

# SQLskills Online Immersion Event

IEVLT: Very Large Tables  
Optimizing Performance and Availability  
through Partitioning




Kimberly L. Tripp  
President / Founder, SQLskills.com  
Kimberly@SQLskills.com  
@KimberlyLTripp





## Kimberly L. Tripp

Founder / President, SQLskills

-  [Kimberly@SQLskills.com](mailto:Kimberly@SQLskills.com)
-  [@KimberlyLTripp](https://twitter.com/KimberlyLTripp)
-  [www.sqlskills.com/blogs/Kimberly](http://www.sqlskills.com/blogs/Kimberly)



PLURALSIGHT



## Consultant / Trainer / Speaker / Writer

- Author / instructor for SQL Server Immersion Events: IEPTO1, IEPTO2, and IEHADR
- Author / presenter for Pluralsight
- Instructor and exam author for SQL Server and Sharepoint MCMs
- Author / manager of SQL Server 2005 and 2008 Launch Content
- Author / speaker at Microsoft TechEd, SQLPASS, ITForum, TechDays, ...
- Author of SQL Server Whitepapers on MSDN/TechNet: Partitioning, Snapshot Isolation, Manageability, SQLCLR for DBAs


- Author / presenter for MSDN and TechNet (25+)
- Instructor / SME for Microsoft University

## Data Platform MVP



- 17 years: 2002-2019

## When I'm not working with SQL

- Traveler/adventurer and avid diver / photographer: [www.BlueWaterImages.com](http://www.BlueWaterImages.com)
- @KimberlyLTripp on Insta 



# Daily Course Format: Mon, Tue, Wed, and Thu

- **Class runs for four half days**

- Monday, May 17 through Thursday, May 20, 2021

- **Each day**

- Lecture: 90 minutes from 9am until 10:30am PT
  - Please use “CHAT” for questions *during* the lecture
- Open Q&A: 30 minutes from 10:30am until 11:00am PT
  - Will address unanswered questions from chat
- Mandatory break: 30 minutes from 11:00am until 11:30am PT
  - Everyone needs a bit of a break!
- Lecture: 90 minutes from 11:30am until 1:00pm PT
  - Please use “CHAT” for questions *during* the lecture
- Open Q&A: up to 60 minutes from 1:00pm until 2:00pm PT
  - Will address unanswered questions from chat
  - Can open up for “open mic” questions

## Time Conversions

9:00am PT

= Noon ET

= 4:00pm UTC

Use this to see the exact time  
in YOUR time zone:

Event Time

Announcer -

SQLskills IEVLT

(timeanddate.com)

# Class Format: No LABS???

- **Class time – lecture / demo / questions! And, please ASK questions!**
  - There are no stupid questions... everyone started with ZERO knowledge about SQL Server!
  - If you're thinking it – you're probably not alone... speak up!
- **Why not lab time?**
  - Historically folks defer on labs to respond to email, etc.  
⇒ losing valuable class time
  - We can spend more time on content  
⇒ covering both MORE content and in more depth!
  - You receive all of the demo code and samples; you can reproduce everything we show you and pick where to really focus your attention!
    - You can take time to really study the content that's most appropriate to you
    - You can skip content that's not appropriate
    - You can take the RIGHT time for each/every exercise (rather than some being done quickly and others needing more time... potentially getting frustrated without enough time)

# Course Resources, Pluralsight Access, and Alumni

- Send Paul ([paul@sqlskills.com](mailto:paul@sqlskills.com)) an email for a FREE 30-day access to ALL SQLskills online training classes on Pluralsight
  - No strings attached, no credit-card required
  - If you don't receive it, send mail to Paul
- Course resources will be posted on the course membership page for class upon course completion
- We may want to update a slide or two, add some notes/resources based on questions and discussions, and process the recordings
- Email us: [Training@SQLskills.com](mailto:Training@SQLskills.com) if you have any problems finding or accessing any of your course materials
- And, now that you've attended one of our courses; you are an alumnus in our system with discounts for additional training courses!



# Course Resources

- **Course content (this slide deck)**
  - One “two-slides per side” PDF for optimal printing (save some trees!)
  - One “full slide” PDF for optimal viewing and/or online note-taking
- **Demos**
  - All instructor-led demo scripts
  - All reference scripts
- **Reference links and additional resources**
- **Online recordings for these sessions**
- **Upon completion don’t forget to request: “BadgeMe”**



**Please do not redistribute: These resources are for attendees only.**  
**Please respect our IP and time in creating these resources**  
**and do not share/forward.**



# Recommendation:

## Leverage Great Products!

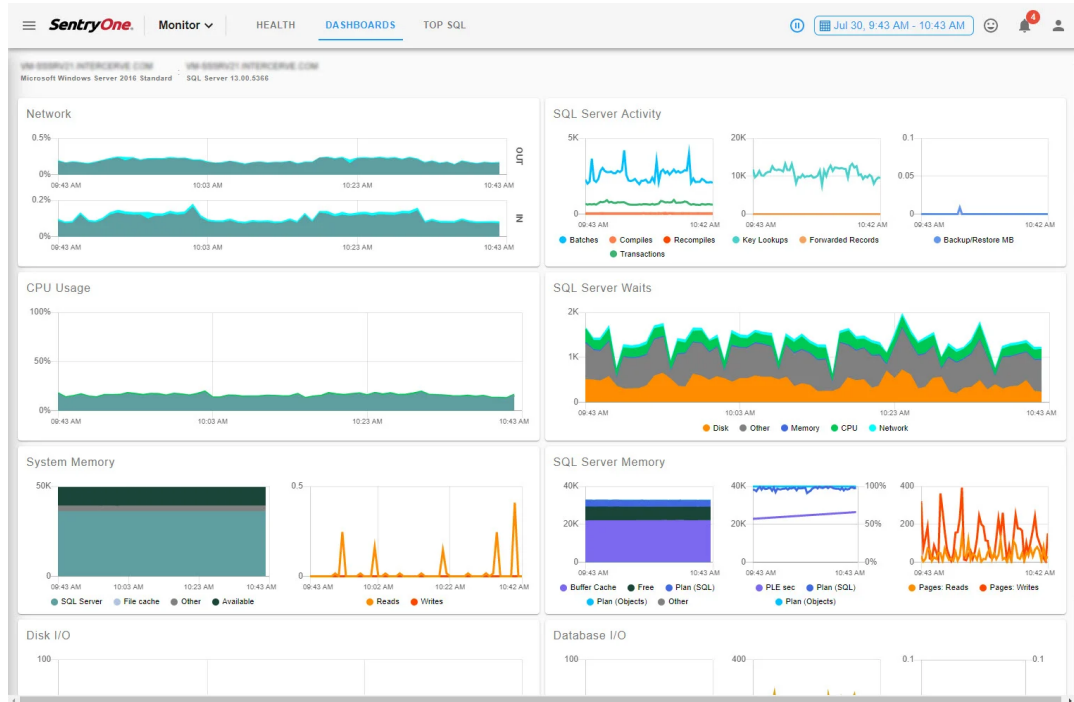


- Even with as much training as we will provide, you can't know everything
- SQL Server is powerful, complex, and has lots of different potential issues let third party software help!
- SQLskills partners with SentryOne because we really feel they've made the best suite of SQL Server products available
- **LEARN MORE:**
  - SQL Server Monitoring: <https://www.sentryone.com/sql-server/sql-server-monitoring>
  - Plus, get your questions answered!
    - Email Andy / Devon your specific monitoring questions
    - MailTo: [ayun@sentryone.com](mailto:ayun@sentryone.com) and MailTo: [dwilson@sentryone.com](mailto:dwilson@sentryone.com)
- **Plan Explorer**
  - <https://www.sentryone.com/plan-explorer>
  - [Advanced Plan Explorer Usage for Tuning Execution Plans](#)
- **Special SQLskills-Only Presentation Coming in May!**
  - May 24, 2021 from 10am to noon PT (1-3ET)



[SQL Server Monitoring](#)

# SentryOne™



SQLskills' strategic partners and class sponsor

<http://www.SentryOne.com>



# VLT = Very Large Table

- **Change your thinking**
  - Instead of table think “very large DATA SET” (it does NOT *have* to be in ONE table)
- **What does that actually mean?**
  - 100GB?
  - 10TB?
- **It’s really MUCH more hardware specific...**
  - If you only have 128GB of memory then a 600GB table will present problems...
- **When and why do you care?**
  - When you’re keeping more data around longer (and generating more and more data)
  - When you need to access this data (regularly for current/new data and *probably* irregularly for older data)
  - When different data has different requirements for performance and availability
  - When you see these “varying access patterns” or know they are expected...

# Module 1: Horizontal Partitioning Strategies

- **Workloads**
- **Time-based data patterns**
- **Horizontal partitioning: motivation?**
- **Horizontal partitioning**
  - Partitioned Views
  - Partitioned Tables
- **Functional partitioning**

# Module 2: Partitioned Views

- **Partitioned views**
  - Creation / implementation
  - Check constraints (filtering)
  - Trusted constraints
  - Partitioned view challenges
  - Stored procedure estimates and recompilation
- **Demo – partitioned views walkthrough**

# Module 3: Partitioned Tables

- **Range-based partitioned tables**
- **Partition function**
  - Left or right?
  - Partitioning: date column
- **Demo – partitioned tables walkthrough**
- **Partition scheme**
  - Aligned schemes
  - Storage-aligned schemes
- **Verifying partition data / location**

# Module 4: Tables, Indexes, Keys, and Relationships

- **Partitioning keys**
- **Primary and foreign keys**
- **Relationships**
- **Partitioned object indexing strategies**
  - Indexing strategies based on workload
  - Row-based indexes
  - Column-based indexes
  - Base table structure key points
  - Converting an existing table

# Module 5: Implementing the Sliding Window Scenario

- **Implementing the sliding window scenario**
- **The sliding window scenario with partitioned views**
- **The sliding window scenario with partitioned tables**
- **Switching data IN**
  - Staging area/loading in a heap
  - Why load into a staging area?
- **Switching data OUT**
  - For archiving
  - For removal
    - New in 2016
- **The sliding window scenario key points**
- **Concerns with Switch (IN and OUT)**

# Module 6: Key Partitioning Concerns and Considerations

- **Special considerations for partitioned tables**
- **Online operations, indexes, and partitioning**
- **Incremental statistics updates**
- **Partition-aligned index views**
- **Partitioned table with filtered indexes**
- **Interval subsumption**
  - Partitioning / fast-switching
- **Partition-level lock escalation**

# Module 7: Partitioning Techniques Combined

- Partitioning: partitioned views vs. partitioned tables
- Damaged “partitions” on Enterprise Edition
- Partition-level features and frustrations
- Other features and partitioning
- Partitioning for performance



# Module 1: Horizontal Partitioning Strategies



# Overview

- **Workloads**
- **Time-based data patterns**
- **Horizontal partitioning: motivation?**
- **Horizontal partitioning**
  - Partitioned Views
  - Partitioned Tables
- **Functional partitioning**

# Workloads

- **Hybrid – OLTP with real-time analysis**
  - Larger and larger tables...
  - New data coming in and performance problems with existing / older data
  - Maintenance takes longer and longer to maintain tables (larger and larger amount that's not changing)
  - Management nightmares (backup / restore / downtime)
  - Don't generally consider "partitioning" in OLTP environments (meant more for management of the sliding window scenario) but here it's ideal
- **Isolated – Primary OLTP server that's isolated from DSS and periodically data is migrated**
  - Limited to no analysis done on the OLTP server
  - Data is read-only (except for periodic loads) to DSS
  - This is the canonical case for "sliding window" but how you implement this is surprising...

# Time-based Data Patterns

- Structures designed for “sliding window” (or, “rolling range”) scenario
- Tables created and data flows on regular / consistent basis (weekly, monthly, yearly, etc.)
- Data may be archived / removed to keep only “current” timeframe (year, two years, etc.)
- Implementation benefits vary with implementation choice:
  - Partitioned views
  - Partitioned tables



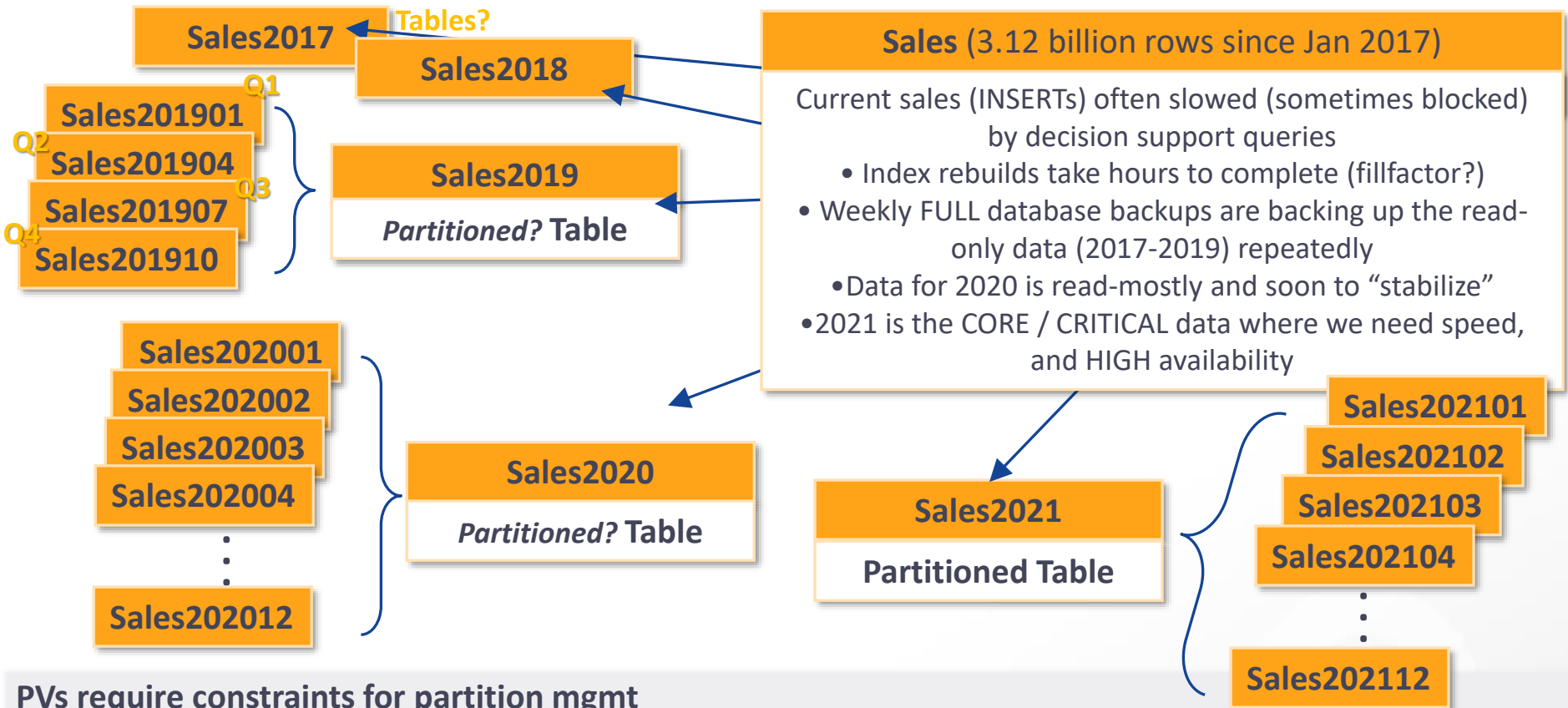
# Horizontal Partitioning: Motivation?

- **Resource contention limitations**
- **Varying access patterns**
- **Maintenance restrictions**
- **Availability requirements**
- **To remove resource blocking or minimize maintenance costs**
- **Important notes:**
  - Partitioning does not automatically mean Distributed Partitioned Views (DPVs)
  - Partitioning is more of a scale-up / manageability enhancement; DPVs are a form of scale-out partitioning

# Table's Shared Data/Indexes

- **A single large table...**
  - Presents different types of problems
    - Management
    - Index create / build or rebuilds
    - Backup / restore and recovery
    - Escalation
  - May have different access patterns
    - Some data new – OLTP (inserts / updates for new and current sales)
    - Some data old for historical lookups – small singleton for OTLP lookups or larger for analysis
- **Does NOT have to be one table!**

# Horizontal Partitioning (PVs or PTs)



PVs require constraints for partition mgmt  
`CHECK ([SalesDate] >= '20210101'  
AND [SalesDate] < '20210201')`

PTs require a partition function and a partition scheme for partition management

# Functional Partitioning

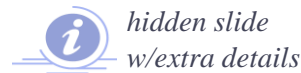
- **More granular control**
  - Varying access patterns
  - Varying index defrag patterns
  - Very fast sliding window control
- **More backup/restore choices**
  - File / filegroup backups
  - File / filegroup restores from anything larger
    - Full backups support PAGE, FILE, FILEGROUP or FULL restores
    - Filegroup backups support PAGE, FILE or FILEGROUP restores
    - File backups support PAGE or FILE restores
  - Even support for file / filegroup in simple recovery model (backups separated between RW and RO backups and RO backups can be performed at any frequency!)
- **Significantly better availability**



# Functionally Partitioning Data

- **Partitioned tables (requirement: Enterprise Edition prior to SQL Server 2016 SP1)**
  - But, for ALL Enterprise ADMIN features such as online operations – you still need EE
  - Lots of requirements around “switching in / out” sets of data
  - Problems around large tables
    - Manageability isn’t perfect (varies release to release)
    - Statistics aren’t ideal (more details coming up – some differences in 2014+)
- **Partitioned views (benefit: available in any edition)**
  - Creating separate tables and managing them individually...
    - Offers LOTS of options
    - Requires automation / planning
  - Definitely more work to architect, manage, design – payoff is often worth it!
- **Generally, partitioning is done through partitioned views alone or partitioned views with partitioned tables but rarely partitioned tables alone!**

# Functionally Partitioning Data



- **Partitioned tables (requirement: Enterprise Edition prior to SQL Server 2016 SP1)**
  - But, for ALL Enterprise ADMIN features such as online operations – you still need EE
  - Can convert an existing table as an ONLINE operation IF the table doesn't have any LOB columns in 2005 / 2008 / R2 (fixed in 2012)
    - Might run into problems around “unique” index requirements for PTs in that the partitioning column must be a member of the key – for all unique indexes
  - Cannot do fast switching in 2005 if Indexed Views
  - Cannot do fast switching if iFTS desired
- **Partitioned views (benefit: available in any edition)**
  - Might be able to replace an existing table with a view (even for DML) if you meet the correct criteria
    - Might not be able to replace all statements, can programmatically direct modifications (for INSERTs)
  - Conversion may require downtime or time where certain data is inaccessible
  - Definitely more work to architect, manage, design – payoff is often worth it!

# Review

- **Workloads**
- **Time-based data patterns**
- **Horizontal partitioning: motivation?**
- **Horizontal partitioning**
  - Partitioned Views
  - Partitioned Tables
- **Functional partitioning**

# Questions!



## Module 2: Partitioned Views



# Overview

- **Partitioned views**
  - Creation / implementation
  - Check constraints (filtering)
  - Trusted constraints
  - Partitioned view challenges
  - Stored procedure estimates and recompilation
- **Demo – partitioned views walkthrough**

# Partitioned Views

Individual tables are created on filegroups – manually.



⋮



Sales202101

| ID | SalesDate | ... |
|----|-----------|-----|
| 1  | 20210101  | ... |
| 2  | 20210101  | ... |
| 3  | 20210101  | ... |
|    |           |     |
| n  | 20210131  | ... |

Sales202102

| ID | SalesDate | ... |
|----|-----------|-----|
| 1  | 20210201  | ... |
| 2  | 20210201  | ... |
| 3  | 20210201  | ... |
|    |           |     |
| n  | 20210228  | ... |

⋮

Sales202112

| ID | SalesDate | ... |
|----|-----------|-----|
| 1  | 20211201  | ... |
| 2  | 20211201  | ... |
| 3  | 20211201  | ... |
|    |           |     |
| n  | 20211231  | ... |

The view is logically bringing the data together in a tabular data set.

| ID | SalesDate | ... |
|----|-----------|-----|
| 1  | 20210101  | ... |
| 2  | 20210101  | ... |
| 3  | 20210101  | ... |
|    |           |     |
| n  | 20211231  | ... |

```
SELECT * FROM Sales202101
UNION ALL
SELECT * FROM Sales202102
UNION ALL
SELECT * FROM Sales202103
UNION ALL
...
SELECT * FROM Sales202112
```

# Partitioned Views

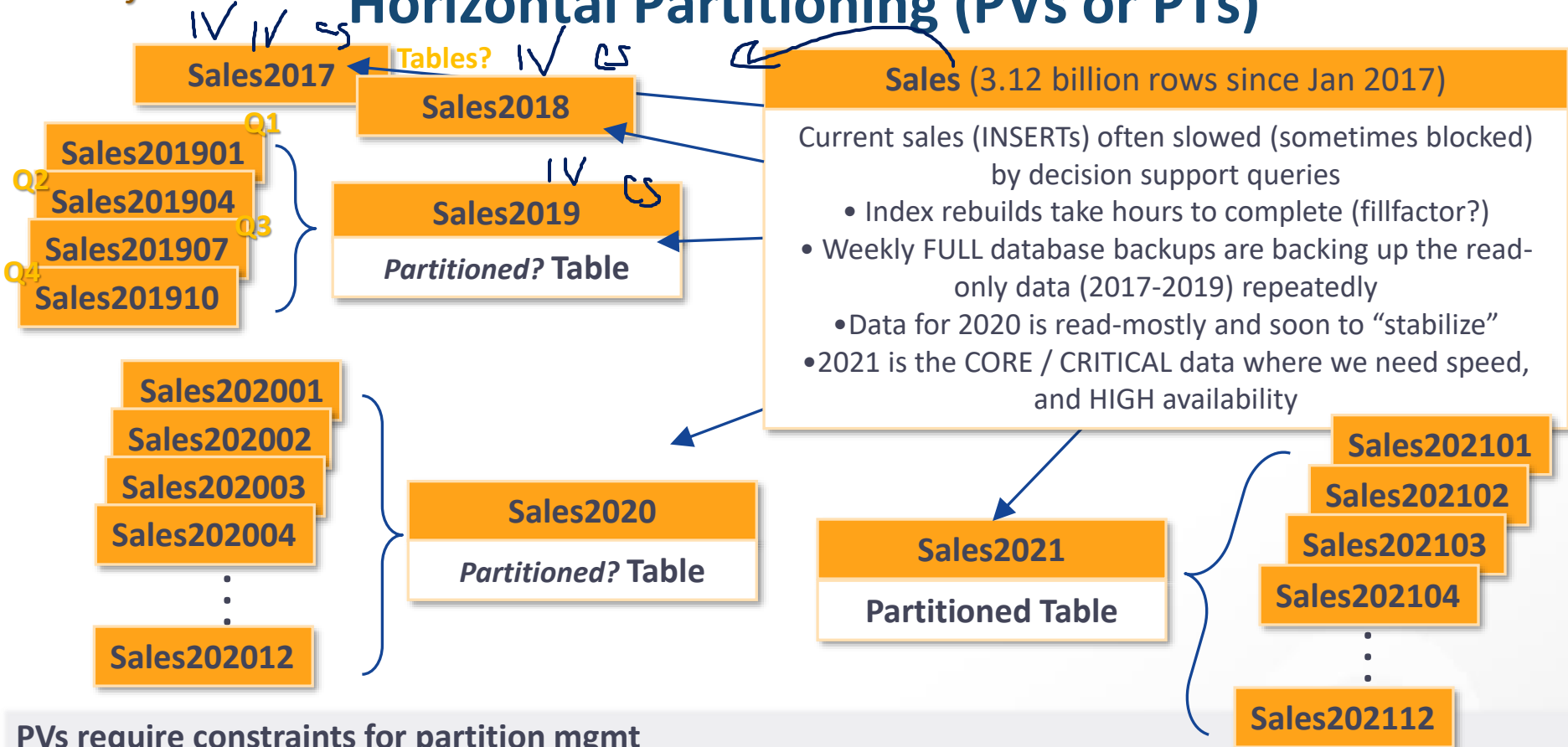
## Still Motivation for Creating in SQL Server 2005+

- **Queryable partitioned views added in SQL Server 7.0**
  - Query Optimizer removes irrelevant tables from query plan (partition elimination)
- **Updateable partitioned views added in SQL Server 2000**
  - Inserts / updates / deletes may need to be directed to correct base table (if you can't meet UPV requirements that partitioning key has to be the first column of the primary key)
  - Data modification statements can be directed to appropriate base tables IF the partition key is part of the primary key and it's enforced by a CHECK constraint
- **Requires separately named / created tables**
- **Manual placement on different filegroups**
- **Checked / verified constraints – and tested for logic problems (what will happen on February 29th?)**
- **UNION ALL views**



Added for review

# Horizontal Partitioning (PVs or PTs)



PVs require constraints for partition mgmt  
`CHECK ([SalesDate] >= '20210101'  
AND [SalesDate] < '20210201')`

PTs require a partition function and a partition scheme for partition management

# Creating Partitioned Views

- **Create files / filegroups**
- **Create individual tables on filegroups**
- **Create base table indexes (CL, NCs, views with indexes, and columnstore indexes) as per performance requirements on ALL partitions**
  - It's a pro and a con that each "partition" can have different indexes:
    - PRO: you can reduce indexes for OLTP and increase for DSS
    - CON: it's harder for the optimizer to optimize – each table has to be analyzed individually
- **Create base table constraints to restrict each table's partitioning column**
  - Range must be protected on ALL partitions using a constraint; the constraint MUST be checked and verified (= "trusted" constraints)
  - Can create restrictions on multiple columns and SQL Server can eliminate on any combination of them (you can only do this with partitioned views)
- **Create views (must use UNION ALL (not UNION))**

# Check Constraints (Filtering)

- Defines column's domain of legal values
- All data checked on INSERT / UPDATE

- Only range constraints supported

**CHECK (SalesDate >= '20210101'  
AND SalesDate < '20210201')**

- Limited subset of range operators supported for partitioning (i.e. filtering) are:  
BETWEEN, AND, OR, <, <=, >, >=, =
  - Others are supported for data integrity purposes only
- Added during CREATE TABLE
- Add later using ALTER TABLE

*But what about the existing data?*

Default is to CHECK the existing data...

# Verify Constraint is “Trusted”

- Check to see if constraint is trusted:

`OBJECTPROPERTY(CONSTRAINT_ID, 'CnstIsNotTrusted')`

- Check to see if any data violates constraints:

`DBCC CHECKCONSTRAINTS('charge')`

`DBCC CHECKCONSTRAINTS('constraint')`

- However, even if DBCC CHECKCONSTRAINTS returns NO errors or warnings about your data... your constraint is still NOT trusted!
- You must DROP / re-CREATE or re-CHECK (syntax is confusing)

`ALTER TABLE [tablename]`

`WITH CHECK`

`CHECK CONSTRAINT [constraint_name]`

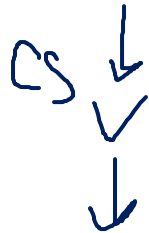
# DEMO: Partitioned Views Walkthrough



7V



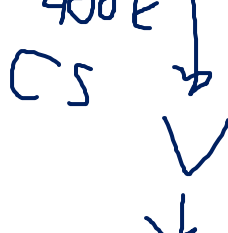
202k June



5k

IV

400k July



71k IV

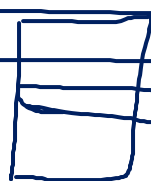
Aug



IV

Sept

Country  
Cat



PV

9008



# Partitioned View Challenges

- Administration of separate tables, index creation on many tables
- Constraints (and business logic) are not guaranteed with partitioned views; must be managed individually on all tables
- Constraints must be trusted; if you disable constraints make sure you not only “enable” but recheck the data
- Queries can work well but data modifications may not work through the view
  - You cannot insert through a PV if the base table has an identity column
    - TIP: Use application-directed inserts
  - Partitioning column must be leading column of PK (*and all PKs must have same definition*)
    - Some people really hate this but it does have benefits
      - ✓ It can help query performance to lead the clustered index with the partitioning column
      - ✓ The negative is that you often have to “push-down” a column in related tables that may not have been necessary
- Harder for the optimizer to optimize as tables could be different (some problems with optimizing certain predicates):
  - <http://blogs.msdn.com/craigfr/archive/2006/11/27/introduction-to-partitioned-tables.aspx> (<http://bit.ly/Qyu6Zp>)
  - <http://blogs.msdn.com/sqlcat/archive/2006/02/17/Partition-Elimination-in-SQL-Server-2005.aspx> (<http://bit.ly/Qyu4Rx>)

# Stored Procedure Estimates and Recompilation

- **Some features cannot support a single plan or perform well with one that's cached**
  - Filtered objects
    - SQL Server cannot save a single plan based on filtered indexes or statistics
    - SQL Server won't use a filtered object unless recompilation is performed:
      - Using `OPTION (RECOMPILE)`: this is the most safe as it forces the statement to be recompiled and is not affected by the database setting for parameterization
      - Using a dynamically constructed string (unless forced parameterization is on)
  - Partitioned objects (both PVs and PTs)
    - SQL Server CAN do partition elimination correctly (even if the first execution is only against one partition) so you will get the correct data
    - However, the first execution will set the “estimates” for the plan which could vary across the sets and lead to poor performance
  - CHOOSE
    - This is similar to partitioned views in that you will get the correct data but subsequent executions will have estimation issues



Slide from Pluralsight course:  
SQL Server Optimizing Stored  
Procedure Performance



# Review

- **Partitioned views**
  - Creation / implementation
  - Check constraints (filtering)
  - Trusted constraints
  - Partitioned view challenges
  - Stored procedure estimates and recompilation
- **Demo – partitioned views walkthrough**

# Questions!



## Module 3: Partitioned Tables



# Overview

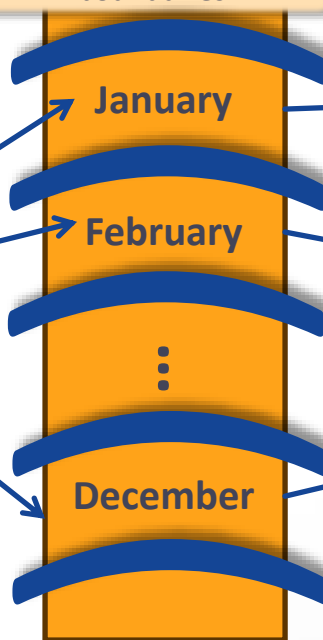
- **Range-based partitioned tables**
- **Partition function**
  - Left or right?
  - Partitioning: date column
- **Demo – partitioned tables walkthrough**
- **Partition scheme**
  - Aligned schemes
  - Storage-aligned schemes
- **Verifying partition data / location**

# Partitioned Tables

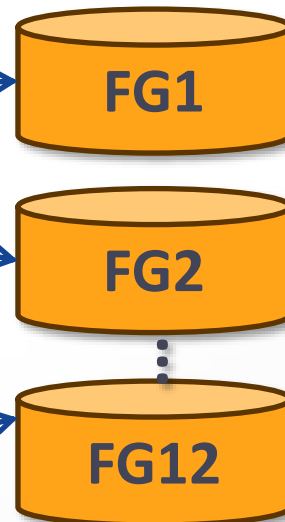
The table is created on a scheme, which references a function

| ID | SalesDate | ... |
|----|-----------|-----|
| 1  | 20210101  | ... |
| 2  | 20210101  | ... |
| 3  | 20210101  | ... |
|    | ⋮         |     |
| n  | 20211231  | ... |

Partition function  
defines the logical  
boundaries



Partition scheme  
defines the physical location  
for the data

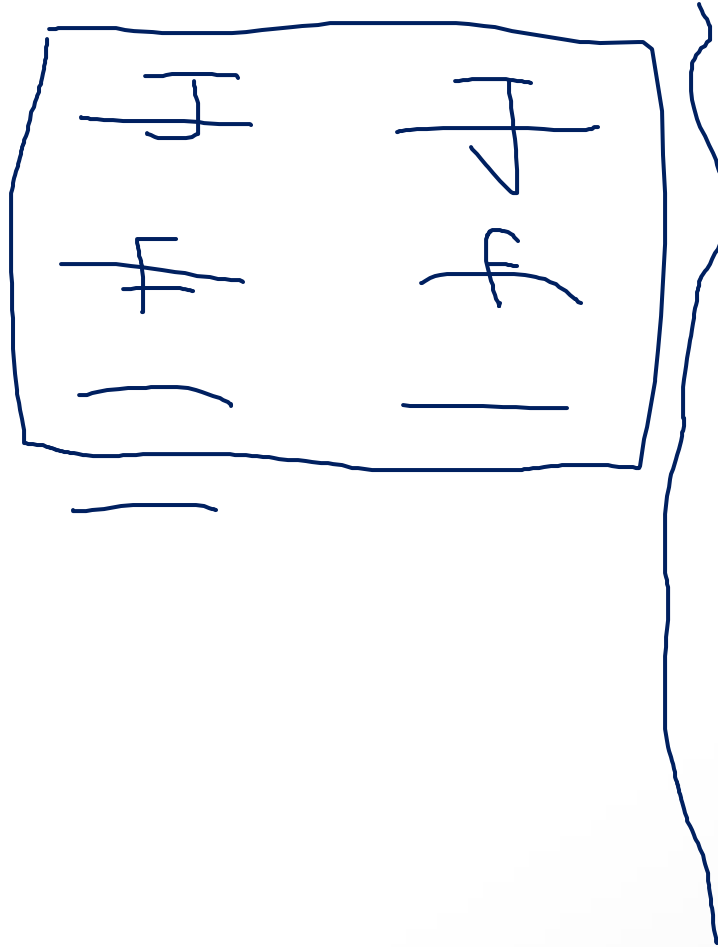


WIP

12 Cmn

24 months

A Q1



D

J  
A  
S  
...  
D  
J  
...  
W


# Range Partitioned Tables

- Scripts from samples, lab, and whitepaper (updated) in file: **PartitioningScripts.ssmssln**
- ✓ Step 1: Create filegroups
- ✓ Step 2: Create files in filegroups
- Step 3: Create partition function
- Step 4: Create partition scheme
- Step 5: Create table(s) on partition scheme
- Step 6: Create index(es) on partition scheme
- Optionally, verify data with catalog views and functions
- Add data to tables
  - SQL Server redirects data and queries to appropriate partition

NOTE: As I reach each of these topics, you'll actually see slides that say (Step 4, 5, etc.) over the next few modules



# Whitepaper Demo Scripts Updated: Why?

 *hidden slide  
w/extra details*

- Problems with “left” in terms of logic
- Decided to write whitepaper to show these flaws
- Many people “copied” this example as a base
- In hindsight, general preference is for RIGHT-based
- All scripts are updated with RIGHT
  - ❑ Can go through whitepaper scripts using LEFT
  - ❑ Can go through the entire scenario again using RIGHT (or, just use RIGHT)

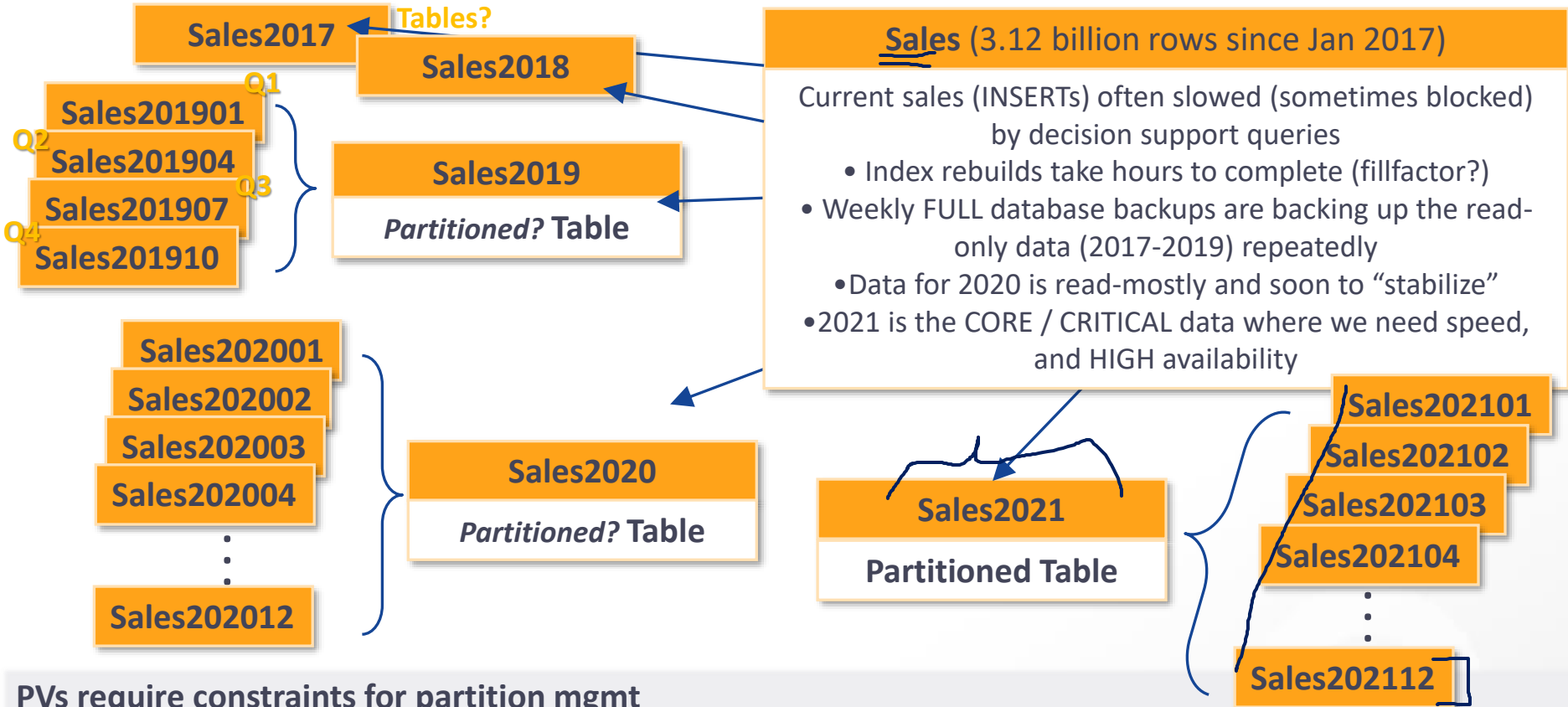


## Step 3: Partition Function

- Not related to scalar / table-valued functions
- Must define the data type over which the function will calculate
  - Partition function can partition over ONLY ONE column of a table
- Always includes the entire domain of possible values from negative infinity ( $-\infty$ ) to infinity ( $\infty$ )
- Table should use constraints to limit the domain of data values (they are not required as they are with partitioned views)

Added for review

# Horizontal Partitioning (PVs or PTs)

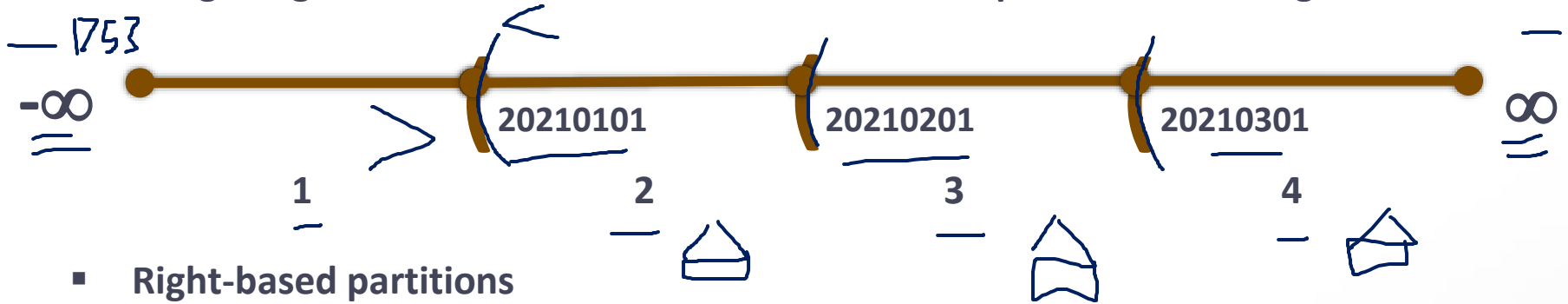


PVs require constraints for partition mgmt  
`CHECK ([SalesDate] >= '20210101'`  
`AND [SalesDate] < '20210201')`

PTs require a partition function and a partition scheme for partition management

# Partition Function

- Range “right” means the value is contained in the partition to the right



- Right-based partitions

- Use lower boundaries

- n boundaries creates n+1 partitions

- ✗ One table, one set of indexes, simplified management (to a point)



# Partition Function

## Left or Right?

- Defines whether or not the first boundary point should fall to the LEFT or the RIGHT within the full domain of values ( $-\infty$  to  $\infty$ )
- Values can use functions to calculate boundary point but boundary point will always be stored as a literal value based on calculation of the function at time the partition function is created
- Only the boundary point is stored, not the expression used to calculate it
- Can see boundary points stored within the catalog view (`sys.partition_range_values`)

# Partition Function

## Left or Right?

### ■ LEFT

- Defines that the boundary point falls into the FIRST (LEFT, of the first two) partition
- All other boundary points follow this pattern
- An “ending point” makes more sense when using LEFT
- In date-based partitioning, this can be a very frustrating value with which to work

### ■ Right

- Defines that the boundary point falls into the SECOND (RIGHT, of the first two) partition
- All other boundary points follow this pattern
- A “beginning point” makes more sense when using RIGHT
- In date-based partitioning, this is an easier value to define (recommended)

# Choosing Upper Boundary (LEFT)

## Added complexity with datetime data type



hidden slide  
w/extra details

See *whitepaper* for complete  
text for these diagrams

- Upper boundary is inclusive ( $\leq$ ) so must be exact
- Datetime value of 23:59:59.999 is not “available” due to the precision with datetime data. If entered it will round to 00:00:00.000 of the next day
  - 23:59:59.999 rounds up to 00:00:00.000
  - 23:59:59.998 rounds down to 23:59:59.997
  - 23:59:59.997 can be stored and is the last explicit value supported
- The partition function should use either of these two values for the upper boundary:
  - date 23:59:59.997
  - DATEADD(ms, -3, date) 🟡 Much safer to use this format for left!

# Create the Partition Function

## Using LEFT

- Orders and Order Details will include the OrderDate column (yes, duplicated date needed in the Order Details table)
- For example, if orders ranged from November 2016 through November 2018

```
CREATE PARTITION FUNCTION TwoYearDateRangePFN(datetime)
AS
RANGE LEFT FOR VALUES
( '20161130 23:59:59.997',      -- Nov 2016
  '20161231 23:59:59.997',      -- Dec 2016
  '20170131 23:59:59.997',      -- Jan 2017
  '20170228 23:59:59.997',      -- Feb 2017
  ...
  '20181130 23:59:59.997')      -- Nov 2018
```



# Create the Partition Function

## Using LEFT with DATEADD Function



hidden slide  
w/extra details

See *whitepaper* for complete  
text for these diagrams

- Can use a function in place of a literal value
- Function is evaluated at creation and only the calculated value is stored
- See sys.partition\_range\_values for values

```
CREATE PARTITION FUNCTION TwoYearDateRangePFN(datetime)  
AS  
RANGE LEFT FOR VALUES  
(  
    dateadd (ms, -3, '20161201') -- Nov 2016  
    , dateadd (ms, -3, '20160101') -- Dec 2016  
    , dateadd (ms, -3, '20170201') -- Jan 2017  
    , dateadd (ms, -3, '20170301') -- Feb 2017  
    ...  
    , dateadd (ms, -3, '20181201') -- Nov 2018
```



# Create the Partition Function

## Using RIGHT

- Orders and Order Details will include the OrderDate column (yes, duplicated date needed in the Order Details table)
- For example, if orders ranged from November 2016 through November 2018

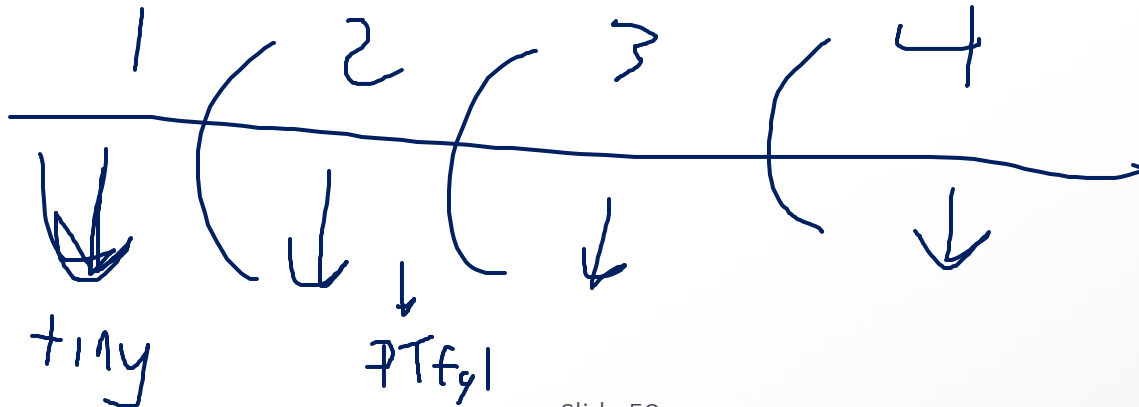
```
CREATE PARTITION FUNCTION TwoYearDateRangePFN(datetime)
AS
RANGE RIGHT FOR VALUES
( '20161101',      -- Nov 2016
  '20161201',      -- Dec 2016
  '20170101',      -- Jan 2017
  '20170201',      -- Feb 2017
  ...
  '20181101')      -- Nov 2018
```

# Partitioning: Date Column

- **SQL Server 2008 added many new date-related data types**
  - date, time, datetime2, etc.
- **Avoids the lack-of-precision problem (and therefore, programmability) with datetime and partition switching in the sliding window scenario**
- **You no longer need to write your partition boundaries with '23:59:59.997' to match the last time value in a day**
- **More details about the SQL Server 2005 limitation in the MSDN Whitepaper: Partitioned Tables and Indexes in SQL Server 2005**
  - ✓ <http://msdn2.microsoft.com/en-us/library/ms345146.aspx>
- **Examples using date in the SQL Server 2008 whitepaper:**
  - ✓ <http://msdn.microsoft.com/en-us/library/dd578580.aspx>

## Step 4: Partition Scheme

- Maps the partitions (number defined by the function) to the filegroups (and therefore files) with the database
- Because both the extreme left and extreme right boundary cases are included, a partition function with  $n$  boundary conditions creates  $n+1$  partitions
- This first (in RIGHT) partition will remain empty as data slides / rolls forward
  - Note: this is not a requirement at the time of definition but it's a good idea if you want your management procedure to be consistent from the first switch procedure



## Step 4: Partition Scheme

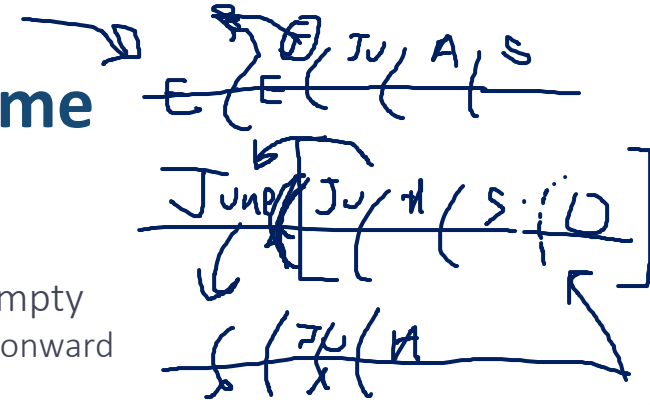
- **You can have empty partitions**

- In RIGHT-based partition functions, the first partition is empty
  - Pro – Your automation routines are the same from the first onward
  - Con – None, except a few extra partitions to define
- If LEFT-based partition functions, the last partition is empty (similar pros / cons)

- **If using an empty partition:**

- Use a special “tiny” filegroup
  - Create a file/filegroup that’s restricted to 1MB with autogrowth disabled
    - Fast fail if you “MERGE” the wrong boundary point
    - Data should never reside in it
- Constraints should be used to restrict the table's data.

- **Explicitly name a filegroup for the "active" partitions and then use “tiny” for the empty partition (currently, and always to remain empty)**



## Step 4: Partition Scheme

- If we hold 2 years worth of Order/Order Details data in 24 partitions we will create a partition scheme to handle 25 partitions where the FIRST one will begin/remain empty:

```
CREATE PARTITION SCHEME [TwoYearDateRangePScheme]
AS PARTITION [TwoYearDateRangePFN] TO
( [Tiny],
  [FG1], [FG2], [FG3], [FG4], [FG5],
  [FG6], [FG7], [FG8], [FG9], [FG10],
  [FG11], [FG12], [FG13], [FG14], [FG15],
  [FG16], [FG17], [FG18], [FG19], [FG20],
  [FG21], [FG22], [FG23], [FG24])
GO
```

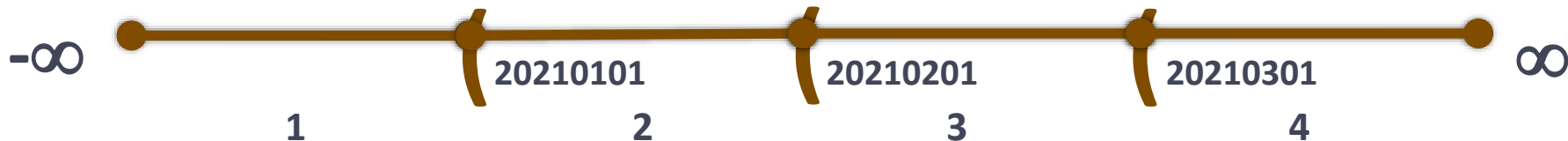
Handwritten annotations: A circle around [Tiny], an arrow pointing to [FG3], a '3' above [FG5], a '4' and 'next' to the right, and 'next' written below the closing parenthesis of the partition list.

# DEMO: Partitioned Tables Walkthrough



# Aligned and Storage-Aligned Schemes

Function defines 3 boundary points  $\therefore$  4 partitions



```
CREATE PARTITION SCHEME [Scheme1]
AS PARTITION [FNwith3BoundaryPoints]
TO ([Tiny], [FG1], [FG2], [FG3]);
GO
CREATE TABLE [T1] ... ON [Scheme1];
CREATE TABLE [T2] ... ON [Scheme1];
```

```
CREATE PARTITION SCHEME [Scheme2]
AS PARTITION [FNwith3BoundaryPoints]
TO ([Tiny], [FG4], [FG5], [FG6]);
GO
CREATE TABLE [T3] ... ON [Scheme2];
```

- Scheme1 and Scheme2 reference the same function  $\Rightarrow$  Aligned
- If an object uses the same scheme  $\Rightarrow$  Storage Aligned
  - T1, T2, and T3 are all aligned
  - T1 and T2 are storage aligned

# Verify Partition Data / Location

- Programmatically determine which partition...

```
SELECT $partition.PFN('date-to-modify')
```

- See all data across all partitions...

```
SELECT $partition.[PFN]([s].[Date]) AS [Partition#]  
      , min([s].[Date]) AS [Min Sale Date]  
      , max([s].[Date]) AS [Max Sale Date]  
      , count(*) AS [Rows In Partition]  
FROM [dbo].[Sales] AS [s]  
GROUP BY $partition.[PFN]([s].[Date])  
ORDER BY [Partition#]
```

- BEST: Selectively review a specific index/partition

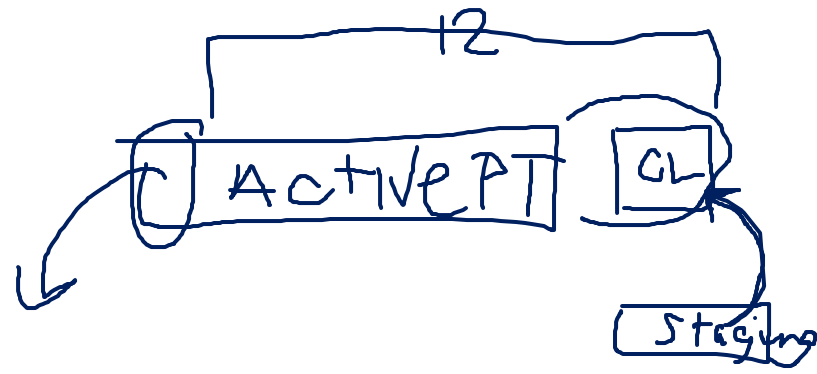
```
SELECT * FROM sys.dm_db_index_physical_stats  
(db_id(), object_id('name'), IDX_ID, PART_ID, 'mode')
```



# Review

- **Range-based partitioned tables**
- **Partition function**
  - Left or right?
  - Partitioning: date column
- **Demo – partitioned tables walkthrough**
- **Partition scheme**
  - Aligned schemes
  - Storage-aligned schemes
- **Verifying partition data / location**

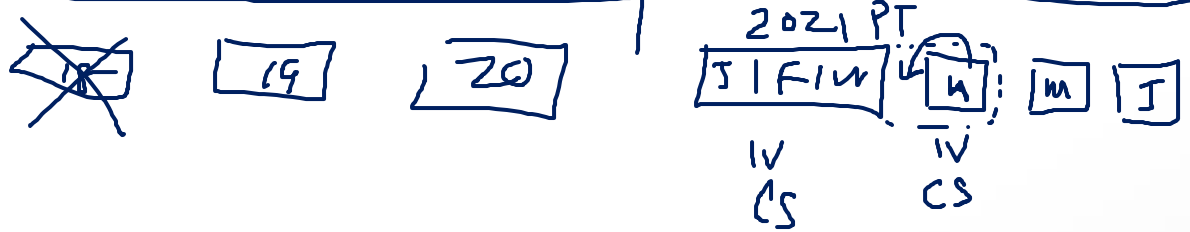
PW



OLTP

OLTP

RO | RW



# Questions!



## Module 4: Tables, indexes, Keys, and Relationships



# Overview

- **Partitioning keys**
- **Primary and foreign keys**
- **Relationships**
- **Partitioned object indexing strategies**
  - Indexing strategies based on workload
  - Row-based indexes
  - Column-based indexes
  - Base table structure key points
  - Converting an existing table

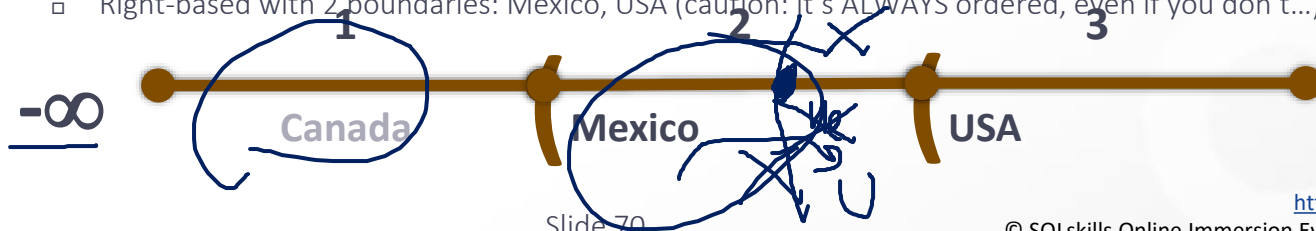
# Partitioning Keys

## ■ Partitioned Views

- Must choose the primary partitioning column – for updateable partitioned views
  - UPDATES and DELETES must specify this column in the WHERE clause
  - ✓ □ INSERTs might not use the updateable partitioned view (remember “challenges”)
- Can have ~~multiple columns~~ over which elimination can occur
  - Constrained date column (`Date >= '20210101' AND Date < '20210201'`) ✓
  - Constrained identifier column (`OrderID >= '1' AND OrderID < '2000000'`) ✓

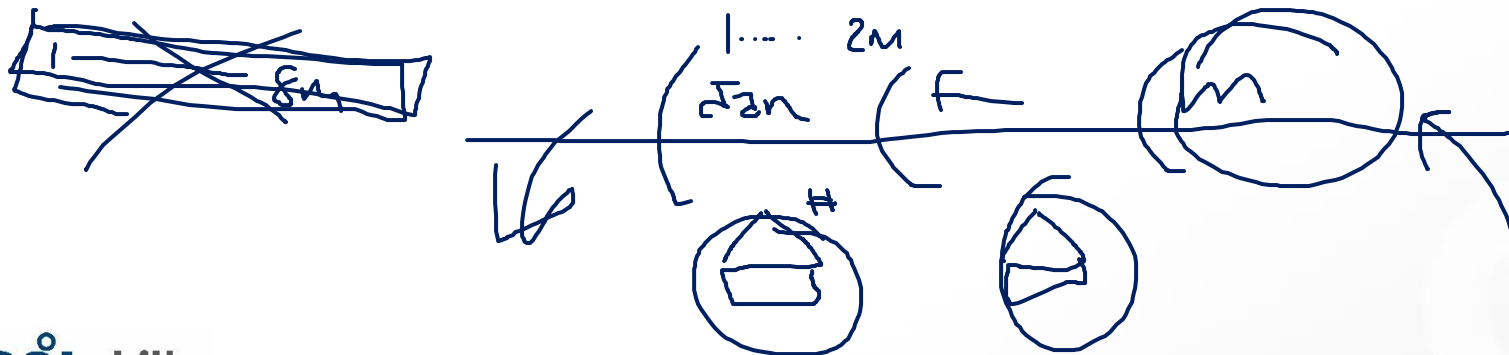
## ■ Partitioned Tables

- Only one column
- Can be a persisted computed column
- ✓ □ Always “range-based” but a range can also be a list without any other values (“list-based”)
  - Want to have 3 partitions: Canada, Mexico, USA
    - Right-based with 2 boundaries: Mexico, USA (caution: it's ALWAYS ordered, even if you don't...)



# Primary Keys (1 of 2)

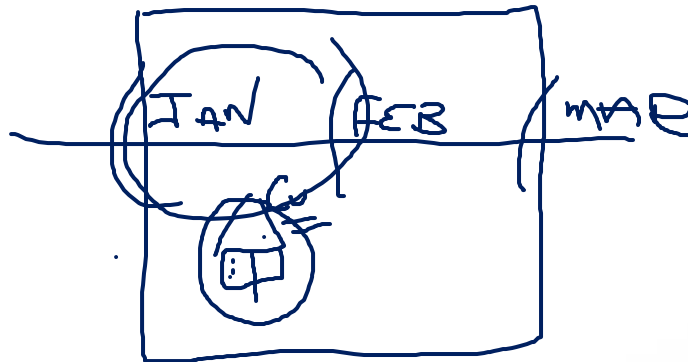
- **Partitioned Views** X X X X
  - Updateable partitioned views require the partitioning key to be the leading column of the primary key
    - Partition by OrderDate then PK composite key = OrderDate, OrderNumber
  - No other requirements on indexes / keys
- **Partitioned Tables**
  - Require the partitioning column to be **part of ALL ALIGNED, UNIQUE indexes**  
(doesn't have to be the leading column)



# Primary Keys (2 of 2)

## ■ Partitioned Tables

- Partitioning column = OrderDate
- Primary key = OrderDate, OrderNumber
- Secondary key on OrderNumber
  - If UNIQUE
    - If aligned / partitioned then must be either:
      - OrderNumber, OrderDate or OrderDate, OrderNumber
    - If UN-aligned / NOT-partitioned then can be
      - UNIQUE on OrderNumber
      - No fast-switching
  - If non-unique / partitioned then partitioning column is “non-selective leading edge to the index”
    - Meaning they know the data is “June” but without the partitioning column then they don’t know where within June
    - Optimal when the query is over an entire partition’s data set but not when subset over partitioning column (can look up all rows for a customer and then eliminate by date)





# Foreign Keys

- **Partitioned object referencing other objects**

- No problem
  - Partitioned Views – underlying Orders202101 references Products
  - Partitioned Tables – Orders references Products

- **Other tables referencing the partitioned object should be aligned**

- OrderHeaders and OrderDetails

- OrderHeaders

- Partitioning column = OrderDate
    - Primary key = OrderDate, OrderNumber

- OrderDetails

- Partitioning column = OrderDate (this probably wouldn't have even been in the table...)
    - Primary key = OrderDate, OrderNumber, OrderLineNumber
    - Foreign key OrderDate, OrderNumber REFERENCES OrderHeader (OrderDate, OrderNumber)
    - Concept is to switch in the referenced table first; however, SQL doesn't allow FKs on fast switching
      - Drop constraint, switch in, recreate the constraint WITH CHECK
    - Concept is to switch OUT the referencing table first; however, SQL doesn't allow FKs on fast switching
      - Drop constraint, switch out, recreate the constraint WITH CHECK

# Partitioned Object Indexing Strategy

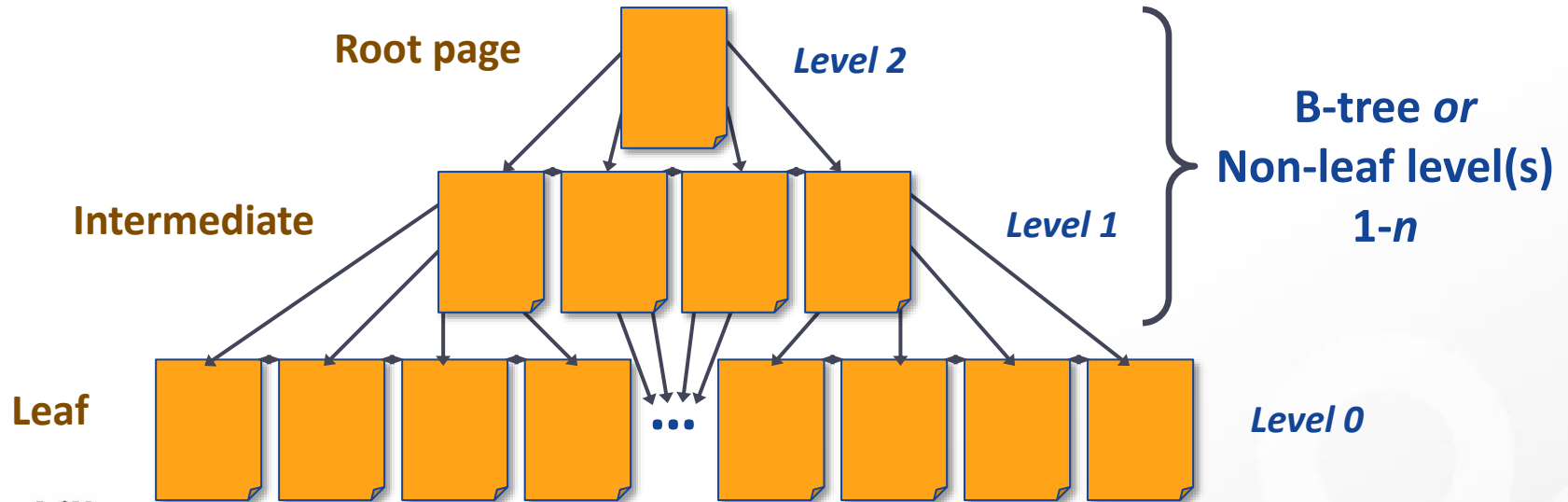
- **SQL Server 2008 is the lowest (IMO) version for large tables, performance, scalability**
  - Added data compression (row and page compression)
  - Added filtered indexes / filtered statistics
  - Fixed fast-switching for partition-aligned, indexed views
- **SQL Server 2012 adds read-only, nonclustered, columnstore indexes**
  - Some frustrating “batch-mode” limitations for partitioned views (UNION ALL)
    - If you’re using PVs then you should use a minimum of SQL Server 2014!
- **SQL Server 2014 adds updateable, clustered, columnstore indexes**
  - Many of the most frustrating limitations with CS fixed – for example, UNION ALL supports batch mode (which means you can use these with partitioned views)
  - Added “incremental statistics” to help reduce when to rebuild as well as time to rebuild
- **SQL Server 2016+ takes columnstore indexes even further with updateable, nonclustered, columnstore indexes and row-based, nonclustered indexes with clustered, columnstore indexes!**

# General Indexing Strategies Based on Workload

- **OLTP – Online transaction processing**
  - Traditional row-based clustered and nonclustered indexes
- **Dedicated Decision Support System / Relational Data Warehouse**
  - Prior to SQL Server 2012
    - Traditional row-based clustered and nonclustered indexes (with indexed views)
  - SQL Server 2012+:
    - Consider adding a read-only, nonclustered, columnstore index for partitioned objects leveraging partition switching as additional data is added
  - SQL Server 2014+:
    - Use the SQL Server 2012+ strategy above
    - Or, consider the new, updateable, clustered, columnstore index
    - NOTE: Still quite a few restrictions in 2014 – best to consider **2016 and higher** instead!
- **Hybrid**
  - Most likely to use traditional, row-based clustered and nonclustered indexes and possibly add a nonclustered, columnstore index for the read-only data – **if you've partitioned your data in the hybrid scenario**

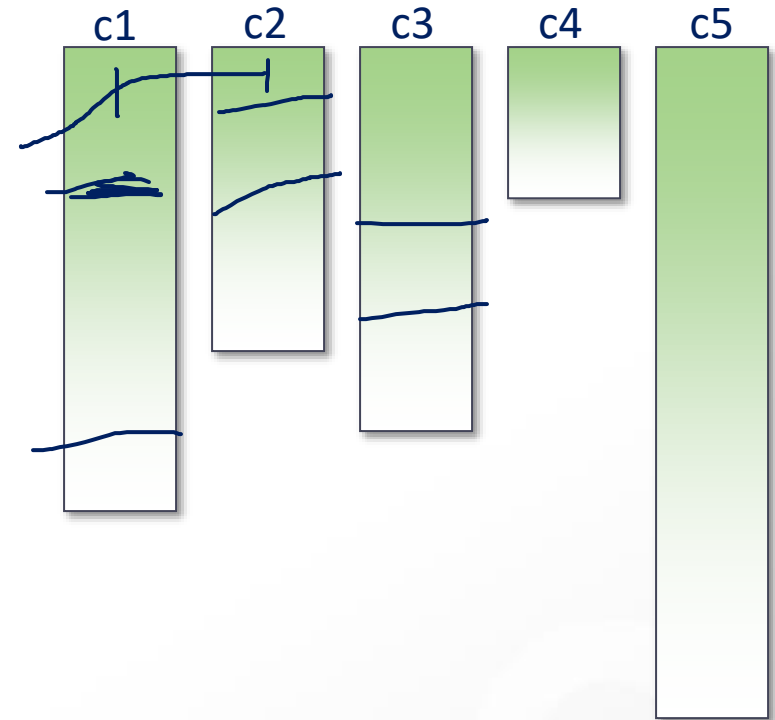
# Index Structures: Row-based Indexes

- **Leaf level:** contains something for every row of the table in indexed order
- **Non-leaf level(s) or B-tree:** contains something, specifically representing the FIRST value, from every page of the level below. Always at least one non-leaf level. If only one, then it's the root and only one page. Intermediate levels are not a certainty.

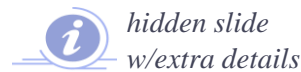


# Index Structures: Column-based Indexes *hidden slide w/extra details*

- Column-based
- Only data from that column is stored on the same page – potentially **HIGHLY** compressible!
- Data is segmented into groups of ~1M rows for better batch processing
- SQL Server can do “segment elimination” (similar to partition elimination) and further reduce the number of batch(es) to process!

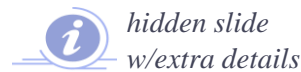


# Columnstore: Data Types NOT Supported



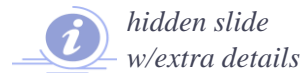
- **Data types not supported in some versions:**
  - uniqueidentifier (applies to SQL Server 2012 (11.x))
  - nvarchar(max), varchar(max), and varbinary(max) (applies to SQL Server 2016 (13.x) and prior versions, and nonclustered columnstore indexes but can be used in SQL Server 2017 clustered, columnstore tables)
- **Data types unlikely to be supported in any versions**
  - ~~n~~text, text, image, and XML ?
  - ~~rowversion (and timestamp)~~
  - CLR types (hierarchyid and spatial types)
  - sql\_variant
  - Sparse columns ⇒
- **NOTE: You CAN have a filtered NC columnstore index in SQL Server 2016 (13.x)+**
- **Review: <https://bit.ly/2F2LgJ5> (MSDN: CREATE COLUMNSTORE INDEX)**

# Features and Capacity Specifications



- **Editions and supported features of SQL Server 2016**
  - <https://bit.ly/2qK9X93>
- **Editions and supported features of SQL Server 2017**
  - <https://bit.ly/2JWsmHz>
- **Maximum Capacity Specifications**
  - <https://bit.ly/2EZWGO3>
- **NOTES (since these are what *partially* led to needing these links/references)**
  - **LIMIT:** Tables per SELECT statement ( view ) => Limited only by available resources
  - **LIMIT:** Memory for Columnstore segment cache per instance of SQL Server Database Engine by Edition:
    - Enterprise: Unlimited
    - Standard: 32GB

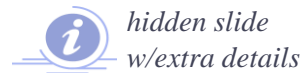
# Base Table Structure Key Points



- **If you're not on SQL Server 2012+ your options are limited to row-based indexes**
- **For DSS / RDW workloads with partitioning (partitioned views or partitioned tables):**
  - At a minimum, create and test a nonclustered, columnstore index on your large fact tables
    - They're super easy to create
      - You can have only one
      - There's no column order
      - It's highly compressed so it doesn't take a lot of space
      - You might get HUGE gains for large scan / aggregate queries – especially those that are narrow (in terms of columns)
    - For better batch processing you must use SQL Server 2014 or higher
- **If you're debating about your next version – don't!**
  - Go straight to SQL Server 2016 (or higher!)



# Columnstore Performance Gains



- **Two complimentary technologies:**

- Storage

- Data is stored in a compressed columnar data format (stored by column) instead of row store format (stored by row)
      - Columnar storage allows for less data to be accessed when only a sub-set of columns are referenced
      - Data density/selectivity determines how compression friendly a column is – example “State” / “City” / “Gender”
      - Translates to improved buffer pool memory usage and fewer IOs

- New “batch mode” execution

- Data can then be processed in batches (1,000 row blocks) versus row-by-row
    - Depending on filtering and other factors, a query may also benefit by “segment elimination” - bypassing million row chunks (segments) of data, reducing I/O

# Batch Mode Processing

- **Allows processing of 1M row blocks as an alternative to single row-by-row operations**
  - Enables additional algorithms that can reduce CPU overhead significantly
  - Batch mode “segment” is a partition broken into million row chunks with associated statistics used for Storage Engine filtering
- **Batch mode can work to further improve query performance of a columnstore index, but this mode isn’t always chosen:**
  - Some operations aren’t enabled for batch mode:
    - E.g. outer joins to columnstore index table / joining strings / NOT IN / IN / EXISTS / scalar aggregates / and UNION ALL (meaning PVs)
  - Row mode might be used if there is SQL Server memory pressure or parallelism is unavailable
  - Confirm batch vs. row mode by looking at the graphical execution plan

# Row-based Indexes v. Column-based Indexes

- **Supports data compression**
  - Row compressed
  - Page compressed
- **Can support point queries / seeks**
- **Wide variety of supported scans**
  - Full / partial table scans (CL)
  - Nonclustered covering scans (NC)
  - Nonclustered covering seeks with partial scans (NC)
- **Biggest problems**
  - More tuning work for analysis: must create appropriate indexes per query  
Must store the data (not as easily compressed)
- **Column-based indexes**
  - Significantly better compression
  - COLUMNSTORE / COLUMNSTORE\_ARCHIVE
- **Supports large scale aggregations**
- **Support partial scans w/“segment” elimination**
  - Only the needed columns are scanned
  - Data is broken down into row groups (roughly 1M rows) and segments can be eliminated
  - Combine w/partitioning for further elimination
  - Parallelization through batch mode processing
- **Biggest problems**
  - Minimum set for reads is a row group (no seeks)
  - Limitations of features for batch mode by version
  - Limitations with other features (less and less by SQL Server version)

# The Clustering Key

- **For columnstore indexes there's no clustering key defined; however, converting an existing clustered index (cannot be a PK) to a columnstore index can provide benefits in segment elimination (use DROP\_EXISTING)**
- **For row-based indexes the clustering key is critical for performance**
  - Clustering key defines data order
    - Some clustering keys are more prone to fragmentation
    - Others can have issues with contention
    - Others cause inefficiencies in the nonclustered indexes
  - Clustering key is used for “lookups” into the data
    - This means that nonclustered indexes are actually different when created on a table that has a clustered index (as opposed to a table that does not)
      - This dependency should affect our choice in clustered index!
- **Check out my Pluralsight course – [SQL Server: Indexing for Performance](#)**
- **Check out Paul's Pluralsight course – [SQL Server: Index Fragmentation Internals, Analysis, and Solutions](#)**

## Step 5: Create Table

```
CREATE TABLE [Sales]
(
    [SalesDate] [datetime2] NOT NULL
    CONSTRAINT [DF_Sales_SalesDate]
    DEFAULT SYSDATETIME()

    ...
) ON [PartitionScheme]([SalesDate])
```

- Column has to match data type of partition function
- Table is created on partition scheme
- Can be done with an online operation for an existing table by rebuilding the clustered index ON the partition scheme
  - Nonclustered indexes are un-aligned until manually rebuilt on the same (or aligned) scheme
  - Table is kept online but fast-switching is not allowed until ALL indexes are aligned
    - Some indexes might not be able to be aligned without their definition changing! Remember, all UNIQUE indexes MUST have the partitioning column somewhere in the KEY of the NC index!

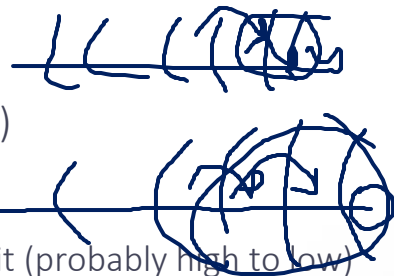
## Step 6: Create Indexes

~~✗~~ CREATE UNIQUE CLUSTERED INDEX [SalesPK]  
ON [Sales] ([SalesID])  
WITH (DROP\_EXISTING = ON, ONLINE = ON)  
ON [Sales4Partitions\_PS]([SalesID])

- Can “move” a table to one or more filegroups as an online operation by rebuilding the clustered index on the new scheme WITH ONLINE = ON
  - ~~✗~~ □ Note: Online only when there are no LOB columns in the table (fixed in SQL Server 2012); legacy LOBs (text, ntext, and image) do not support ONLINE operations in any edition
- ~~✗~~ ■ If clustered index was created through a primary key constraint, you rebuild by using the constraint name and same definition – on new scheme (example above is for a primary key)
- Secondary nonclustered indexes are NOT moved/partitioned
- New indexes are automatically aligned by default but existing indexes (prior to partitioning) must be rebuilt / re-created on the new scheme to be aligned properly
- Unique indexes must contain the partitioning column as part of the key; it does not have to be the leading column

# Converting an Existing Table

- **From non-partitioned to partitioned**
- **From partitioned on scheme 1 to partitioned on scheme2**
  - Going from n partitions to n+1 partitions should just use SPLIT
  - Going from n partitions to y partitions (e.g. from 4 partitions to 8)
    - ~~\*~~ Should create a NEW scheme and rebuild on the new scheme
      - Do not SPLIT, SPLIT, SPLIT, etc.
      - If you absolutely have to split? Then, work out the correct way to split (probably high to low)
  - Going from n partitions to a partitions (e.g. from 8 partitions to 4)
    - ~~\*~~ Should create a NEW scheme and rebuild on the new scheme
      - Do not MERGE, MERGE, MERGE, etc.
      - If you absolutely have to merge? Then, work out the correct way to merge (probably low to high)



# SUMMARY: Partitioned Object Indexing Strategies

- **Lots of limitations to columnstore indexes in 2012 and 2014**
- **If you're not already on 2014 then SKIP it and go straight to SQL Server 2016+!**
  - Consider KEEPING your current CE (the legacy CE) and then testing the new CE when troubleshooting (*see next set of [hidden] slides for more details*)
- **Can create any combination of indexes to support ALL types of queries**
  - Row-based, clustered index
    - With row-based, nonclustered indexes (to support point queries)
    - With read-write, nonclustered, columnstore index (for large scale aggregations)
  - Read-write, clustered, columnstore index
    - With row-based, nonclustered indexes (to support point queries)
- **Plus, bug fixes with batch-mode processing...**
- **Plus, SQL Server 2016 adds query store**
  - SQL Server 2017 adds further enhancements including adaptive query processing...
  - \* SQL Server 2019 adds even further query processing and performance enhancements...




# Trace Flags vs. Query Hints

- **Trace flags are meant more for administrative use**
  - Some are “global” only and do nothing at the session level
    - See [BOL](#) under: scope “global only”
    - Using DBCC TRACEON (#, -1) sets a trace flag globally but only until next restart
    - Set as a startup option (-T #) if you want this set for each service restart
  - Check FIRST to see if there’s a better way to set these AND re-check on each SP / upgrade
- **Query hints are a MUCH better way of enabling these behaviors:**
  - OPTION clause for your query
  - SELECT ...  
OPTION (USE HINT ('query\_hint', 'query\_hint'))
  - Example – instead of using trace flag 9481  
SELECT ...  
FROM ...  
OPTION (USE HINT ('FORCE\_LEGACY\_CARDINALITY\_ESTIMATION'))

# Legacy Cardinality Estimation Model:

Cardinality Estimation, Compatibility Mode, and Query Hints

 *hidden slide  
w/extra details*

- ~~YUCK: Service-wide trace flag on start up (set in service manager)~~  
→ ~~-T 9481~~
- ~~Testing?: Temporary, but server-wide trace flag~~  
→ ~~DBCC TRACEON (9481, -1)~~
- Better: Session-level Testing! → DBCC TRACEON (9481)
- SQL Server 2014: any compatibility mode less than 120
- SQL Server 2016+:
  - Any compatibility mode less than 120
  - Database compatibility mode  $\geq 120$  but scoped database configuration option is on:  
**ALTER DATABASE SCOPED CONFIGURATION  
LEGACY\_CARDINALITY\_ESTIMATION = { ON | OFF | PRIMARY }**
- Seeing which CE you're using: (Properties Window with Showplan)
  - CardinalityEstimationModelVersion: 70 (legacy)

# New Cardinality Estimation Model (1 of 2)

## Cardinality Estimation, Compatibility Mode, and Query Hints

- **Yuck: Service-wide trace flag on start up → -T 2312**
- **Testing: Temporary / server-wide trace flag → DBCC TRACEON (2312, -1)**
- **Better: Session-level Testing! → DBCC TRACEON (2312)**
- **SQL Server 2014: compatibility mode 120**
- **SQL Server 2016+:**
  - Any compatibility mode greater than or equal to 120
  - Each compatibility mode has optimizer fixes AND sometimes subtle changes / fixes to the cardinality estimation model
  - If you want to use the cardinality estimation model for that database compatibility model
    - **OPTION (USE HINT ('FORCE\_DEFAULT\_CARDINALITY\_ESTIMATION'))**
    - Overrides using the legacy cardinality estimation model when set through the database scoped configuration
- ***Continued on next slide***

# New Cardinality Estimation Model (2 of 2)

## Cardinality Estimation, Compatibility Mode, and Query Hints

- If you want to get ONLY the RTM optimizer fixes for that version
  - `OPTION (USE HINT ('QUERY_OPTIMIZER_COMPATIBILITY_LEVEL_n'))`
  - Does NOT affect the cardinality estimation model when set through the database scoped configuration
- If you want to get the optimizer hotfixes post-RTM
  - `OPTION (USE HINT ('ENABLE_QUERY_OPTIMIZER_HOTFIXES'))`
  - Does NOT affect the cardinality estimation model when set through the database scoped configuration
  - Equivalent to trace flag 4199 (for more information see: [DBCC TRACEON - Trace Flags \(Transact-SQL\)](#))

### ■ Seeing which CE you're using: (Properties Window with Showplan)

- CardinalityEstimationModelVersion: #
  - 120 = SQL Server 2014
  - 130 = SQL Server 2016
  - 140 = SQL Server 2017
  - 150 = SQL Server 2019

# Migrations / Upgrades / Regressions

## ■ Option 1: Least surprise

- Upgrade existing databases (through backup / restore)
- Leave their existing compatibility level intact
  - Restoring / attaching does not “upgrade” the compatibility level
- When troubleshooting, test trace flag 2312 against queries whose estimates are inaccurate (see if they benefit from the new CE model)
  - If so, add the OPTION hint in the specific query that benefits

## ■ Option 2: Better / safer process + with ideal testing (2016 and higher)

- Change to that version’s compatibility level 130+ (for optimizer fixes)
- Set the legacy CE using the scoped configuration option
- TEST (using the query hints to see where beneficial)
- Change to the new CE by turning the database scoped configuration off  
**LEGACY\_CARDINALITY\_ESTIMATION = OFF**
- TEST (possibly using the legacy CE if you find a query that needs it)
- ✓ Add the optimizer fixes post RTM with the database scoped configuration option
- TEST

# Review

- **Partitioning keys**
- **Primary and foreign keys**
- **Relationships**
- **Partitioned object indexing strategies**
  - Indexing strategies based on workload
  - **Row-based indexes** (*be sure to review the hidden slides*)
  - **Column-based indexes** (*be sure to review the hidden slides*)
  - Base table structure key points
  - Converting an existing table

# Questions!



## Module 5: Implementing the Sliding Window





# Overview

- **Implementing the sliding window scenario**
- **The sliding window scenario with partitioned views**
- **The sliding window scenario with partitioned tables**
- **Switching data IN**
  - Staging area/loading in a heap
  - Why load into a staging area?
- **Switching data OUT**
  - For archiving
  - For removal
    - New in 2016
- **The sliding window scenario key points**
- **Concerns with Switch (IN and OUT)**

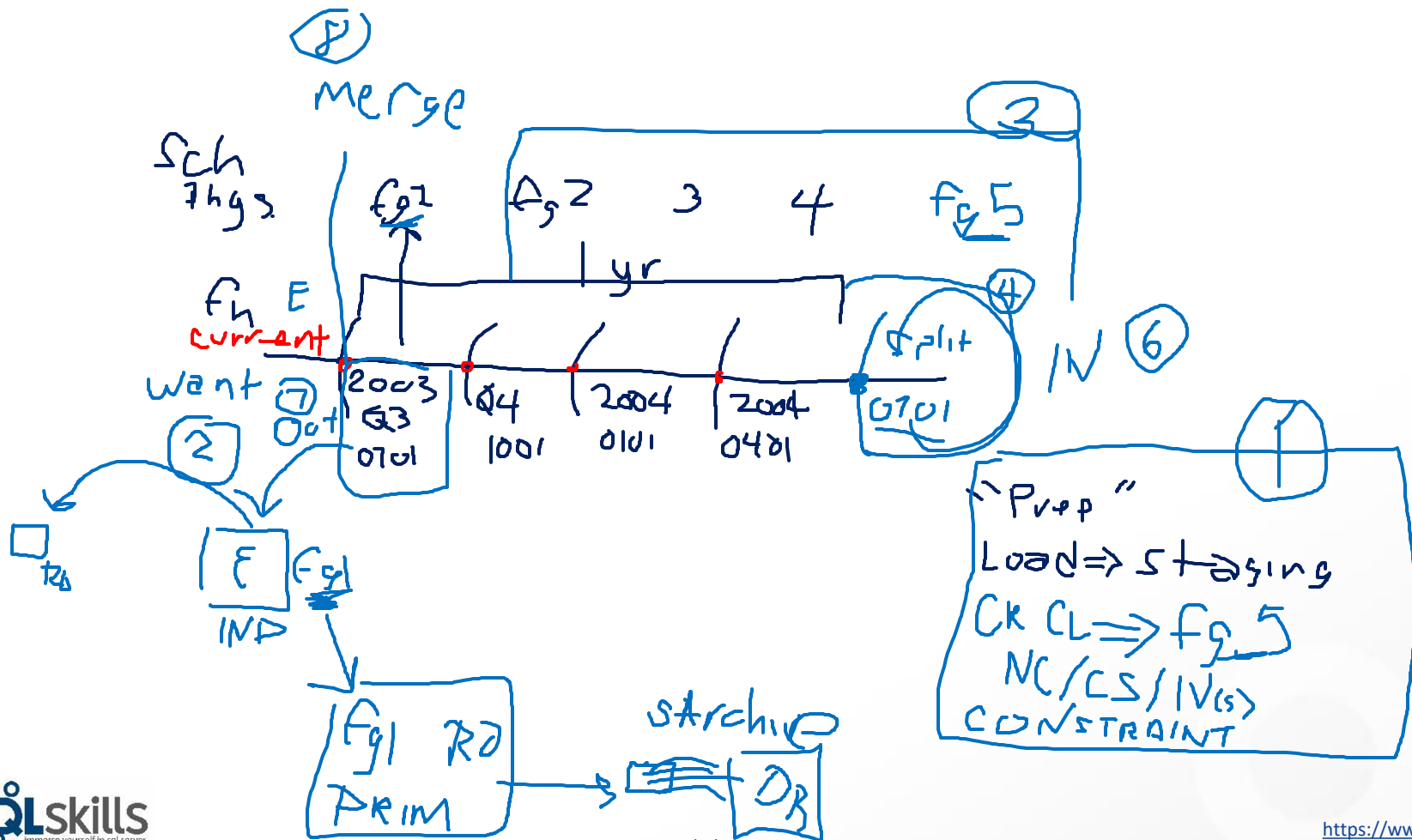
# The Sliding Window Scenario



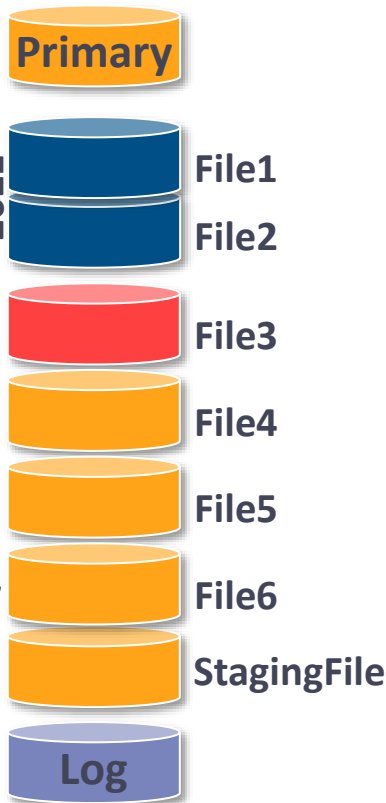
- New data must come in
- Old data must be removed
- The table should only change it's overall data set... with simple metadata changes
- To create metadata ONLY changes
  - Create a new table on the same filegroup where the partition resides
  - Structure it IDENTICALLY to that of the partitioned table
  - SWITCH the partition OUT – the data from the partition “moves” to the empty table
  - The empty table to be switched in
    - Is initially created on a staging area
    - Is moved to the final location by creating a clustered index on the appropriate filegroup
    - SWITCH the new table IN so that the newly manipulated data becomes the partition

# Sliding Window Scenario


- **Get stakeholders in a room...**
- **Whiteboard your data flow**
  - Understand your needs first
  - Technology LAST (meaning syntax and actual implementation)
- **I'll "whiteboard" what I mean...**
  - I think you need to see the flow of things once... we'll go through it
  - Additionally, the following hidden slides are from the SQL Server 2005 Partitioning whitepaper so you can also reference that for more help!



Partitioned DB



# Staging Area/Loading in a Heap

 *hidden slide  
w/extra details*

**Database** *consists of...*

**Filegroups** *consist of...*

**Files** *consist of...*

**Objects** *are placed on a filegroup*

**Modify the view for PVs (switch in for PTs)! (step 3)**

⬅ **Move to final FG (step 2)**

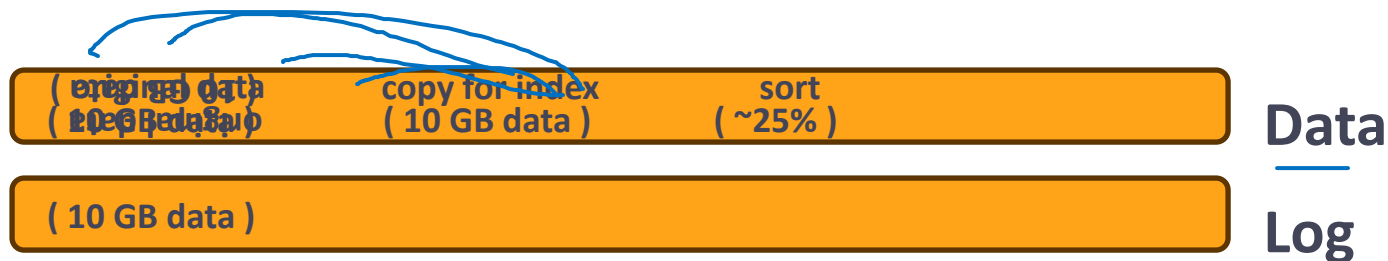
Create CL index on FG

**Remember – the CL Index IS the data**

⬅ **Data load (step 1)**

Create heap on staging FG

# Why Load Into a Staging Area?



- **Log space required depends on recovery model:**
  - 3.25x for FULL recovery model = Log will have size of data
  - 2.25x for BULK\_LOGGED or SIMPLE = Minimally logged operation
- **If you load into destination file then you will have a gap at the beginning of the file after the clustered index is created...**
- **How do you get rid of this gap?**
- **Shrink??**
- **NOOOOOOOOOOOOOOOOO! ☺**

# The Sliding Window Scenario with Partitioned Tables

- **More complicated**
- **Specific syntax required**
  - SPLIT
  - MERGE
- **Requires strategy and choreography**

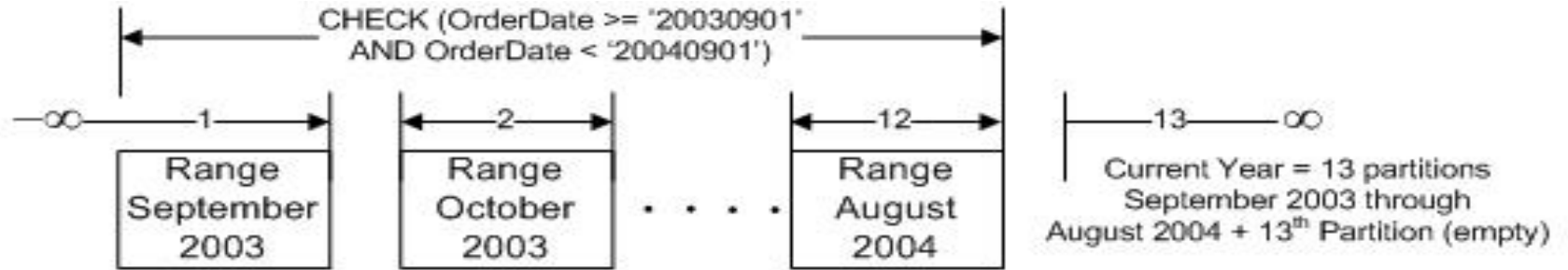
# The Sliding Window: Switch



hidden slide  
w/extra details

See whitepaper for complete  
text for these diagrams

- Whiteboard discussion...
- For speed in manipulation, the sliding window must be implemented as a metadata ONLY “switch” (no data should ever move on SPLIT / MERGE)

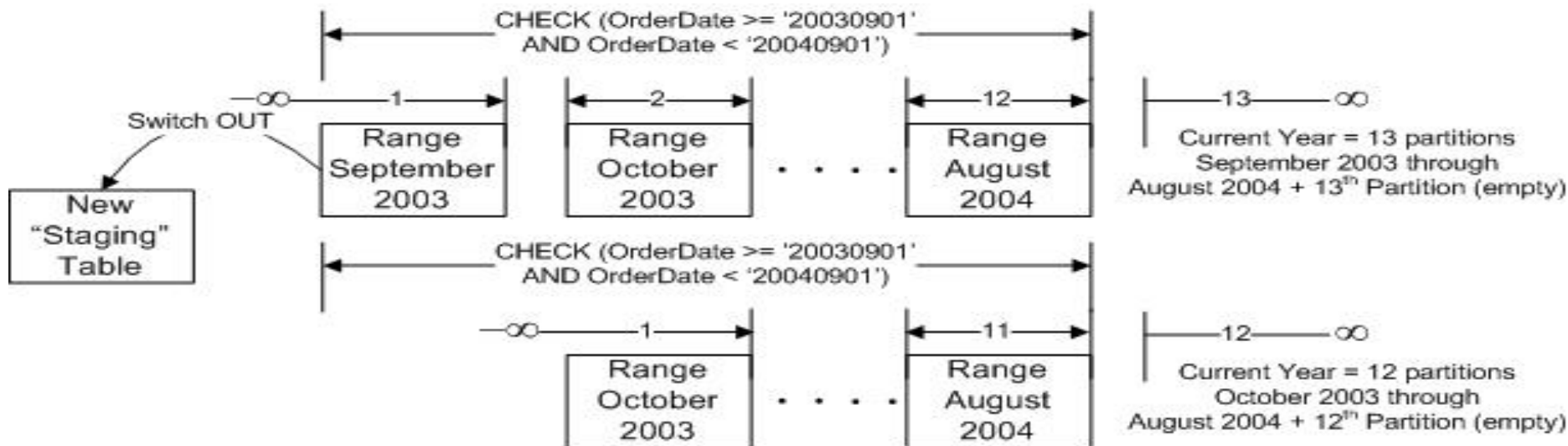


- Most importantly, make sure to “staging” tables on the same physical filegroup as the partition you are switching IN or OUT...




# The Sliding Window: Switch

- Create new table (same structure / indexes)
- Switch OUT the old data

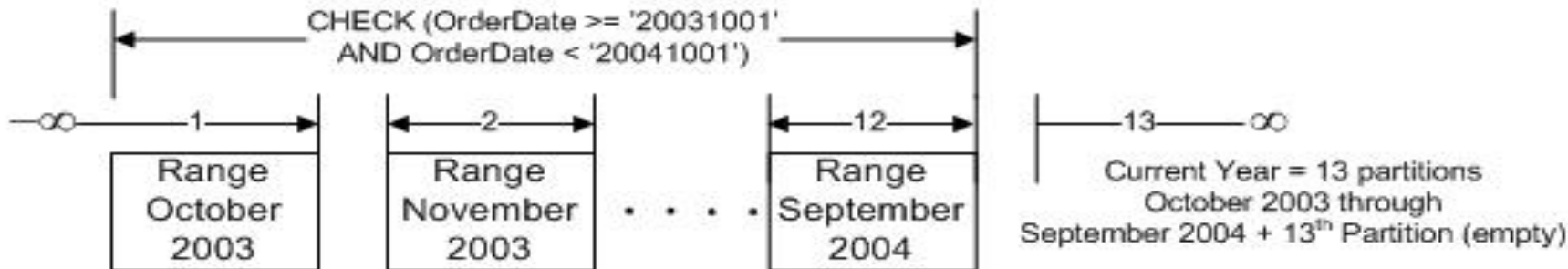
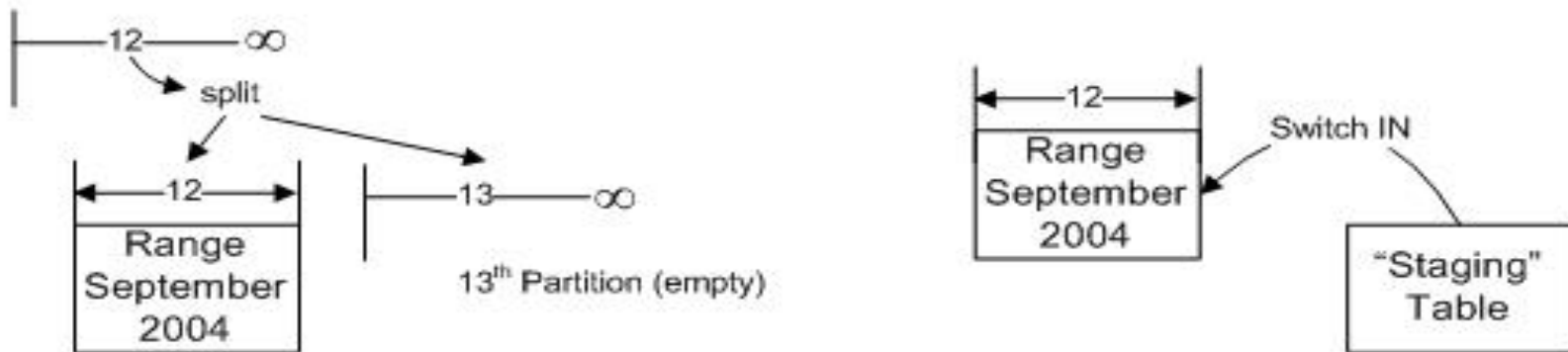


# The Sliding Window: Switch

 hidden slide  
w/extra details

See [whitepaper](#) for complete  
text for these diagrams

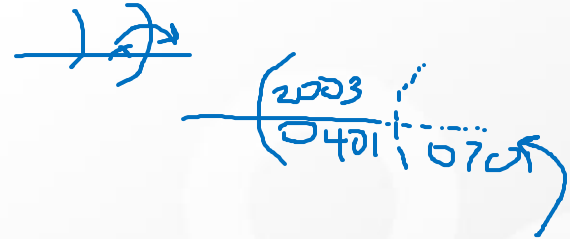
Create new table as a heap, load data, build indexes, then switch IN the current data



# The Sliding Window Summary (1)

- 1) If you're removing data, prepare the "staging out" table
- 2) If you're bringing data in
  - Load / transform / cleanse – in a staging area
  - Move the data by building the clustered index on the destination filegroup
- 3) **Alter the partition scheme to set NEXT USED (based on the filegroup where you created the clustered index above)**
- 4) Split the function to add the new boundary point (this will put the boundary point on the filegroup specified in the scheme's next used setting). **IMPORTANT NOTE:** be sure that NO data needs to move (if using LEFT, then the partitioning split should be empty)

*Note: none of these first few steps impact the actual table!*



## The Sliding Window Summary (2)

- 5) Switch “IN” the new data
- 6) Switch “OUT” the old data

*Note: steps 5 and 6 can be in either order – and are FAST (re: metadata only operations)*

- NEW IN SQL Server 2016: Truncate Table Partitions

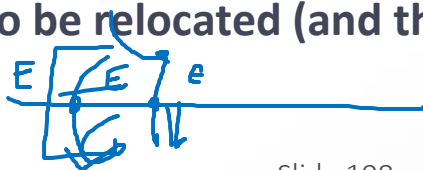
```
TRUNCATE TABLE PartitionTable1 WITH (PARTITIONS (1, 5, 9 TO 11, 14))
```

- 7) Backup / remove the old data (drop the table)

*Note: file-level backups can be restored with the primary filegroup (plus In-Memory filegroup) to another location but not directly into another database [you must INSERT / SELECT]*

- 8) Merge the boundary point (from the emptied [switched out] partition)

**NOTE:** If the “merge” process is slow then it’s likely that you had data in the partition being merged. Be careful never to split or merge where there’s already data; the data will need to be relocated (and the process of moving the data is slow).



# The Sliding Window Key Points

- **The staging tables must match in every way**
  - Same column types
  - Same indexes (and ALL indexes)
  - SQL Server 2008+, all indexed views and their indexes
  - SQL Server 2012+, all columnstore indexes
- **The staging tables must be on the same filegroup at the time of the switch (can build the data in a staging area first and then move to destination)**
- **Be sure that NO data needs to move... test!**
- **One final special consideration for the table that you're switching IN: it must have a "trusted" constraint to guarantee the data matches the partition into which it's switching**

# Concerns with Switch (IN and OUT)

- **Potential blocking concerns**

- Use my script to programmatically wait until the partitioned table is available
- Or, on SQL Server 2014 and higher, use low-priority lock waits:

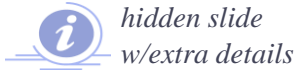


```
WITH ( ONLINE = ON  
      ( WAIT_AT_LOW_PRIORITY  
        (MAX_DURATION = n minutes  
          , ABORT_AFTER_WAIT = BLOCKERS)))
```

- **Will want to update statistics – potentially with FULLSCAN**

- Beginning with SQL Server 2012, statistics are not created by scanning all the rows in the table when a partitioned index is **created** or **rebuilt**. Instead, the query optimizer uses the default sampling algorithm to generate statistics.
- USE CREATE STATISTICS or UPDATE STATISTICS with FULLSCAN to have the statistics updated without sampling

# Sliding Window Scenario with Partitioned Views



## ■ Bringing data in

- Create the new table
- Stage the data IN (possibly a staging area)
- Build the clustered index on the correct/destination filegroup
- Add an NC indexes, Views/IVs, and possibly (probably) a NCCS Index
- Modify the view <- this could create blocking (consider using my script to reduce blocking – just as important with MERGE or using the low priority lock waits)

## ■ Removing data

- Modify the view <- this could create blocking (consider using my script to reduce blocking – just as important with MERGE or using the low priority lock waits)
  - NOTE: TRUNCATE TABLE on an “unused table” **won't work** as ANY query through the view will require a lock (at least an IS lock) on all tables to protect them!
- Archiving
  - Backup the primary + filegroup where table resides
  - Restore PARTIAL just the primary + filegroup (plus In-Memory filegroup)
  - Import into the archive database
  - Drop the table
- Removal
  - Drop the table

# Review

- **Implementing the sliding window scenario**
- **The sliding window scenario with partitioned views**
- **The sliding window scenario with partitioned tables**
- **Switching data IN**
  - Staging area/loading in a heap
  - Why load into a staging area?
- **Switching data OUT**
  - For archiving
  - For removal
    - New in 2016
- **The sliding window scenario key points**
- **Concerns with Switch (IN and OUT)**



# Questions!



## Module 6: Key Partitioning Concerns and Considerations



# Overview

- **Special considerations for partitioned tables**
- **Online operations, indexes, and partitioning**
- **Incremental statistics updates**
- **Partition-aligned index views**
- **Partitioned table with filtered indexes**
- **Interval subsumption**
  - Partitioning / fast-switching
- **Partition-level lock escalation**

# Special Considerations for Partitioned Tables



- If read-write portion is only current month then that's the only place where fragmentation will occur
- If current month is October then only rebuild October:

```
ALTER INDEX [ChargesPTPK] ON [ChargesPT]  
REBUILD PARTITION = 9  
WITH (ONLINE = ON)
```

Msg 155, Level 15, State 1, Line 3

'ONLINE' is not a recognized ALTER INDEX REBUILD PARTITION option.

- **Fixed in SQL Server 2014+ you CAN rebuild a partition as an online operation. But, does that actually solve ALL of the problems with rebuilding partitions?**  
**IMPORTANT: You cannot do an ONLINE partition-level rebuild if ANY of the other partitions (of the partitioned table) are on read-only filegroups (YIKES!!!)**

# Online Operations, Indexes, and Partitioning

- **Partitioned Views – accessible in all editions**
- **Partitioned Tables – all editions, as of SQL Server 2016 SP1**
  - SQL Server 2016 SP1: Making innovation more accessible to all applications
    - <https://bit.ly/2f0tLvW>
- **Indexed Views – accessible in all editions BUT**
  - Require different coding if you want to access them in non-Enterprise Editions
    - VIEW must be named and the NOEXPAND optimizer hint is REQUIRED  
**FROM viewName WITH (NOEXPAND)**
    - Base table queries will NOT benefit from Index
    - ONLY queries that directly name and directly access the view will benefit
- **Online operations are Enterprise-only (Enterprise, Enterprise Eval, Developer)**

# Online Operations, Indexes, and Partitioning

- **Table-level index rebuilds can be done online if there are no LOB columns in the table (fixed in SQL Server 2012)**
  - Legacy LOB columns (text, ntext, image) do NOT support online rebuilds in any version
- **Partition-level index rebuilds can be done online in SQL Server 2014 and SQL Server 2014 introduced incremental statistics (more details on next slide)**
- **Rebuilding/partitioning the clustered index does NOT rebuild any of the nonclustered indexes – these must be rebuilt on the new scheme as well**
  - Until the nonclustered indexes are rebuilt the partitioned table will not support fast-switching; everything else will work!
  - Unique indexes must have the partitioning column as a member of the unique key (to support fast-switching). If you don't need fast-switching then you don't need to do this!

# Incremental Stats Updates

NOTE: See next (new) slide for more information on invalidation thresholds

- New feature for SQL Server 2014 partitioned tables
- Statistics update triggered when the threshold is reached at the PARTITION-level; SQL Server then:
  - Builds partition-level statistic over just that data
  - Compresses the table-level statistic
  - Merges the partition-level statistic in with table-level statistic
  - Optimizer uses the resulting table-level statistic for query estimation
- **Positive:** statistics updates are triggered a lot earlier for partitions that are updates (especially useful in ever-increasing tables)
- **Positive:** less data has to be read for statistics updates (especially useful in ever-increasing tables)
- **Negative:** statistic the optimizer uses for optimization is still limited to 200 steps and can become lossy for VLTs and especially earlier partitions (because of repeated compression)
  - Article: SQL Server 2014 Incremental Statistics on SQLperformance.com
  - <http://bit.ly/1uhEbr1>

# Auto Updating Threshold

- **Automatically updated statistics (updates only when used AFTER invalidation threshold is reached); when auto update statistics is ON (for *both* the DB and the statistic)**
  - Statistics are **invalidated** when:
    - In versions PRIOR to SQL Server 2016: If a minimum of 500 + 20% of the data changes
    - In SQL Server 2016 OR in 2008SP1+ with TF 2371: the threshold is dynamic and tied to the number of rows in the table **130+**
- **Manually update statistics is really what you want for “off hours” and more effective maintenance**
  - Executing UPDATE STATISTICS
    - Might want to **decrease** the frequency of updating for highly volatile tables where distribution isn't changing significantly and you see a lot of “statistics” events
    - Might want to **increase** the frequency of updating for large table where distribution is changing significantly but you're not reaching 20%
  - If you have problems with sampling, consider turning off *auto update stats* at the statistic-level instead of the database-level (more granular control); use:
    - STATISTICS\_NORECOMPUTE on the index definition
    - NORECOMPUTE on the statistics definition



# Summary: How Does Auto-Updating Work?



- **You don't really want auto-updating to be your method of updating statistics**
  - 1) It could happen during production
  - 2) It's likely to be later than desired (causing performance problems along the way)
- **But, LEAVE IT ON (auto\_update\_statistics) as a safety measure**
- **Auto-updating in all versions prior to 2016**
  - Percentage-based threshold unless trace flag 2371
- **Auto-update in version 2016 and higher**
  - Dynamic auto-updating tied to table size (not percentage); same as having turned on 2371
- **Incremental stats – good, but not great**
  - **Positive:** less data has to be read for statistics updates (especially useful in ever-increasing tables)
  - **Negative:** statistic the optimizer uses for optimization is still limited to 200 steps and can become lossy for earlier partitions

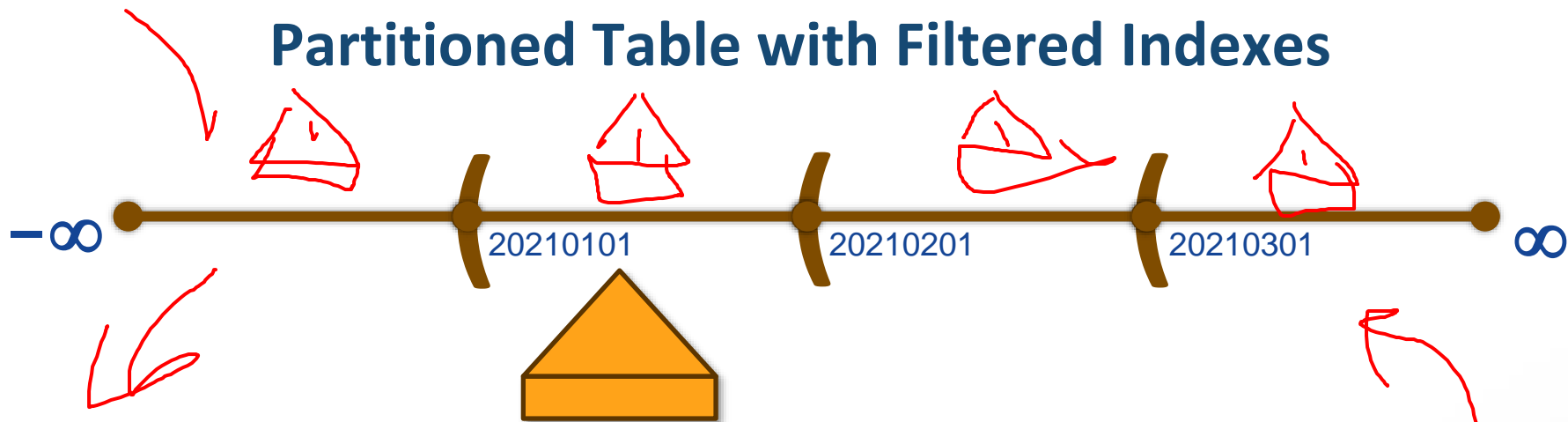
# RECOMMENDATION: Updating Statistics

- **Manually: but automated through a job**
  - Executing sp\_updatestats
    - Sledgehammer maintenance. Only one row has to have been modified.
  - Ola Hallengren's code
    - <http://ola.hallengren.com/>
    - Use @UpdateStatistics parameter / settings
  - Roll your own?
    - Programmatically evaluate the stat\_header data
    - Use sys.dm\_db\_stats\_properties: If 2-5% of the data has changed since last stats update OR (stats\_date older than 1 week AND last update was sampled (sampled < rows)) then UPDATE STATS with FULLSCAN
- **Auto update stats: only as a safety measure**
  - As a fallback if your code doesn't catch EVERY statistic
- **Asynchronous update stats**
  - Unlikely to cause a problem

# Partition-Aligned Index Views

- **SQL Server 2005 does not support partition switching while index views are defined**
  - Drop the index(es) to switch partitions ☹️
  - Then re-create the index(es) once switched
  - SLOW to do (and those queries using the indexed views are slow during this period too!)
- **SQL Server 2008 has partition-aligned indexed views**
  - Allows the switch to happen with a meta-data operation (including the indexed view partition data)
  - Much better support for Data Warehouse maintenance
- **Do you still need indexed views with column-based indexes?**
  - In relational data warehouses with many millions or billions of rows – pre-aggregated data might still be faster for aggregates that have very few groupings
    - E.g. SUM(Sales) by Country

# Partitioned Table with Filtered Indexes



Fast-switching is NOT allowed when filtered indexes are created on individual sets (and are NOT aligned):

```
CREATE INDEX [SubsetOfItems]
ON [Table] ([col1], [col2])
WHERE [Date] >= '20210101'
AND [Date] < '20210201'
```

# Interval Subsumption

## Partitioning / Fast-Switching

- Undesirable to use filtered indexes over specific sets due to the lack of fast-switching
- Can use a filter over the ENTIRE set (and aligned) (e.g. status = 1)
- Filtered indexes cannot be used when the query's predicate is not a subset of a particular filter interval:

- Filter: January data

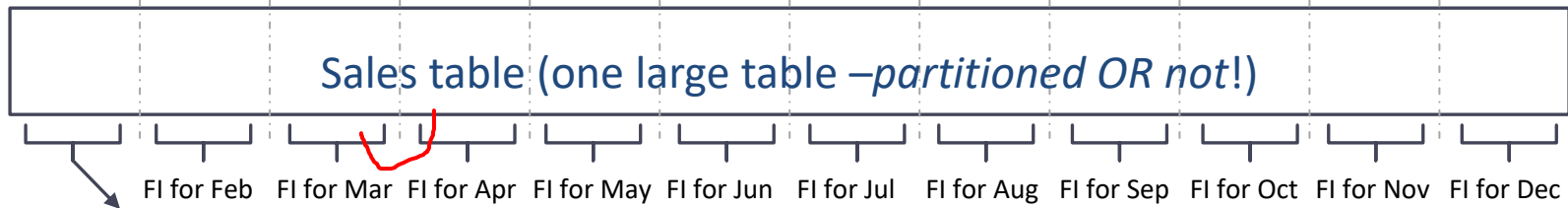
```
WHERE date >= '20210101'  
      AND date < '20210201'
```

- Filter: February data

```
WHERE date >= '20210201'  
      AND date < '20210301'
```

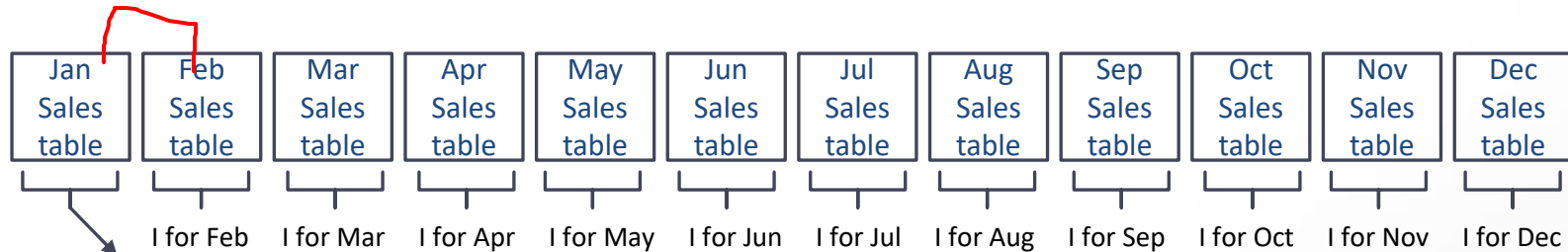
- Predicates and their usage:

```
WHERE date = '20210115' OK  
WHERE date BETWEEN '20210105'  
      AND '20210115' OK  
WHERE date BETWEEN '20210115'  
      AND '20210215' Cannot be used
```



Filtered Index for January

```
CREATE INDEX JanuaryItems
ON Sales (col1, col2)
WHERE SalesDate >= '20210101'
      AND SalesDate < '20210201'
```



Non-filtered index for January

```
CREATE INDEX JanuaryItems
ON Sales (col1, col2)
```

NOTE: Each table has a constraint

```
ALTER TABLE JanSalesTable
ADD CONSTRAINT JanuaryItems
CHECK (SalesDate >= '20210101'
      AND SalesDate < '20210201')
```

# Partition-level Lock Escalation

- Available in SQL Server 2008+
- Escalation setting is per-table, set with ALTER TABLE:  

```
ALTER TABLE [mytable]  
SET (LOCK_ESCALATION = {AUTO | TABLE | DISABLE} )
```

  - AUTO: Partition-level escalation if the table is partitioned
  - TABLE: Always table-level escalation
  - DISABLE: Don't escalate unless absolutely necessary
- How to tell what setting a table already has?  

```
SELECT [st].[lock_escalation_desc]  
FROM [sys].[tables] AS [st]  
WHERE [st].[name] = 'mytablename'
```
- With partitioned views, lock escalation is reduced by having separated the tables...

# Review

- **Special considerations for partitioned tables**
- **Online operations, indexes, and partitioning**
- **Incremental statistics updates**
- **Partition-aligned index views**
- **Partitioned table with filtered indexes**
- **Interval subsumption**
  - Partitioning / fast-switching
- **Partition-level lock escalation**



# Questions!



## Module 7: Partitioning Techniques Combined



# Overview

- **Partitioning: partitioned views vs. partitioned tables**
- **Damaged “partitions” on Enterprise Edition**
- **Partition-level features and frustrations**
- **Other features and partitioning**
- **Partitioning for performance**

# 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 before SQL 2016 SP1
- 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 *FS?*
  - 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
    - “Incremental stats” do NOT fix this!
- Prior to SQL Server 2014, partition-level rebuilds are offline or you have to rebuild the ENTIRE table online (not desirable)
- Can only support partitioning over a single column

# Damaged “Partitions” on Enterprise Edition

## Interesting Partitioning Management Note

- Damaged files do NOT render the database unavailable; only the damaged filegroup is unavailable
- Damaged files do NOT render the partitioned table unavailable as only the damaged data is unavailable but the partitioned table can still be queried/accessed
- Damaged files DO cause problems for accessing partitioned views
  - NOTE: A partitioned view that has tables that are inaccessible generates an error when accessing the view; however, the tables on which the view are defined are still accessible. A partitioned table does NOT.
  - TIP: What you need to do with PVs is modify the view on EVERY restore (to remove / add tables to the view based on what's inaccessible / accessible after each restore).
- Partial database availability and online piecemeal restore are fantastic EE features:
  - There is one minor hiccup here...
  - Really minimal impact to downtime as files are brought offline
  -  Users can access the database during online restore (on Enterprise Edition only)

# Partition-Level Features and Frustrations

- **Row-store compression**

- FOR PTs: Can be set at the partition-level
  - Page for older / less-often used data
  - Row for more recent data that's still being accessed often
  - No compression for critical / hot data
- FOR PVs: Can do whatever you want per table or, if PTs under the PVs then see above

- **Columnstore compression**

- All columnstore indexes are highly compressed by default
- In SQL Server 2014+, an even higher level of compression (COLUMNSTORE\_ARCHIVE) can be used for older / less-often used data

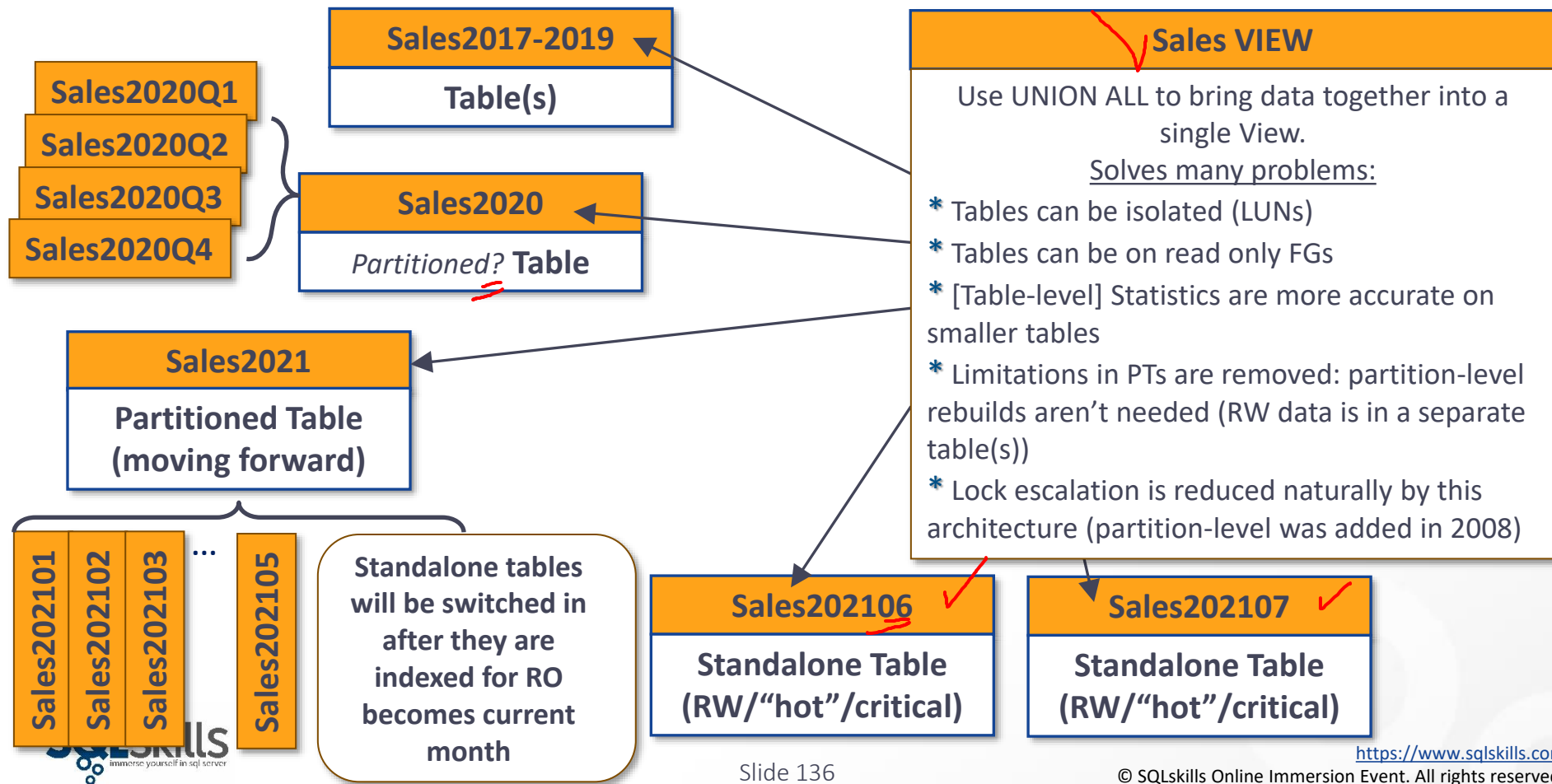
- **FILLFACTOR / PAD\_INDEX cannot be set at the partition-level**

- Having separate tables (PVs) allows you to set these for RO vs. RW and for different patterns in RW (read-mostly might have a higher FF than critical/hot OLTP)

# Other Features and Partitioning

- ~~iFTS~~
  - PTs: cannot do fast-switching if PT has iFTS index(es) – all releases
- ~~Indexed views~~
  - ~~2005~~ PTs: cannot do fast-switching if PT has IVs – must drop, switch, recreate (nightmare! fixed in 2008)
- **Identity columns**
  - PVs: cannot use UPVs for INSERT if they have an identity column on base tables (must use application directed INSERTs)
  - PTs: can have an identity
- **Multiple partitioning columns?**
  - PTs: no, only one
  - PVs: yes, as many as makes sense!
    - For the PT column (the leading column of the PK), you can get an UPV so you can do partition elimination for selects, updates, deletes, and possibly inserts (depending on things like identity)
    - For secondary columns with constraints can do query elimination only

# Functionally Partitioning Data





# Partitioning for Performance

- **Separate read-only into a partitioned table or even multiple partitioned tables (eg. by year, re: probably better stats depending on data distribution)**
- **Keep read-write as standalone table or additional partitioned table**
- **For queries, use a partitioned view to UNION ALL of the tables together**
- **For DML:**
  - Use an updateable partitioned view for selects / updates / deletes
  - Use application directed inserts (programmatically determining date)
  - Consider using a synonym for the “current month” to simplify direct base table access  
HOWEVER, this can be problematic around the timing of the change (better to use DSE)
- **Other benefits:**
  - Partitioned tables are easier for the optimizer to optimize (same indexes)
  - Partitioned tables are easier to manage wrt creating / dropping indexes
  - Partitioned views allow differences in the data to be treated as such:
    - Can index read-only one way (w/ more indexes), read-write another (w/ fewer indexes)!
    - Statistics are separated between drastically different data (volatile v. static, current v. historical, time periods where certain differences occur naturally in the data)
    - Lock escalation is solved architecturally

# Review

- **Partitioning: partitioned views vs. partitioned tables**
- **Damaged “partitions” on Enterprise Edition**
- **Partition-level features and frustrations**
- **Other features and partitioning**
- **Partitioning for performance**

# Questions!



## Module 8: Review



# What to do next?

- **Lots of content...**
  - Reviewing sooner rather than later will help!
  - Using a mix of mediums will help you see things in different ways and likely for some of you one will be better than another – multiple will be the way to make it stick!
- **Don't go straight into designing a partitioning scenario – lots of factors to consider**
  - Criticality of the data
  - Access patterns
  - Does data eventually become obsolete (some companies “never delete”)
  - OLTP and separate DSS or just queries in OLTP?
    - OLTP with queries
      - Partitioning is key here
      - Also consider versioning!
    - Separate DSS
      - Pattern / frequency for load
      - Data deletion from OLTP?

# Self-paced Study and Exercises (1 of 3)

- Read the SQL Server 2005 whitepaper on Partitioning <http://bit.ly/1qID49k>
- Read the SQL Server 2008 whitepaper on Partitioning <http://bit.ly/1A01UFq>
- Review the course content
  - Review the course materials (and don't forget the hidden slides)
  - Review the recording
  - Review the whiteboard drawings (don't forget the annotations)
  - Review the demo scripts
    - Go through these with the recording
    - Play with them and try out some of the extra things we discussed or anything that you're wondering about
- Go through the HOL lab from your zip

# Self-paced Study and Exercises (2 of 3)

## ■ Videos for VLTs and Partitioning

- **Optionally:** Video - 01 Partitioning (MCM Readiness)
  - Lots of overlap but you might hear something differently
- **Definitely:** Video - 02 Partial Database Availability and Online Piecemeal Restore (MCM Readiness)
  - This is great for recovery scenarios and online piecemeal restores
  - You have the scripts to play with as well in the ILL project
- **Definitely:** Video - 03 Isolated Disasters in VLDBs (PASS 2011)
  - Human errors and dealing with isolated disasters (some overlap with video 02)
- **Definitely:** Video - 04 Skewed Data, Poor Cardinality Estimates, and Plans Gone Bad (PASS 2013)
  - VLTs and filtered statistics solutions (a lot of helpful code for automation)

**NOTE: Many PASS Videos are now here: PASStv – YouTube**  
**(<https://www.youtube.com/channel/UCCN1vyLawxrXAIiQoi3INow>)**

# Self-paced Study and Exercises (3 of 3)

## ■ Videos for Indexing

- Video - 05 SQL Server Indexing for Performance (PASS 2015)
  - Good overview of indexing, versions, and some tips/tricks (cool demo)
- Video - 05a SQL Server Indexing for Performance (Pluralsight)
  - 7+ hours on index internals and indexing strategies

## ■ Statement Execution and the Plan Cache

- Pluralsight: SQL Server: Optimizing Ad Hoc Statement Performance
  - <https://www.pluralsight.com/courses/sqlserver-optimizing-adhoc-statement-performance>

## ■ Stored Procedure Performance

- Start with the blog post: BLOG Building High Performance Stored Procedures
- Video - 06 Building High Performance Stored Procedures (PASS 2014)
- Pluralsight: SQL Server: Optimizing Stored Procedure Performance
  - <https://www.pluralsight.com/courses/sqlserver-optimizing-stored-procedure-performance>
- Pluralsight: SQL Server: Optimizing Stored Procedure Performance - Part 2
  - <https://www.pluralsight.com/courses/sqlserver-optimizing-stored-procedure-performance-part2>



# THANK YOU!

