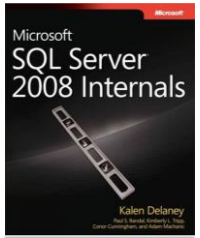
SQLintersection

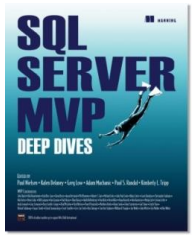
Indexing for Performance: Row-based Indexes

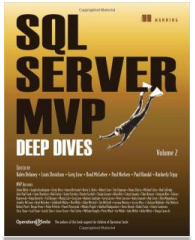
Kimberly L. Tripp
President / Founder, SQLskills.com
Kimberly@SQLskills.com
@KimberlyLTripp

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

- **Indexing for Performance**
  - Indexing strategies overall
  - Design goals
  - The clustering key
- **Plan of attack**
  - Unused indexes
  - Healthy indexes
  - Additional indexes
    - Understanding the tools
    - Using the tools to help tune critical queries
- **Too many cooks in the kitchen**
- **Consolidation**

Row-based Indexes v. Column-based Indexes

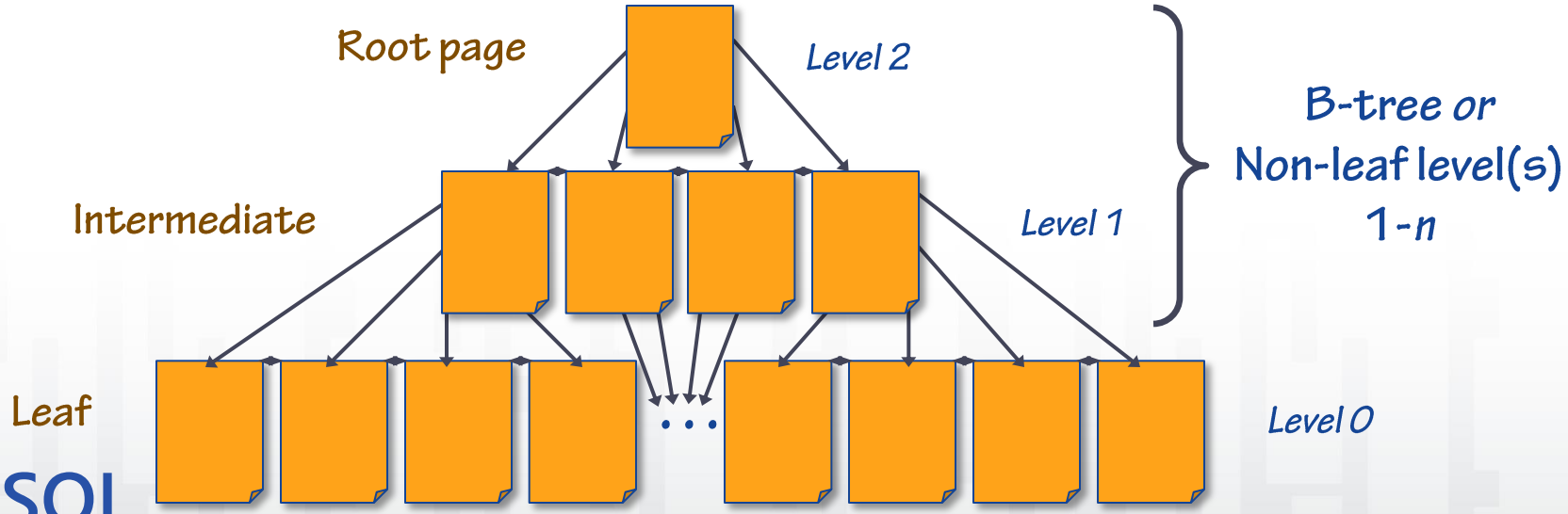
- **Supports data compression**
  - Row compressed
  - Page compressed
- **Can support point queries / seeks**
- **Wide variety of supported scans**
  - Full / partial table scans (CL)
  - Nonclustered covering scans (NC)
  - Nonclustered covering seeks with partial scans (NC)
- **Biggest problems**
  - More tuning work for analysis: must create appropriate indexes per query and then consolidate
  - Must store the data (not as easily compressed)

This is covered in the final indexing session today after lunch

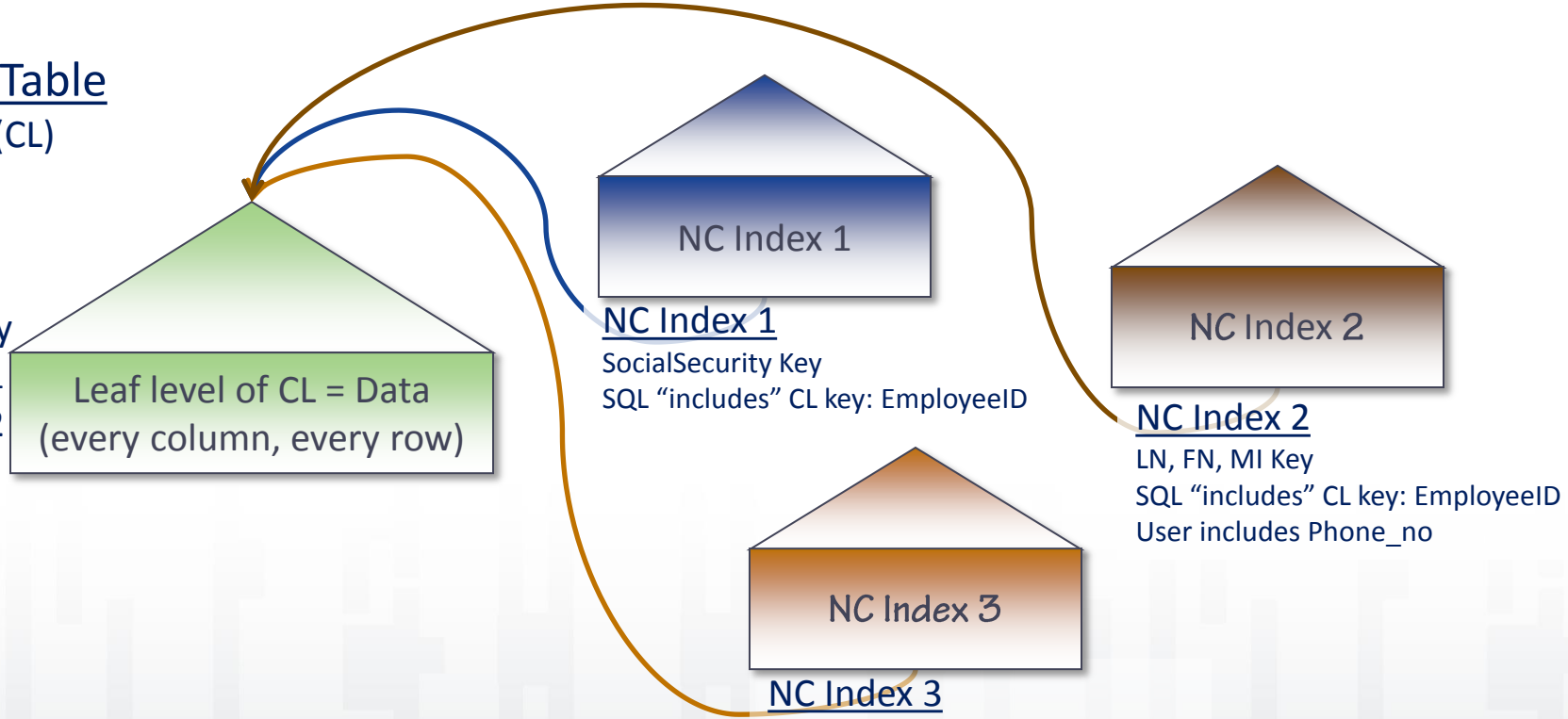
This room 2:15-3:15

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.



More Redundancy: Pros / Cons (1 of 2)



More Redundancy: Pros / Cons (2 of 2)

- **Nonclustered index is challenging**
- **Each index is initially added for specific query gains**
  - Requires tuning experience / knowledge
  - Additional indexes are needed for different (but, often similar) queries
- **Each index adds overhead:**
  - Disk space
  - Every INSERT / DELETE needs to modify the index
  - UPDATE is affected only if a column within the index is modified
  - Maintenance
- **MUST use consolidation to avoid added redundancy / overhead**
- **Use patterns change over time; indexing strategies must as well**

Clustered Index Criteria

Keeping our Clustering Key as Streamlined as Possible!

- **Unique**
  - Yes: No extra time/space overhead, data takes care of this criteria
  - NO: SQL Server must “uniquify” the rows on INSERT
- **Static**
  - Yes: Reduces overhead
  - NO: Costly to maintain during updates to the key
- **Narrow**
  - Yes: Keeps the NC indexes narrow
  - NO: Unnecessarily wastes space
- **Non-nullable/fixed-width**
  - Yes: Reduces overhead
  - NO: Adds overhead to ALL nonclustered indexes
- **Ever-increasing**
  - Yes: Reduces fragmentation
  - NO: Inserts/updates might cause splits (significant fragmentation)

Clustering on an Identity

- **Naturally unique**
  - (should be combined with constraint to enforce uniqueness)
- **Naturally static**
  - (should be enforced through permissions and/or trigger)
- **Naturally narrow**
  - (only numeric values possible, whole numbers with scale = 0)
- **Naturally non-nullable/fixed-width**
  - (an identity column cannot allow nulls, a numeric is fixed-width)
- **Naturally ever-increasing**
  - Creates a beneficial hot spot…
  - Needed pages for INSERT already in cache
  - Minimizes cache requirements
  - Helps reduce fragmentation due to INSERTs
  - Helps improve availability by naturally needing less defrag

Clustering Key Suggestions

- **Identity column**
  - Adding this column and clustering on it can be extremely beneficial – even when you don't "use" this data
- **DateCol, identity**
  - In that order and as a composite key (not date alone as that would need to be "uniquified")
  - Great for partitioned tables
  - Great for ever increasing tables where you have a lot of date-related queries
- **GUID**
  - NO: if populated by client-side call to .NET client to generate the GUID. OK as the primary key but not as the clustering key
  - NO: if populated by server-side NEWID() function. OK as the primary key but not as the clustering key
  - Maybe: if populated by the server-side NEWSEQUENTIALID() function as it creates a more sequential pattern (and therefore less fragmentation)
    - But, this isn't really why you chose to use a GUID…
- **Key points: unique, static, as narrow as possible, and less prone to require maintenance – by design**

Indexing Strategies at Design and Beyond

- First and foremost: choose a GOOD clustering key
- Create your primary keys and unique keys
- Create your foreign keys
  - Manually index your foreign keys with nonclustered indexes
- Create any nonclustered indexes needed to help with highly selective queries (lookups are OK for highly selective queries)
- **STOP:** this is your "design" base
- Add indexes slowly and iteratively while evaluating your workload characteristics as well as you know them during testing / QA
- Continue to fine tune them in production based on query priorities and always re-evaluate if / when things change!

Plan of Attack

1. **Index cleanup**
   - Get rid of the dead weight
   - Remove duplicate indexes
   - Remove unused indexes
2. **Index health**
   - Determine health of existing (and useful) indexes
3. **Missing indexes**
   - Slowly and iteratively – add missing indexes
   - Database Tuning Advisor
   - Performance Data Collection

(1) Remove Unused Indexes

- **DMVs Don't Tell You Everything…**
- **Removing redundant / duplicate indexes**
- **Duplicate indexes can still show as used**
- **MUST review your indexes manually**
  - Don't forget INCLUDEd columns (in 2005) or filtered indexes (in 2008)
  - sp_helpindex doesn't show these columns
  - Use my updated version of sp_helpindex (blog category: sp_helpindex rewrites) to get better information and determine if one index really is redundant/duplicate
  - Blog post: Removing duplicate indexes (http://bit.ly/rusI9U)
- **Don't rely on sys.dm_db_index_usage_stats alone as it lies**

(1) Remove Unused Indexes

hidden slide w/extra details

DMVs Don't Tell You Everything…

- **In addition to completely duplicate indexes, must prune out the redundant indexes…**
- **Pruning existing indexes is not quite as simple as removing all left-based subsets:**
  - It's true that a query using an index on LastName alone COULD use an index on LastName, Firstname OR an index on LastName, Firstname, MI but, always be careful of how much larger the indexes are and the type of queries using them
  - Should you drop indexes that are left-based subsets of others?
    - Typically yes BUT, consider the width of the additional columns
    - If the additional column(s) are relatively wide and only needed for a couple of queries whereas the narrower version is used by a lot of queries, you might want both!

(1) Remove Unused Indexes

hidden slide w/extra details

What Do the DMVs Tell You?

- **Verify index usage with sys.dm_db_index_usage_stats**
- **Tracks the following and the date / time of their last occurrence:**

  Seeks Scans Lookups Updates
- **The cache is flushed at shutdown as well as when**
  - The object is rebuilt (not reorged) in SQL Server 2012 (check EACH SP)
    - Joe Sack blog post on this: http://bit.ly/JdunGW
  - Entries for all indexes in a database are removed when the database is closed (via AUTOCLOSE (if enabled)), taken offline, or detached
  - There's no way to manually flush the cache
- **Persist this data and analyze over business cycle**

(1) Dropping an Index

hidden slide w/extra details

What Could Go Wrong?

- **Queries that use index hints will ERROR if the index no longer exists**
  - This is the reason for why SQL Server allows duplicate indexes to be created in general…
- **Plans guide might no longer work**
  - They're just invalidated, a new plan will be used
- **Plans could change**
  - This could be good… or bad?!

(2) Verify Health of Existing (and Useful) Indexes…

- **Fragmentation can mean a lot of things (completely overloaded term) and not an entirely simple thing to address…why? It depends on many factors…**
  - Table size and usage patterns
  - Impact to availability – can you use an online operation?
    - ALTER INDEX…REBUILD…WITH (ONLINE = ON)
      - Not if the index has a LOB column in it (fixed in 2012)
      - Not if you try to rebuild only a single partition (fixed in 2014)
    - ALTER INDEX…REORGANIZE (always online)
  - Reorganizing always uses 'full' logging but the amount of log information generated will depend on how much fragmentation exists
  - There are trade-offs between rebuilding and reorganizing in terms of log space, disk space, run-time, impact to tempdb and even benefit…
  - Resources
    - Whitepaper: Microsoft SQL Server 2000 Index Defragmentation Best Practices (concepts)
    - Whitepaper: Online Indexing Operations in SQL Server 2005
    - Paul's Pluralsight course: SQL Server: Index Fragmentation Internals, Analysis, and Solutions

(3) Are You Missing Any Indexes?

- **Have you tried other options?**
- **Ask DTA what it thinks?**
- **Ask SQL Server what it thinks?**
  - SQL Server 2005+
    - DMV queries
      - Critical / frequently executed procedures (dm_exec_procedure_stats)
      - Missing index DMVs
    - Other resources/blogs/sites…
  - SQL Server 2008+
    - [Performance] Data Collector
    - DMVs add analysis for ad hoc queries using dm_exec_query_stats with query_hash and query_plan_hash)

The “Cumulative Effect” of Queries

- **SQL Server 2008 added query_hash and query_plan_hash to sys.dm_exec_query_stats**
  - Aggregate by query_hash to find similar queries
  - Aggregate by query_hash, query_plan_hash to find similar queries along with their plans (possibly more than one per query_hash which is why the statement wasn’t safe)
- **SQL templatizes the parameters – similar to sp_get_query_template**
- **See the queries in the demo scripts…**
- **Check out BOL: Finding and Tuning Similar Queries by Using Query and Query Plan Hashes**
  - http://bit.ly/QfUmRY

(3) Are You Missing Any Indexes?

Ask SQL Server What it Thinks…

- **sys.dm_db_missing_index_group_stats (probably the most detailed):**
  - user_seeks, user_scans
  - last_user_seek and last_user_scan are both datetime
  - avg_total_user_cost: higher costs give relative numbers to determine which are more "costly" to the system
  - avg_user_impact: the improvement (in terms of percent that the user should see from the index addition)
- **sys.dm_db_missing_index_groups**
  - Many to many relationship table to tie together the index details and the usage stats (index group stats is effectively the index usage stats for these needed indexes)
- **sys.dm_db_missing_index_details**
  - Details the table, key columns and included columns that you should consider for these indexes…

SQL intersection

NOTE: The important part isn't the query,
it's what you do with the results (evaluation, testing, consolidation)

(3) Asking the Tools

- **USE the tools!**
  - STATISTICS IO
  - Showplan's "green hint" / missing index DMVs
  - Database [Engine] Tuning Advisor
- **BEWARE of the limitations of the tools!**
  - Missing index DMVs (and therefore showplan) only tune the plan that was executed; they do not "hypothesize" about alternatives (like DTA does)
  - All of the index recommendation from tools tend to go for "the best" choice rather than good enough choices
  - NONE of the tools do index consolidation…
- **Resources:**
  - Search "Bart Duncan Missing"
  - Find Missing Indexes on SQLServerPedia
  - A bit of searching as lots of good stuff out there

Using the Tools for Join Tuning

- **Understand how to break down a join**
- **Understand how to force your join for performance comparisons**
- **Understand the pros/cons of the showplan recommendations**
- **Understand how to best use DTA for a more well-rounded recommendation**
  - Use DTA from SSMS to see all of the recommendations
  - Know how to use DTA's recommendations iteratively!

Demo

Tuning Joins Using Nonclustered Indexes

Using the tools to help simplify this process

(IMPORTANT: simplify <> simple)

SQL *intersection*

Indexing for Joins

hidden slide w/extra details

- **Multiple possible join strategies: do you *need* to care?**
- **Items on which to focus:**
  - Most expensive table in the join (you have to start somewhere?!)
  - Most expensive join in the plan (it's probably downstream from the most expensive table and a join on that table)
  - Once you know the problem table AND the problem join, focus on tuning that particular table within that specific join!

Best Options for Joins: Phase I

hidden slide w/extra details

Do you already have individual indexes on each and all of these columns?

Foreign key???

- **One join strategy might use Table1's SARG1 index to Table2's join key index (loop join)**
- **Another could use Table2's SARG1 index to Table1's join key index (loop join)**
- **Another could use only the join key indexes (merge)**
- **What's best depends on the data!**
- **If ALL four indexes exist then the optimizer has the best choices (for phase I)**

Cover the Combination: Phase II

hidden slide w/extra details

Still not working?

- **Not using these indexes?**
- **Performance still stinks?**
- **Cover the combo**
  - Problem table (SARG, Join): priority for the SARG
  - Problem table (Join, SARG): priority for the join
- **Only works when the cardinality of the join is low**

Cover the Query: Phase III

hidden slide w/extra details

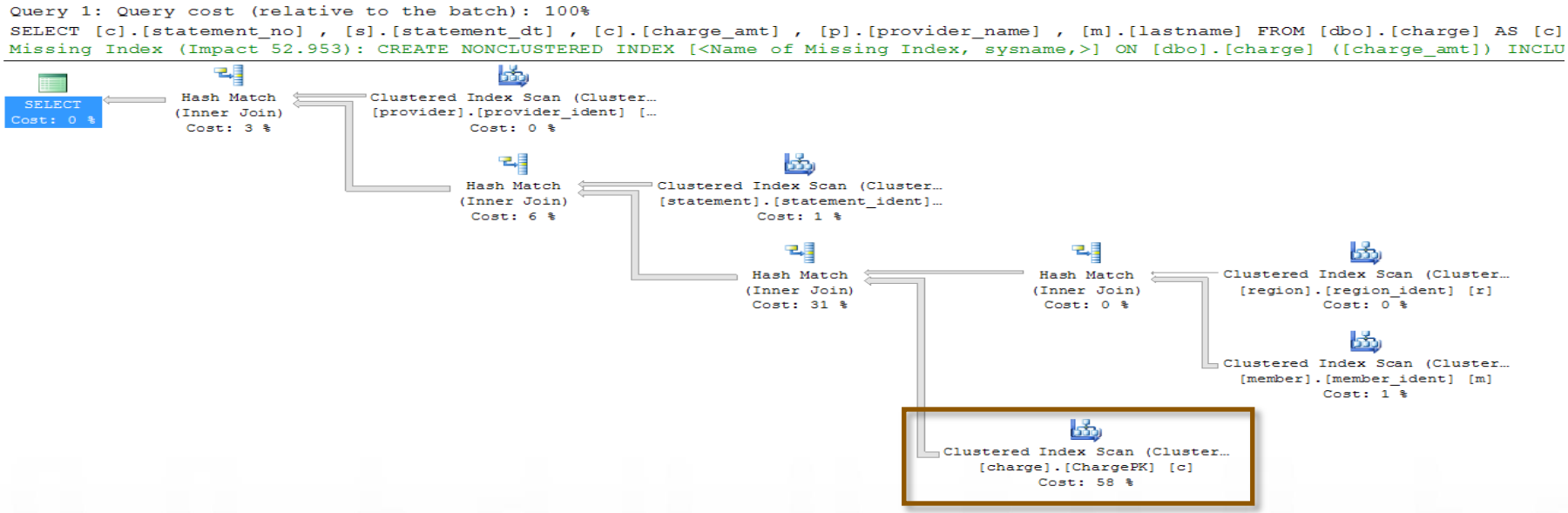
- **Covering the query/queries**
- **Cover the combo first, THEN add the additionally requested columns, with INCLUDE**
  - Priority for the SARG: problem table (SARG, Join) INCLUDE (col*x*, col*y*, …)
  - Priority for the Join: problem table (Join, SARG) INCLUDE (col*x*, col*y*, …)

Bringing It All Together (Long Demo)

- **Pulling apart a plan and describing a lot while I do it…**
- **Hard-coding a query to create a base-line to go against**
- **Deciding where to start**
  - Analyzing where we have a problem(s)
  - Finding the problem table
  - Finding the problem join
- **Evaluating whether or not an index is a good idea**
  - Reviewing/debating the “green hint”
  - Using DTA from SSMS to see if the hint is different

Table Scans – Are They Necessary?

hidden slide w/extra details

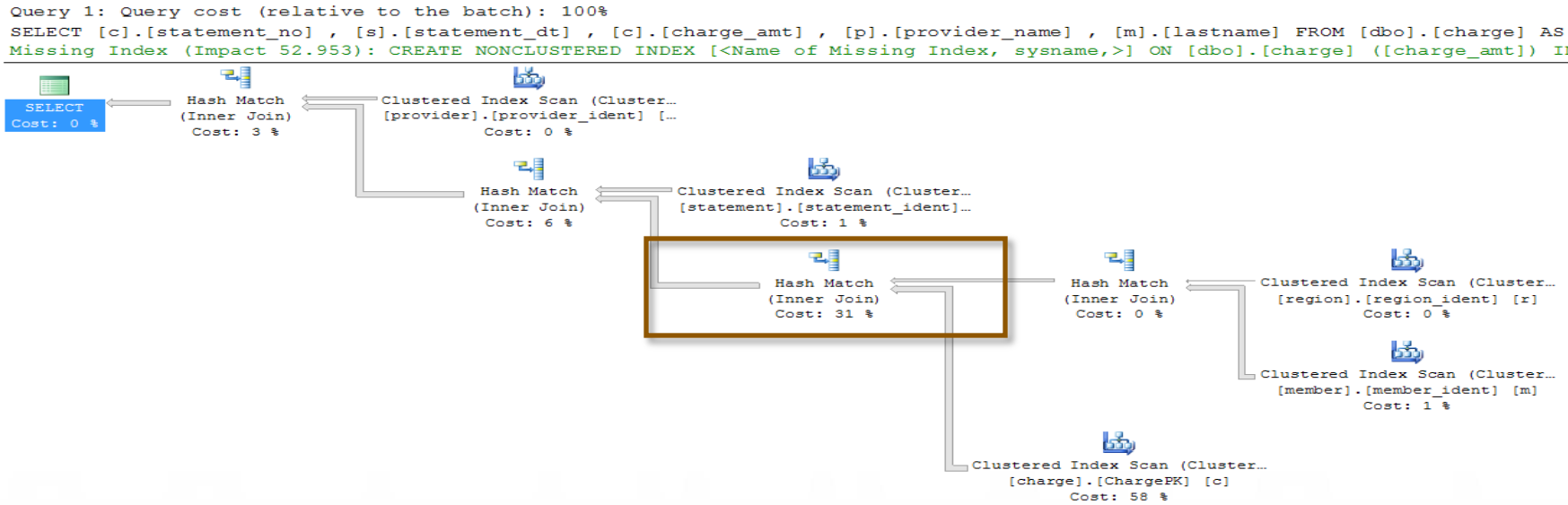


- Is the table scan because you're returning the entire table/all columns?
- Or, it is because the right indexes don't exist?
- What table has the highest cost?

SQL intersection

Joins – Are All of the Joins Hash Joins?

hidden slide w/extra details



- Are all of these hash joins because you're tables are large?
- Or, are they because the right indexes don't exist?
- What join has the highest cost?

SQL intersection

Manual Tuning Process

hidden slide w/extra details

- **Find the most expensive table in the query (charge)**
  - Are there any SARGs – consider what the key would look like with these SARGs independent of the join conditions
- **Find the most expensive join in the query (charge joining to member)**
  - Figure out which join is the join that your problem table is joining to
- **Phase I should be already done**
- **Phase II should be considered**

```
CREATE NONCLUSTERED INDEX Charge_PriorityForSARG
ON [dbo].[charge] ([charge_amt], [member_no])

CREATE NONCLUSTERED INDEX Charge_PriorityForJoin
ON [dbo].[charge] ([member_no], [charge_amt])
```

- **Phase III adds any columns not already present, to the INCLUDE**

```
INCLUDE ([statement_no], [provider_no])
```

Tuning Goal

hidden slide w/extra details

- **Find the most expensive table in the query (charge)**
- **Find the most expensive join in the query (charge joining to member)**
- **Try to tune CHARGE for its join to member… How?**
  - Review the green hint:
    - CREATE NONCLUSTERED INDEX MissingIndexDMVRecommendation
    - ON [dbo].[charge] ([charge_amt])
    - INCLUDE ([member_no],[provider_no],[statement_no])
  - Double-check using DTA (Query, Analyze Query in DTA):
    - CREATE NONCLUSTERED INDEX [DTA_K6_K7_K3_K2]
    - ON [dbo].[charge]
    - ([member_no], [charge_amt], [statement_no], [provider_no])
- **Notice how similar these indexes are?**
- **What do they do, how do they differ?**

Result of Manual and Tool-based Tuning

hidden slide w/extra details

- **The missing index DMVs (via the green hint in showplan) came up with:**

```
CREATE NONCLUSTERED INDEX MissingIndexDMVRecommendation
ON [dbo].[charge] ([charge_amt])
INCLUDE ([member_no],[provider_no],[statement_no])
```

- **Manually, we came up with:**

```
CREATE NONCLUSTERED INDEX Charge_PriorityForSARG
ON [dbo].[charge] ([charge_amt], [member_no])
INCLUDE ([statement_no], [provider_no])
```

```
CREATE NONCLUSTERED INDEX Charge_PriorityForJoin
ON [dbo].[charge] ([member_no], [charge_amt])
INCLUDE ([statement_no], [provider_no])
```

- **The Database Tuning Advisor came up with:**

```
CREATE NONCLUSTERED INDEX [DTA_K6_K7_K3_K2]
ON [dbo].[charge]
( [member_no], [charge_amt], [statement_no], [provider_no])
```

Benefits of These Indexes?

- **The green hint/the Missing Index DMVs recommendation:**
  - Leads with the column charge_amt
  - This removes the table scan and changes to an index seek
  - This allows filtering by our search argument (charge_amt > 2500)
  - PRO: This helps tune the plan that was chosen/executed
  - CON: They did not hypothesize for alternatives
- **The DTA's recommendation:**
  - Leads with the column member_no
  - This removes the table scan and changes to an index seek
  - This allows the join to change to a loop join
  - PRO: This significantly reduces the cost/time for the join
- **Of the tools – what's better? What gives better performance?**
  - DTA

Database [Engine] Tuning Advisor

- **It's not always perfect**
- **It sometimes yields the same index recommendation that the missing index DMVs**
- **It often OVER recommends indexes (this is why you want to use it ITERATIVELY after really analyzing where to begin)**
- **It doesn't recommend any forms of index consolidation**
  - This is one of the reasons that a lot of development environments end up over-indexed
- **IF you end up creating the index that was recommended for a particular table, then, go ahead and create the statistics that are recommended for that table**
  - DTA can create multi-column statistics
  - The can give the optimizer other (sometimes VERY useful) ways of using the recommended index!

Be Careful: Too Many Cooks in the Kitchen!

- **You find a query that needs indexes…**
- **Your colleagues find queries that need indexes…**
- **ITW/DTA find queries that need indexes…**
- **The missing index DMVs find queries that need indexes…**
- **We think these tools/people are helping – and the indexes *definitely* help queries (they're not wrong)**
- **But, you may end up with a lot of similar indexes**
  - (hopefully only similar – and not identical? – indexes)

SQL Server will let you create as many useless indexes as you'd like…

Index Consolidation for *Better* Covering

- **Index structures**
  - Key is used for navigation
    - Must preserve the left-based seeking capability of the key
  - INCLUDE is for covering
    - Order of the columns in the include is irrelevant
- **Imagine the following indexes:**
  - Ind1: (Lastname, Firstname, MiddleInitial)
  - Ind2: (Lastname) INCLUDE (Phone)
  - Ind3: (Lastname, Firstname) INCLUDE (SSN)
- **COMBINE all three:**
  - (Lastname, Firstname, MiddleInitial) INCLUDE (Phone, SSN)
- **Before you create ANY new indexes, review the current indexes!**
- ***This is overlooked/missed all too often!***

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
- **Getting your database developers to create a more solid "base" set of indexes is a great start**
- **Plan on needing some time to fine tune your environment after you see how production is *really* running**
- **Consistently, analyze / tweak / test / repeat…**

Session Review

- **Indexing for Performance**
  - Indexing strategies overall
  - Design goals
  - The clustering key
- **Plan of attack**
  - Unused indexes
  - Healthy indexes
  - Additional indexes
    - Understanding the tools
    - Using the tools to help tune critical queries
- **Too many cooks in the kitchen**
- **Consolidation**

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

XML Resources

- **Good starting point:** XML Indexes in SQL Server 2005 (http://bit.ly/1Lpt5vk)
- **Then, catch up in the books online with new features, etc.**
- **Then, for selective XML indexes (SXI) check out Bob Beauchemin's series of blog posts (www.SQLskills.com/blogs/BobB):**
  - Getting started with Selective XML Indexes in SQL Server
  - Selective XML Index – Implementation Details
  - Selective XML Index – Why is this compelling?
  - Selective XML Indexes – Learning the rules through error messages
  - Selective XML Index – Secondary Selective XML Indices

  and more!

Questions?

SQLskills
immerse yourself in sql server

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

Indexing for Performance: Row-based Indexes

Session by Kimberly L. Tripp

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

SQL *intersection*

Thank you!

Special Index Considerations: Bits and bitmasking

- **What if you have a lot of true / false conditions**
  - Options on a car
  - Options on a home for sale
  - Properties…
- **You have two options for storage:**
  - Separate columns for each
    - Lots of columns to work with
    - Depending on data type – can be efficient (SQL Server combines bits into a byte)
    + Easier to read / understand / index
  - One column for a bitmask
    + Only one column per 15 bits (fewer columns in the table)
    + Always stored as an efficient numeric value (int / bigint)
    - Harder to read / understand / index (with one exception)

Indexing Options: Bits and bitmasking

- **SQL Server 2008+ has filtered indexes (a MUST here IMO)**
- **Bit columns**
  - Index individually for those that need to be searched
    - With regular indexes each indexed bit column will be stored in the leaf level of a nonclustered index
    - Bit columns are always combined to make storage efficient but if only one bit in the index then it will take a byte
    - Filtered indexes drastically reduce the wasted space and CAN be used incredibly efficiently IFF you can deal with the requirements
- **Bitmask column**
  - Index with one index to improve scanning but no ability to seek for individual bits
  - GREAT when your searching MANY values in one query and some are not very selective but horrible if the criteria IS selective…

Special Considerations: XML Indexes

- **XML indexes**
  - Primary XML indexes – SQL Server 2005 and higher
    - Primary XML index can be costly in terms of storage / backups /cache / rebuilds
    - Requires the primary key to be a row-based clustered index
  - Secondary XML indexes – SQL Server 2005 and higher
    - Requires the primary to have been created first
    - PATH secondary XML index
    - VALUE secondary XML index
    - PROPERTY secondary XML index
  - Selective XML Indexes (SXI) – SQL Server 2012 and higher
    - Doesn't *require* a primary XML index to have been created first
    - Implemented using sparse columns (sparse column limitations apply)
    - Indexing of only certain paths in an XML column

XML Design Considerations?

- **My main issues with XML are these…**
- **Is this ONLY used to get the data IN to SQL Server?**
  - Do you return XML to the clients and then get XML back?
- **Can you "shred" this data and make it relational**
  - Are some columns included in every XML set?
    - These should be regular columns
  - Are some data values highly sparse?
    - Can these be stored as sparse columns?
    - Can they be stored as name-value pairs?
- **There are often multiple ways to store the data and each has pros / cons**

XML Resources

- **Good starting point:** XML Indexes in SQL Server 2005 (http://bit.ly/1Lpt5vk)
- **Then, catch up in the books online with new features, etc.**
- **Then, for selective XML indexes (SXI) check out Bob Beauchemin's series of blog posts (www.SQLskills.com/blogs/BobB):**
  - Getting started with Selective XML Indexes in SQL Server
  - Selective XML Index – Implementation Details
  - Selective XML Index – Why is this compelling?
  - Selective XML Indexes – Learning the rules through error messages
  - Selective XML Index – Secondary Selective XML Indices

  and more!