

Kimberly L. Tripp

kimberly@SQLskills.com

Whiteboard drawings during
class AND drawings from other
workshops that I thought you
might benefit from...



Queries Gone Wrong: Statistics, Cardinality, Solutions

Things I talked about briefly...

- **scalar functions – BEWARE** (see scalar function script in “other” project)
- **inline table-valued function – are usually OK**
 - ☐ look like view
 - ☐ act like view
- **multi-statement table-valued function - rarely**
 - ☐ look like sps but not as optimizable
 - ☐ try the TF
 - ☐ if still a problem -> sp
 - ☐ Consider changing to use the new **trace flag 2453** - be sure to read this article
 - ☐ <http://sqlperformance.com/2014/06/t-sql-queries/table-variable-perf-fix>



Notes I wrote up in SSMS (2)

- SARGs with a column called msal
- What most people would write would be:
- `where msal * 12 <= @annualsalary`
 - Non-seekable expression
 - Doesn't allow SQL Server to "seek" for the rows – must process every value (requiring a scan)
- `where msal <= @annualsalary / 12`
 - Isolate the column to **one side of the expression** and SQL can evaluate once
 - This CAN use an index – but, only if it's selective enough to warrant it



Table

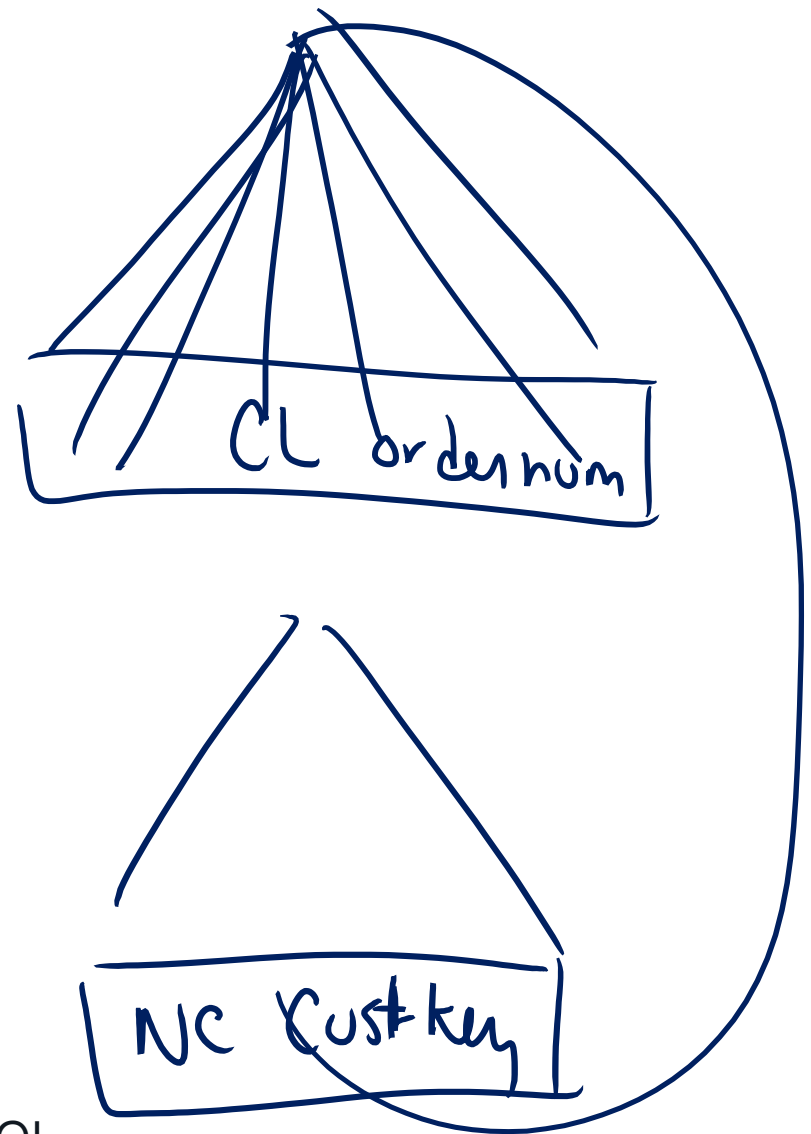
factinternet sales

Query

sum (Salesamt)

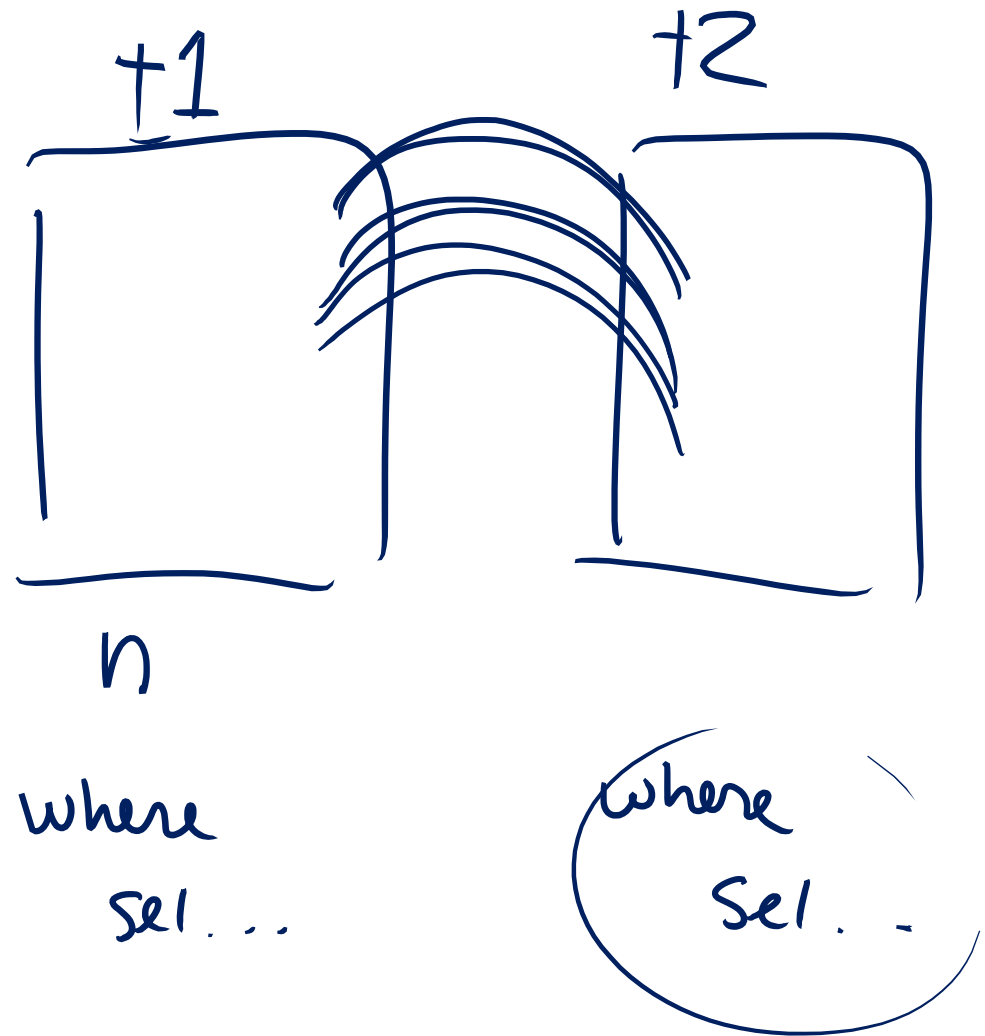
by cust

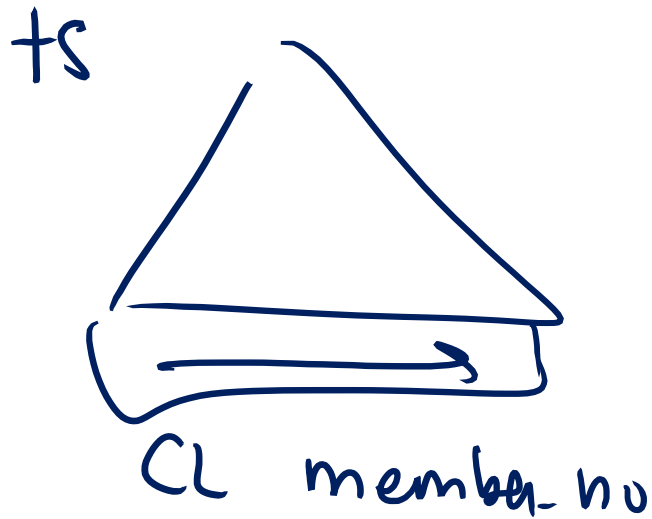
We were discussing the way that SQL Server will access the data for "sales by customer" when the nonclustered index is only on CustomerKey and the query needs additional information (like the SalesAmount). This NC index does not cover the query. As a result, SQL Server has to do bookmark lookups for every row – which can be very costly. At some point, SQL Server's NOT going to do this... that's the tipping point.



Having statistics without indexes can still be beneficial for other optimization techniques – for example joins. When a loop join is performed SQL Server wants to position the smaller set as the driver (or the first table) and then iterate into the second table. So, if we know the selectivity (even without indexes) then SQL Server might change the position of the tables in a join...

This is why statistics – even without indexes – can be really helpful!





Slide 27: Indirect Index Usage (Set Selectivity)

This image describes the two options that SQL Server has to access the data for WHERE firstname LIKE 'Kim%'

SQL Server always had an option of doing a table scan...

But, if the number of rows where firstname is like Kim is very selective then the cost of using the nonclustered (even though we need to scan) is still cheaper – even with the needed bookmark lookups to get the rest of the data (the index does not cover the query)



VLT

UPV ← S/D/U

Ro hist.
less crit

Rw critical
table per year

(2012 - 2014)
UA

(2015)
UA

(2016)
UA

2017
(PT)
UA

(2017)
Q5
UA

(2017)
Q6
UA

↑
Insite
DSE

Recurring theme – dealing with VLTs

There's more on this at the end of this whiteboard file...

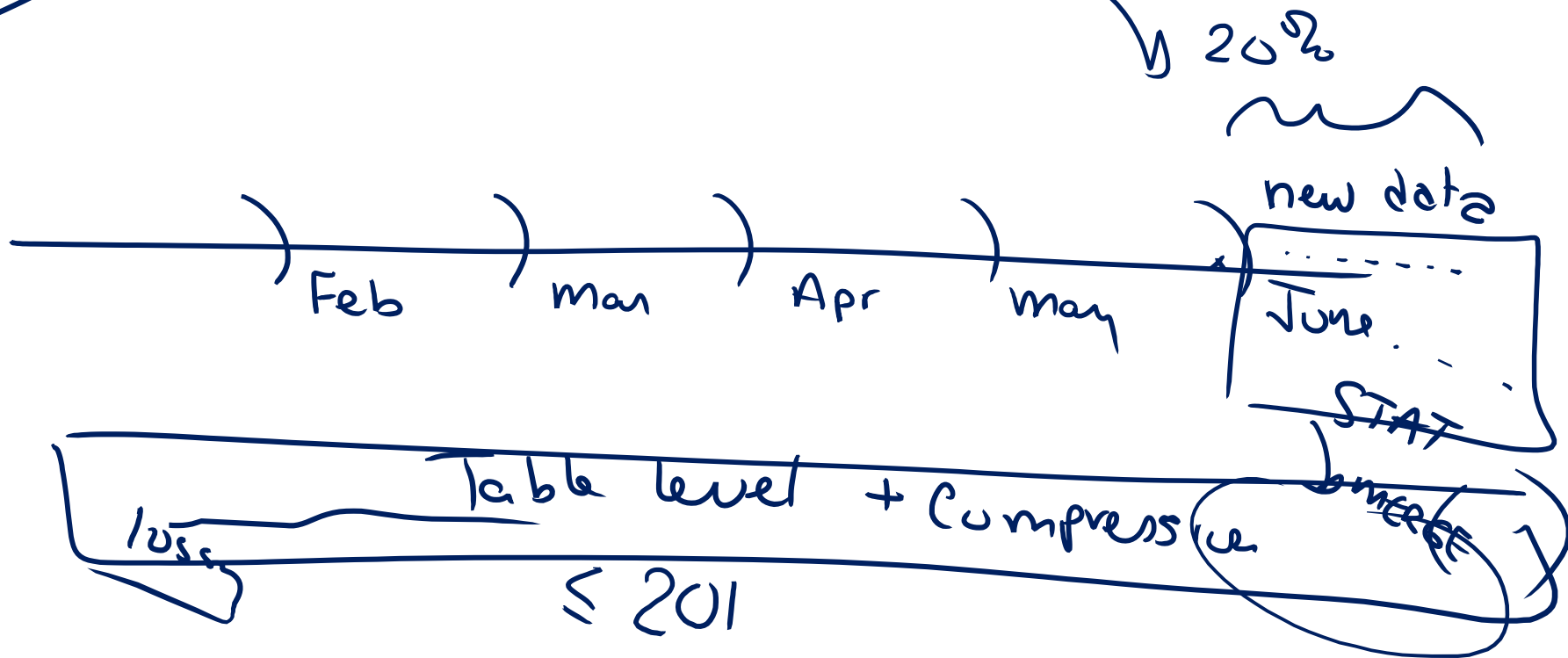
Simply put, a single, very-large-table (even if partitioned) will have "issues" related to performance, management, etc.

However, breaking that large table down – into more manageable chunks (e.g. separate tables) offers a tremendous number of options/solutions.



PT

w/ incremental

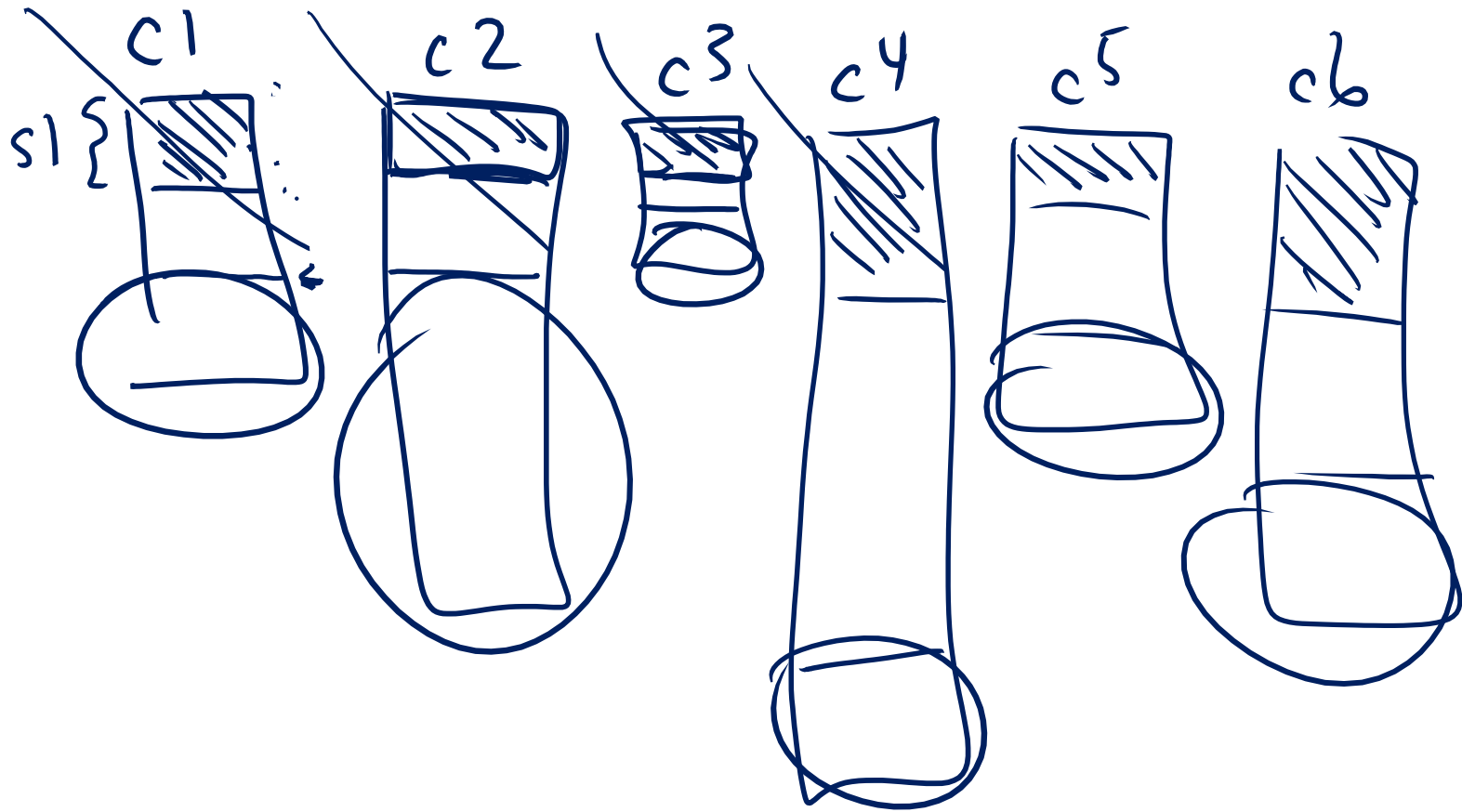


Statistics are table-level (just one of the many problems with VLTs)

Statistics are table-level even when the table is partitioned.

Unfortunately, you just can't accurately describe the data (especially when heavily skewed) in only 200 (or 201) steps.





Columnstore indexes

Separate data into columnar storage – allowing for much better compression (because the data is often similar).

Furthermore, SQL Server breaks those columnstore structures into segments and can do better parallelism AND segment elimination... which helps to improve colmstore performance.



Shape

$\frac{1}{4}$ round
=

Color

$\frac{1}{3}$ red

$$\text{Legacy CE} = \frac{1}{4} \times \frac{1}{3}$$

$$\boxed{\frac{1}{12}}$$

$$\text{New CE} = \frac{1}{4} \times \frac{1}{9}$$

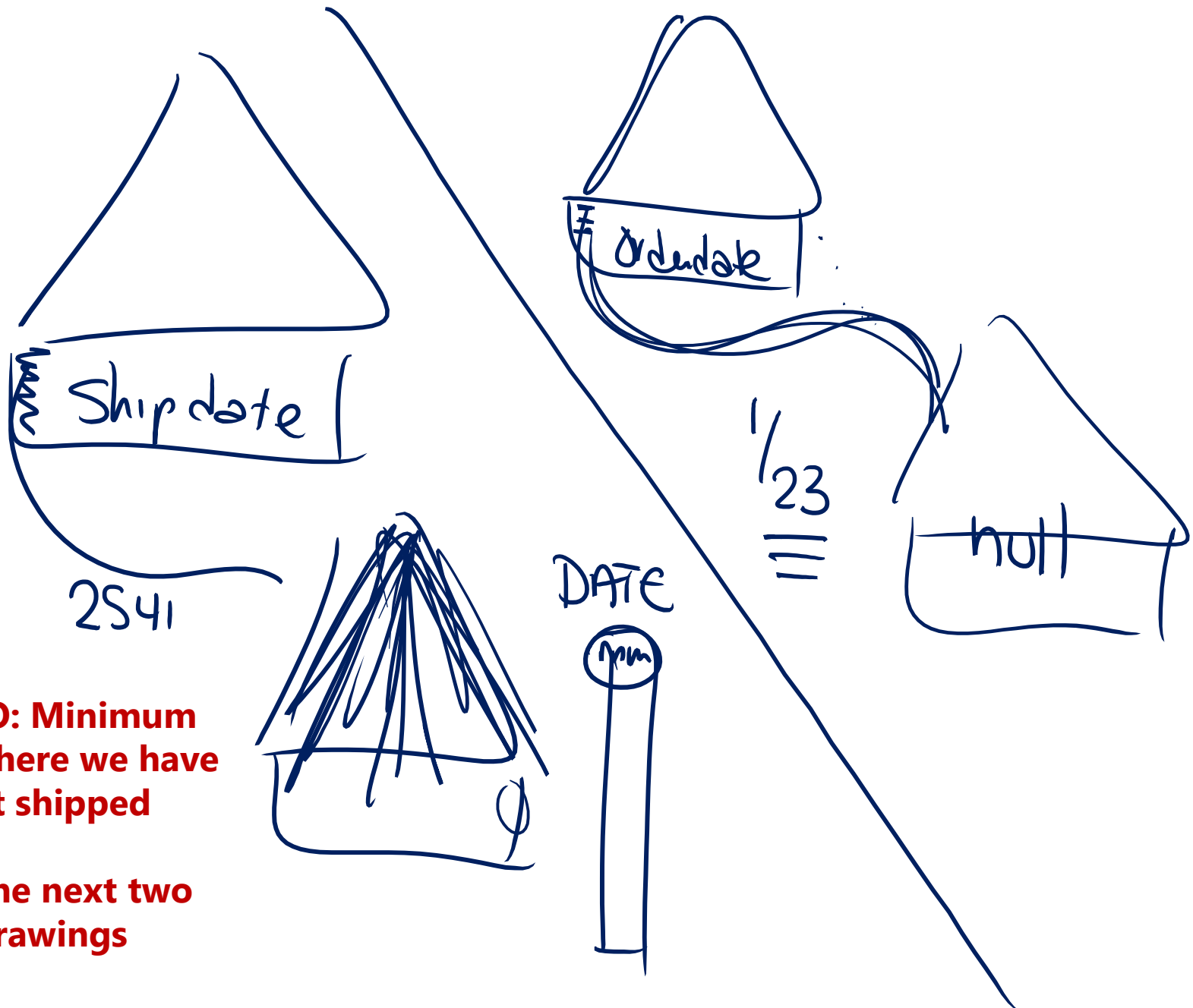
$$\boxed{\frac{1}{36}}$$

The New CE in SQL Server 2014 and higher uses different calculations for estimations...

If you want to get some insight into the specific calculations you should check out Joe Sack's whitepaper: **Optimizing Your Query Plans with the SQL Server 2014 Cardinality Estimator** (<https://msdn.microsoft.com/en-us/library/dn673537.aspx>)

Is the new CE "better"... in many cases, yes but it's not a guarantee. It's really just different. Some queries might benefit. Some queries might not... but, now you have TWO CEs from which to choose, test, and use in your queries – THAT's what's cool!



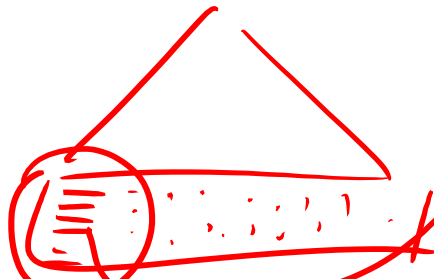


**DEMO: Minimum
date where we have
not shipped**

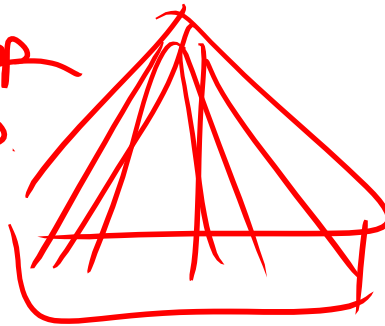
**See the next two
drawings**



TS
DEMO: Minimum date where we have not shipped
Index on ShippedDate
 Index Ship date Key



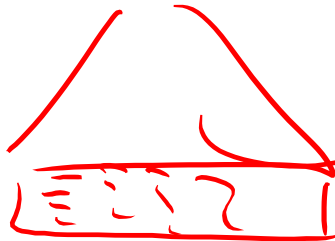
lookup
U.D.



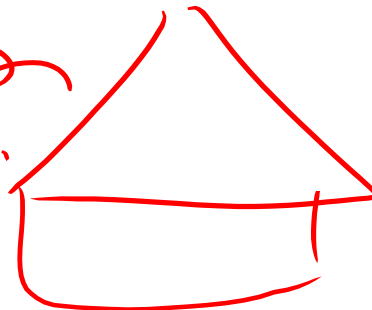
temp
 SORT
 min

Index on OrderDate

NC
 Index OrderDate Key
 lookup
S.D.



lookup
S.D.



when will I hit
 a NULL

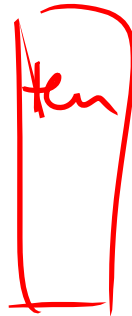
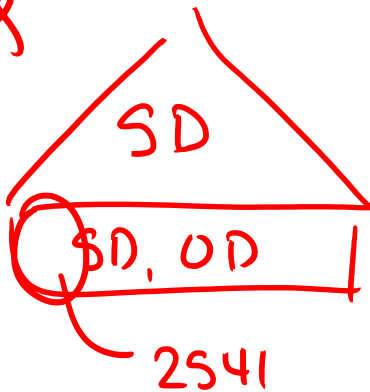
$$\frac{2541}{60348} = \frac{1}{23.7}$$

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.



Missing
Index
DMV



Score

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.



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.

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



These are the additional
whiteboard pages.

Kimberly L. Tripp

kimberly@SQLskills.com



Queries Gone Wrong: Statistics, Cardinality, Solutions

LN, FW, MI, MNO

385 1 1 1

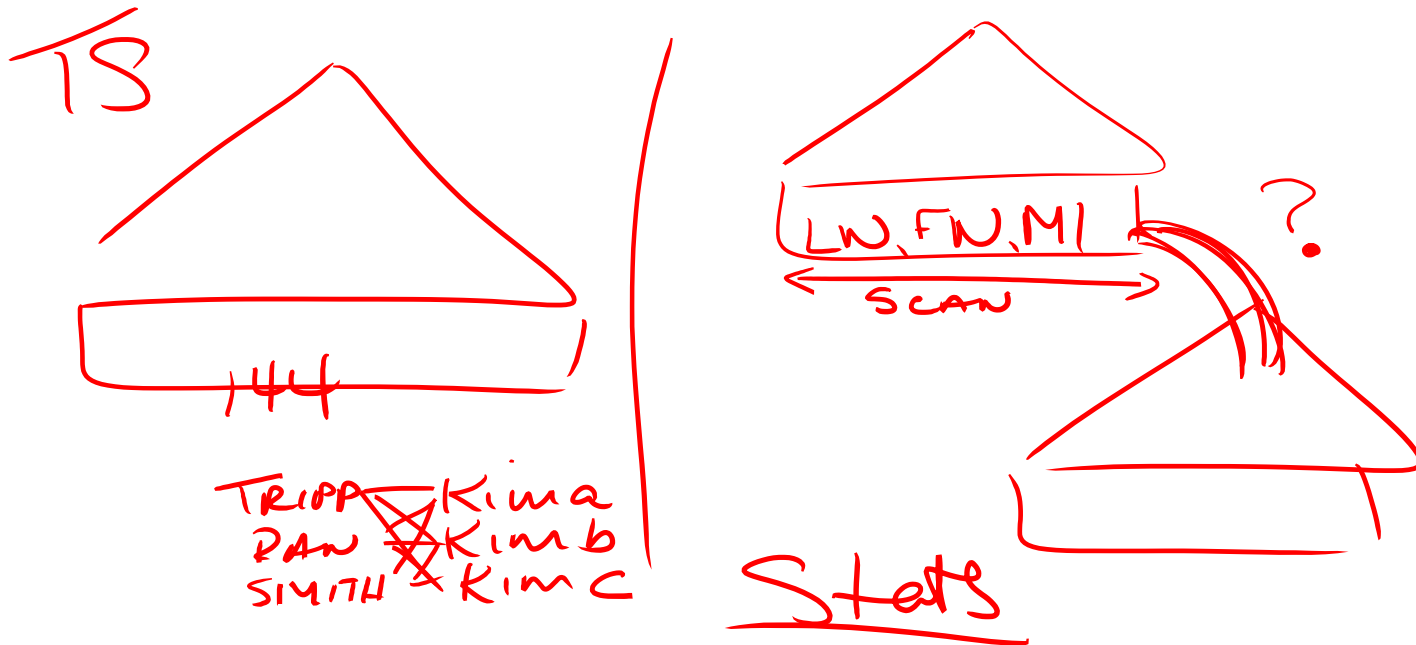
Data selectivity and “need” for additional columns in the key - from left-based density subsets...

If the distribution of the data is unique at the combination of the first and second columns then the third, forth, etc. do not provide any use in terms of seeking (JUST in terms of seeking). However, they might provide use for sorting.

But, it MIGHT be possible to consolidate another index with this one IF you really don't need those extra columns in the key. Something to consider!

Multiple – all density * rows and if unique (= 1 or very close to one) then you can *consider* consolidation with other similar indexes!

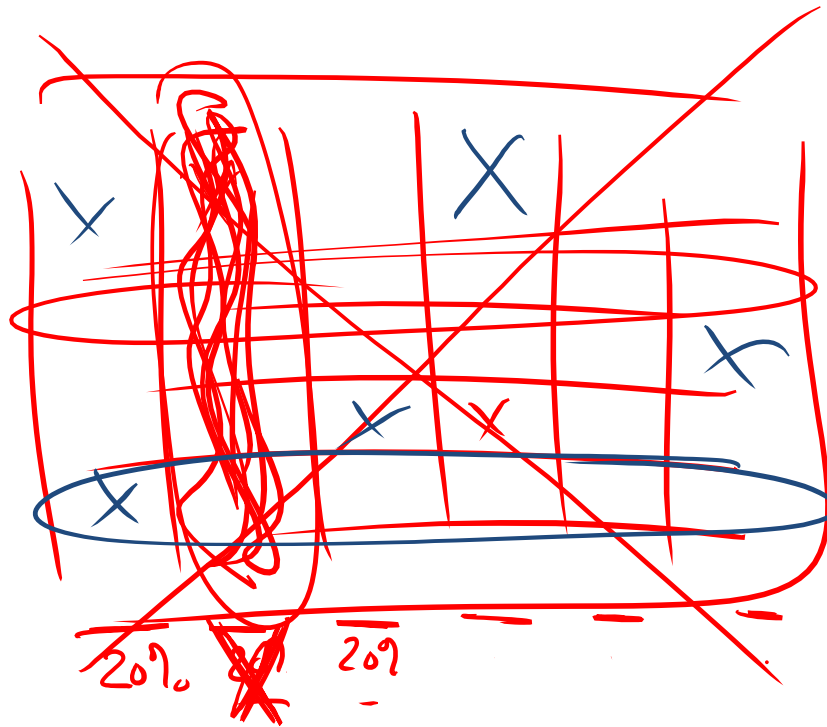




Column-level distribution from left-based density subsets...

This relates to the query on slide 10 and the idea that I was trying to show here is that even though the left-based density shows that last names are horribly NOT 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.





20%?

||||
—
20

SQL 7.0/2000's rowmodctr

This is a diagram showing 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.



in

X				X		X	
		X			X		X

row modified 20%

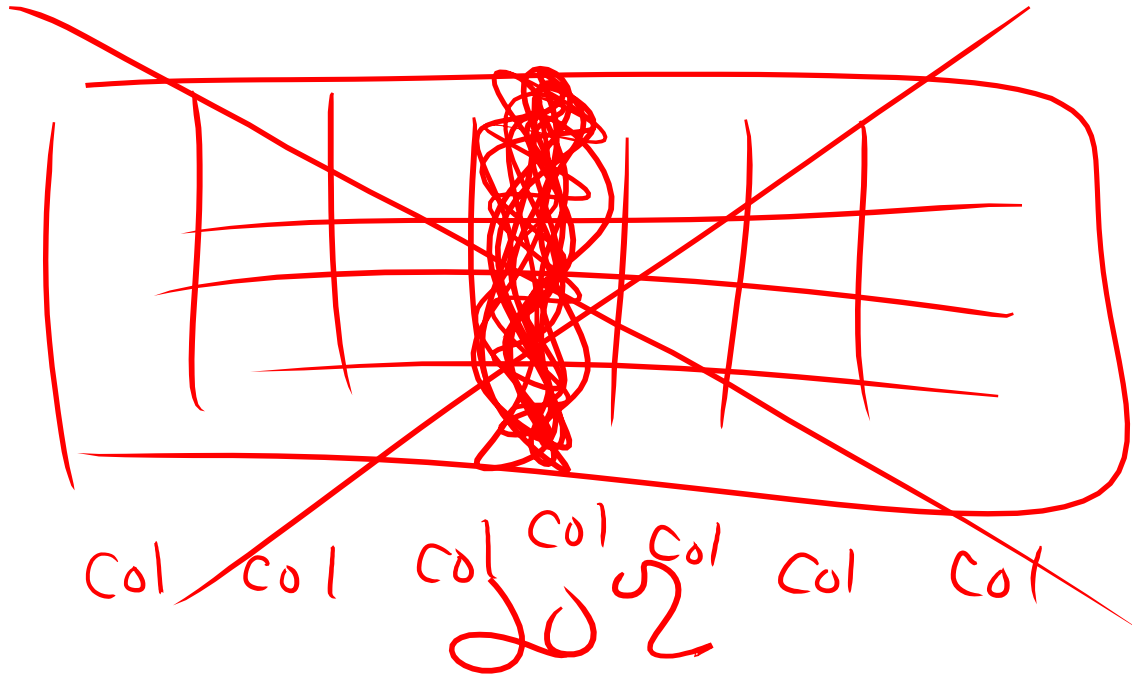
|||||

We are 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).





Overly volatile columns (or rows) could cause invalidation too soon (across the table):

The problem prior to 2005 was that a single overly volatile column would cause ALL statistics to be invalidated.



101
200 [250K] — [5] [Avg 50K]
115 135

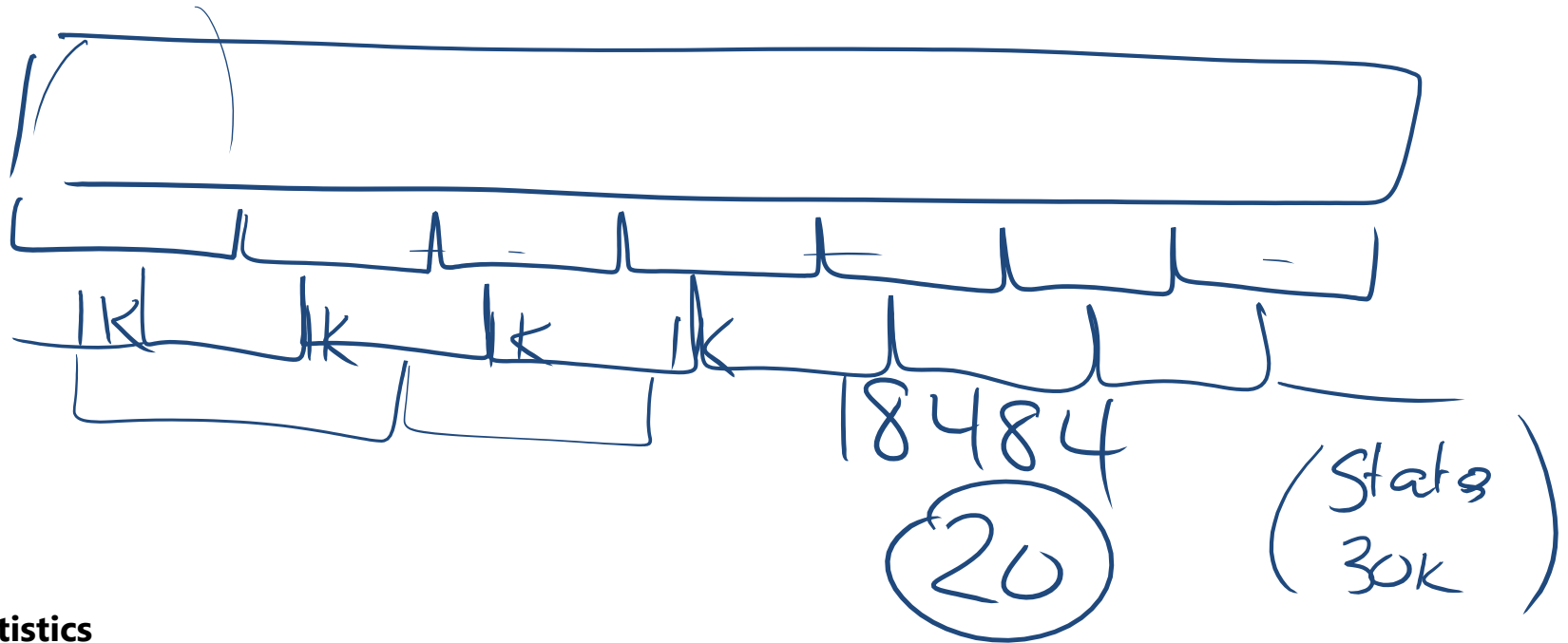
This was around our discussion about the limitation of SQL Server's knowledge of the data with gaps in the step values.

If the step describes the range from 101 to 200 (not including those values)

- There are 250K rows over that range
- There are 5 distinct values

The average will show 50K and EVERY value supplied between 101 and 200 will get an estimate of 50K. There's no way for SQL Server to know WHICH values actually exist.





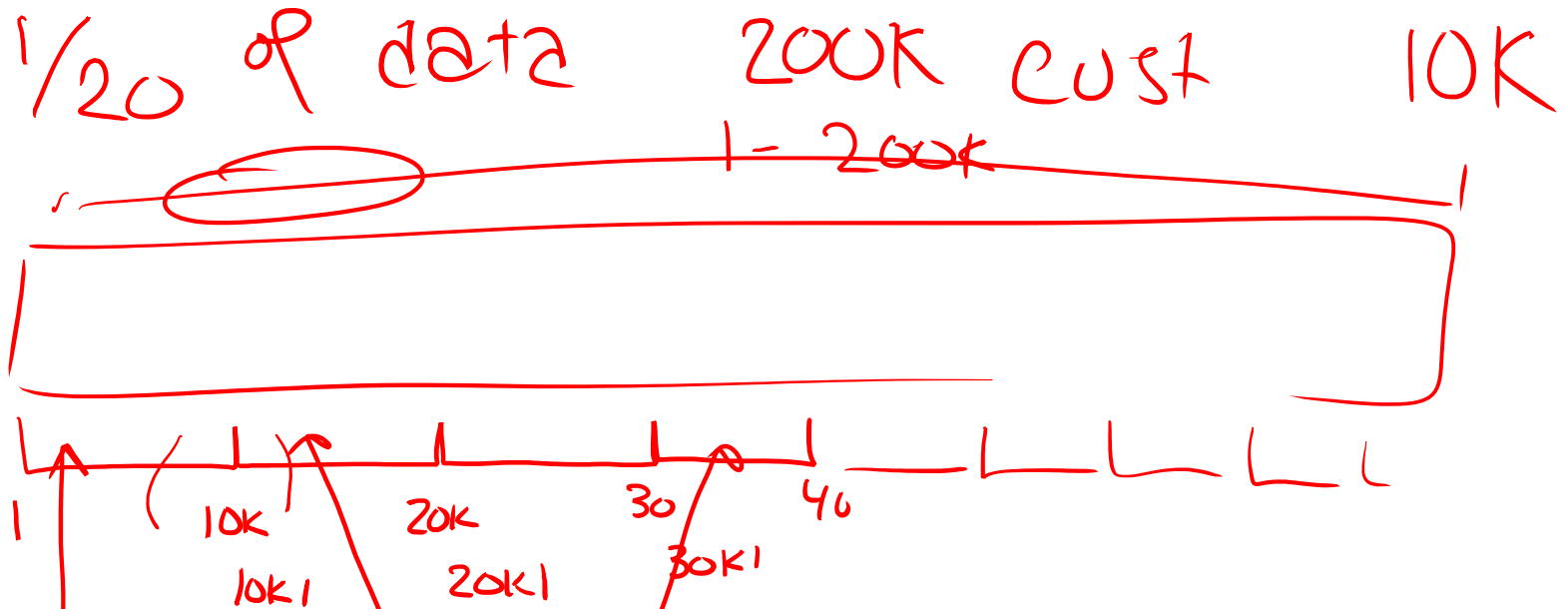
Filtered Statistics

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.



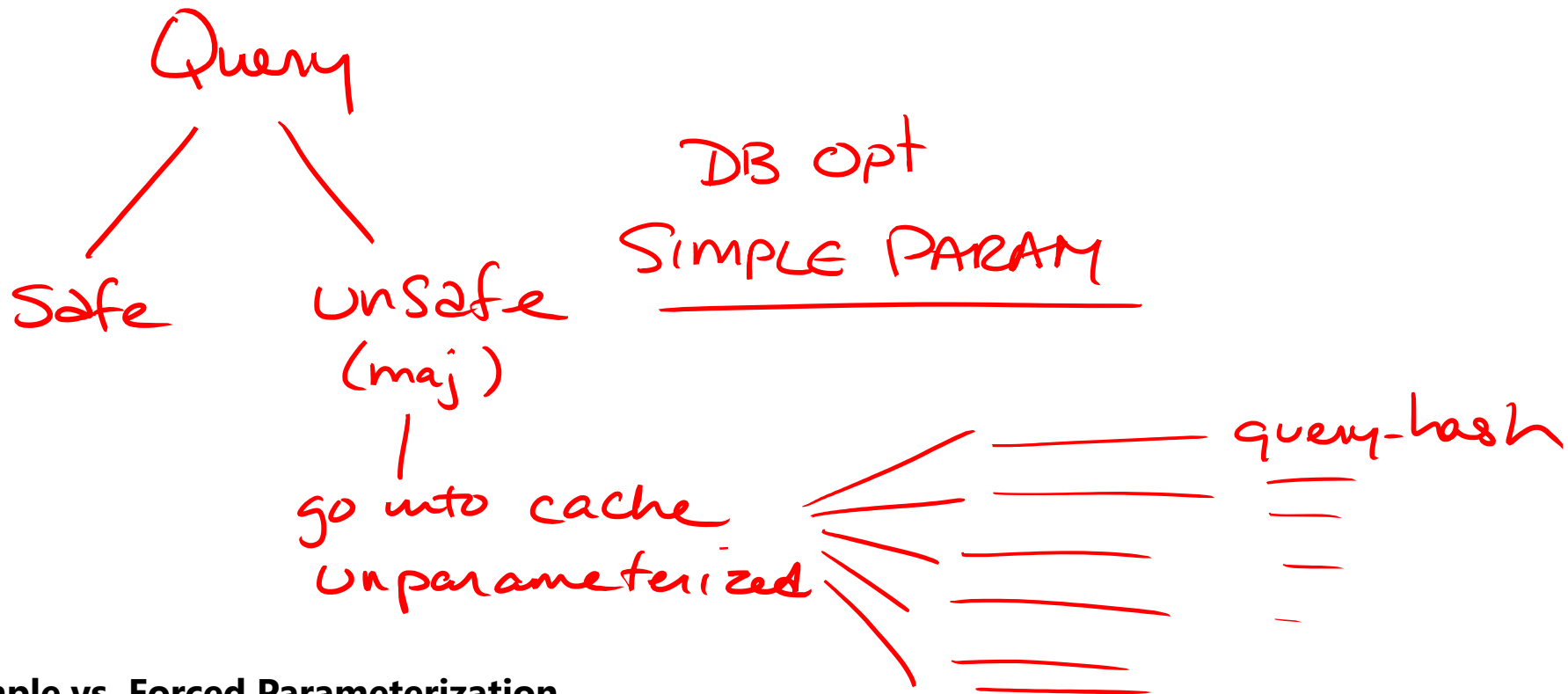


Creating Filtered Stats

The idea is to just have better statistics than what you have currently. To do this you can divide the range into 10-20 buckets. This will give you 10 times (or 20 times) better information in terms of statistics.

You WILL need to regularly/automatically review the values/ranges and add more or divide them up again to ensure that the statistics are good (and stay good)!



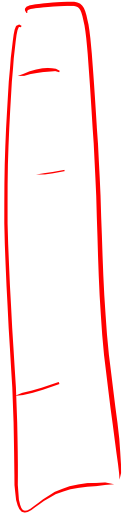


Simple vs. Forced Parameterization

The general process is that SQL Server analyzes a query to determine if it's safe or not (the majority of them will **NOT** be safe). If it's safe then it can get parameterized and reused. If it's unsafe then it goes into memory as an individual query (and it's harder to determine the cumulative effect of these queries). Check out the `query_hash` and `query_plan_hash`.



table



Status = 1

filter = 1

Query



Where Status = 1

Simple analyze use index

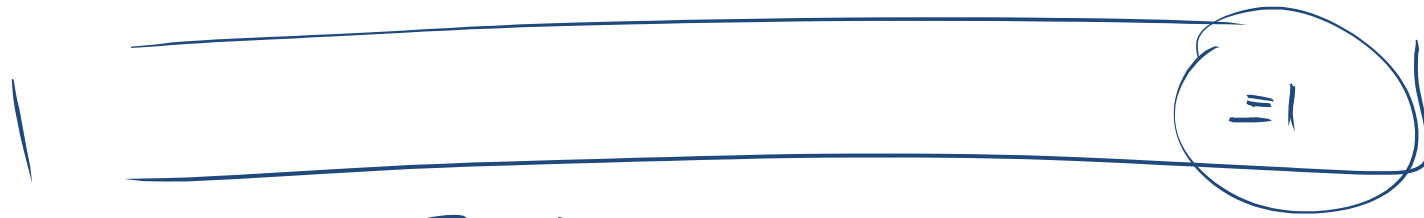
Filtered Stats and Forced Param

The bad news... there's always bad news.

Forced parameterization: even when SQL Server would NOT have parameterized the statement (because it had deemed the statement's plan as "unsafe" to re-use), forced param will force it. In systems where plans are very stable (but A LOT of adhoc) then this could be great. However, the example of status = 1 (in your query) is converted to status = @1 so the filtered index/filtered stat cannot be used.

Forced status = @1





Simple

Status = 1 not
save plan

Forced

Status = @1

Filtered indexes and filtered stats – Auto update

The bad news... there's always bad news.

For Updates:

statistics for a filtered index OR a filtered stat – do NOT get updated until that column's statistics get updated (which is when the colmodctr) is reached



~~F_1~~
 ~~F_1~~
 ~~F_1~~
 ~~F_1~~

~~$status = 1$~~

$status = @1$

Database option: **FORCED** parameterization

Because of how forced parameterization changes each variable to a parameter – the value of that parameter is unknown. As a result, a filtered index (for example, WHERE status = 1) cannot be used when a query has been parameterized to status = @1. There's no guarantee that the value they're searching on is 1.

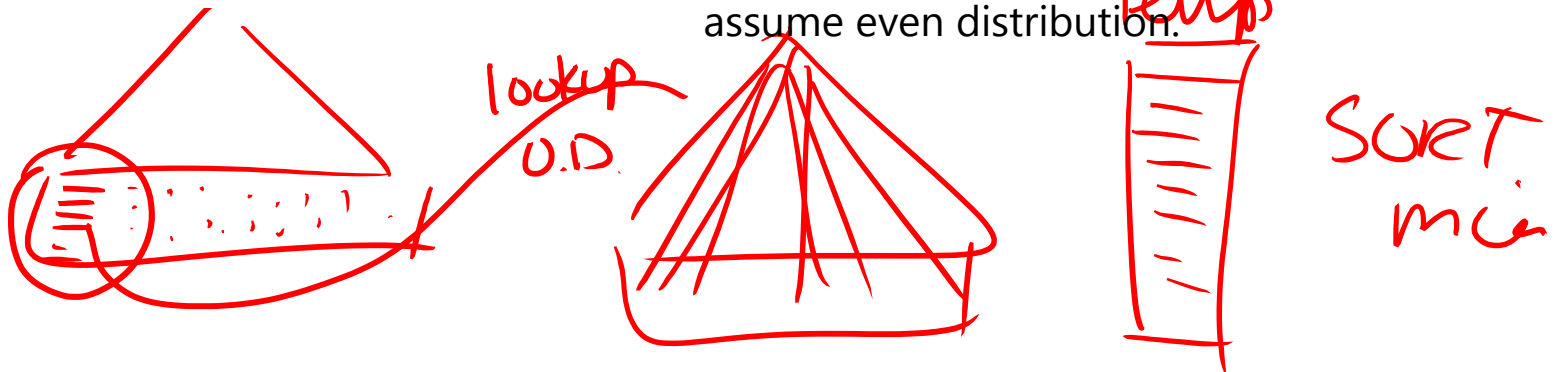
End result. You will NOT have good results with filtered indexes IF you use FORCED parameterization. **The good news:** This is NOT on by default and not likely to be on in most environments.



TS ?

FactInternetSales2

Index on ShippedDate



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.

Temp

Index on OrderDate

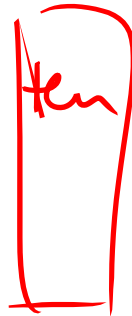
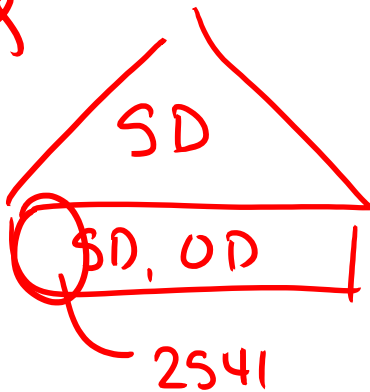


when will I hit a NULL

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



Missing
Index
DMV



Score

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.



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.

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



Poor Man's DW

OLTP → B/R

Poor Man's Datawarehousing

Setting a database to RO (read-only) can be a good idea when you plan to use it solely for reads. However, how does SQL Server update statistics or add statistics?

Really, it can't.

So, the main point... if you're going to move a backup to another server for read-only access – you should automate some basic optimizations before setting it to RO.

- ① Update stats
- ② Rebuild Indexes

FF = 100

~~Update stats~~

Sampling?

- ③ sp_createstats
Index only

Full scan - ?

READ ONLY



Kimberly L. Tripp

Kimberly@SQLskills.com



SQLskills

VLDB Design Strategies / Techniques

Discussions / drawings around VLDB
and “partitioning” large data sets (VLTs)

Partitioning: PVs vs. PTs

Partitioned views

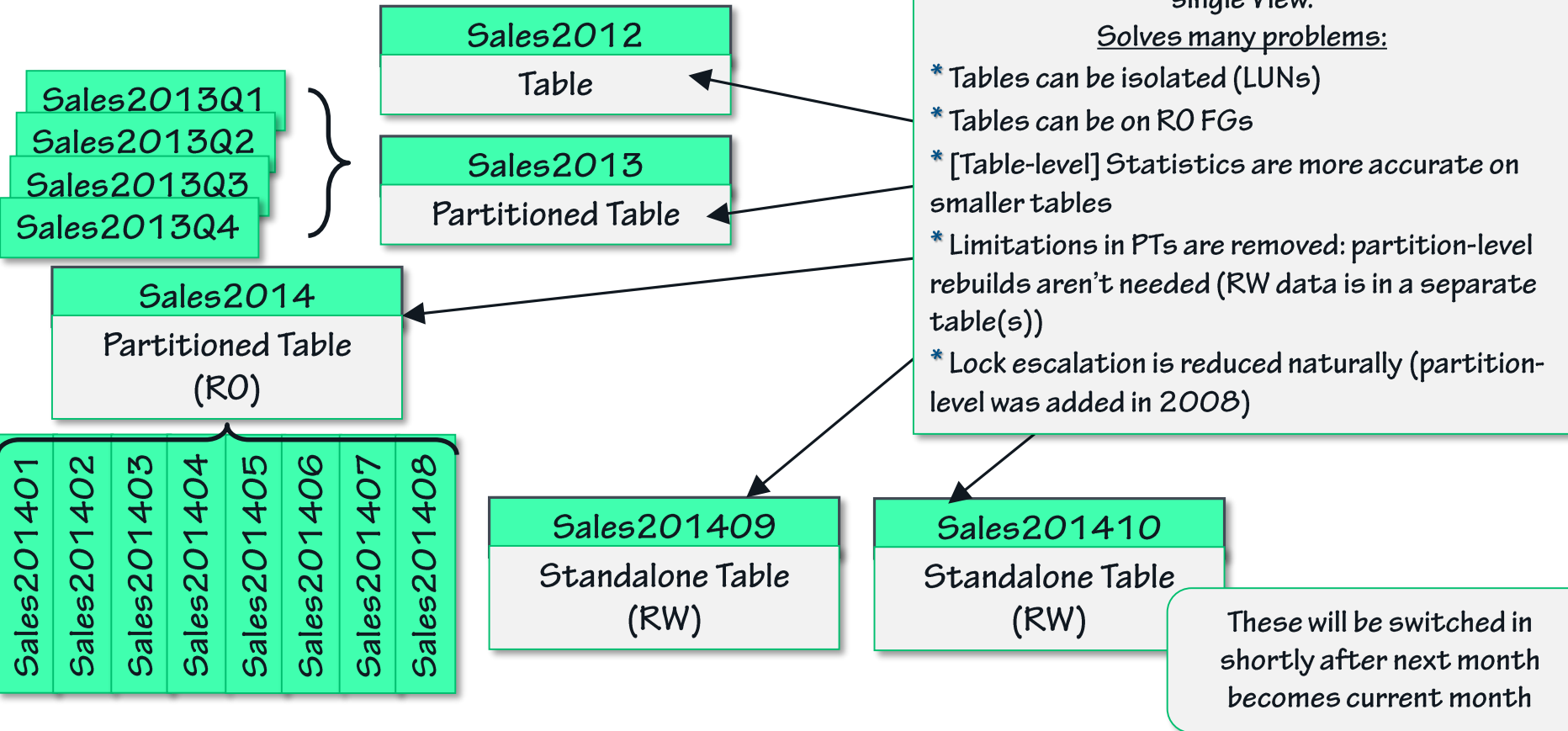
- Any edition
- Lots of tables to administer
 - Must create/drop indexes on all base tables
 - Can have different indexes
 - Harder for the optimizer to optimize with so many indexes
 - Must verify business logic so that there are no gaps or overlapping values
 - Each table has [potentially] better statistics as the tables are smaller
- Can rebuild any of the tables ONLINE (if using EE)
- Can support multiple constraints on one or more columns

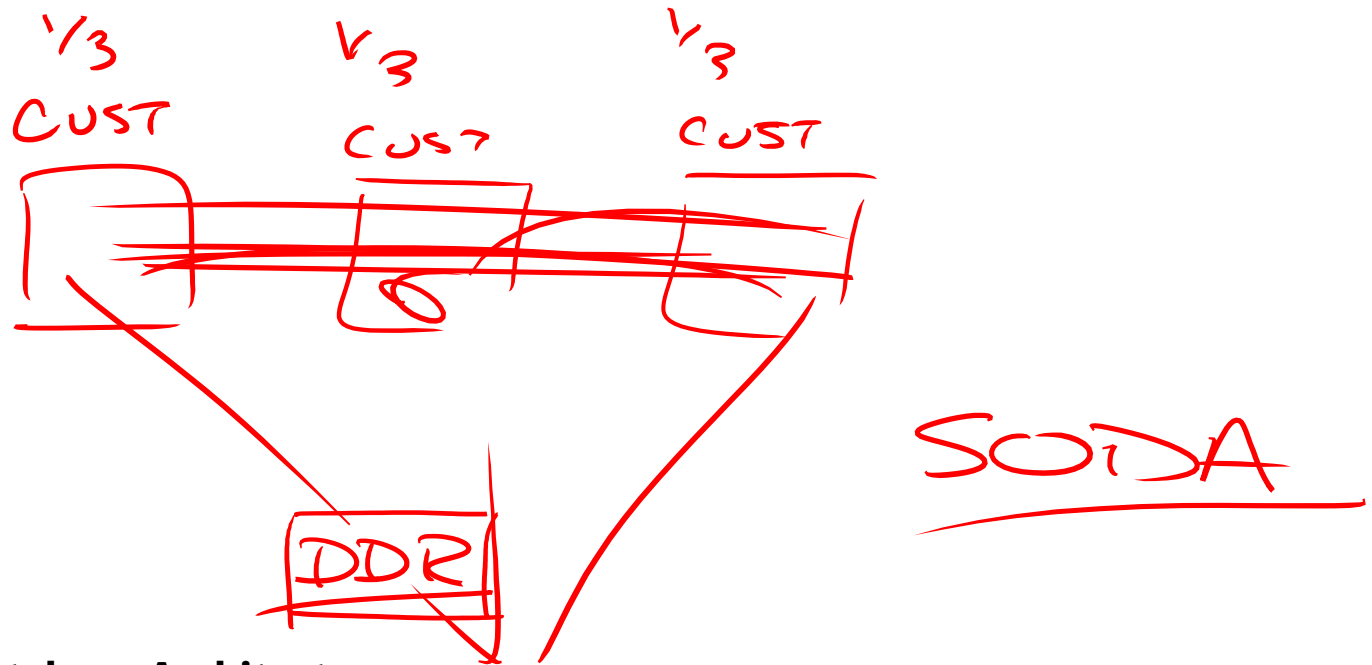
Partitioned tables

- Enterprise Edition only
- Only 1 table to administer
 - Only 1 table to create/drop indexes
 - All partitions have same indexes (which is easier for the optimizer to optimize)
 - Can create different indexes with filtered indexes
 - No possibility of errors (or gaps or overlapping values)
 - Table-level statistics can be less accurate for very large tables when there's a lot of skew to the data
- Partition-level rebuilds are offline but can rebuild the ENTIRE table online (not desirable)
- Can only support partitioning over a single column



Functionally Partitioning Data



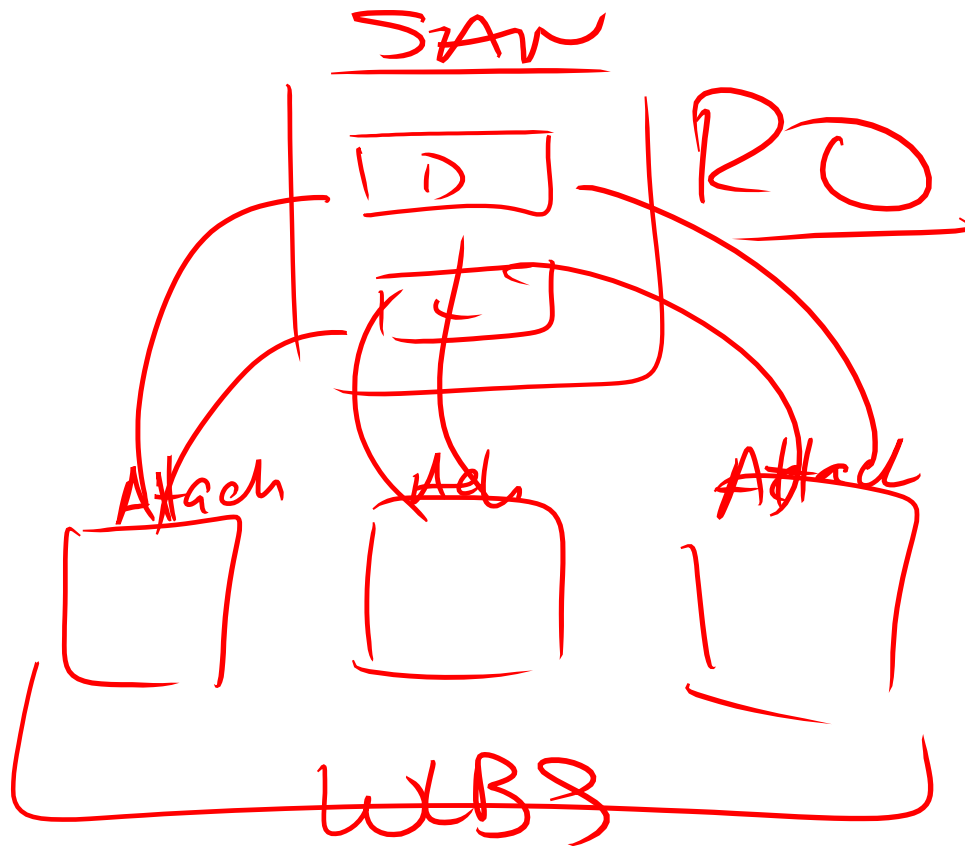


Service-oriented Database Architectures

Partitioning in our module was all within a single database. However, we had a side/note about sharding and scale-out design. The most important thing that I can highlight is that scaling out is most ideal through middle-tier DDR (data-driven routing) where the applications are directed to the appropriate server. If every user randomly goes to any of these instances and all requests go through [distributed partitioned] views then performance will likely be worse!

Check out the whitepapers on SODA (service-oriented database architectures).





Scalable Shared Databases

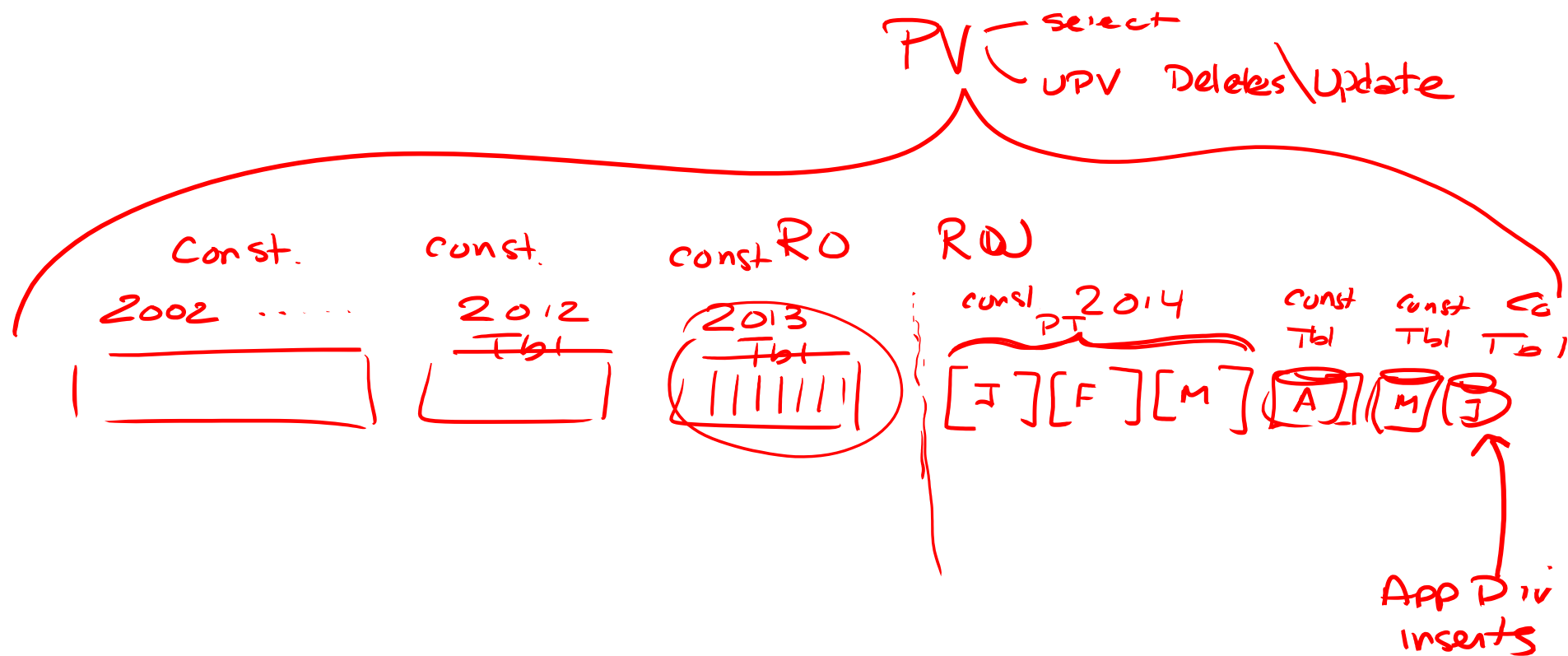
Partitioning in our module was all within a single database. However, SQL Server does support Scalable Shared Databases. These are RO databases that are attached to multiple servers and then balanced through WLBS.

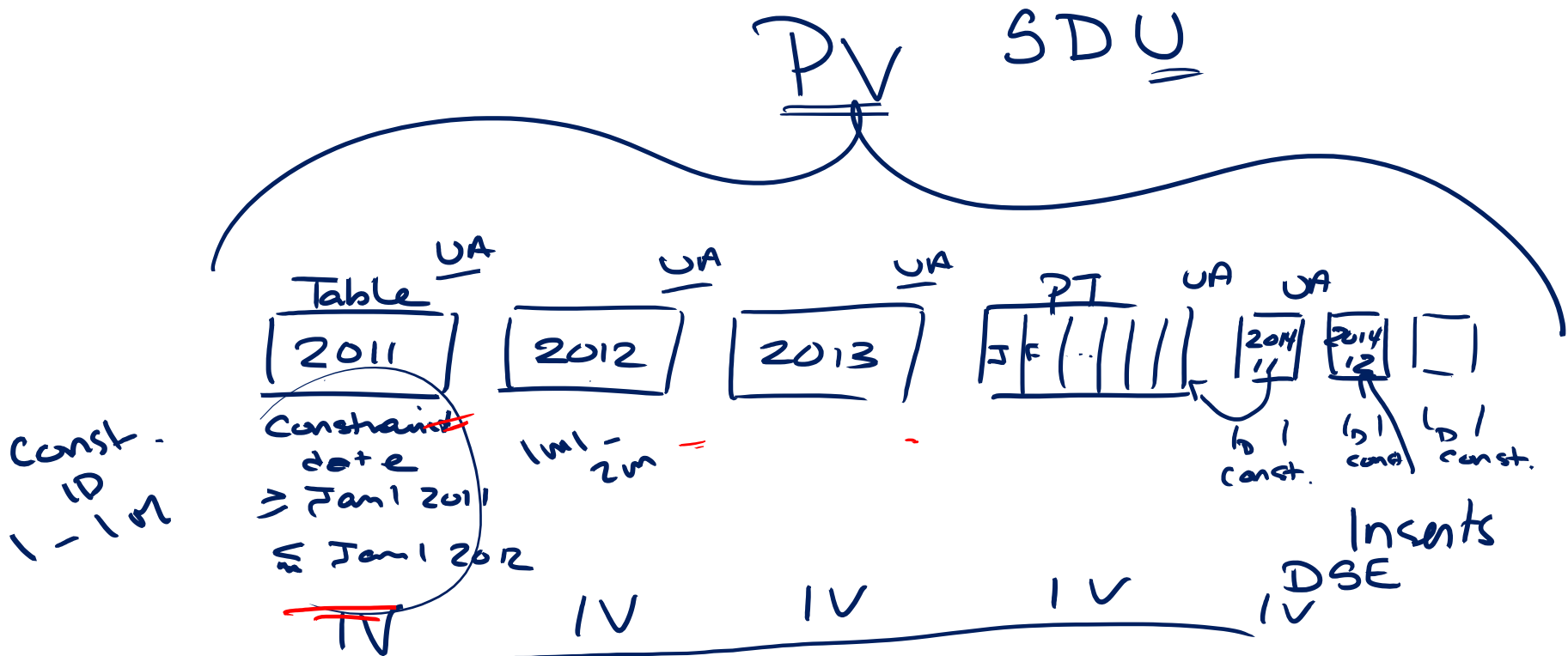
[Scalable Shared Databases](#)



Architecting a “layered” approach to your table design

A better option is to separate the RW from the RO and put the RW in separate tables. Then, you can do online rebuilds at the table level. How do you deal with queries – put a partitioned view over them!





Optimizing the VLT (Very Large Table)

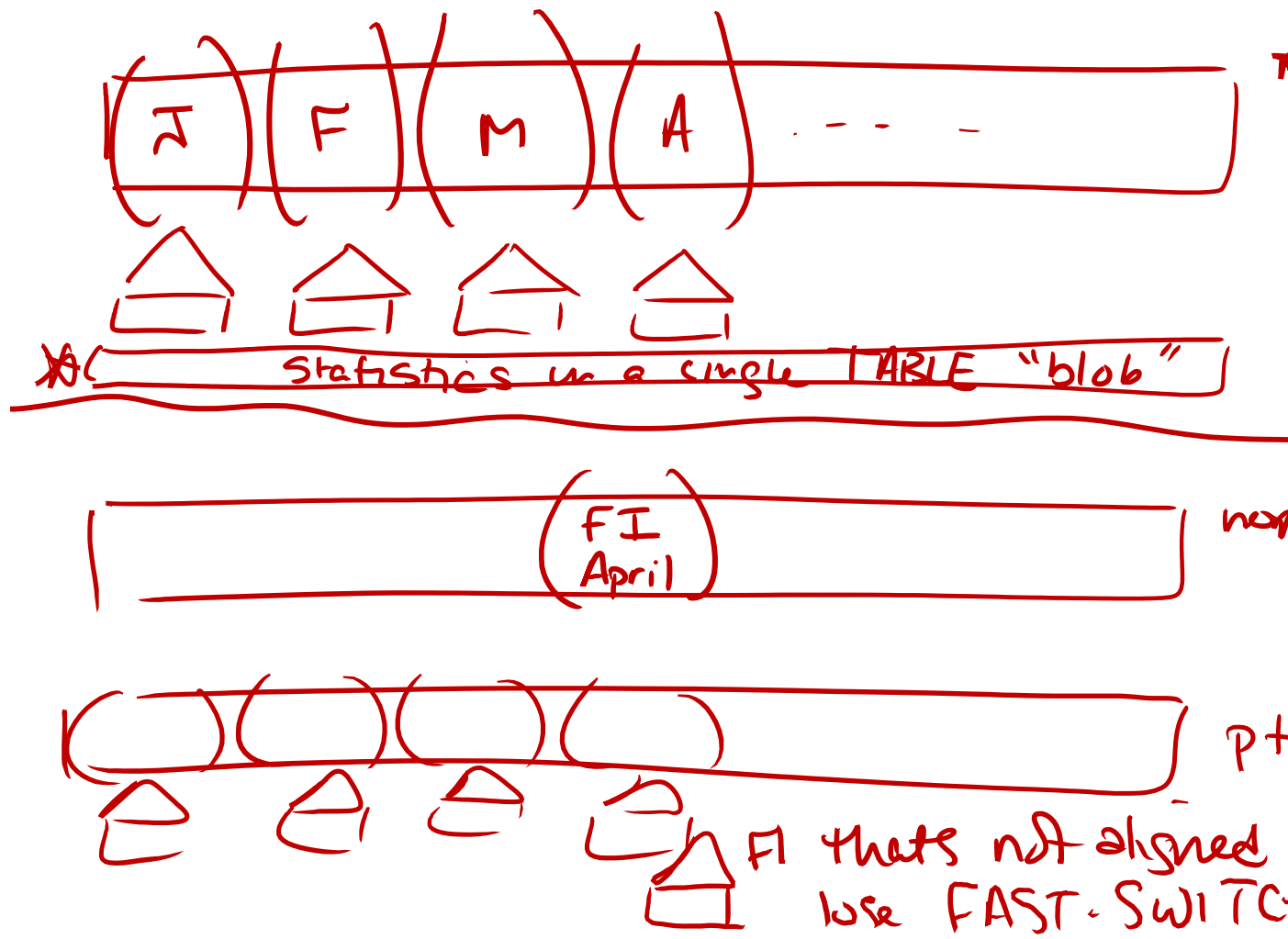
A very large table has many "problems"... to reduce those – don't have just ONE VLT – have smaller tables that are unioned (using UNION ALL) into a view. If you also have restrictive constraints (CHECK constraints) across all of the base tables this combination is called Partitioned Views. You get numerous benefits with this architecture including: better control / manageability, better statistics on each table, online operations, and the reduction of some operations all together (on the historical data).



Filtering v. Partitioning

One of the frustrating things about a large table is that – even when partitioned, the statistics cover the entire table (leading to inefficiencies).

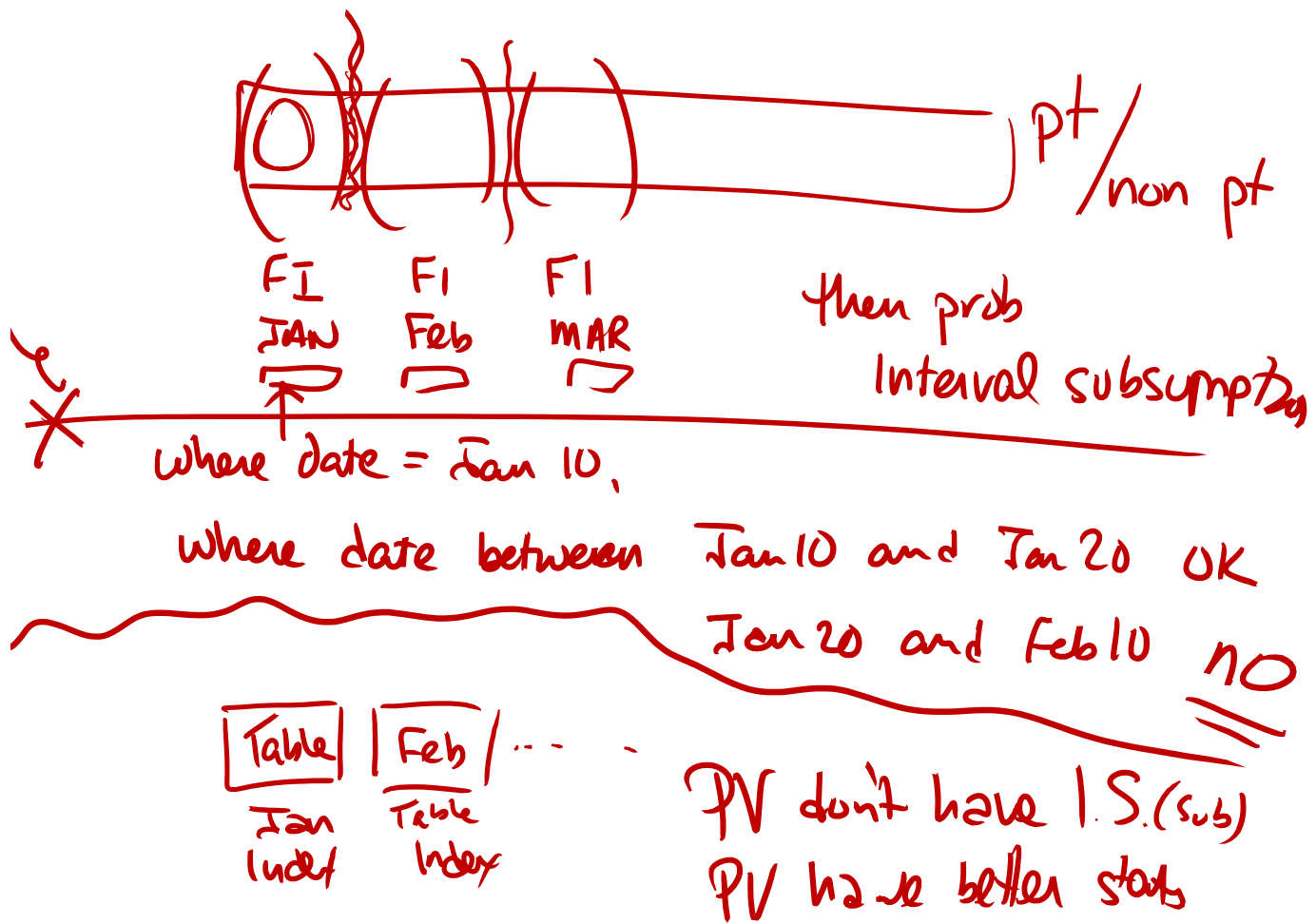
If you create a filtered index over just a single partition you get better statistics and an isolated index (just for the queries that need it) but then you lose the ability to do fast switching.

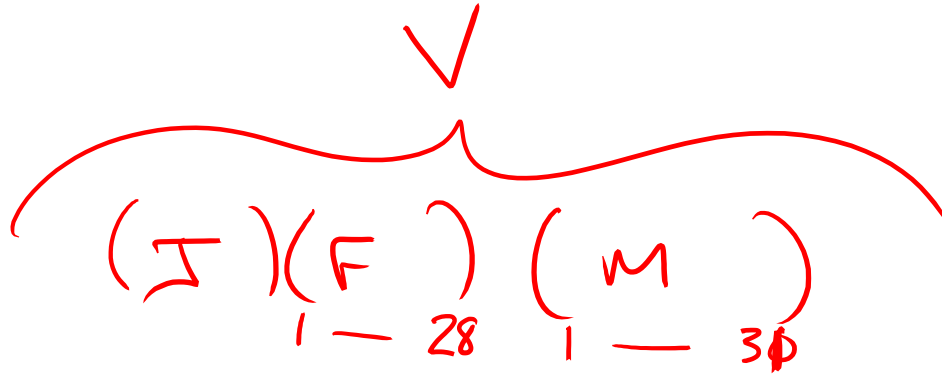


Filtering v. Partitioning

You might think that using JUST a filtered index approach would be better but then there's the interval subsumption problem.

Architecting the RIGHT solution and breaking down a VLT into smaller tables can be ideal. Partitioned Views (PVs) do NOT have interval subsumption problems. And, PVs have better statistics...





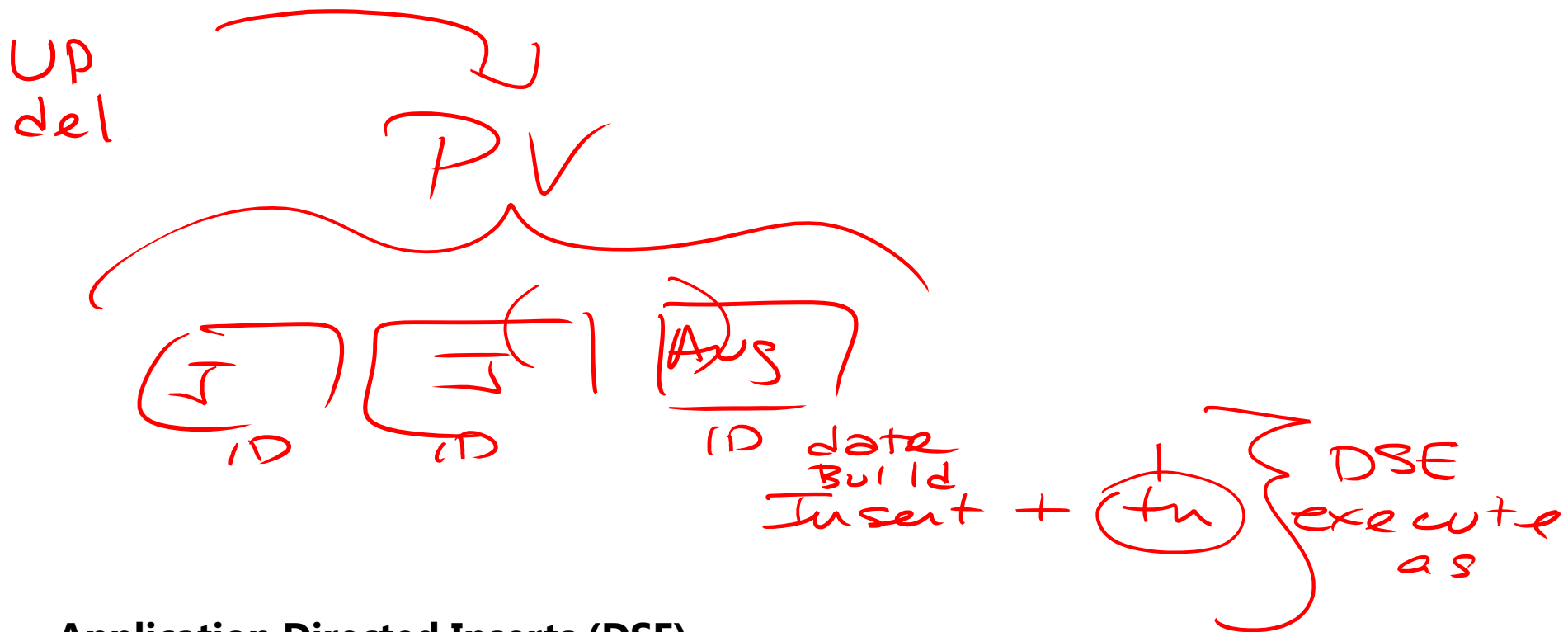
Partitioned Views and business logic

Be careful with partitioned views... there's nothing that's going to test your business logic. As a result, if you miss a day (for example Feb 29, 2008) then inserts into the view will fail because there's no view that can store that date.

Alternatively, using partitioned tables – there's no way to have a gap or overlapping ranges.

But, there's a lot more to PVs and PTs (this isn't enough to choose one over the other) so this is just a bit more info to add to the list!

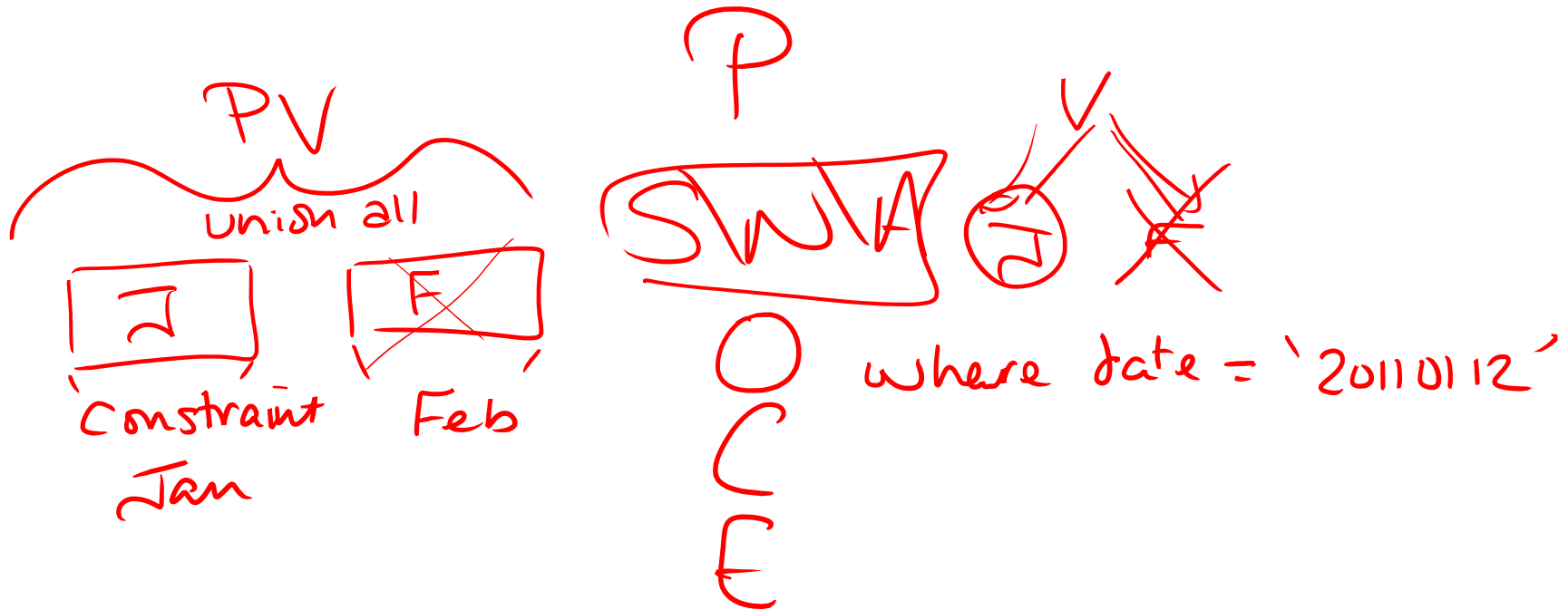




Application Directed Inserts (DSE)

Even with updateable partitioned views, I usually use application directed inserts. If you're concerned about dynamic string execution check out the Little Bobby Tables blog post: <http://www.sqlskills.com/BLOGS/KIMBERLY/post/Little-Bobby-Tables-SQL-Injection-and-EXECUTE-AS.aspx>

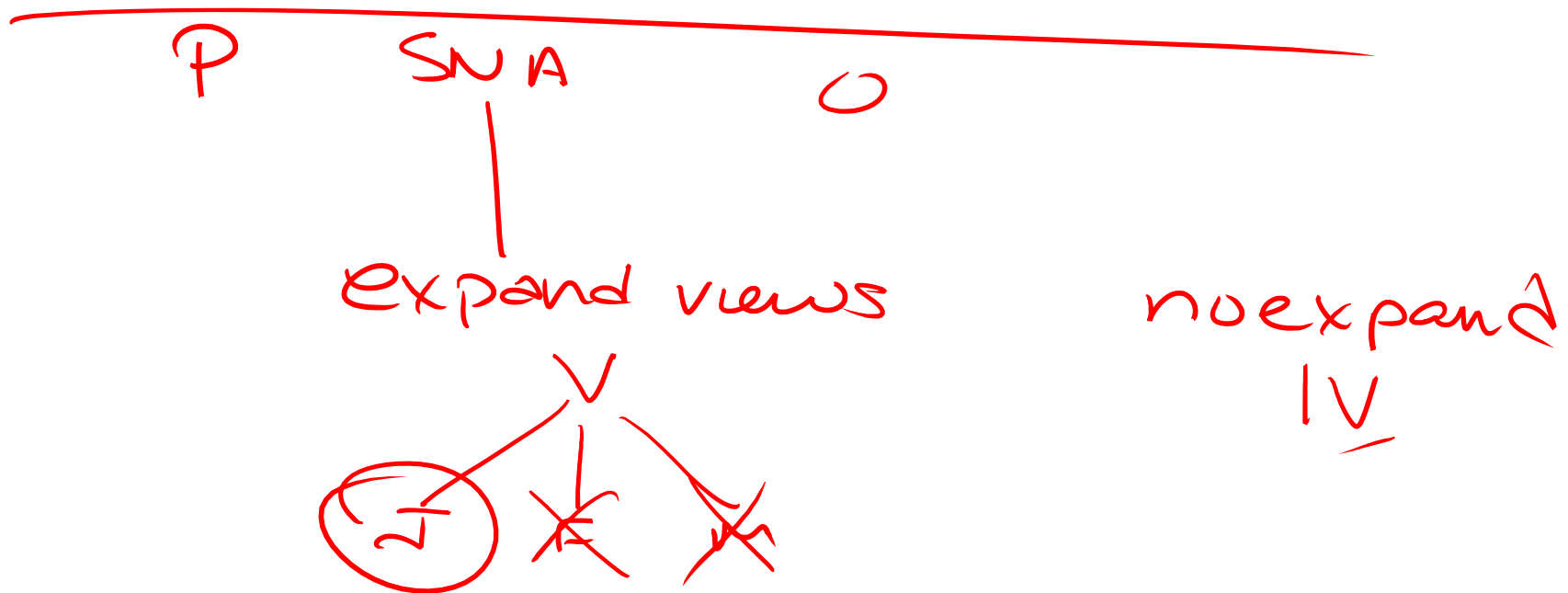




Partition Elimination

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. 😊)

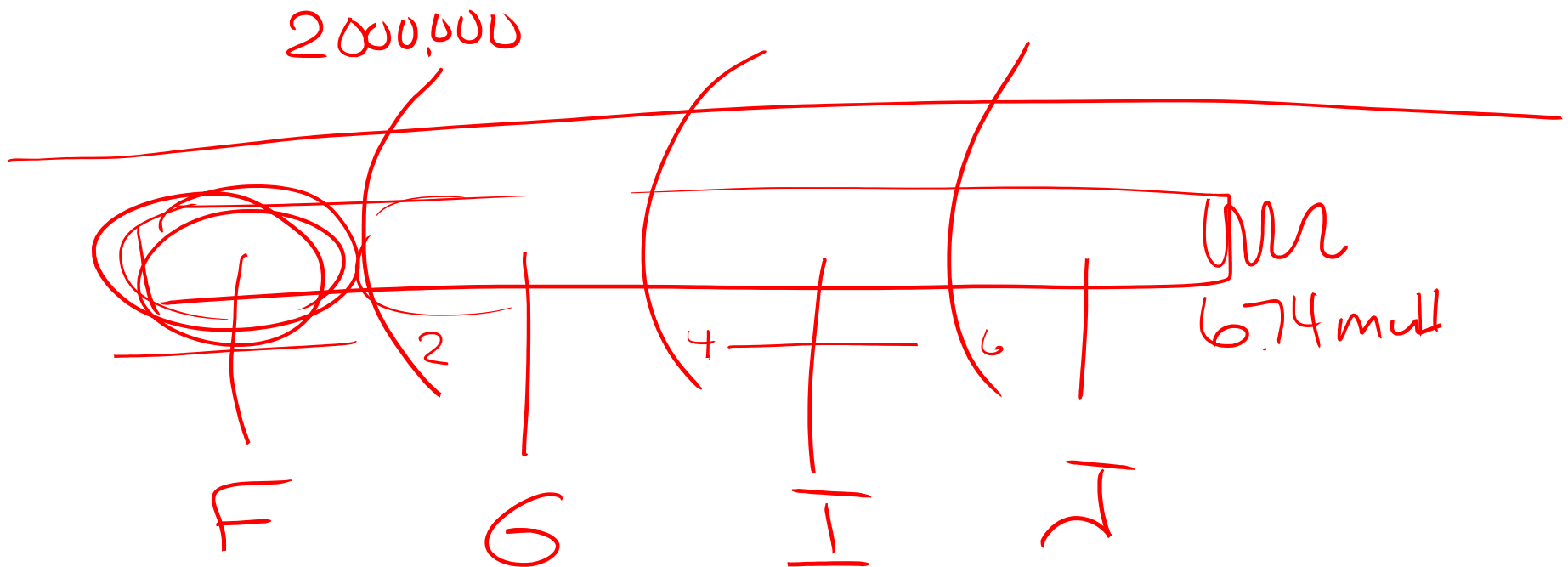




Expand Views

Another way to look at this is that there are multiple phases of query processing. The second phase (standardization, normalization, algebrization) is where views are expanded to their base tables (known as EXPANDVIEWS). This might sound familiar because of the hint NOEXPAND – which is used to force SQL Server to use indexes on views (if for some reason SQL Server isn't using the IV).





Partitioning on integers (rather than date)

Another example of a partitioned table with 3 boundary points: 2million, 4 million and 6 million. It's a RIGHT-based partition function so the rows that match the boundary point will be to the RIGHT side. Specifically this translates into:

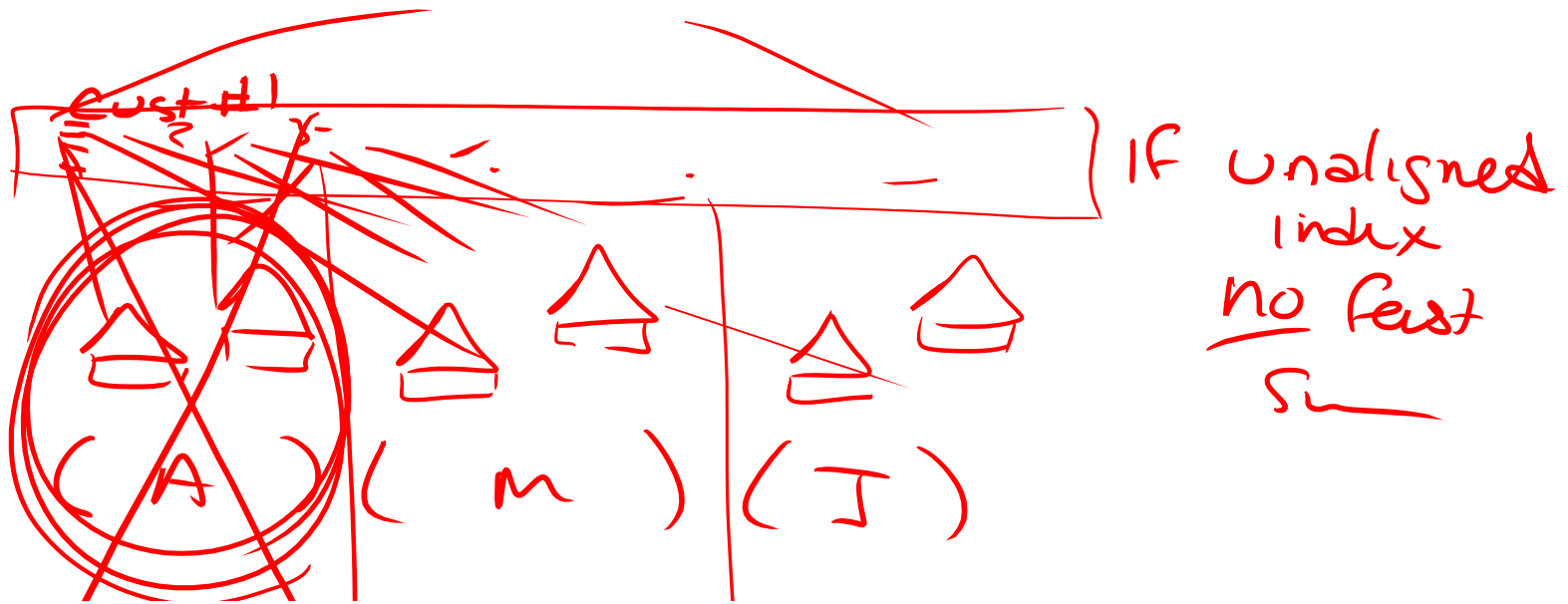
- In partition 1: all rows less than 2 million

- In partition 2: all rows equal to or greater than 2 million and less than 4 million

- In partition 3: all rows equal to or greater than 4 million and less than 6 million

- In partition 4: all rows equal to or greater than 6 million





Aligned Indexes

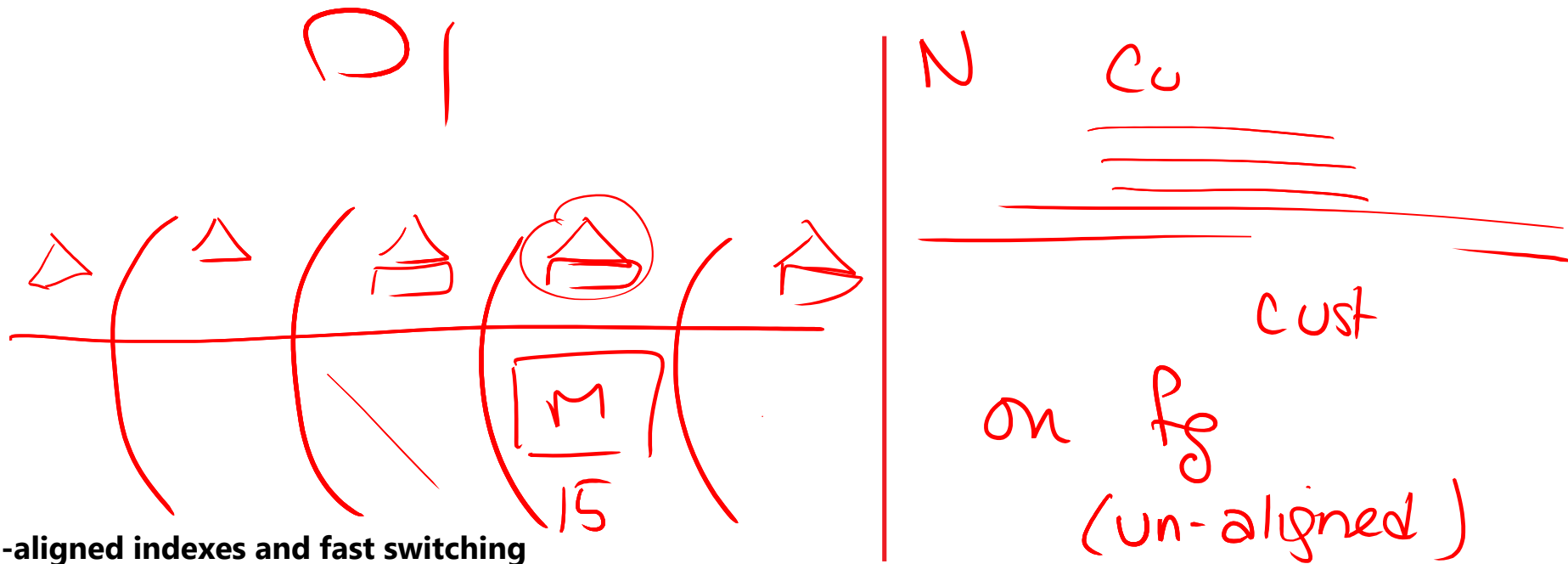
Indexes can be aligned to the same partition scheme:

- Either by creating them ON SCHEME(col) or
- Accepting the default behavior during creation. Nonclustered indexes default to being created on the same scheme as the clustered index – unless they are an UNIQUE index. If the index is unique then the partitioning column must [explicitly] be part of the key.

Indexes can be unaligned

- The index has all of the nonclustered index data for the table in one leaf structure
- Fast-switching is NOT allowed if unaligned indexes exist





Un-aligned indexes and fast switching

For fast switching to be allowed – you must have ONLY aligned indexes. For an index to be aligned – there are rules. If the index is going to be unique (and aligned) then it MUST include the partitioning key. Indexes will default to being created on the same scheme as the table (they will default to being aligned – and therefore have all of these requirements).

However, **if you're not interested in fast switching** – then you can have un-aligned indexes. In this case you create these indexes on a specific filegroup (or on a different scheme). These un-aligned indexes can be unique and do NOT have to include the partitioning column.



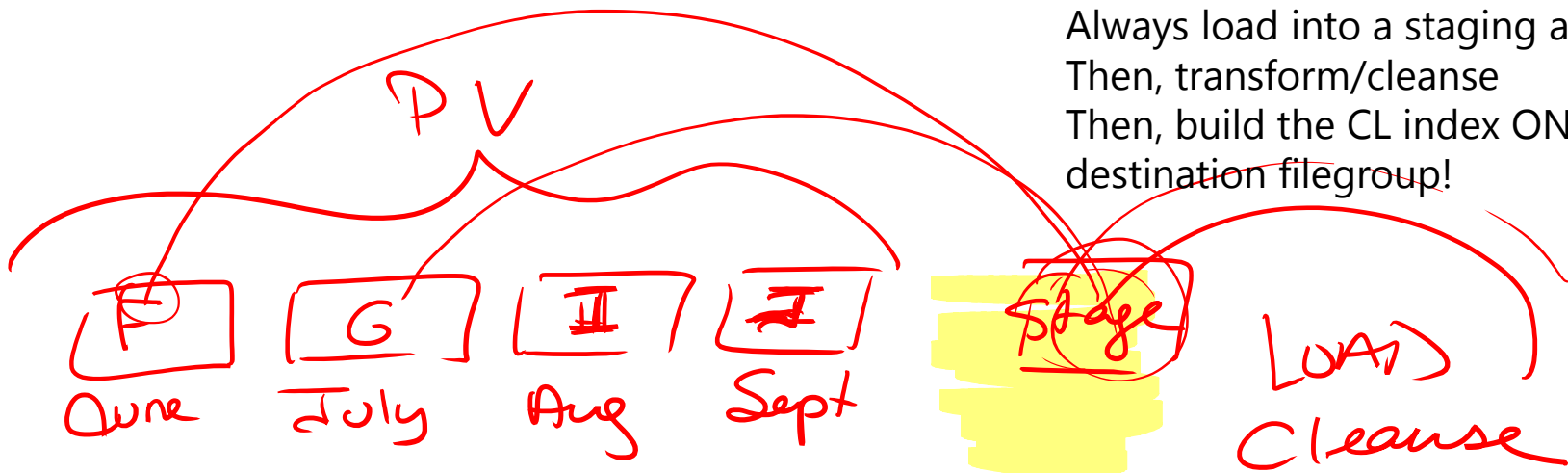


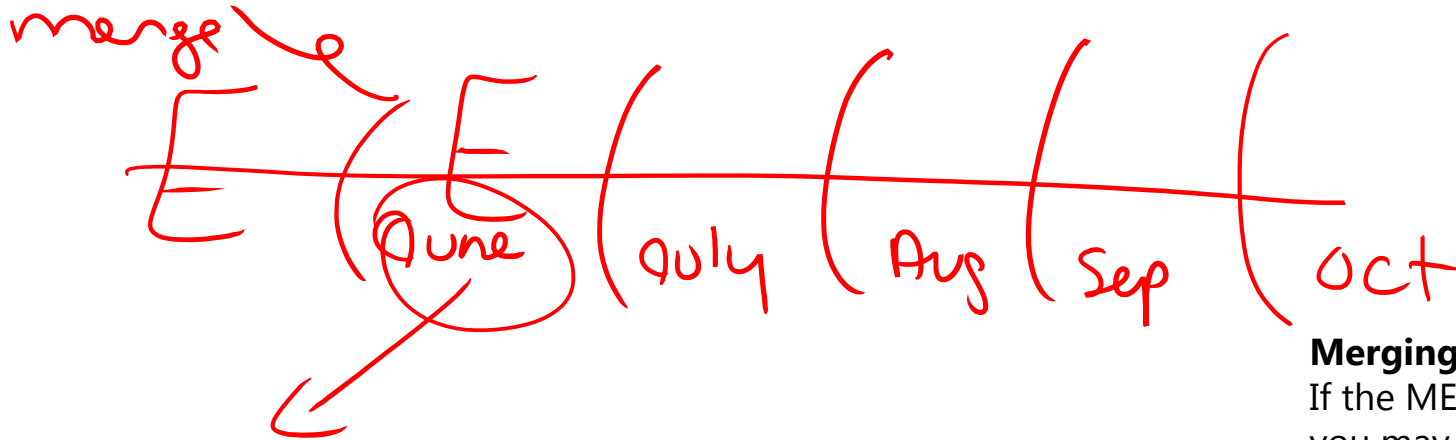
Staging Data

Why do you need a staging area?

- (1) So that you don't need to overallocate space within destination filegroups
- (2) So that you don't have to shrink (see s54)
- (3) So that you can optimize the process...

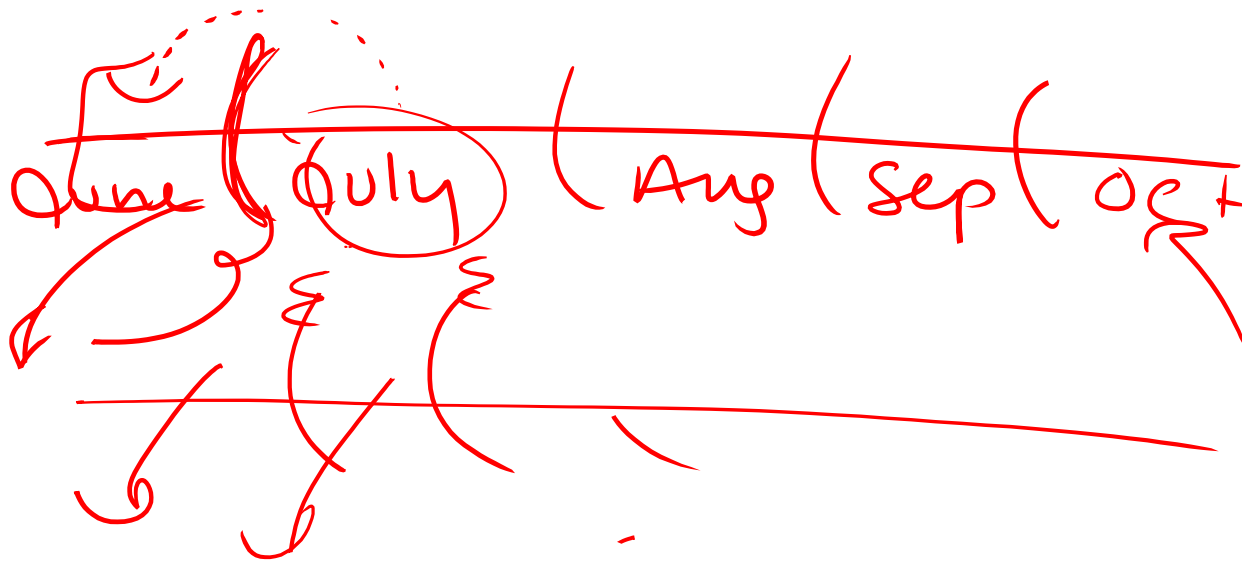
Always load into a staging area first.
Then, transform/cleanse
Then, build the CL index ON the destination filegroup!





Merging the right boundary

If the MERGE process is slow – you may have merged a boundary point that was NOT empty.



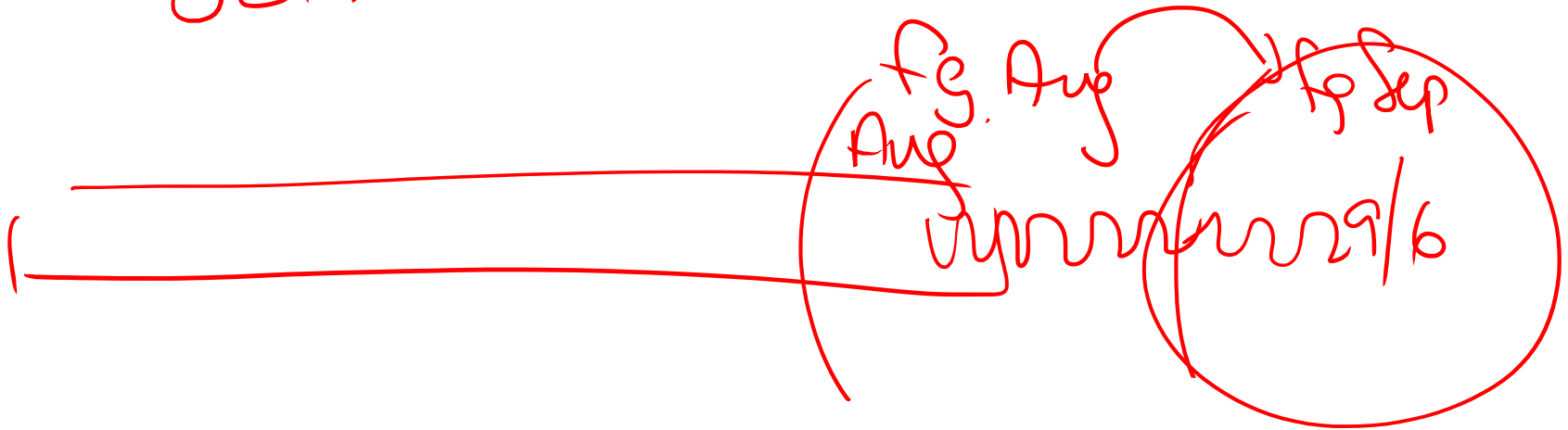
The most common case of this is when you start with a non-empty first partition using RIGHT-based partitioning. With RIGHT-based partitioning you should not MERGE until you have TWO empty partitions that surround the emptied boundary. Then, you can MERGE (top diagram).



Rebuilding the active partition – OFFLINE ☹️

With PTs – it's likely that your last partition (often, the current data) will be active. This is also where it's most likely that you'll have fragmentation. If you want to rebuild ONLY that last partition – you'll need take it offline to do it.

OLTP



Criteria
Always 1 yr.
What if Q3 0

Always 1 yr.
what if $q_3 = 0$

merge
tiny

After switch

Scheme fs use next fgx ←
function Split 20040701 boundary

Trust?

↳ Constraint

OUT

CR 2003Q3 on ~~FS1~~

CR CL

CR NC

CR $\vee \vee \vee$

CR CS

drop table?

IN

Heap Staging
Cleanse (CI?)

Ready for prod

CR CL on $f_g x$

CR NC on fix

12 V / IV

CR CS

2008+ →

2012 \rightarrow

S49 – The Sliding Window Scenario

See the next slide for full details



The prior slide showed the sliding window scenario – but we went through it slowly:

(1) Preparing the table into which we'll switch OUT our old partition

Review all of the slides for the requirements here but the key point is that this “staging” table MUST be on the same filegroup as the partition you are going to switch out.

(2) Preparing the table for our data load and what will become our new partition to switch IN

Again, be sure to review all of the slides for the requirements here but one thing you need to make sure of is that there's a TRUSTED constraint on this table before you switch in.

(3) Change the partitioned table to support the new filegroup and data range

Always set the NEXT USED filegroup with ALTER SCHEME

Once you have the correct filegroup specified then ALTER FUNCTION...SPLIT

- Now, you're ready

(4) and (5) can be in any order. SWITCH IN the new partition and SWITCH OUT the old.

(6) Clean up

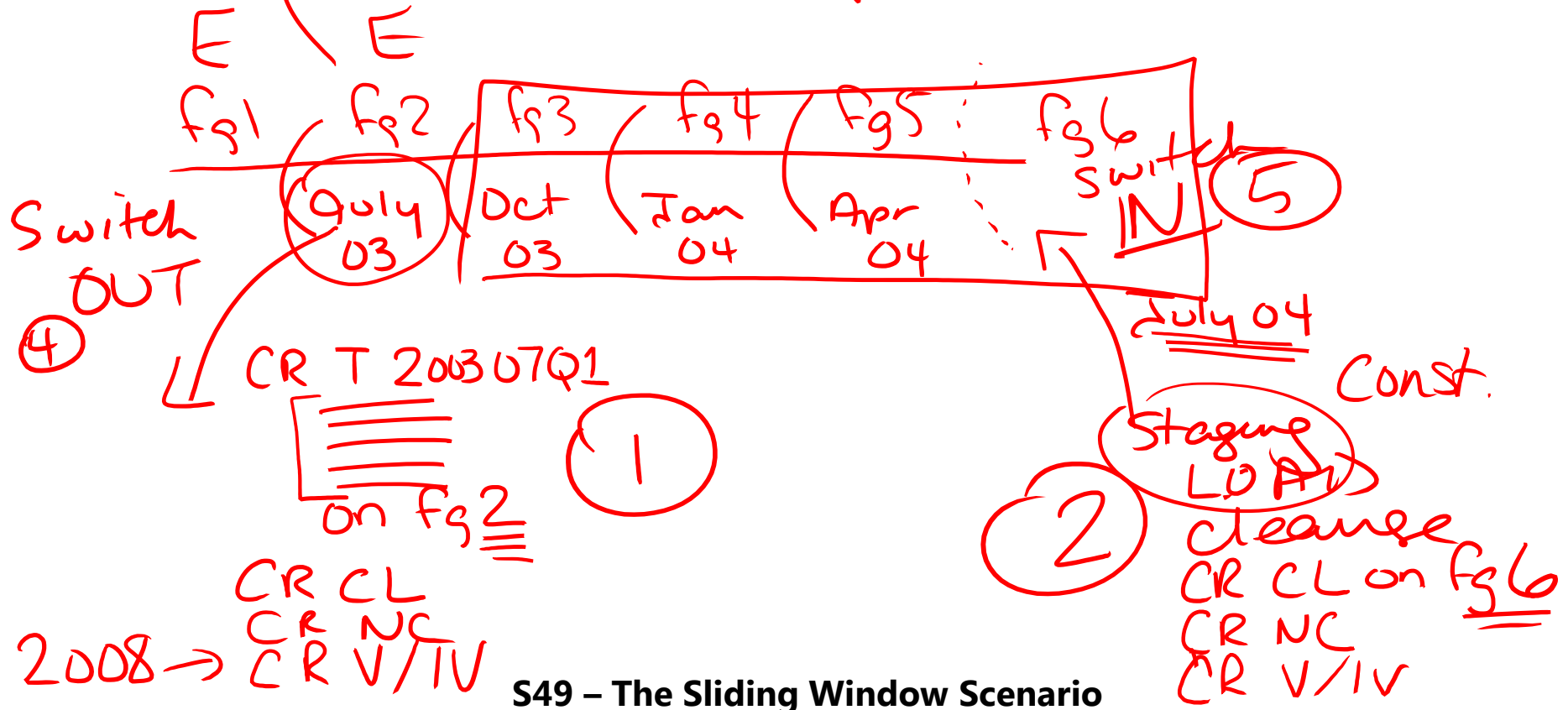
Merge the boundary point but ONLY if it's empty (the next picture will remind you of what happens when you don't MERGE an empty boundary point)

Backup the filegroup where the partition resides that you just switched out. OR, drop the table.



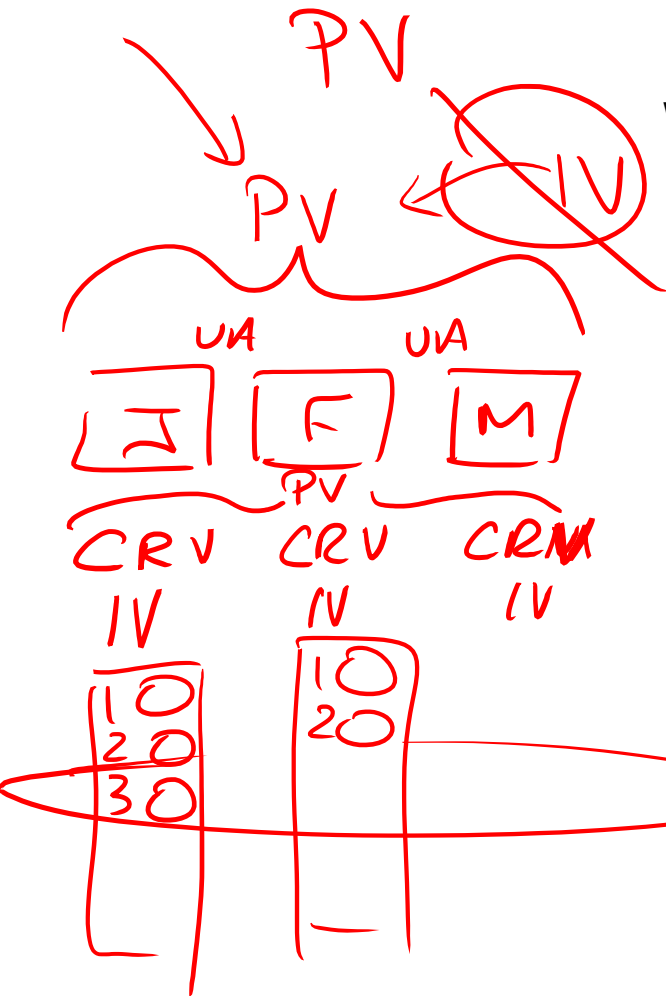
⑥ merge

③ Scheme next used fsg
split @ 20040701



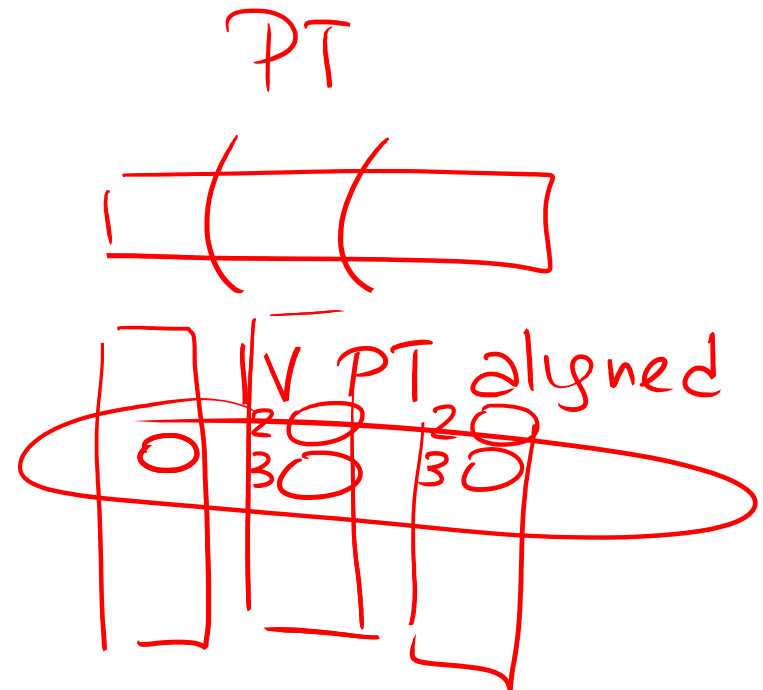
S49 – The Sliding Window Scenario





With PVs - If you want to create indexed views, you must create them individually per table (for the tables that underpin the PV). If you're not on EE then you'll also need to create a specific view to access them with


With PTs - If you want to create indexed views, you must create them as partition-aligned (for fast switching). This is available in SQL Server 2008+.





CL OD

mdf ((Sales) NC1 NC2 SalesID
C/D P/D NC PK (DATE, II)

Unique

fs1 ((JAN)  ID)

fs2 ((FEB)  ID)

fs3 ((MAR) )

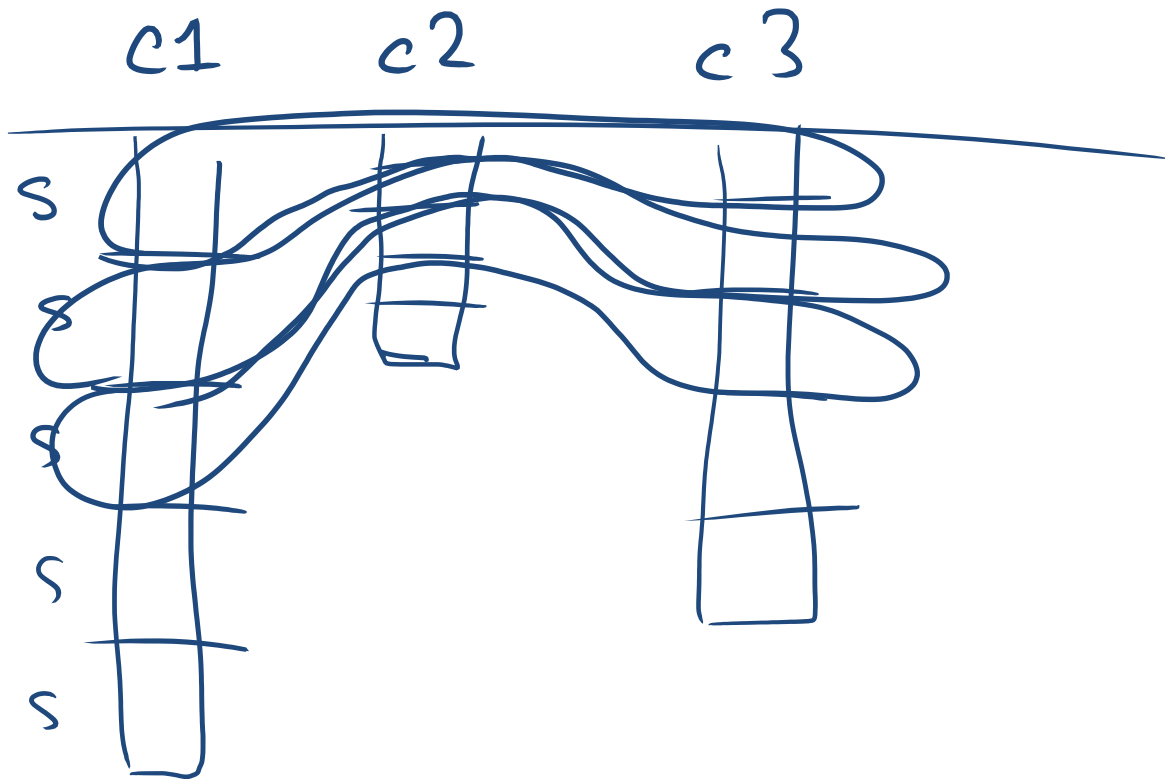
CRUCIAL INDEX
SalesPK on
Sales (SalesID)

Moving/partitioning the CL index

When you rebuild the CL index on a scheme the clustered index (the table) will be partitioned on the new scheme. However, nonclustered indexes will NOT be partitioned. Nonclustered indexes must be built separately/individually. And, they might need to be changed. All unique nonclustered indexes must have the partitioning column defined as part of the key.

If the CL table that you want to partition is clustered by date/id (and is the PK) then rebuilding that on the scheme is fairly easy. But, if it's not the PK and the PK is instead on SalesID then you'll change this PK to have the partitioning key as part of the PK. And, this means all FKs will need to change, etc. This can make the process of converting to a partitioned table become an offline process.

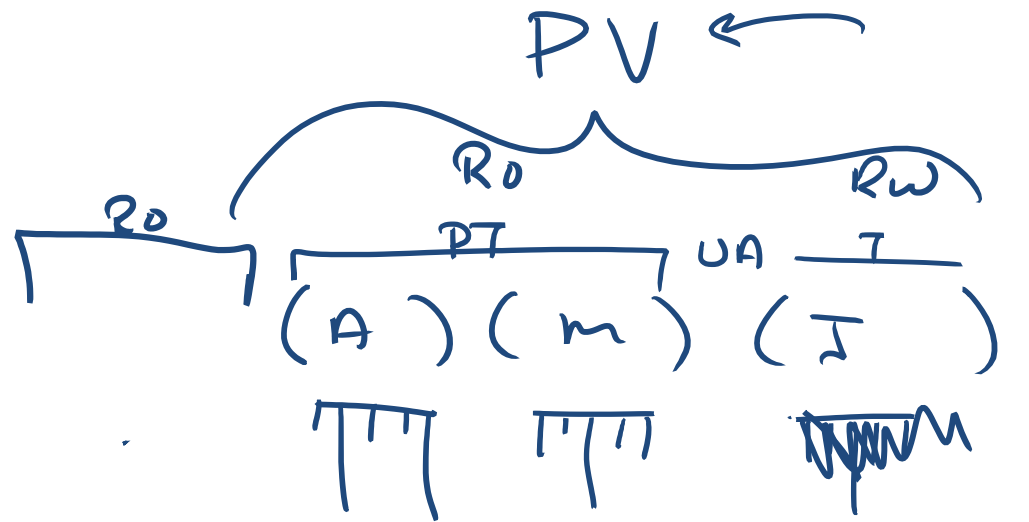




Columnstore Indexes

This was a drawing showing the possible compression of 3 columnstore-based indexes. Then, each columnstore index is broken down into segments. This is the base of batch-mode processing (another core part to the performance gains that can be recognized with columnstore indexes).





Might want to
have CS indexes
only on RO
tables

SP
(Q PT)
(Q T)
As
select:

Batch mode support

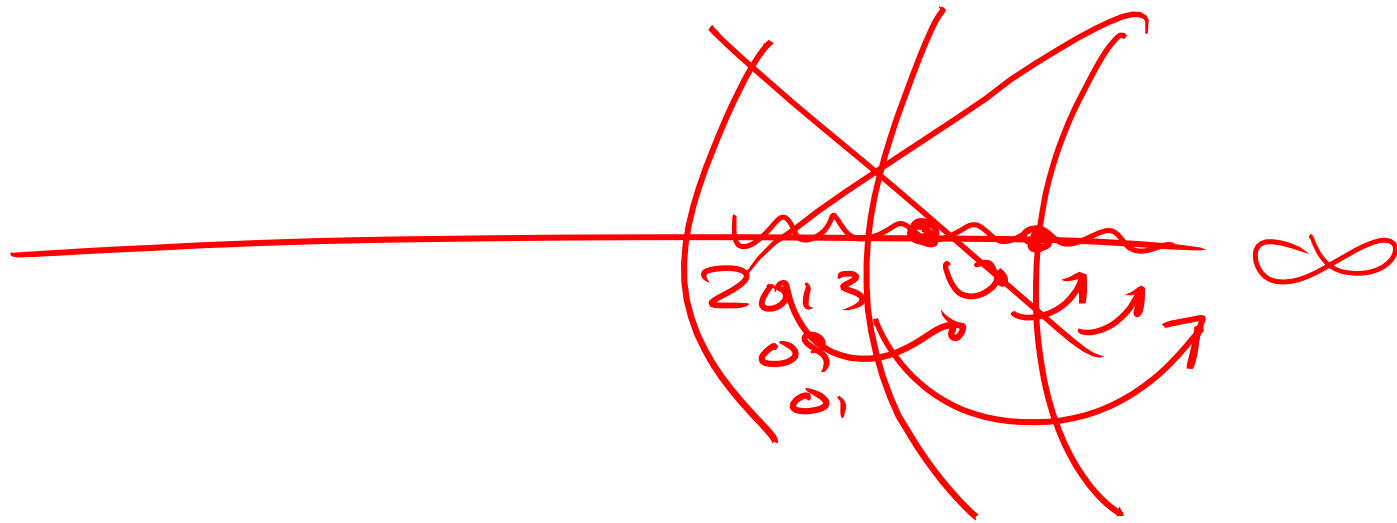
Since columnstore indexes make the table on which they're created read-only then you might want to use PVs to separate read-write and read-only data.

Having said that, one limitation of the current implementation of nonclustered columnstore indexes is that they do not support batch mode processing across views that include UNION ALL. The workaround is to use CTEs.

Check out the discussion: **Perform UNION ALL and Still Get the Benefit of Batch Processing** on the columnstore wiki here:

<http://social.technet.microsoft.com/wiki/contents/articles/perform-union-all-and-still-get-the-benefit-of-batch-processing.aspx>





Splitting a partition that already has data:

There isn't a slide to which this applies. This was a side discussion about splitting a partition after it already has data.

If you forget to split for October and November and the last split was for Sept 1 – then, instead of splitting for October (which has to move BOTH October and November data) and then splitting for November (which has to move November's data again) – you should always split the last set (November) and then the earlier (October). Then, November's data only moves once and October's only once as well

However, if you're going to make significant changes to a partitioning scheme (like 4 partitions to 8) then instead of splitting 4 times – just create a new function and new scheme and then rebuild (possibly online) the object on the new scheme.

