be invalidated (which was overkill).

Now, we might wait a bit too long because now (2005+) we're waiting for 20% of the column to change!



Filtered Statistics

Filtered statistics can be created for specific values but depending on your data distribution – you might want to divide the data into buckets and then create a [filtered] statistic for each of those buckets.

The example was 31 million sales over ~18K customers. By creating a statistic for each 1K customers you have statistics that are effectively 19x more detailed. Even adding only 10 filtered stats gives you 10 times more detail. However, it still might not be detailed enough. You'll need to test it and check the histogram. Because of potential ***interval subsumption**** issues– some have asked if it would be beneficial to create additional stats at different intervals… you could but it would really depend on the queries. Having said that – the bigger the range that the query is interested the more the averages just average out. So, really, this issue is to significantly help queries that are more targeted (where the stats just weren't good).

***** The optimizer can detect whether interval conditions in a filtered index cover, or "subsume" interval conditions of a query.

TS ? **FactInternetSales**

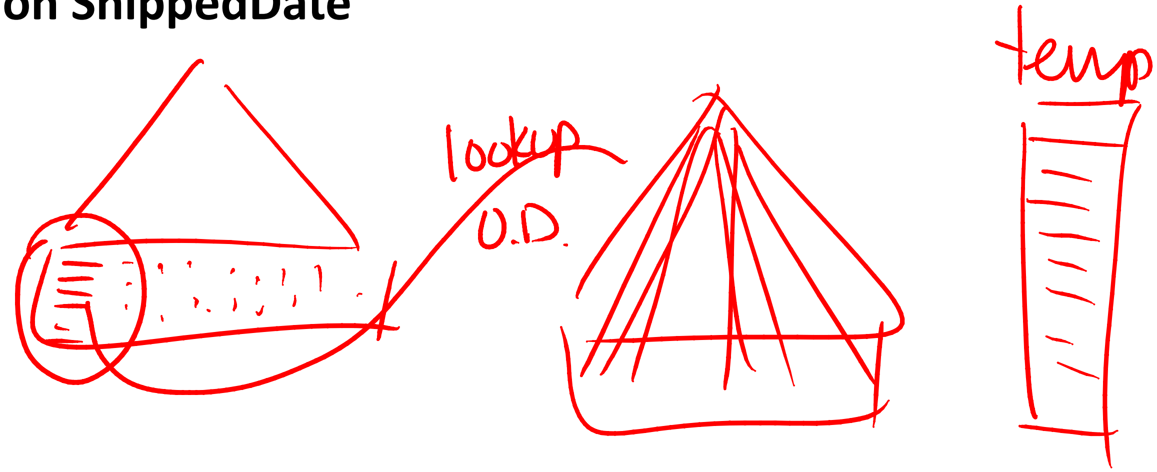
DEMO

This was the demo on uneven distribution.

A table scan is always an option.

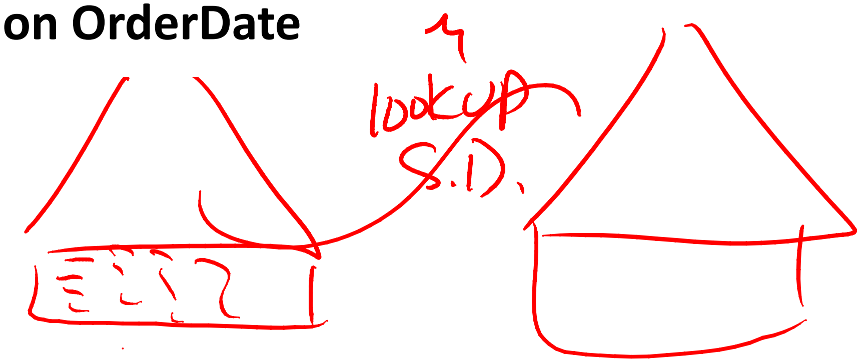
With narrow indexes, SQL Server does not understand the correlation between these columns. As a result, their estimates are going to assume even distribution for where those rows are found related to another column. This might be wrong!

NC **Index on ShippedDate**



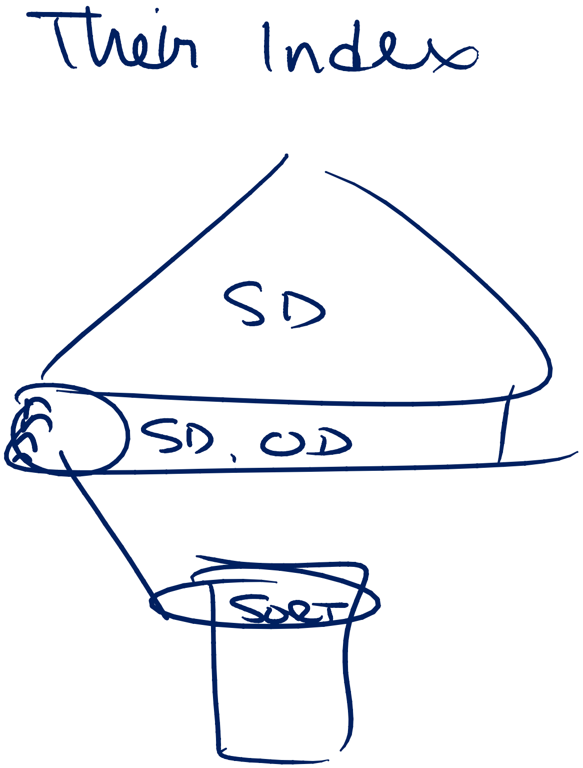
SORT mem

NC **Index on OrderDate**



when will I hit a NULL

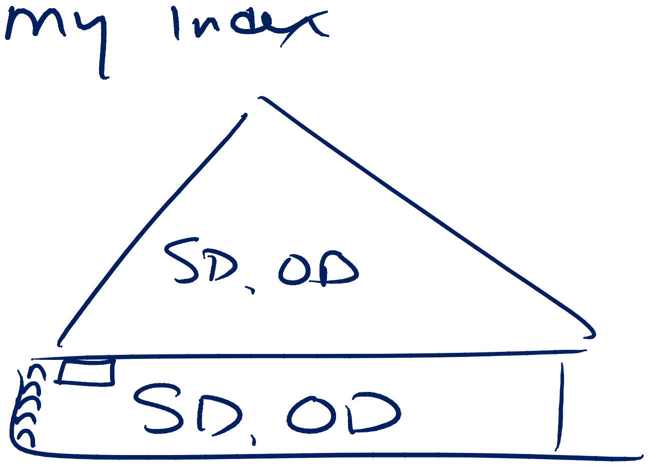
$$\frac{2541}{60348} = \frac{1}{23.7}$$



DEMO
This was the suggested index from the green hint in showplan and while it does make this query faster (and with fewer I/Os) it's not the best index that's possible.

Key point, the missing index DMVs (which is where the green hint gets its information from) – gives you good suggestions but not always the best suggestion.

Furthermore, this is the same index that's recommended by DTA. So… sometimes it does take *manually* defining/choosing the index.



DEMO
This was the index that I suggested (putting OrderDateKey in the key) as a combination of:
ShipDateKey, OrderDateKey
The first record on the first page will be the minimum OrderDateKey where ShipDateKey IS NULL.