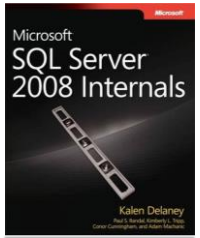
# SQLintersection

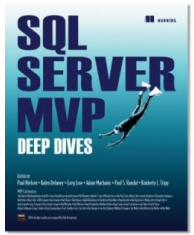
## SQL Server Indexing Strategies

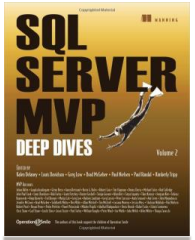
Kimberly L. Tripp
President / Founder, SQLskills.com
Kimberly@SQLskills.com
@KimberlyLTripp

SQL *intersection*

# Author/Instructor: Kimberly L. Tripp




- Consultant/Trainer/Speaker/Writer
- President/Founder, SYSolutions, Inc.
  - e-mail: Kimberly@SQLskills.com
  - blog: http://www.SQLskills.com/blogs/Kimberly
  - Twitter: @KimberlyLTripp
- Author/Instructor for SQL Server Immersion Events
- Instructor for multiple rotations of both the SQL MCM & Sharepoint MCM
- Author/Manager of SQL Server 2005 & 2008 Launch Content
- Author/Speaker at Microsoft TechEd, SQLPASS, ITForum, TechDays, SQLIntersection
- Author of several SQL Server Whitepapers on MSDN/TechNet: Partitioning, Snapshot Isolation, Manageability, SQLCLR for DBAs
- Author/Presenter for more than 25 online webcasts on MSDN and TechNet
- Author/Presenter for multiple online courses at Pluralsight
- Co-author MSPress Title: SQL Server 2008 Internals, the SQL Server MVP Project (1 & 2), and SQL Server 2000 High Availability
- Owner and Technical Content Manager of the SQLintersection conference

# Session Overview

- **SQL Server version**
- **Workload characteristics**
- **Index structures**
- **Base table structures**
  - Clustered row-based index v. clustered column-based index
  - What criteria should you look for in data access patterns and usage patterns?
- **What makes a good clustered key?**
- **Migration Strategies**
- **Indexing Key Points**

# Biggest Concern is SQL Server Version

- **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 limitations but still amazing performance when possible and/or workarounds used (see this WIKI for SQL Server 2012 workarounds: http://bit.ly/1eHVW00)
- **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 on clustered columnstore indexes!**

# SQL Server 2008 / R2 and SQL Server 2012

- **If you're not on SQL Server 2012 – your options are limited to row-based indexes**
  - Skip 2012 on your migration path and go directly to 2014 or 2016*
- **If you're on SQL Server 2012 AND you have DSS / RDW workloads with partitioning (partitioned views or partitioned tables):**
  - 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
    - If you find you're not getting batch mode processing
      - Check out Eric Hanson's WIKI and workarounds: http://bit.ly/ZP63Lr
      - Consider upgrading to SQL Server 2014 (lots of batch mode fixes)

** Backup/restore is not DIRECTLY supported from SS2008x to SS2016. Restore SS2008x backup to SS2012 or SS2014 Eval Edition. Then, backup from that and restore to SS2016.*

# What Type of Workload is Running?

- **OLTP – Online transaction processing**
  - Priority is toward modifications
  - Lots of point queries (highly-selective queries of a small number of rows)

    Search for a sale, search for a customer, lookup a product, …
- **Dedicated Decision Support System / Relational Data Warehouse**
  - Priority is toward large-scale aggregates
  - Very large percentage or even the entire dataset – is being evaluated often
- **Hybrid**
  - OLTP might be the priority and some point query activity
  - Some range-based queries because "management" wants real-time analysis

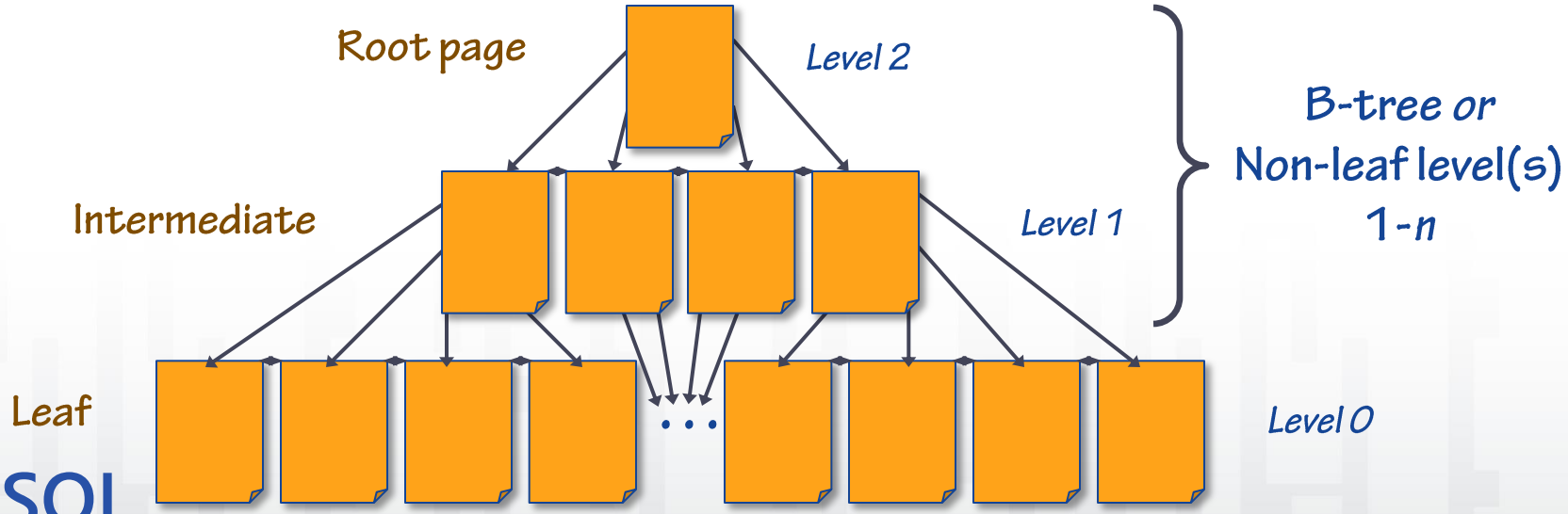
*(NOTE: We're talking indexes in this session but in the hybrid environment you should really be running with versioning enabled! Check out the data option: read_committed_snapshot. For more info, see resources. However, **clustered** columnstore indexes do not work yet in versioned databases until SQL Server 2016.)*

# 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
  - 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
- **Hybrid**
  - Most likely to use traditional row-based clustered and nonclustered indexes and possibly nonclustered columnstore indexes if you've partitioned your data in the hybrid scenario

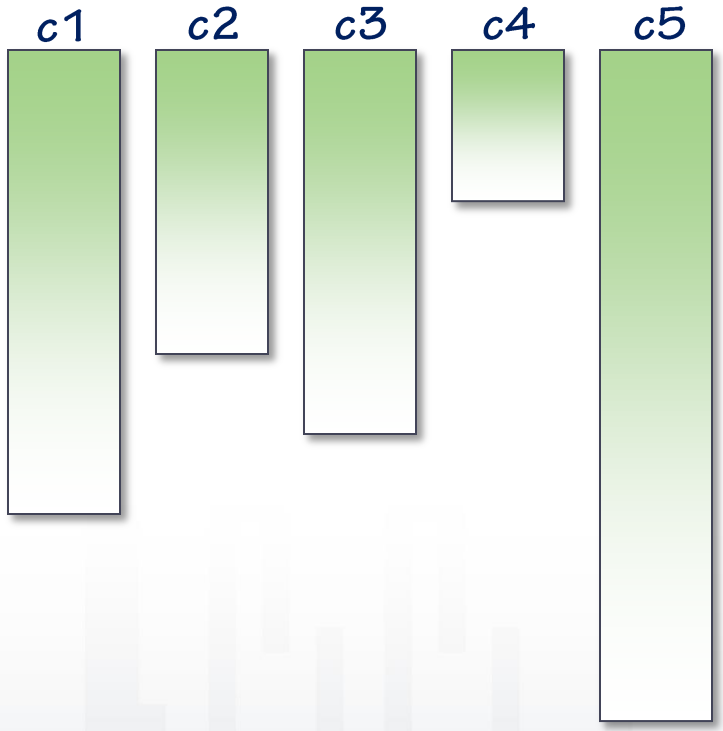
# Index Structures: Row-based Indexes

- **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

- Column-based
- Only 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!



# 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 and then consolidate
  - Must store the data (not as easily compressed)

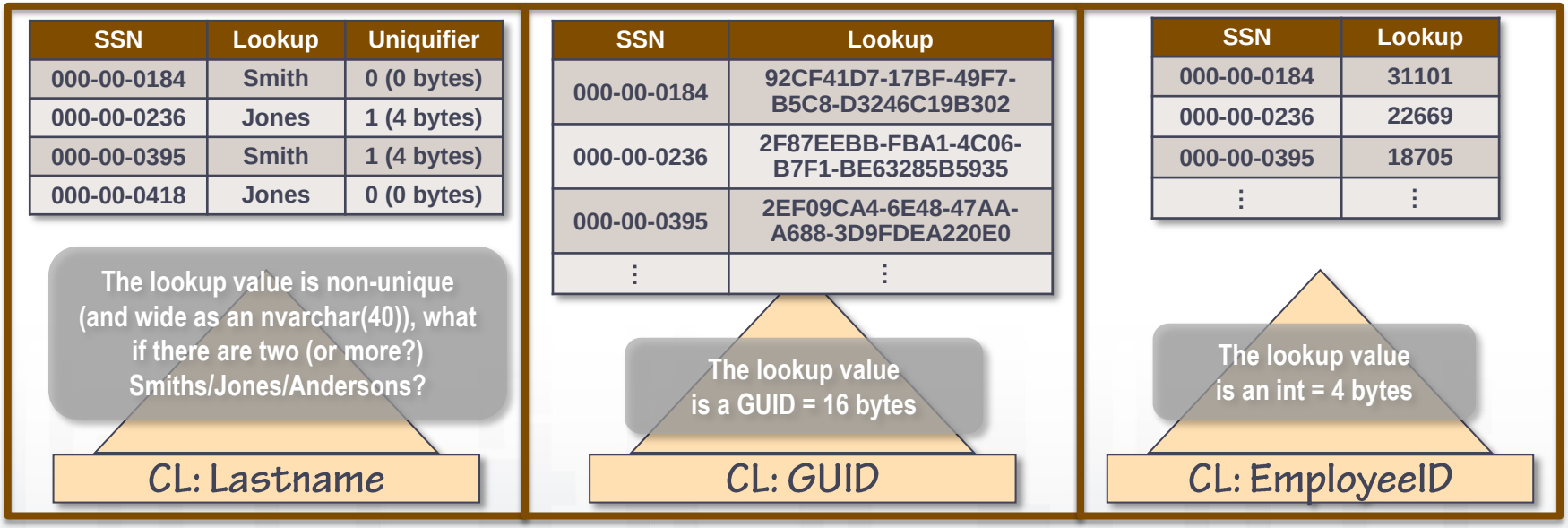
- **Column-based indexes**
  - Significantly better compression
  - COLUMNSTORE / COLUMNSTORE_ARCHIVE
- **Supports large scale aggregations**
- **Support partial scans w/"segment" elimination**
  - Only the needed columns are scanned
  - Data is broken down into row groups (roughly 1M rows) and segments 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 (fixes in 2014 and 2016)
  - Limitations with other features (less and less by SQL Server version)

# The Clustering Key

- **For columnstore indexes there's no clustering key defined; there's no "order" to the data**
  - 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
  - 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!

# How is the Clustering Key Used in Nonclustered Indexes?

**Imagine the internals of a nonclustered index on SocialSecurityNumber with 3 different versions of the Employee table – each with different clustering keys**



# Lookups

## Nonclustered Indexes are Wider!

- **Imagine these costs in a real world scenario…**
  - 10 million rows, 8 nonclustered indexes
- **What's the overhead required (and total space) for the bookmark lookups in the nonclustered indexes:**

| Simple calculations for overhead in the LEAF level of the nonclustered indexes based on CL key | | | | |
|---|---|---|---|---|
| Description | Width of CL key | Rows | NC Indexes | MB |
| int | 4 | 10,000,000 | 8 | 305.18 |
| datetime | 8 | 10,000,000 | 8 | 610.35 |
| datetime, int | 12 | 10,000,000 | 8 | 915.53 |
| guid | 16 | 10,000,000 | 8 | 1,220.70 |
| composite | 32 | 10,000,000 | 8 | 2,441.41 |
| composite | 64 | 10,000,000 | 8 | 4,882.81 |

(NOTE: This is just the overhead of the data type without factoring in nullable/non-unique.)

# Scenario: What is the Real Cost?

**AdventureWorksDW: FactInternetSales**

- **Clustered index:** — **Data type:**
  - SalesOrderNumber — nvarchar(20)
  - SalesOrderLineNumber — tinyint
- **Nonclustered indexes:**
  - IX_FactIneternetSales_ShipDateKey: ShipDateKey
  - IX_FactInternetSales_CurrencyKey: CurrencyKey
  - IX_FactInternetSales_CustomerKey: CustomerKey
  - IX_FactInternetSales_DueDateKey: DueDateKey
  - IX_FactInternetSales_OrderDateKey: OrderDateKey
  - IX_FactInternetSales_ProductKey: ProductKey
  - IX_FactInternetSales_PromotionKey: PromotionKey

# Demo

**AdventureWorks**

The impact of key choice on nonclustered indexes

SQL *intersection*

# Nonclustered Indexes: Key to Better Performance

- **In a row-based indexing strategy performance hinges on your choice of nonclustered indexes:**
  - Indexing strategies are extremely challenging
    - Users lie ☺
    - Workload specific
      - Data modifications are impacted by indexes (indexes add overhead to INSERTs / UPDATEs / DELETEs)
      - The type and frequency of the queries needs to be considered
        - This can change over time
        - This can change over the course of the business cycle
  - To do a good job at tuning (for rowstore indexes) you must:
    - Know your data
    - Know your workload
    - Know how SQL Server works!
      - Rowstore indexes are more query-centric; need to have an understanding of how SQL Server works in order to create the "RIGHT" indexes, you CANNOT just guess!

# Migration Strategy (1 of 2)

- **Columnstore indexes are improving significantly from version to version**
  - Many restrictions in SQL Server 2012
  - SQL Server 2014 is a better choice if you're planning a NEW architecture
  - Ideally, prototyping with SQL Server 2014 to go live with SQL Server 2016
- **SQL Server 2014's clustered columnstore indexes DRASTICALLY reduce the overall size of the table (and IOs) BUT no other indexes are allowed (YET…)**
  - If you do a lot of point queries then clustered columnstore might not be ideal (you'll really need to test to be sure because IOs are so small even scanning a "row group" for a row might not be too bad)
  - SQL Server 2016 will offer many additional choices
    - Updateable, clustered columnstore index with row-based nonclustered indexes
    - Row-based clustered index with an updateable, nonclustered, columnstore index

# Migration Strategy (2 of 2)

- **If not clustered columnstore then you might go with a row-based clustered index and then a read-only, nonclustered, columnstore index for partitioned objects**
  - You'll need to deal with the row-format and storage concerns – consider compression for older data (this is still nowhere near as efficient as columnstore or columnstore_archive)
  - Partitioned tables – fast-switching is supported for read-only columnstore indexes
  - Partitioned views – the performance enhancements around batch-mode processing (UNION ALL) weren't supported in SQL Server 2012 but are now in SQL Server 2014
  - SQL Server 2016 is going to take that even further

# Indexing Key Points

- **Long term scalability doesn't happen by accident**
- **SQL Server has a tremendous number of indexing options available but they all have trade-offs**
- **Prototyping and doing some early analysis is critical to getting it right**
  - Can learn where combinations of features do or don't work well together
  - Can see the disk space requirements and do estimates to scale
- **The sooner you begin the code, the longer it's going to take!**

# Session Review

- **SQL Server version**
- **Workload characteristics**
- **Index structures**
- **Base table structures**
  - Clustered row-based index v. clustered column-based index
  - What criteria should you look for in data access patterns and usage patterns?
- **What makes a good clustered key?**
- **Migration Strategies**
- **Indexing Key Points**

# Resources

pluralsight hardcore developer training

- **Demo code/samples: SQLskills, Resources, Demo Scripts and Sample Databases**
- **Courses on Pluralsight: www.pluralsight.com**
  - SQL Server: Why Physical Database Design Matters
  - SQL Server: Optimizing Ad Hoc Statement Performance
  - SQL Server: Optimizing Stored Procedure Performance (Parts 1 and 2)
    - Part 2 has an entire section on session settings (for performance-related features)
  - If you want to know more about columnstore indexes then check out Joe Sack's Pluralsight course on SQL Server 2012's read-only, nonclustered columnstore indexes:
    - SQL Server 2012: Nonclustered Columnstore Indexes (http://bit.ly/1PYVU2a)

# Resources

- **Paul's index "fanout" blog post: On index key size, index depth, and performance**
  - http://www.sqlskills.com/blogs/paul/on-index-key-size-index-depth-and-performance/
- **Additional columnstore resources:**
  - ColumnStore Index: Microsoft SQL Server 2014 and Beyond
    - https://channel9.msdn.com/Events/Ignite/2015/BRK4556
  - SQL Server 2014: Security, Optimizer, and Columnstore Index Enhancements
    - http://www.microsoftvirtualacademy.com/training-courses/sql-server-2014-security-optimizer-and-columnstore-index-enhancements?prid=ch9courselink
- **BI Foundations Sessions from PASStv**
  - http://www.sqlpass.org/summit/2015/PASStv/Microsoft.aspx

# Questions?

SQLskills
immerse yourself in sql server

**Don’t forget to complete an online evaluation on EventsXD!**

## SQL Server Indexing Strategies

*Session by* Kimberly L. Tripp

**Your evaluation helps organizers build better conferences and helps speakers improve their sessions.**

SQL *intersection*

# Thank you!