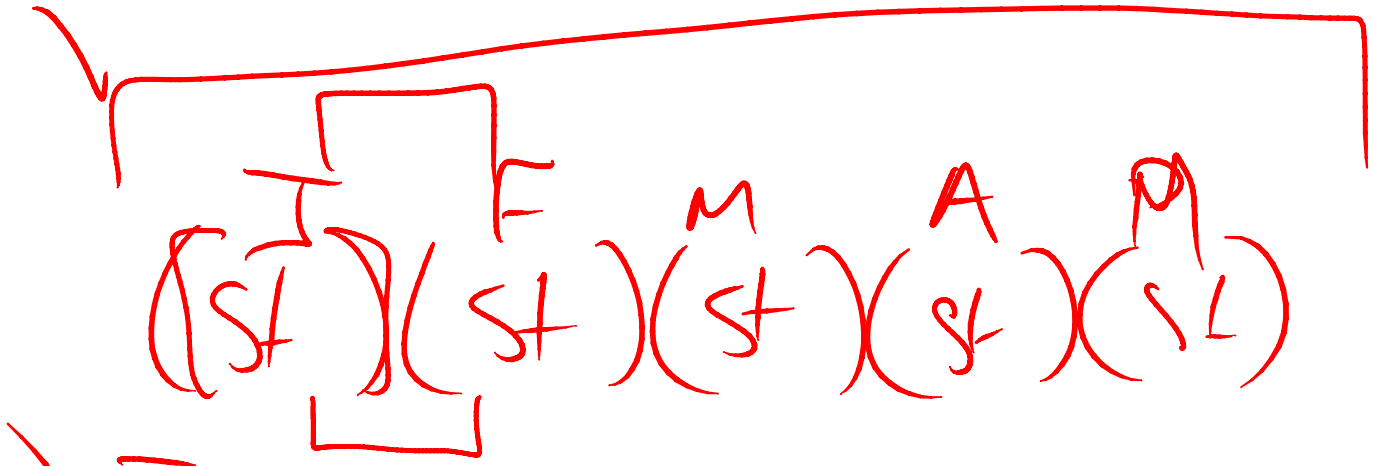


Filtering /partitioning and the concept of interval subsumption

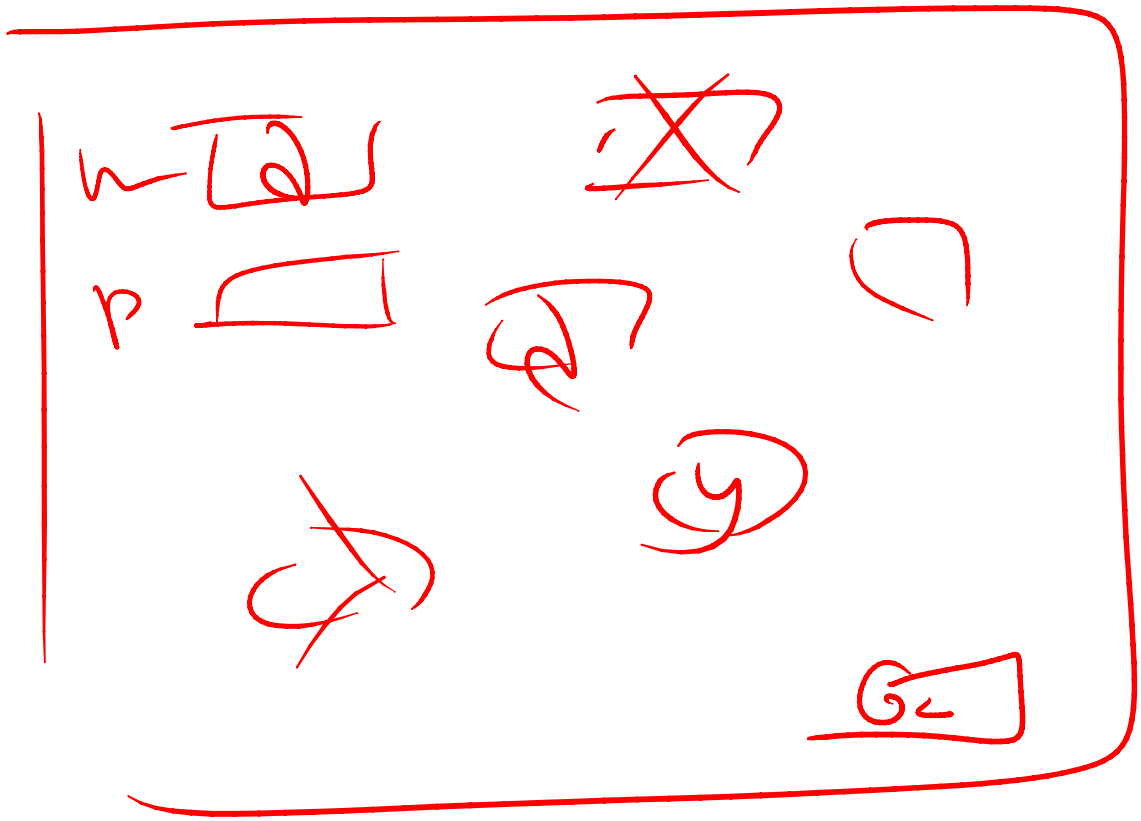
This was part of our end of day (Tue) discussion around statistics. If filtered statistics are creating for each partition then do you have the same as partition-level statistics? No. The reason is that SQL Server cannot combine these statistics to analyze sets. Each filter predicate must be a subset of exactly ONE filtered statistic (or index) in order to be used. So, if you were to have a filtered index or statistic over each month (Jan, Feb, Mar, etc.) then the following predicates **could** use the appropriate filtered statistic:

```
WHERE date = '20120112'
WHERE date between '20120112' and '20120115'
```

But, the following predicate SPANs two filters and therefore the filtered index/statistic cannot be used:

```
WHERE date between '20120120' and '20120205'
```

The same sets – if they were separate tables unioned together in a view – would require SQL Server to use BOTH statistics. This can often provide better (and more accurate) plans but at the cost of being more complicated to optimize. So, you don't want LOTS of little tables but you don't want ONE huge table either.



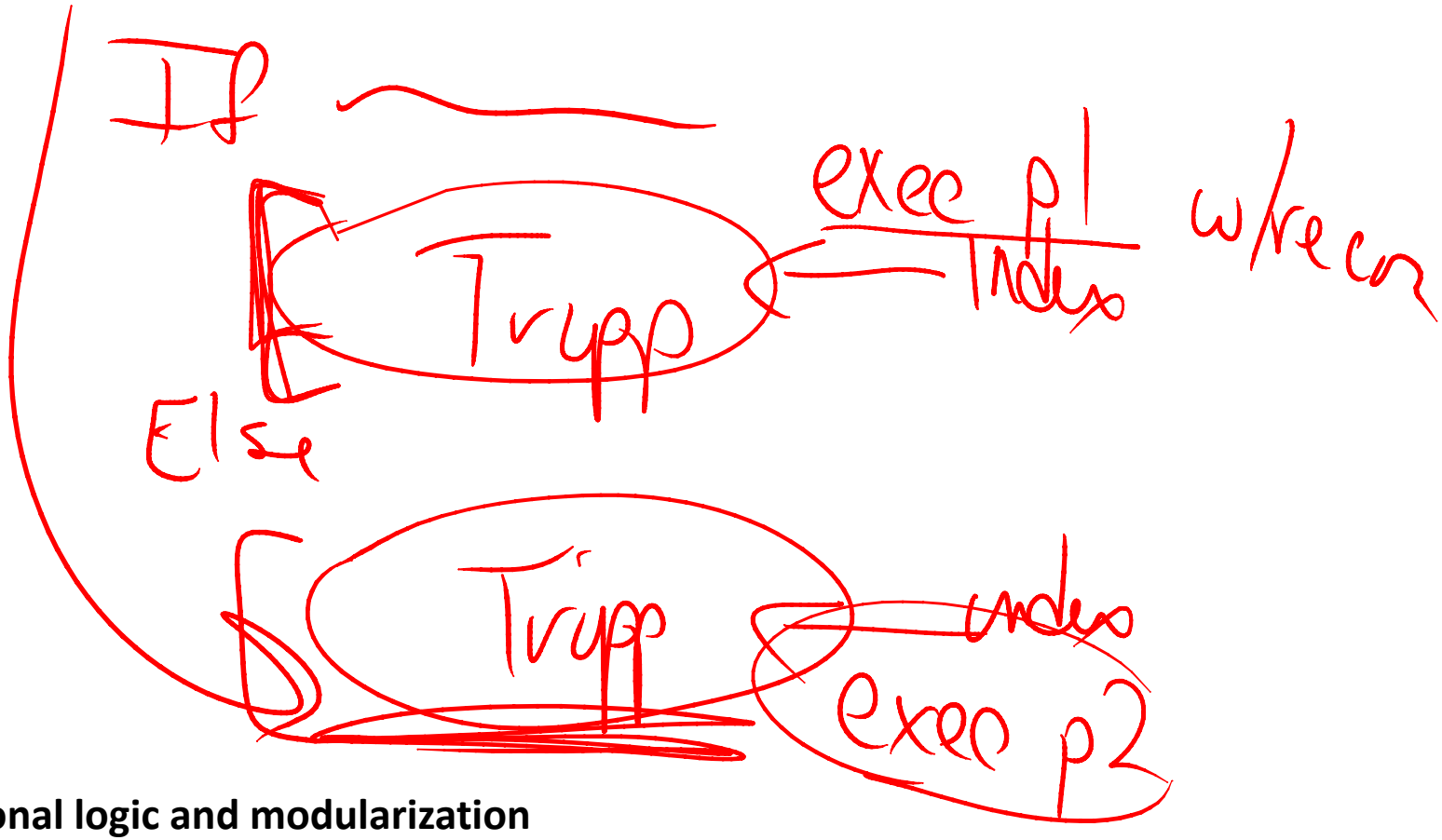
Multipurpose screens/dialog boxes that lead to multipurpose procedures

They looked/sounded good at the time?

But, they're often VERY hard to optimize with where clauses that look like:

```
WHERE (Column1 = @Value1 OR @Value1 IS NULL)
        AND (Column2 = @Value2 OR @Value2 IS NULL)
        AND (Column3 = @Value3 OR @Value3 IS NULL)
        …
```

Check out the demo scripts in order to fully see how to better optimize this – using sp_executesql (for combinations that are stable) and EXEC (@String) for combinations that need to be recompiled. Or, use sp_executesql with strings that include OPTION (RECOMPILE) for the parameters that might not generate consistent plans.



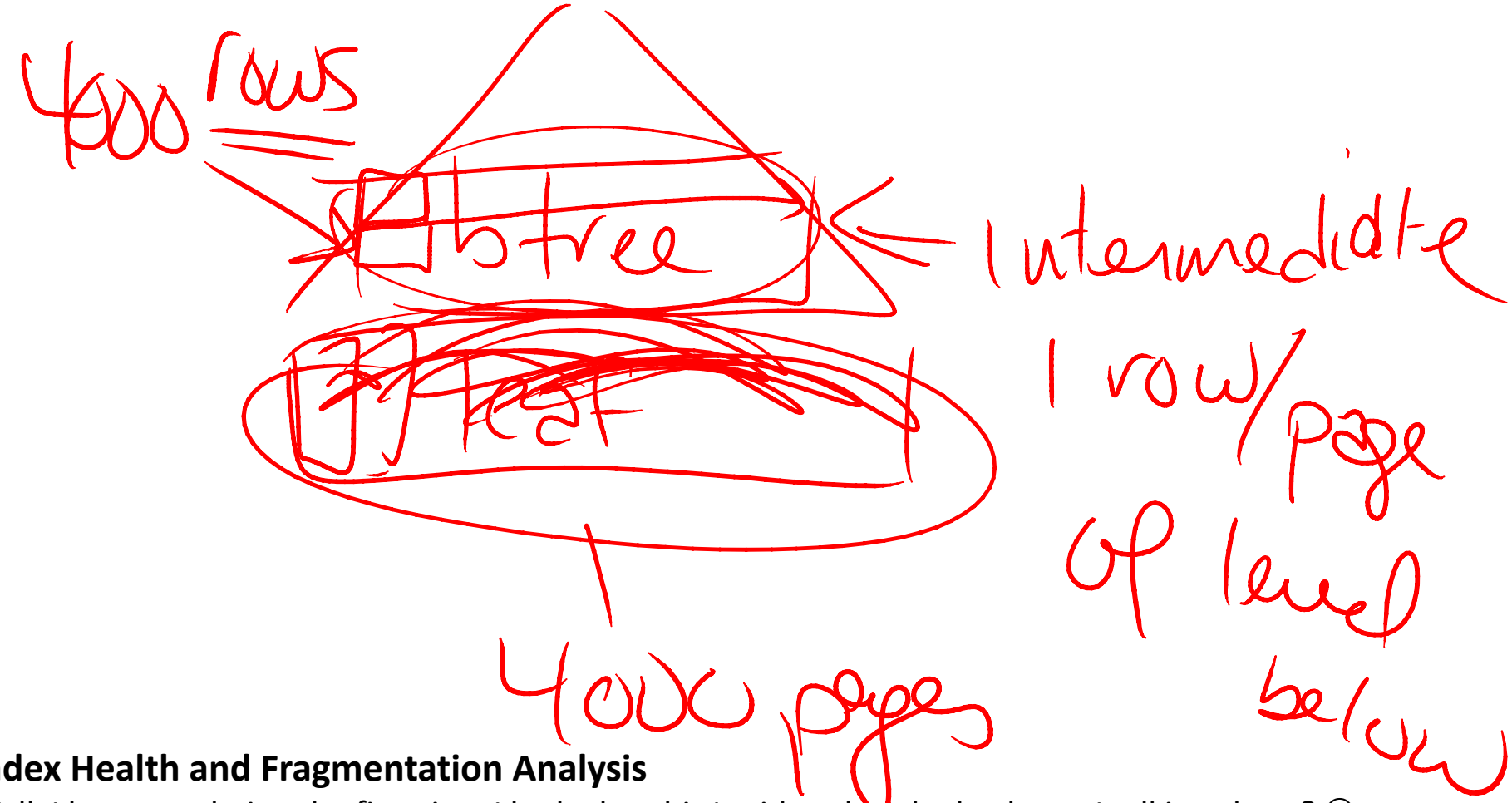
Conditional logic and modularization

Creating branching logic for different types of parameters seems logical but because SQL Server optimizes the process of optimization (where they try to optimize anything that's optimizable)

But, they're often VERY hard to optimize with where clauses that look like:

```
WHERE (Column1 = @Value1 OR @Value1 IS NULL)
        AND (Column2 = @Value2 OR @Value2 IS NULL)
        AND (Column3 = @Value3 OR @Value3 IS NULL)
        …
```

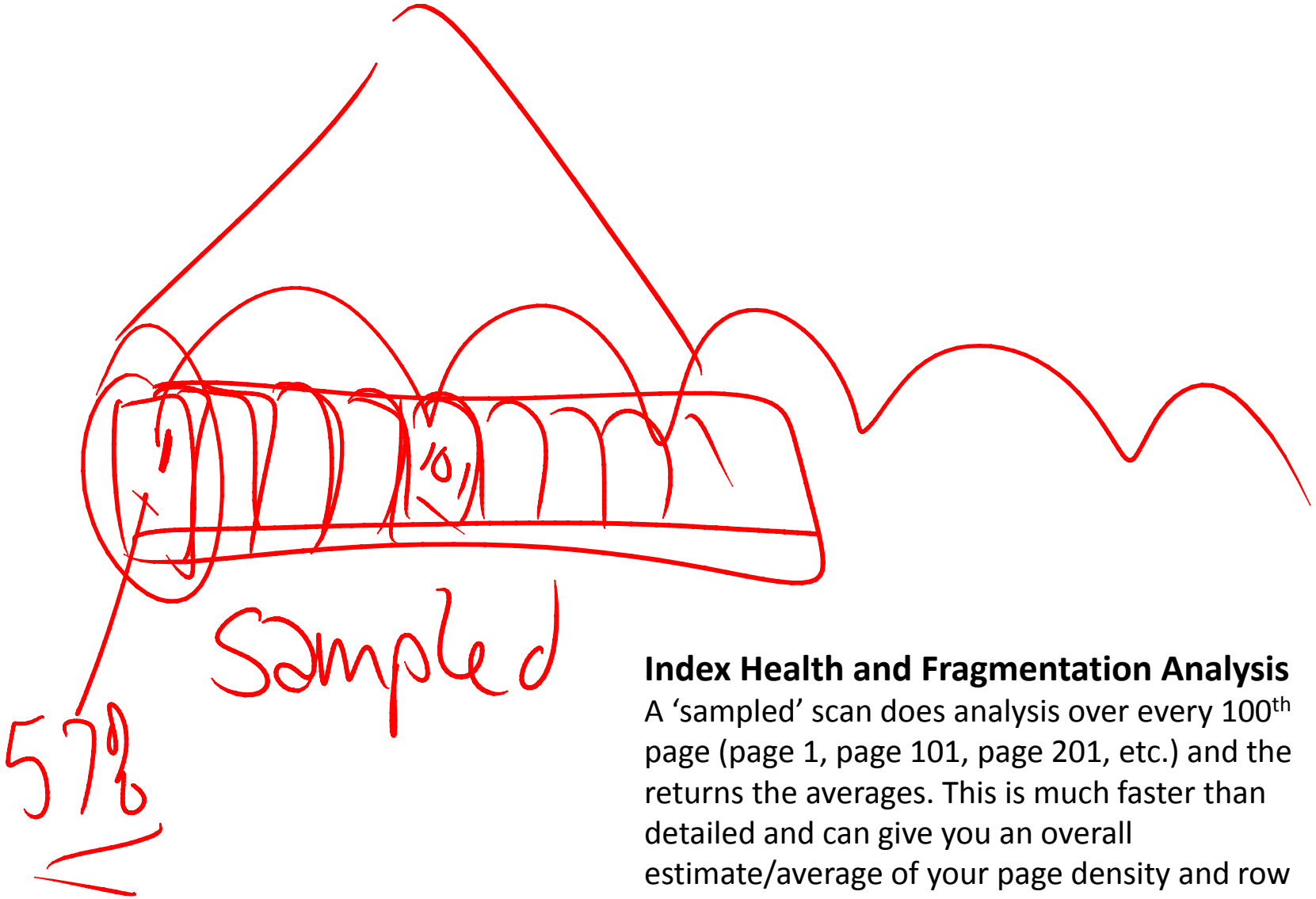
Check out the demo scripts in order to fully see how to better optimize this – using sp_executesql (for combinations that are stable) and EXEC (@String) for combinations that need to be recompiled. Or, use sp_executesql with strings that include OPTION (RECOMPILE) for the parameters that might not generate



Index Health and Fragmentation Analysis

Well, I have to admit… the first time I looked at this I said… what the heck was I talking about? ☺

Then, I remembered we had a bit of a tangent on phase 2 of Index Analysis (Index Health). The main point was that the avg_fragmentation_in_percent is ALWAYS analyzed by doing a scan of the first intermediate level (this is always the first level ABOVE the leaf) in the index. And, this is always true – regardless of the "mode" in which you query sys.dm_db_index_physical_stats. So, if you're ONLY using avg_fragmentation_in_percent to determine what to do – then perform a "limited" scan. Do not bother with SAMPLED or DETAILED as those provide better information for avg_page_space_used_in_percent which is for page density (and things like min, max and average row size). These are all important but (typically) not used for fragmentation analysis. This is comparable to DBCC SHOWCONTIG… WITH FAST.



Index Health and Fragmentation Analysis

A ‘sampled’ scan does analysis over every 100th page (page 1, page 101, page 201, etc.) and the returns the averages. This is much faster than detailed and can give you an overall estimate/average of your page density and row sizes.