Whiteboard
annotations from
Friday, April 12

# SQLintersection

## PostCon 04

# Solving Query Performance Problems

Kimberly L. Tripp, SQLskills.com
Kimberly@SQLskills.com

Joe Sack, SQLskills.com
Joe@SQLskills.com

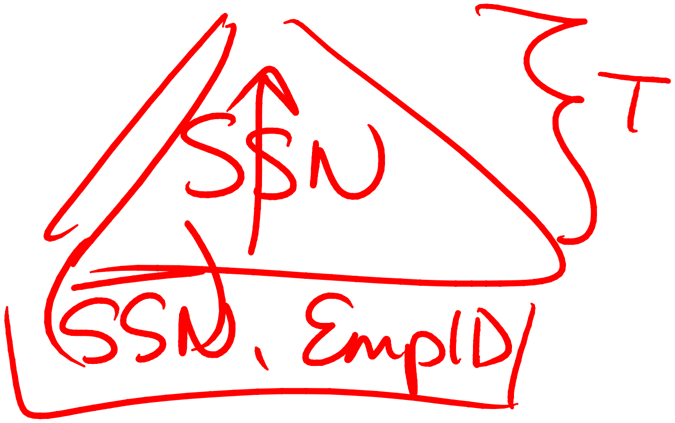
immerse yourself in sql server

Key = Nav

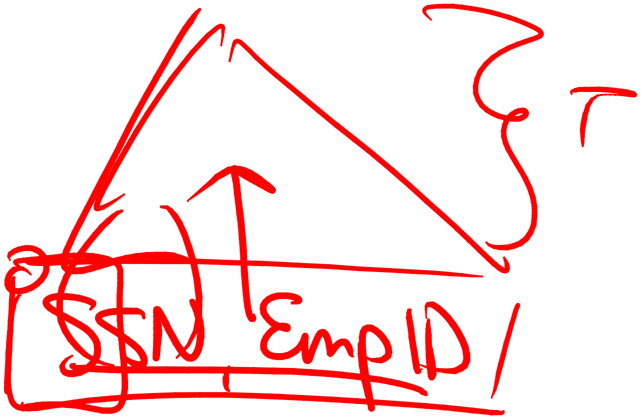
include = covering

CL empID (int)

UNC SSN



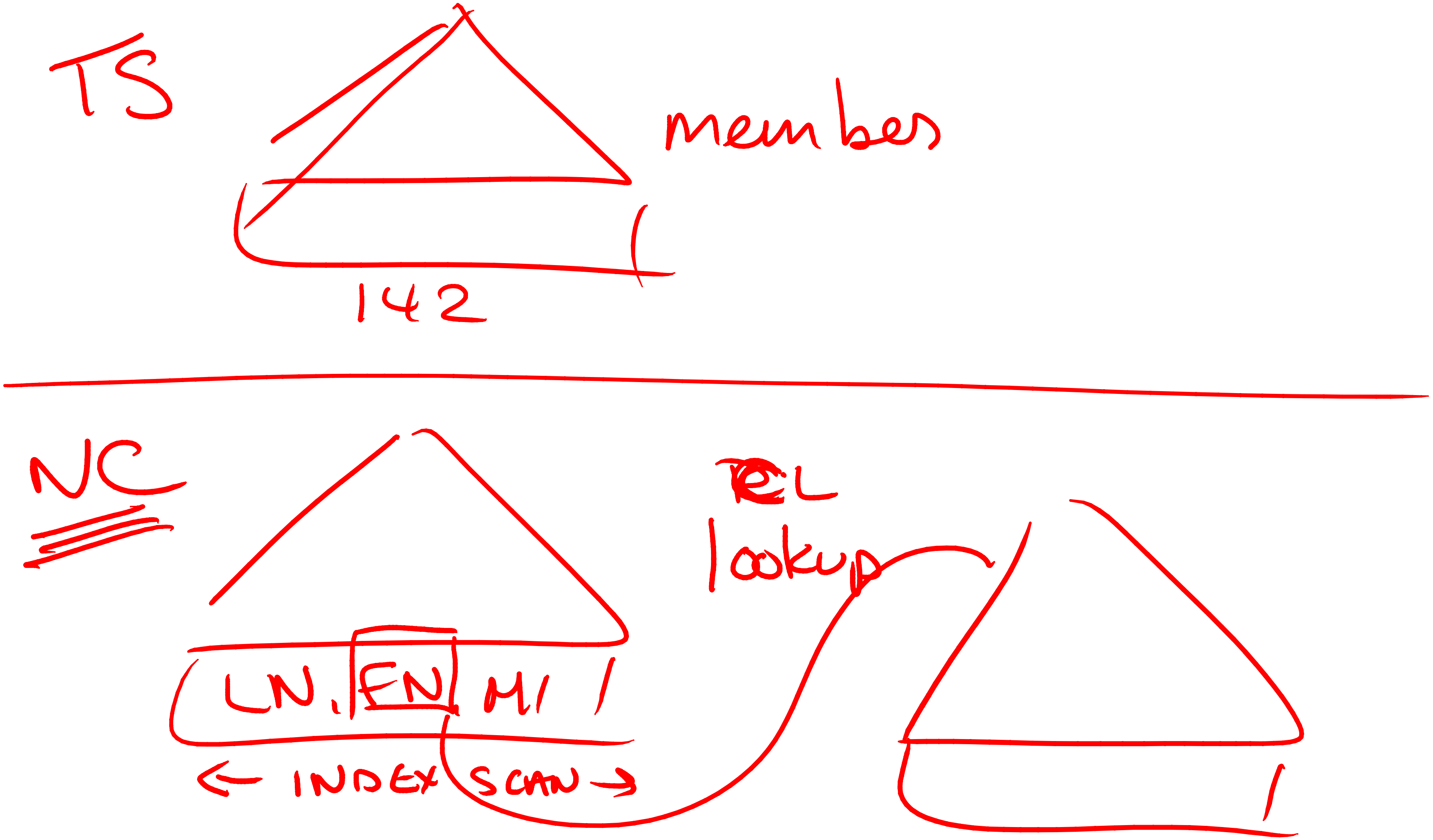
NC SSN



**Where does the CL key go?**

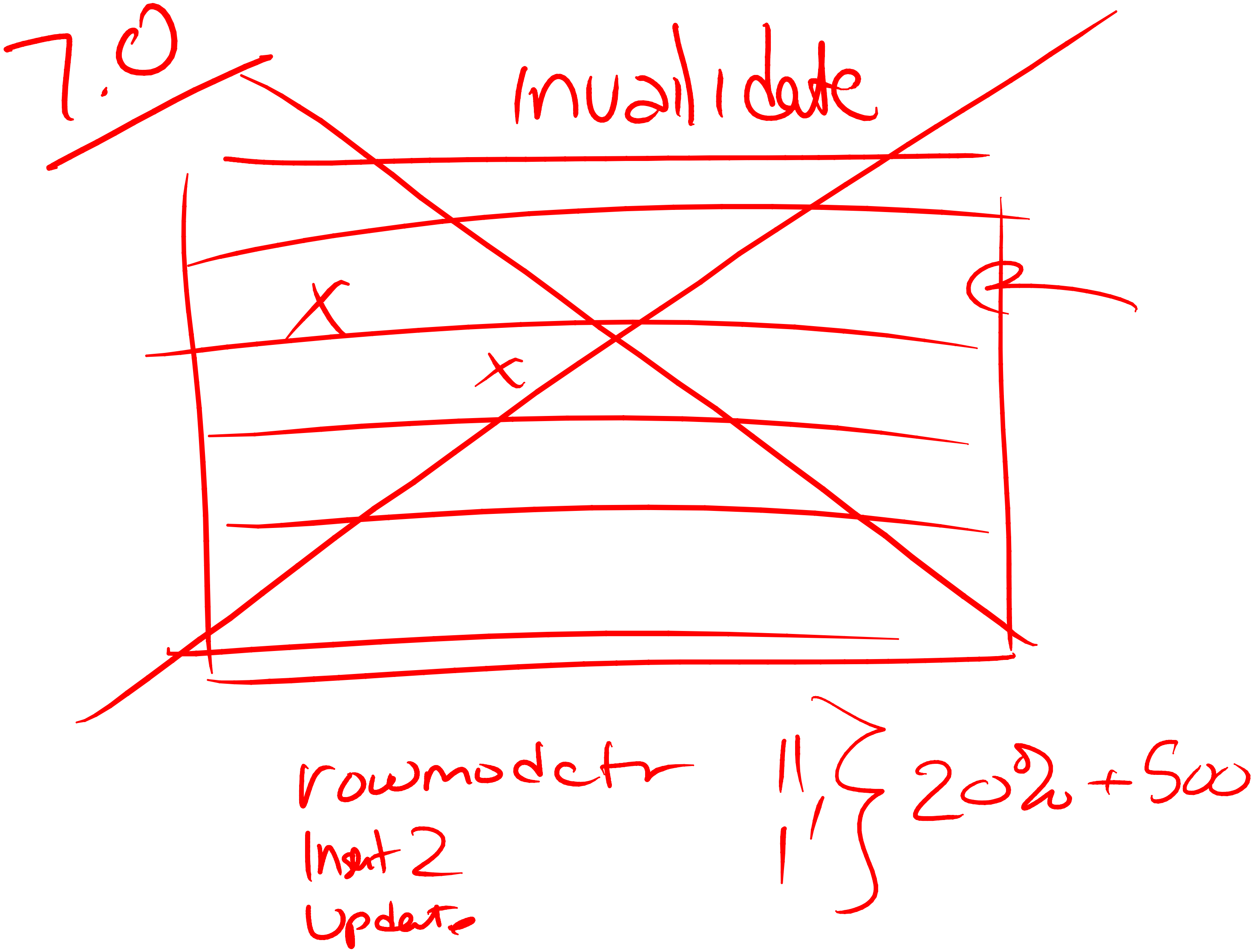
If the index is unique – then the CL is added only to the leaf-level.

If the nonclustered is non-unique – then the CL is added to leaf level as well as up the tree. Specifically, it’s added immediately after the key; included columns are added after that.



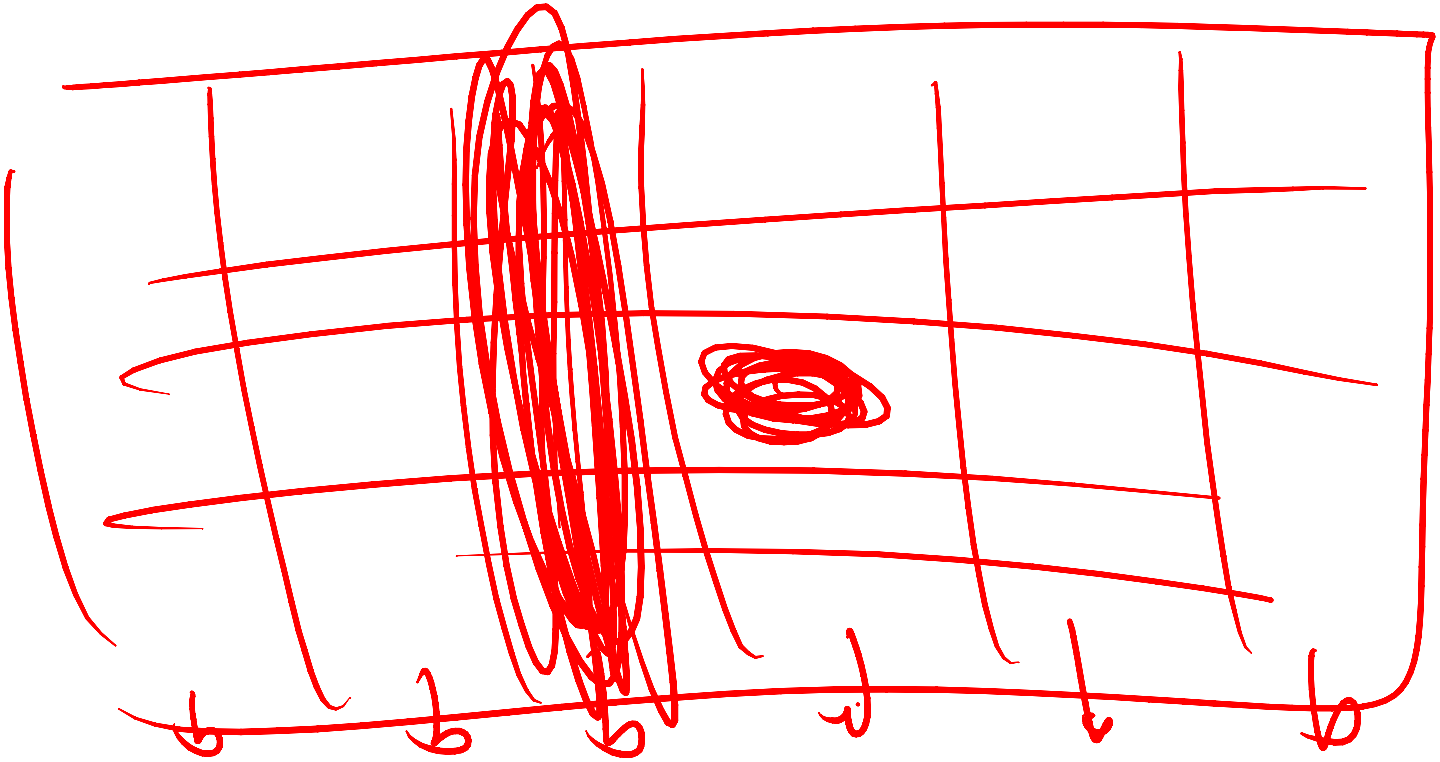
**Column-level distribution from left-based density subsets…**

This relates to the demo – even though the density vector shows the density of the left-based combinations all we know is that last names are horribly NOT unique and the combination of last name & first name IS unique. But, 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.



This is a 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 the backward compatibility view sysindexes).

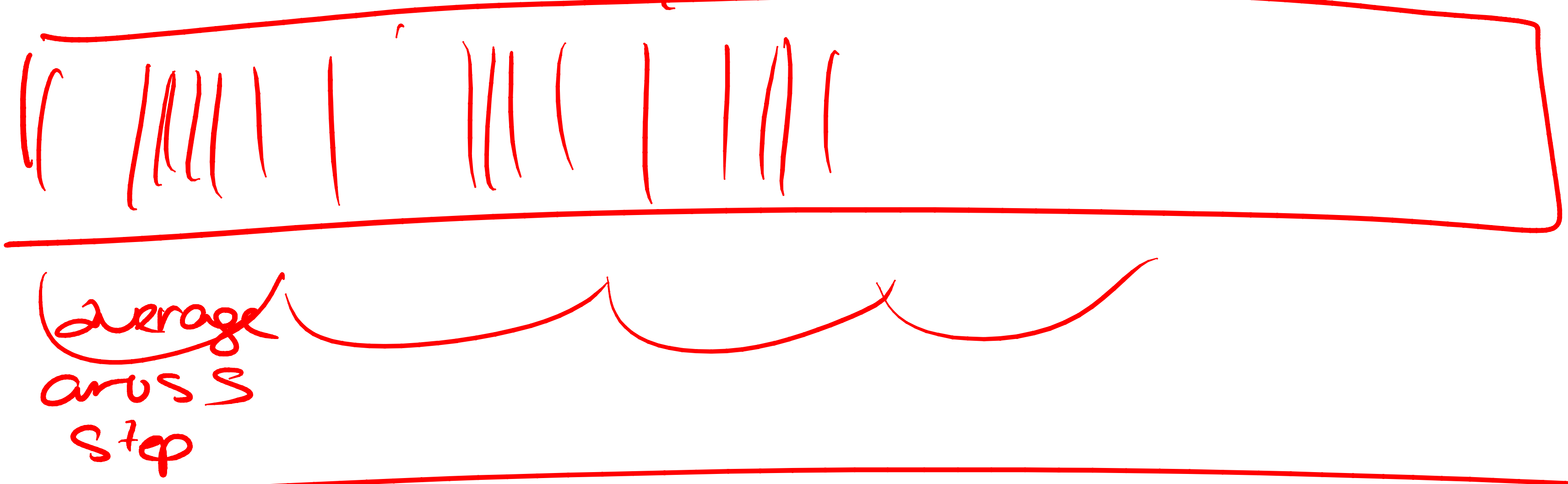
# PROBLEM rowmodctr

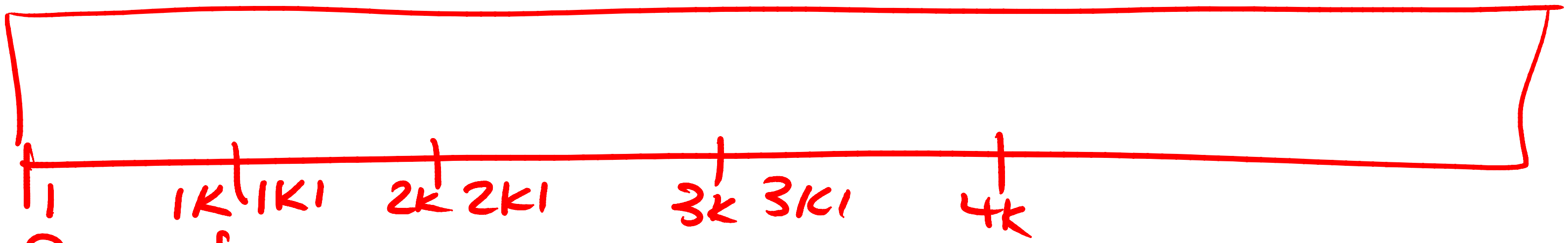


In 2005+ they moved to an internally defined colmodctr:

- 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.

Sales by customer





# of cust = 20K

DIVIDE in 10 or 20 buckets

**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!**

# FI or FS issues

SPs won't use unless statement recompiles
OPTION (RECOMPILE)

Session settings must be consistent
table CR, index CR
query execution
D/I/U will FAIL unless session settings match

The next couple of slides have a bit more detail on what we discussed here!

# Session Settings

- See BOL topic: Set Options that Affect Results
- Session settings control behavior – and the result of some computations
- Data in these persisted structures must be consistent
- Session settings that must be on:
  - ☐ ANSI_NULLS
  - ☐ ANSI_WARNING
  - ☐ QUOTED_IDENTIFIER
  - ☐ CONCAT_NULL_YIELDS_NULL
  - ☐ ANSI_PADDING
  - ☐ ARITHABORT
- Session setting that must be off:
  - ☐ NUMERIC_ROUNDABORT

Msg 1934, Level 16,
State 1, Line 1

CREATE INDEX failed because the following SET options have incorrect settings: 'QUOTED_IDENTIFIER'. Verify that SET options are correct for use with indexed views and/or indexes on computed columns and/or filtered indexes and/or query notifications and/or XML data type methods and/or spatial index operations.

# Client Consistency

- **Consistency with table(s), view and the clustered index (on the view) creation**

  *All tables on which the view is based, the view itself and the index, must be **created** with the correct session settings set or the index cannot be created on the view*

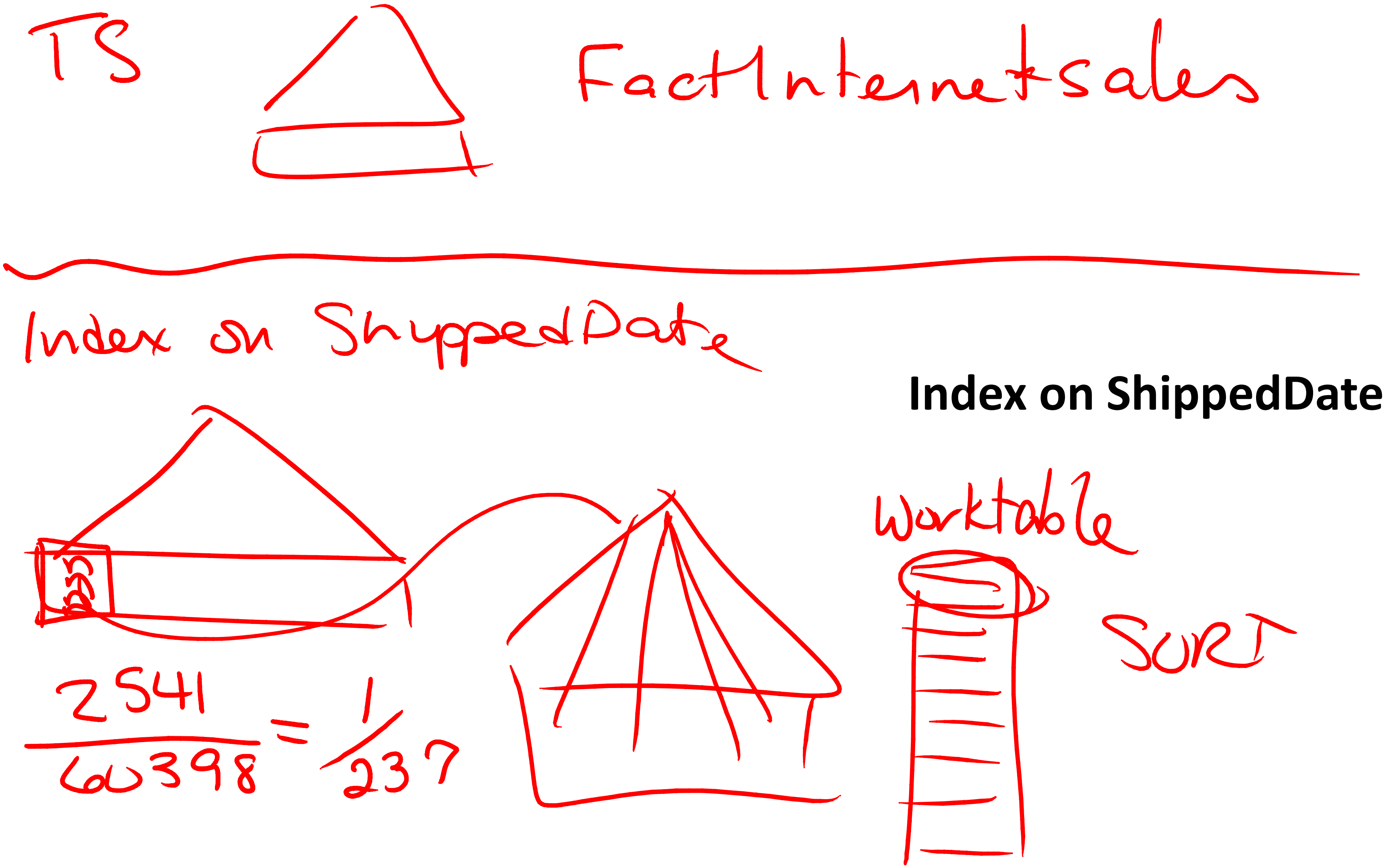
- **Consistency with base table access**

  *All INSERT, UPDATE and DELETE statements must be **executed** with correct session settings or the insert, update or delete will fail*

- **Consistency with view access**

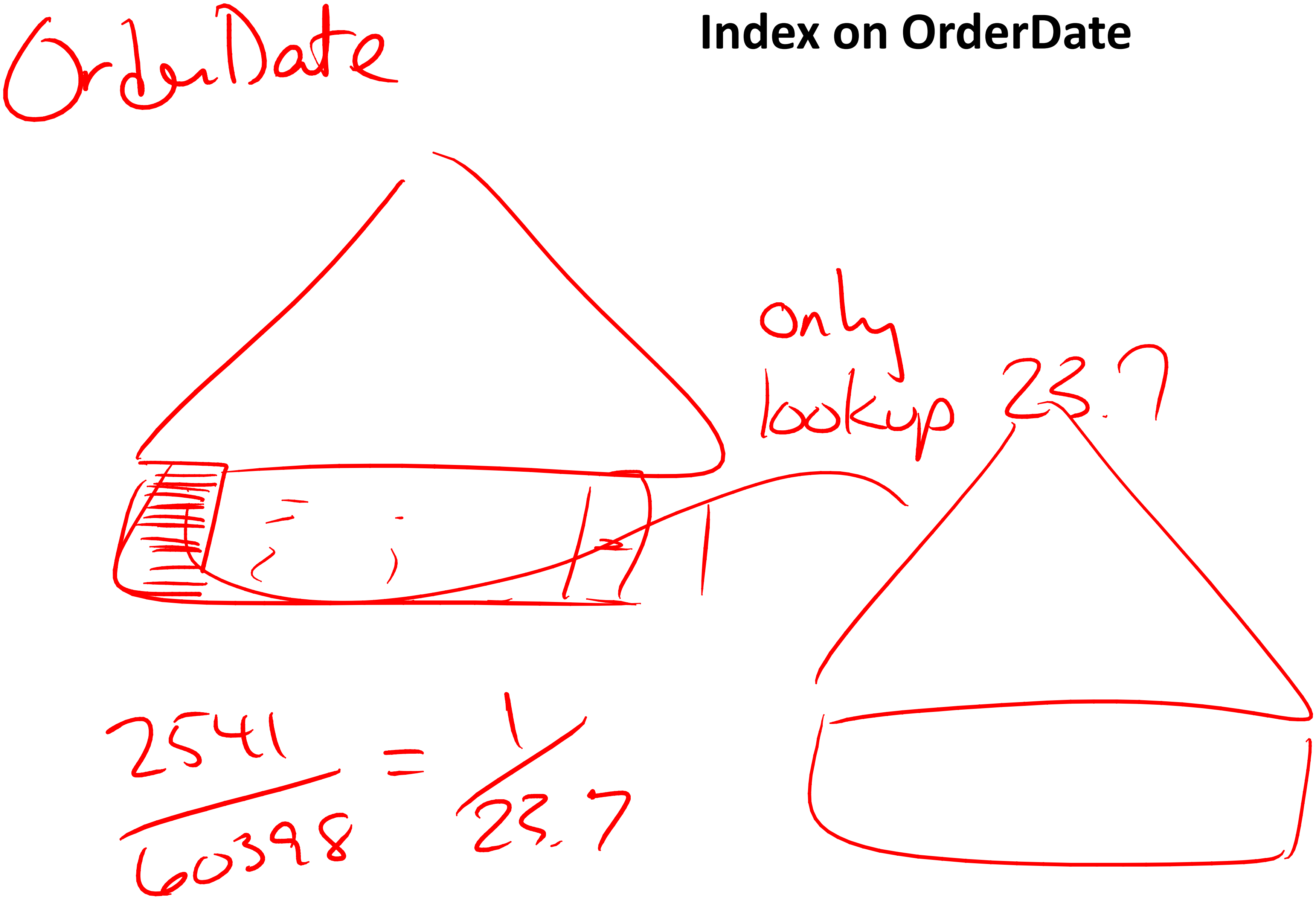
  *All queries that **SELECT** against views with indexes must access them with the correct session settings set otherwise the view's data will need to be recalculated, rejoined or recomputed*

# Index on ShippedDate



This is the diagram to show the options using a table scan vs. an index on ShippedDate vs. an index on OrderDate. The big issue is the cost of the worktable and the sort (at least from what SQL Server thinks).

# Index on OrderDate

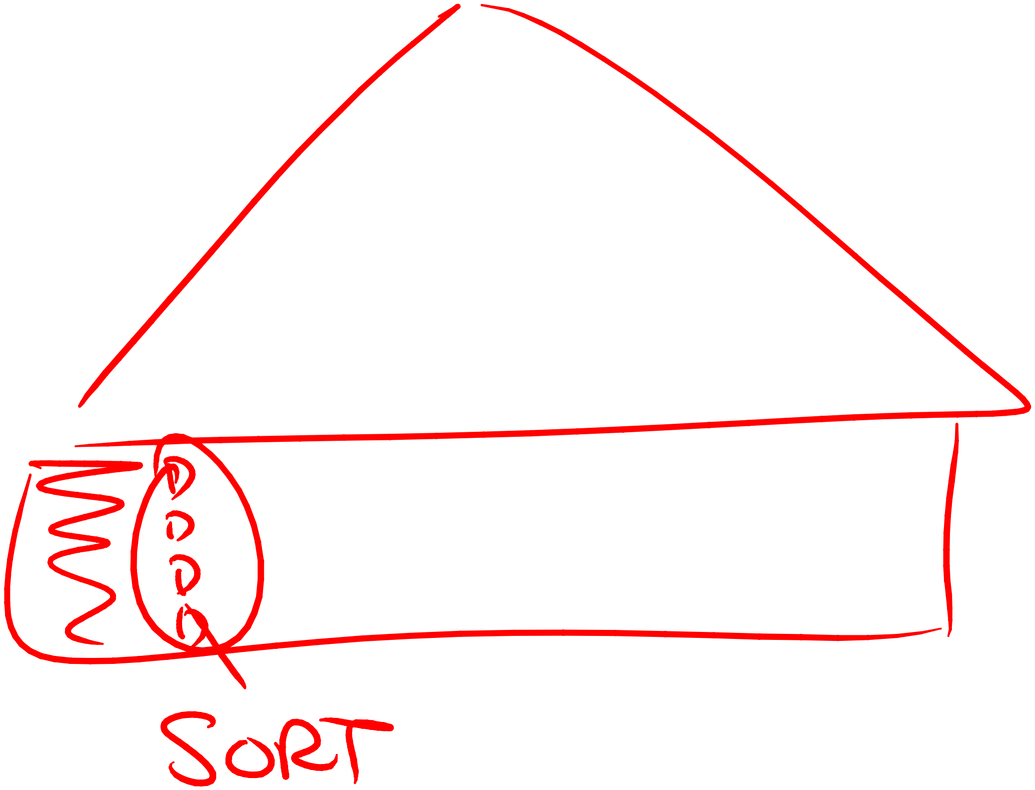


$$\frac{2541}{60398} = \frac{1}{23.7}$$

Here, SQL Server thinks that they'll encounter a null value for Shipped Date within 23.7 (2541/60398) but they're HORRIBLY wrong.

ShipDate Key

Inc OrderDate



What about the right KEY? Check out the demo script to play around with this and see what the green hint recommends vs. creating the index on the combination of ShipDateKey, OrderDateKey.