SQLskills Immersion Event

IEPTO1: Performance Tuning and Optimization

Module 8: Internals and Data Access

Kimberly L. Tripp
Kimberly@SQLskills.com

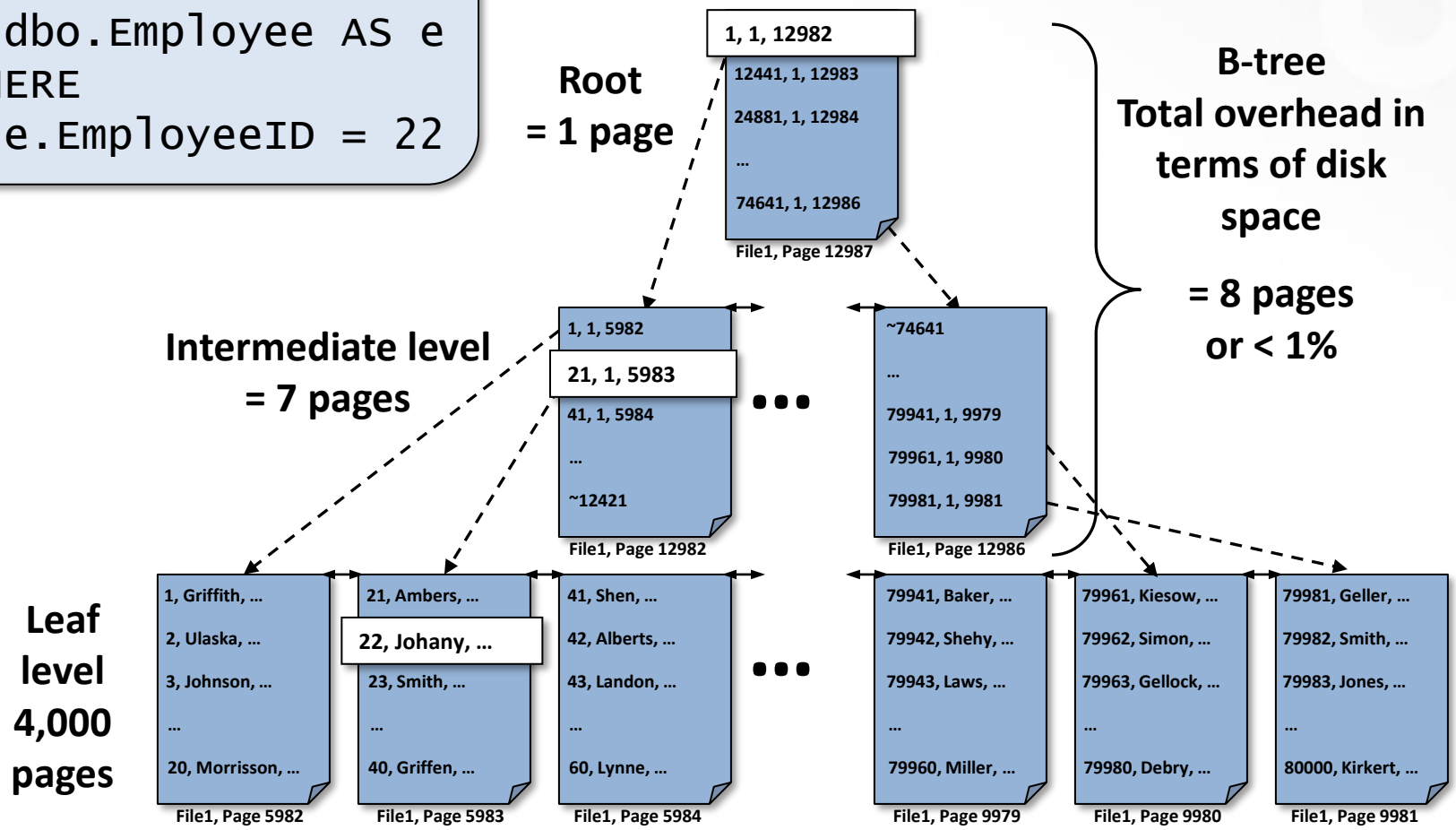
Overview

- **Data access patterns**
- **Covering**
  - Understanding selectivity
  - Understanding the "tipping point"
- **What methods exist for covering?**
  - Nonclustered indexes (all releases)
  - Using indexed views (SQL Server 2000+)
  - Using INCLUDE (SQL Server 2005+)
  - Using filtered indexes (SQL Server 2008+)
  - Using filtered statistics (SQL Server 2008+)
- **Too many cooks in the kitchen…**
- **Index consolidation**

© SQLskills, All rights reserved.
https://www.SQLskills.com

Query Specific Index Usage

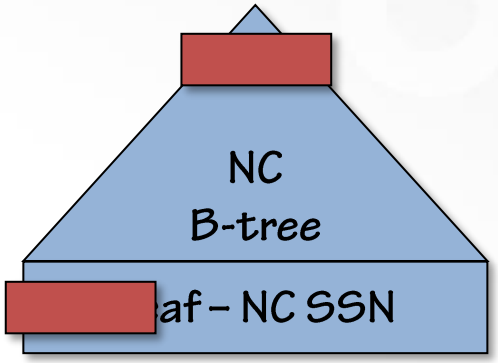
```
SELECT e.*
FROM
  dbo.Employee AS e
WHERE
  e.EmployeeID = 22
```



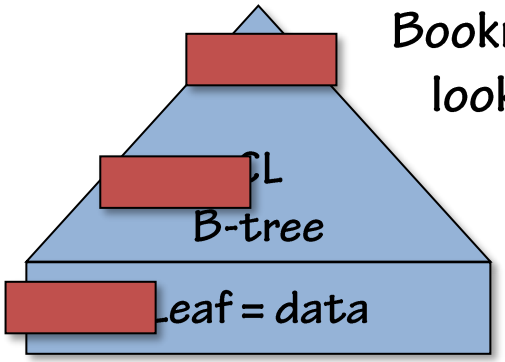
© SQLskills, All rights reserved.
https://www.SQLskills.com

Query Specific Index Usage

```
SELECT e.*
FROM dbo.Employee AS e
WHERE e.SSN = '123-45-6789'
```



- Root, then leaf in NC index on SSN to yield EmployeeID
- = 2 logical reads
- Root, intermediate, leaf in CL index to yield * (all columns)
- = 3 logical reads
- Total of 5 logical reads



© SQLskills, All rights reserved.
https://www.SQLskills.com

Query Specific Index Usage

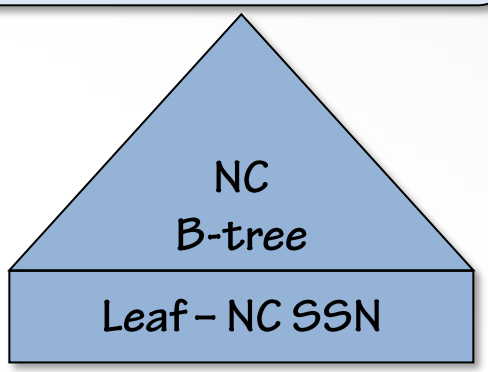
```
SELECT e.*
FROM dbo.Employee AS e
WHERE e.SSN BETWEEN '123-45-6789' AND '123-45-6800'
```

Assumption – 12 Rows

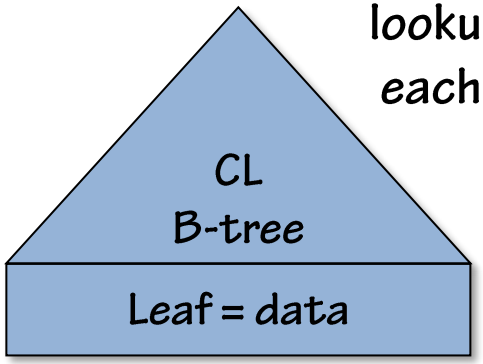
- **Root, then leaf in NC index on SSN to yield 12 EmployeeIDs**

  = 2 to 3 logical reads
- **Root, intermediate, leaf for each row to access * in CL index to yield**

  = 3 x 12 logical reads
- **Total of 38/39 logical reads**



Bookmark lookup for each row



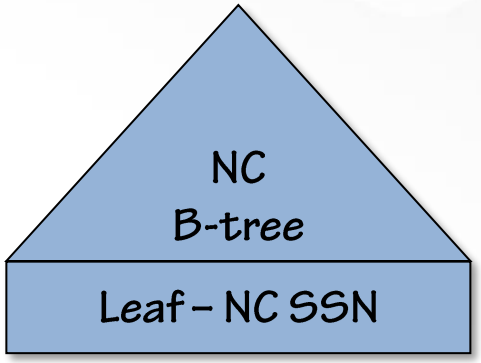
© SQLskills, All rights reserved.
https://www.SQLskills.com

Query Specific Index Usage

- At what range would a bookmark lookup be useless?

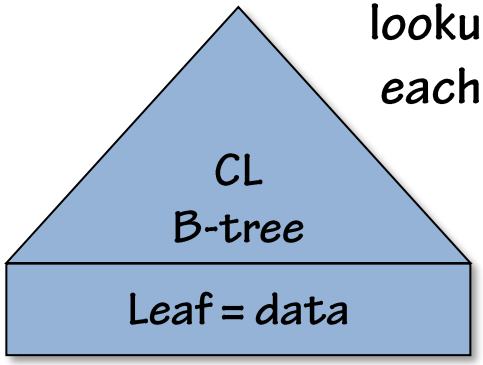
```
SELECT e.*
FROM dbo.Employee AS e
WHERE e.SSN BETWEEN x AND y
```

??? rows



- Table scan = 4,000 pages
- When # of rows >= 4,000, table scan is definitely better!
- In fact, probably well before that… 4,000 sequential reads are better than even fewer random reads

Bookmark lookup for each row



© SQLskills, All rights reserved.
https://www.SQLskills.com

When is a Query Selective Enough?

- *If* bookmark lookups are necessary… then a nonclustered is used ONLY when it's *selective enough*
- SQL Server looks at table size to compare the cost of bookmark lookups (which are random) to the cost of a table scan (which can be performed sequentially)
- To calculate this "tipping point," compare the I/Os to the table size for an estimate
- Take the number of pages (as a percentage of the table) to calculate the tipping point of rows, then determine what that percentage is relative to the table…

© SQLskills, All rights reserved.
https://www.SQLskills.com

Employee Scenario

- Imagine an Employee table with 80,000 rows at 20 rows per page for 4,000 total pages:

- Marking these two points ($^1/_4$ and $^1/_3$) we can see the boundary around where the tipping point is found
- This defines what's OK (less than $^1/_4$) versus what would be too expensive (greater than $^1/_3$)
- Since that represents random I/Os, translate that into rows (lookups):
  - $^1/_4$ mark is 1,000/80,000 = 1.25% and $^1/_3$ mark is 1,333/80,000 = 1.66%
- This means that if a query (against this table) is going to do LESS than 1,000 I/Os, SQL Server will use a nonclustered index because the lookups are OK
- However, if a query is going to do MORE than 1,333 I/Os then SQL Server WILL not use a nonclustered index with bookmark lookups because it's too expensive; instead, SQL Server will scan

© SQLskills, All rights reserved.
https://www.SQLskills.com

The Tipping Point Varies…

Table by Table and Based on PAGES

For more details on these three queries, see the Tipping Point category on my blog: https://www.SQLskills.com/blogs/kimberly/category/the-tipping-point/

- **Tipping point query #1**
  - Table with 1 million rows over 50,000 pages (20 rows/page)
  - 12,500 – 16,666 pages ∴ rows = 1.25 – 1.66%
- **Tipping point query #2**
  - Table with 1 million rows over 10,000 pages (100 rows/page)
  - 2,500 – 3,333 pages ∴ rows = 0.25 – 0.33%
- **Tipping point query #3**
  - Table with 1 million rows over 100,000 pages (10 rows/page)
  - 25,000 – 33,333 pages ∴ rows = 2.5 – 3.33%
- **Roughly $^1/_4$ - $^1/_3$ the number of PAGES in the table, translated to rows in the table = range (defined by the percent which is selective enough (less than $^1/_4$) to the percent not selective enough (> than $^1/_3$))**
  - Why it is not an exact percentage? CPUs/cores, disk affinity…but not actual disk speed (solid state) or whether or not the table is already in cache

© SQLskills, All rights reserved.
https://www.SQLskills.com

UPDATED

Tipping Point of FactInternetSales

From AdventureWorksDW20xx

- **As shipped:**
  - Pages: 1,238
  - Rows: 60,398
  - Tipping point ***estimate*** between
    - 309 ($^1/_4$ pages) and 412 ($^1/_3$ pages)
  - Tipping point ***actual*** = ~356
  - Translated into % of rows
    = 356/60,398 = 0.589%
- **In modified version (key ⇒ integer):**
  - Pages: 475,754
  - Rows: 30,923,776
  - Tipping point ***estimate*** between
    - 118,938 ($^1/_4$ pages) and 158,584 ($^1/_3$ pages)
  - Translated into % of rows – from .41 - .45% (depending of DOP)

Tipping point demo query values
The "small table" version:
11058 (estimate 354 rows)
11059 (estimate 358 rows)

The modified version (beefy machine):
MAXDOP 1 =
11043 (est. 138,695)
11044 (est. 141,505)
about 140,000 = .45%
MAXDOP *n* =
11038 (est. 124,644)
11039 (est. 127,454)
about 126,000 = .41%

© SQLskills, All rights reserved.
https://www.SQLskills.com

The Tipping Point Varies

- Use our formula as an *estimate* – more of a concept rather than a hard number
- It's not exact but the idea is important!
- Narrow indexes have very FEW uses! (often tip)
- If a query has to do a bookmark lookup then the query is going to have to be EXTREMELY HIGHLY selective in order to use the index *(you can calculate/predict this)*
- If a query is not highly selective but critical/important and you want consistent results/plans then you MUST look at some form of covering… why?
- Covering is always good – there is NO TIPPING POINT!

© SQLskills, All rights reserved.
https://www.SQLskills.com

Bookmark Lookups are Expensive!

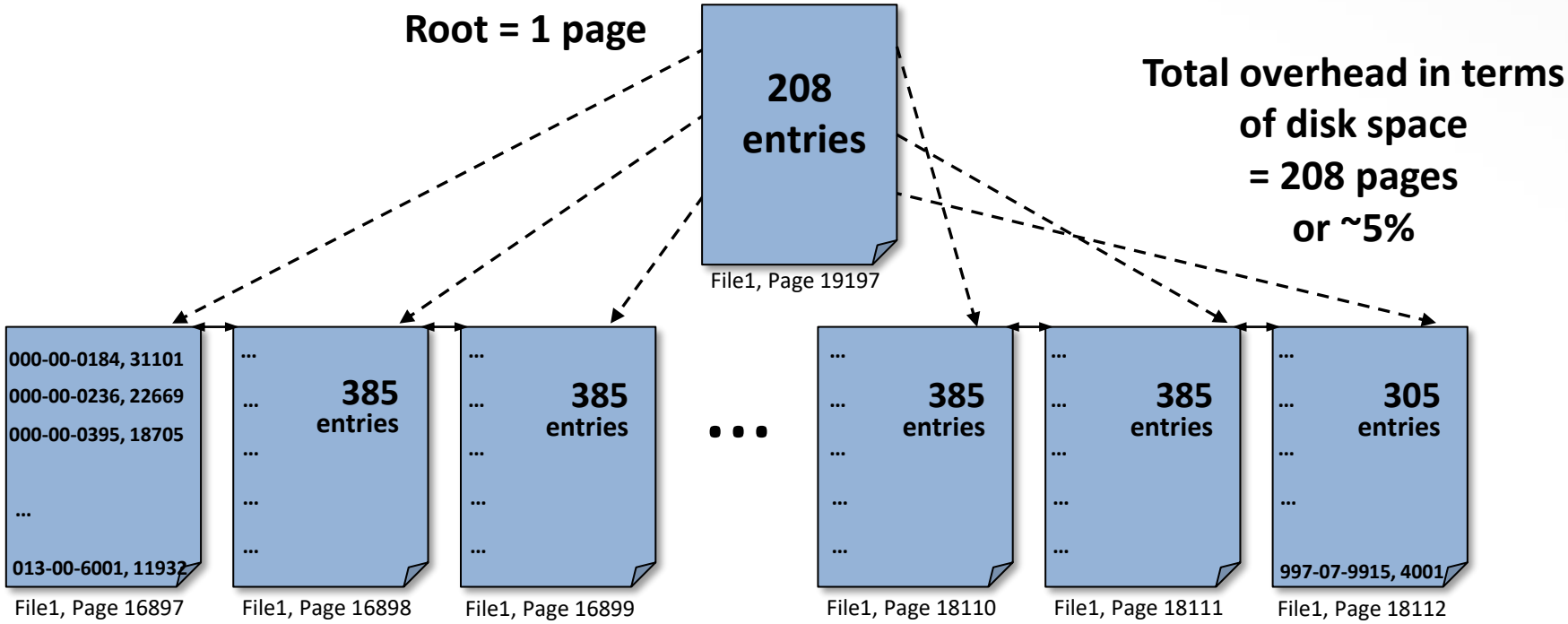
- **How can we get rid of the bookmark lookup?**
- **Are there other algorithms?**
  - Joins that reduce the set before bookmark lookups are needed
  - Index intersection (using multiple indexes to cover a query)
  - Covering
    - Using nonclustered indexes **without** bookmark lookups
    - Indexed views… (2000+)
    - INCLUDEd columns… (2005+)
    - Filtered indexes… (2008+)
- **But, you have to be careful not to over-index!**

© SQLskills, All rights reserved.
https://www.SQLskills.com

Nonclustered Index

Unique Key SSN

- Think back to our nonclustered index structure
- Leaf level contains the nonclustered key column(s) – index indexed order
- Includes either the heap's fixed RID or the table's clustering key

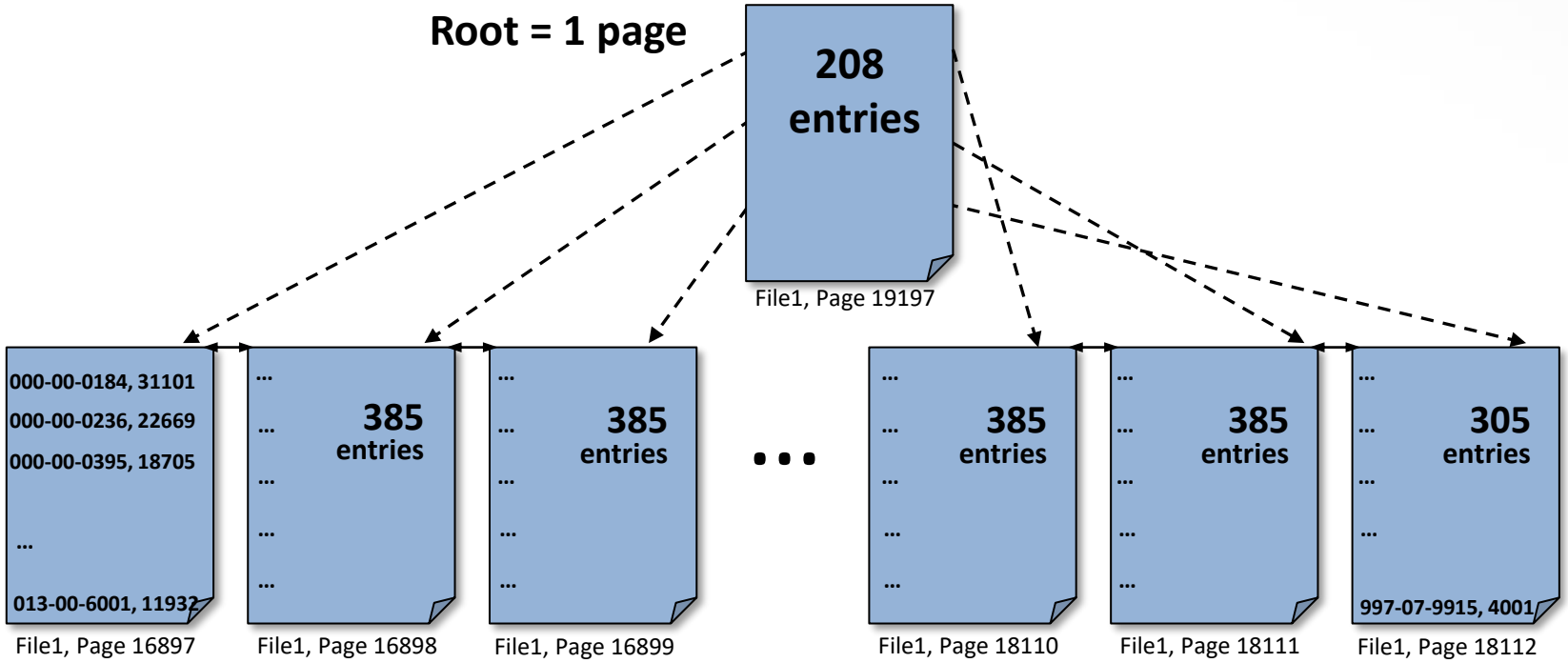


© SQLskills, All rights reserved.
https://www.SQLskills.com

What if You Didn't Know?

Unique Key SSN

- Could this structure be anything else?
- What if you created a table with JUST EmpID and SSN and then clustered it on SSN

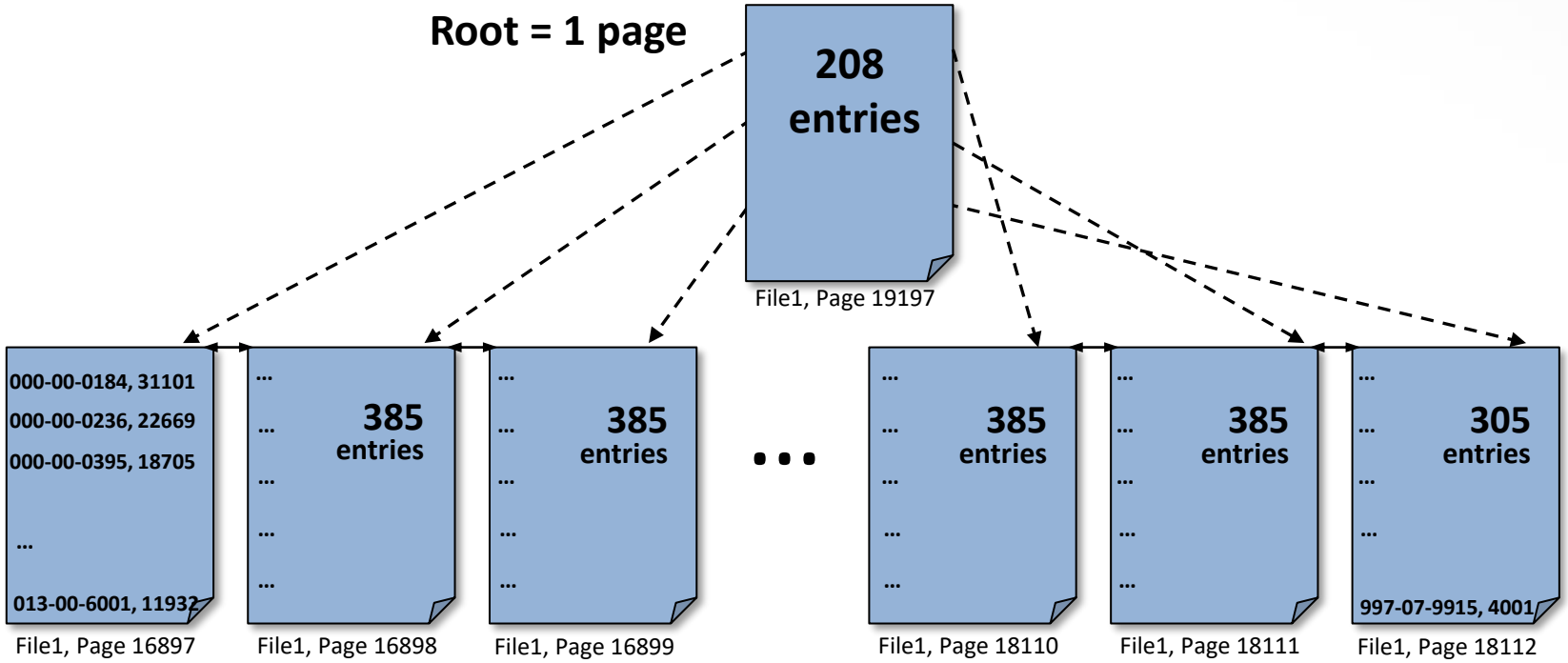


© SQLskills, All rights reserved.
https://www.SQLskills.com

Nonclustered Index

Fairly Obvious Index Access

```
SELECT e.EmpID, e.SSN
FROM dbo.Employee AS e
WHERE e.SSN BETWEEN '623-45-6789' AND '623-45-6800'
```

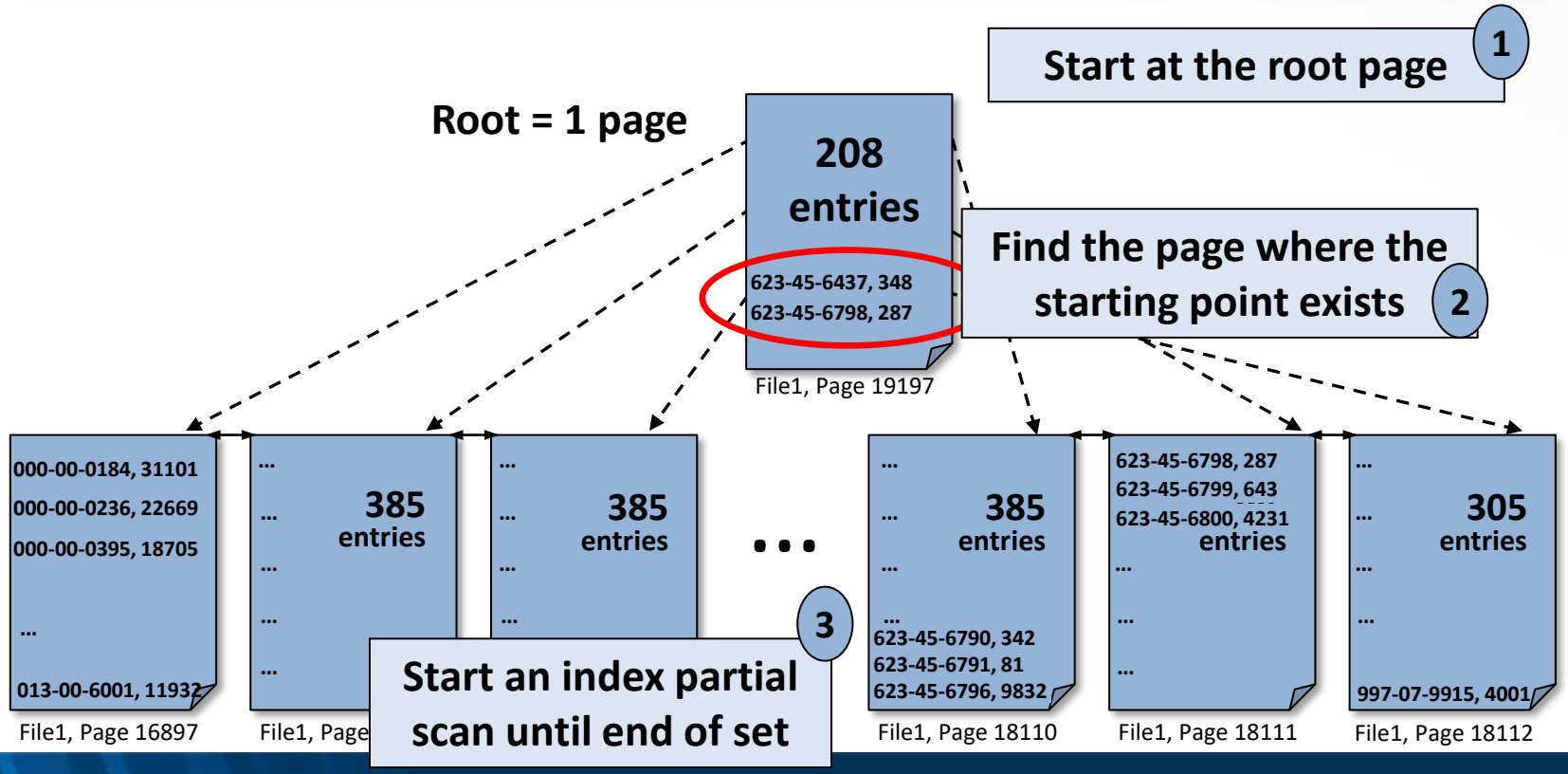


© SQLskills, All rights reserved.
https://www.SQLskills.com

Nonclustered Index

Fairly Obvious Index Access

```
SELECT e.EmpID, e.SSN
FROM dbo.Employee AS e
WHERE e.SSN BETWEEN '623-45-6789' AND '623-45-6800'
```



© SQLskills, All rights reserved.
https://www.SQLskills.com

Similar Query – How to Process?

Less Obvious Index Access

```
SELECT e.EmpID, e.SSN
FROM dbo.Employee AS e
WHERE e.EmpID < 10000
```

- **Clustered index on EmpID = 500 I/Os**
  - Seekable with partial scan
  - If table has 80,000 rows at 20 rows per page then the table has 4,000 pages
  - If 10,000 is $^1/_8$ of 80,000 then this SEEKABLE query will cost $^1/_8$ of the 4,000 pages.
- **Nonclustered index on SSN, EmpID = 208 I/Os**
  - Not seekable, must scan
  - If the leaf level has 80,000 rows at 385 rows per page then the leaf level of the nonclustered index has 208 pages
  - If this index is not seekable, then we must scan all 208 pages.

© SQLskills, All rights reserved.
https://www.SQLskills.com

Similar Query

Is There More to This Example?

- **What if the number of rows WHERE EmployeeID < 10000 was not 9,999?**
- **Would the optimizer make a different choice between the two indexes (and algorithms)?**
  - When 9,999, then
    - Seekable clustered index = 500 I/Os
    - Nonclustered covering index scan = **208 I/Os**
  - If only 1,000, then
    - Seekable clustered index = **50 I/Os** (@20 rows per page)
    - Nonclustered covering index scan = 208 I/Os
- **Because the covering index isn't seekable, the best index varies**
- **If this query were critical, I'd consider covering with a nonclustered index that is seekable…**

```
CREATE INDEX IndexName ON Member(EmployeeID, SSN)
```


© SQLskills, All rights reserved.
https://www.SQLskills.com

What is Covering?

Using an Index to Cover a Query

- **Only applies to nonclustered indexes**
  - The clustered "covers" all requests but it's the largest structure; it's what we're trying to avoid!
- **All columns requested in the query are somewhere in the index regardless of:**
  - Where they are in the query
  - Where they are in the index
- **However, column order does matter…**
  - If an index is seekable – you might be able to drastically reduce I/Os (think of the phone book – seekable by any left-based subset of the key: Lastname, Firstname, MiddleInitial)
  - If an index is not seekable and can only be scanned then you still might have significant savings, depending on how much narrower (than the base table) the index is…

© SQLskills, All rights reserved.
https://www.SQLskills.com

How is Covering Possible?

There is "Data" in the Index

- **Nonclustered indexes (without filters) contain *something* for every row in the base table (in the leaf level)**
  - If the table has 10 million rows then ALL non-filtered nonclustered indexes have 10 million rows
  - The leaf-level naturally has the index key (such as SSN) and also contains the clustering key (if the table is clustered) or the row's RID (if the table is a heap)
    - This structure could be a table in and of itself: a table created with only two columns that is clustered by the index key (SSN for example)
    - This structure can be scanned (like a table)
    - This structure can be seeked into (like a clustered table)
    - This structure can act like a table and it's this that you can leverage!

© SQLskills, All rights reserved.
https://www.SQLskills.com

Improve Low-Selectivity Queries

- **Limited select list**
- **Index on columns requested**

```
SELECT m.LastName, m.FirstName, m.PhoneNo
FROM dbo.Member AS m
WHERE m.LastName LIKE '[S-Z]%'
  -- 10,000 Rows, 3072 in S-Z Range
```

- **Covering!**
  A mini clustered table of just the data you need!

© SQLskills, All rights reserved.
https://www.SQLskills.com

Options to Access Data

- **Table scan**
- **Nonclustered on LastName**
  - Bookmark lookups for every row
- **Nonclustered index on LastName, FirstName and PhoneNo**
- **Nonclustered index on FirstName, LastName, PhoneNo**

© SQLskills, All rights reserved.
https://www.SQLskills.com

Actual I/O Costs

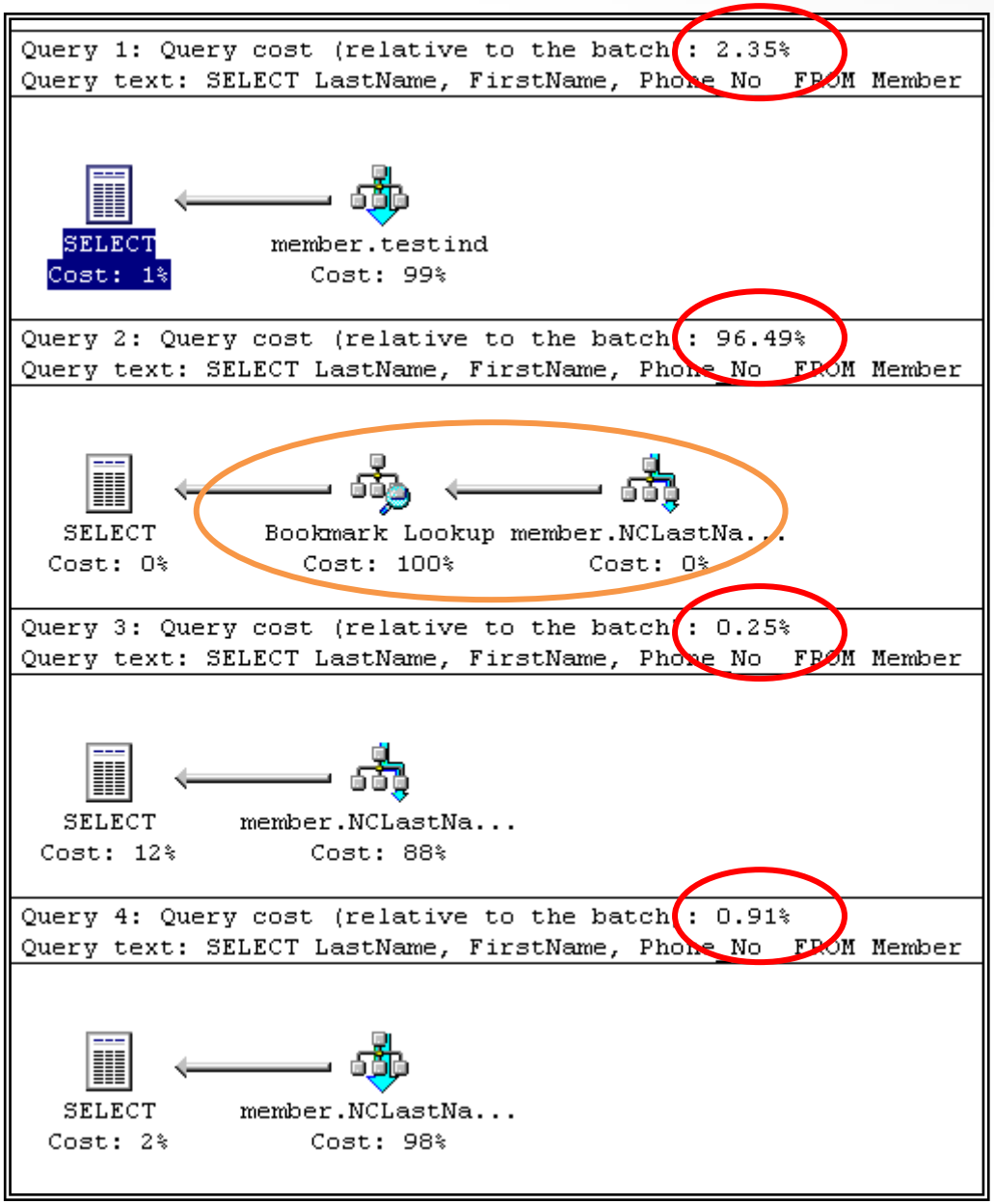


Table scan
= 143 reads

NC LastName
= 6,354 reads

NC covering seek
= 19 reads

NC covering SCAN
= 59 reads

SQL Server 2000 screen shot

hidden slide w/extra details

© SQLskills, All rights reserved.
https://www.SQLskills.com

Actual I/O Costs

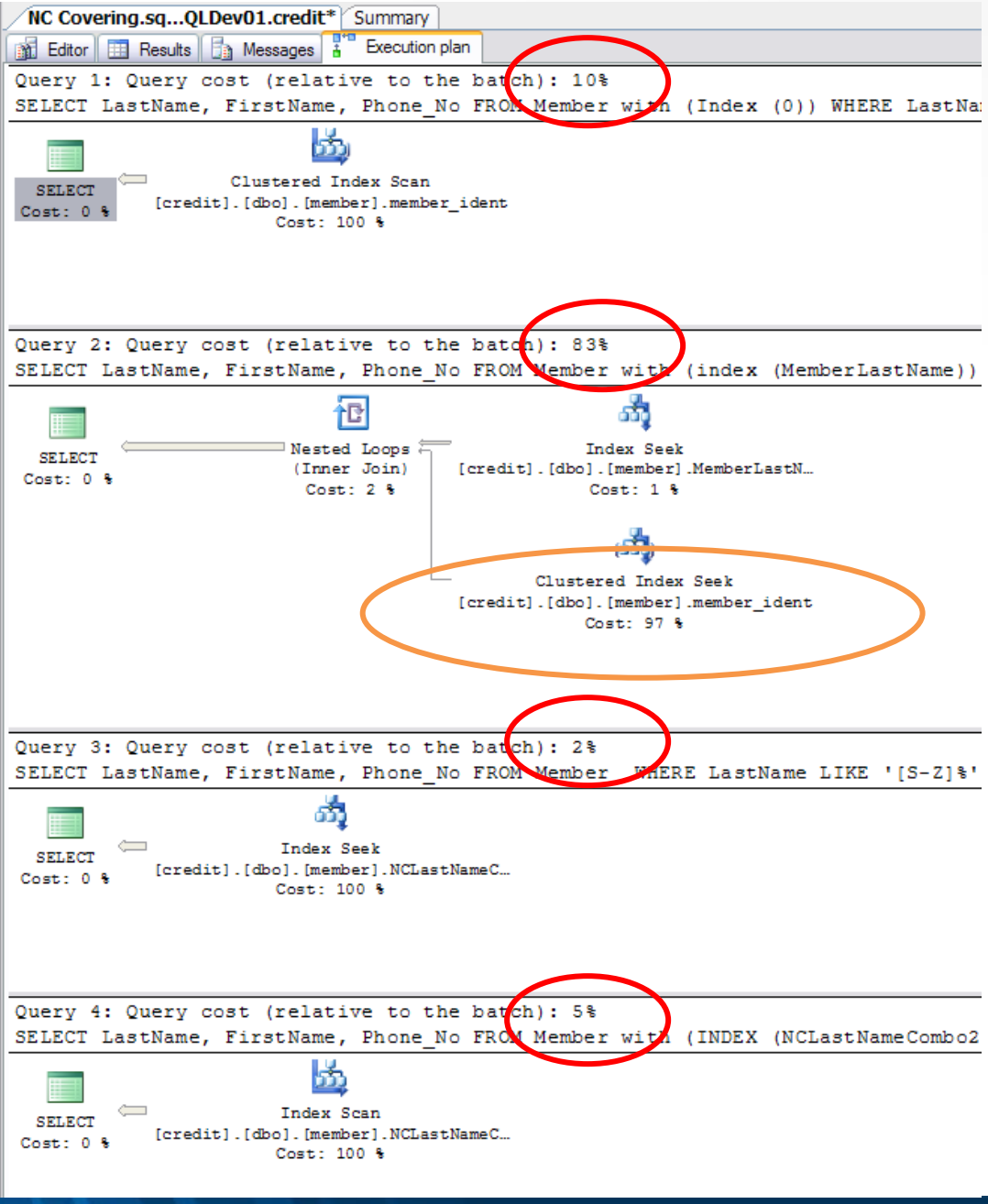


Table scan
= 144 reads

NC LastName
= 6,354 reads

NC covering seek
= 21 reads

NC covering SCAN
= 59 reads

SQL Server 2005 RTM screen shot

hidden slide w/extra details

© SQLskills, All rights reserved.
https://www.SQLskills.com

Actual I/O Costs

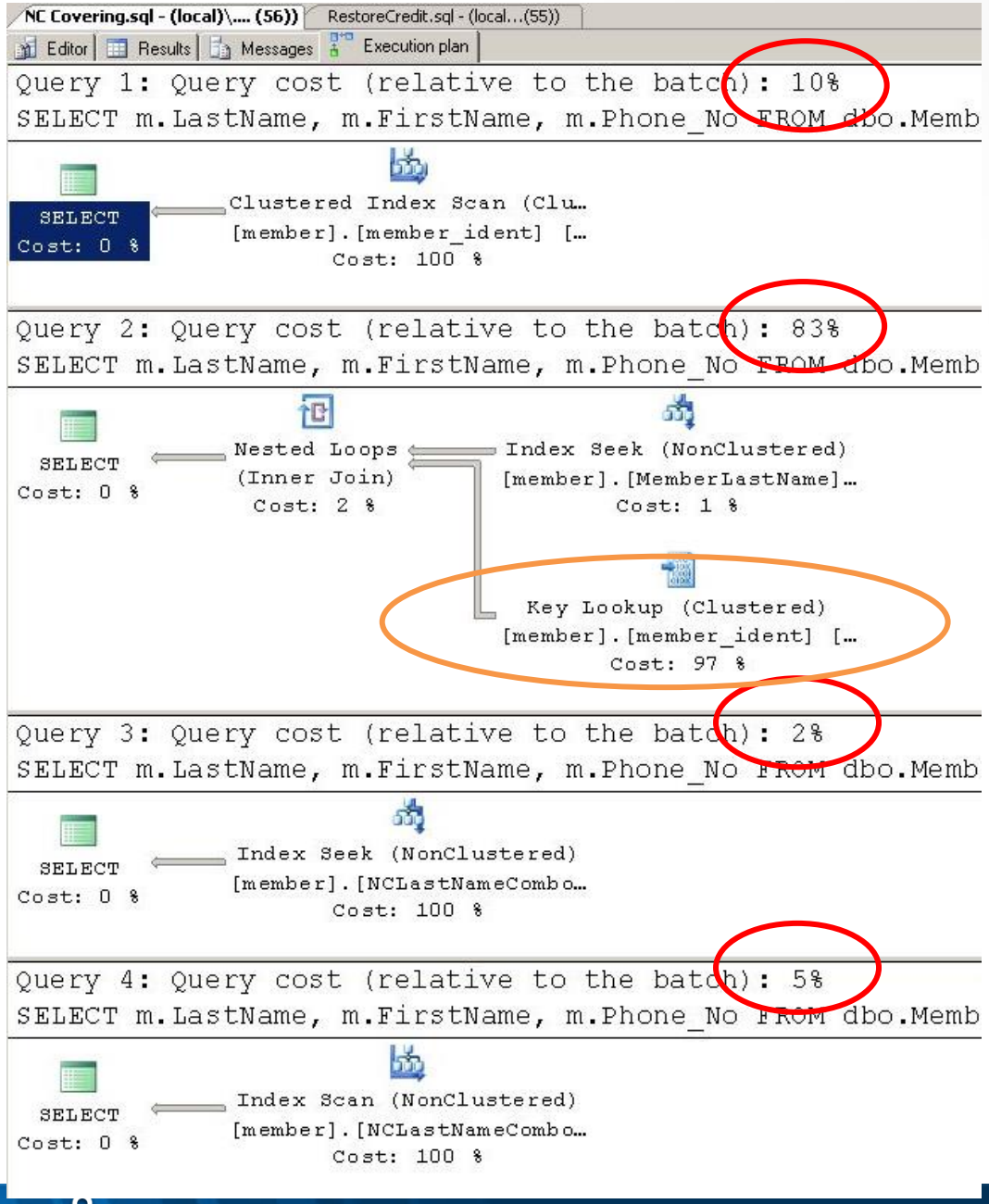


Table scan
= 144 reads

NC LastName
= 6,354 reads

NC covering seek
= 21 Reads

NC covering SCAN
= 59 Reads

SQL Server 2005 SP1+ /2008 screen shot

© SQLskills, All rights reserved.
https://www.SQLskills.com

You CANNOT Cover Everything!

And, I Really Don't Want You To… ☺

- **SQL Server 2005+ theoretically allows you to cover anything and everything (no real restrictions on what can be covered!)**
  - *Just because you can, doesn't mean you should!*

- **Over-indexing can be worse than under-indexing… especially if there's no rhyme or reason to the indexes created (they end up being nothing but costly overhead)**
  - *Do not just automatically put "an index on every column"…BAD!*

- **Too many indexes cost you:**
  - During modifications
  - During maintenance
  - Wasted cache
  - Wasted space (on disk, in backups, etc…)

© SQLskills, All rights reserved.
https://www.SQLskills.com

To Cover or Not to Cover?

Covering: Cleverly, Correctly, and Concisely!

- **Know how these strategies work**
  - *Scalability != "add more indexes"*
- **Get a good feel for your workload (you cannot tune without ALL of the following):**
  - Knowing your data
  - Knowing your workload
  - Knowing how SQL Server works
- **Leverage the tools**
  - 2000: ITW, Profiler, read80trace (PSS)
  - 2005: DTA, Performance Dashboard, Profiler, RML Utilities (PSS)
  - 2008: Add [Performance] Data Collection, query_hash
- **Using covering sparingly and ALWAYS be sure to use index consolidation techniques… ALL of these tools can have a tendency to become shortsighted.**

© SQLskills, All rights reserved.
https://www.SQLskills.com

Methods for Covering

- **Nonclustered indexes (all releases, all editions)**
- **Using indexed views (SQL Server 2000+)**
  - Available on all editions but limited on non-Enterprise editions. If not on Enterprise
    - Only queries which specifically reference the VIEW can leverage the index and ONLY when you use the hint:

```
FROM viewname WITH (NOEXPAND)
```

- **Using INCLUDE (SQL Server 2005+, all editions)**
- **Using filtered indexes (SQL Server 2008+, all editions)**

© SQLskills, All rights reserved.
https://www.SQLskills.com

INCLUDE for Better Covering

- **Key is limited to 900 bytes or 16 columns (whichever comes first)***
  - Allows the tree to be more scalable
  - Only applies to the b-tree not the leaf level of the index
- **Leaf-level can include non-key columns – with NO limitations (can include LOB types – use sparingly if at all!)**
  - Allows the leaf level to cover more queries
  - Can cover ANYTHING
- **In SQL Server 2005+ you CAN cover anything and everything… but just because you can, should you?**
- **In SQL Server 2016:**
  - A clustered index can have up to 32 columns and 900 bytes
  - A nonclustered index can have up to 32 columns and 1700 bytes

© SQLskills, All rights reserved.
https://www.SQLskills.com

Best Uses for INCLUDE

```
SELECT m.lastname, m.firstname,
       m.middleinitial, m.phone_no
FROM dbo.member AS m
WHERE m.lastname LIKE '[S-Z]%'
```

```
CREATE INDEX NCIndexLNOnly
ON dbo.member(lastname)
```

- Only useful for HIGHLY selective queries (which is NOT this one!)

```
CREATE INDEX NCIndexCoversAll4Cols
ON dbo.member
(lastname, firstname, middleinitial, phone_no)
```

- OK, but a bit overkill…

```
CREATE INDEX NCIndexLNinKeyInclude3OtherCols
ON dbo.member(lastname)
INCLUDE (firstname, middleinitial, phone_no)
```

- The *science*

```
CREATE INDEX NCIndexCoveringLnFnMiIncludePhone
ON dbo.member(lastname, firstname, middleinitial)
INCLUDE (phone_no)
```

- Debatable, but this might be my choice (the ***art*** of indexing)

© SQLskills, All rights reserved.
https://www.SQLskills.com

Physical Index Structures

Specific to Index Type

- **The clustered index (only 1 per table)**
  - The leaf level IS the data (the TABLE is clustered)
  - The b-tree is used for navigation
  - Contains *every* row (and every column) of the table
  - Cannot be filtered in any way
- **Nonclustered indexes (SQL2005: 249, SQL2008+: 999 per table)**
  - Without a filter
    - Contains *something* for *every* row of the table
    - Wide variety of uses but with more storage/overhead/maintenance
  - With a filter
    - Contains ONLY the rows that match the condition(s)
    - Fewer, more-specific uses but with [potentially ***significantly***] less storage/overhead/maintenance

© SQLskills, All rights reserved.
https://www.SQLskills.com

Filtered [Nonclustered] Indexes

- **Only applies to nonclustered indexes**
- **Maintenance costs are lower as only DML that affects rows in the index causes index changes**
- **Statistics are more accurate (at the time of creation) as they cover a smaller number of rows**
  - IMPORTANT NOTE: They START out being more accurate but there are some problems keeping them up to date *(more on this coming up)*…
- **Overall size might be significantly lower than a full-table index**
- **Require consistent session settings at many levels of creation and use (same session setting requirements as indexed views and computed column indexes)**

© SQLskills, All rights reserved.
https://www.SQLskills.com

Session Settings

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

Msg 1934, Level 16, State 1, Line 1
CREATE INDEX failed because the following SET options have incorrect settings: 'QUOTED_IDENTIFIER'. Verify that SET options are correct for use with indexed views and/or indexes on computed columns and/or filtered indexes and/or query notifications and/or XML data type methods and/or spatial index operations.

© SQLskills, All rights reserved.
https://www.SQLskills.com

Client Consistency

- **Consistency with table(s), view and the clustered index (on the view) creation OR table and the filtered index**
  - All tables on which the view is based, the view itself and the index must be created with the correct session settings set or the index cannot be created on the view
- **Consistency with base table access**
  - All INSERT, UPDATE and DELETE statements must be executed with correct session settings or the insert, update or delete will fail
- **Consistency with query access**
  - All queries that SELECT against views with indexes (or tables with filtered indexes) must access them with the correct session settings set otherwise the data will need to be recalculated, rejoined or recomputed

© SQLskills, All rights reserved.
https://www.SQLskills.com

Filtered Indexes: Uses

- **Indexing over sparse column values**
- **Indexing over partitions**
  - Subset of (and fewer of them) filtered indexes for current, read-write data
  - Subset (usually different and probably more of them) of filtered indexes for historical, read-only data
- **Indexing over tables with distinct ranges of values**
  - Looking only for a specific type (and relatively small set of data)

    ```
    WHERE Active = 1
    ```

  - Only interested in one salesperson

    ```
    WHERE SalesPerson = 8
    ```

  - Only interested in actual salespeople (OTC Sales have ID of 1)

    ```
    WHERE SalesPerson > 1
    ```

  - NEVER create **identical** individual indexes per set
    - Different columns and/or included columns is OK

© SQLskills, All rights reserved.
https://www.SQLskills.com

Imagine "Types" of Data

- **Insurance types: car, home, life, etc.**
- **Document types: (think SharePoint)**
- **Customers (with subtleties and views where you look at certain customers and specific attributes)**
- **Imagine there are 6 subtle "types" of records stored (distribution can be even or NOT – doesn't matter)**
  - When you look at type = 1, you're most interested in c6, c8, c4, c7
    - A filtered index on type = 1 only stores that % of data (whether large or small)
  - When you look at type = 2, you're most interested in c12, c16, c14, c6
    - A filtered index on type = 2 only stores that % of data (whether large or small)
  - When you look at type = 3, you're most interested in c2, c3, c4, c6

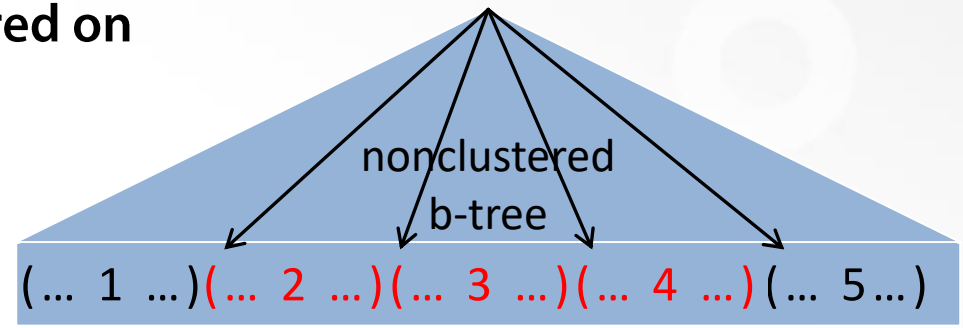
- **The point is that each index is smaller and with the rough equivalent (depending on column sizes) of ONE old (full-table) index you're getting 6 optimal and targeted filtered indexes!**

© SQLskills, All rights reserved.
https://www.SQLskills.com

Filtered Indexes

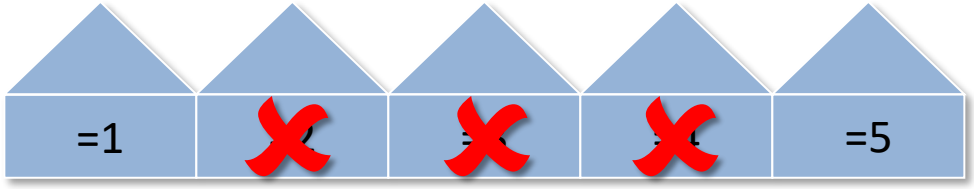
Why Not One Per Set?

- Leaf level of a regular nonclustered on SalesPerson (SalesPerson) INCLUDE (c6, c8)



SalesPerson BETWEEN 2 AND 4

- Leaf level of individual filtered indexes by SalesPerson INCLUDE (C6, c8)
- Total size difference is negligible at best (but, you don't have to include the column over which you are filtering)
  - Predicate uses are MORE limited
  - Covering choices are WAY MORE flexible



© SQLskills, All rights reserved.
https://www.SQLskills.com

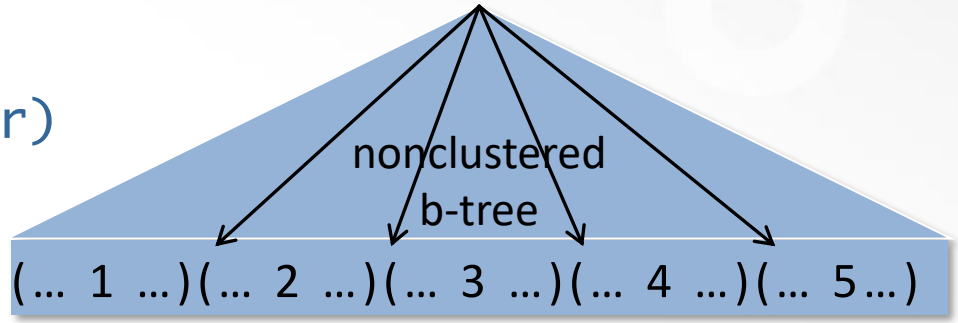
Filtered Indexes

When is One Per Set OK?

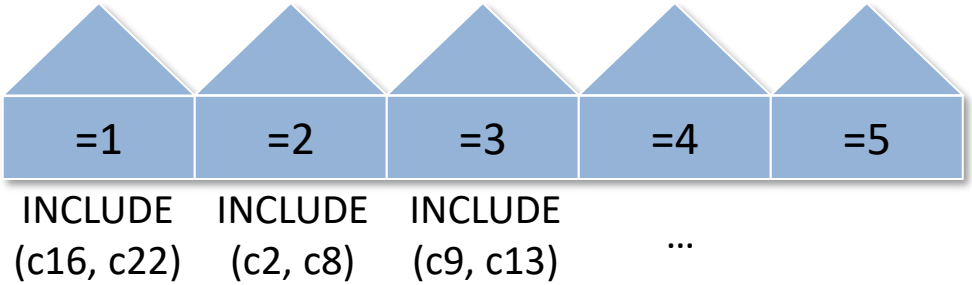
- **Leaf level for a nonclustered on**

```
SalesPerson
   (SalesPerson, Customer)
   INCLUDE (c4, c5, c6)
```

requires same set of columns for all salespeople



- **What if you want different columns per set?**
  - SalesPerson = 1 (c16, c22)
  - SalesPerson = 2 (c2, c8)
  - etc…



© SQLskills, All rights reserved.
https://www.SQLskills.com

Filtered Indexes: Monitoring

- `sys.indexes` has two interesting columns:
  - `has_filter`: if the index has a filter predicate
  - `filter_definition`: expression for the filter definition
- `DBCC SHOW_STATISTICS` will list the filter expression that defines the subset over which the statistics are computed
- `sp_helpindex` doesn't show INCLUDEd columns or filtered indexes– use my tweaked "sp_helpindex" to get better information and determine if one index really is redundant/duplicate:
  - https://www.SQLskills.com/BLOGS/KIMBERLY/category/sp_helpindex-rewrites.aspx

© SQLskills, All rights reserved.
https://www.SQLskills.com

Filtered Indexes: Miscellaneous

- **Have the same "session" requirements as computed columns and indexed views (need to make sure the client applications are consistent)**
- **Other features work well with filtered indexes:**
  - ALTER INDEX … REBUILD/REORGANIZE work with filtered indexes
  - Online index operations work with filtered indexes (but not indexed views)
  - DTA can suggest filtered indexes (NOTE: Missing index DMVs do not suggest filters)
- **Filtered indexes can be accessed for LITERALS within a stored procedure but NOT parameters or variables unless you use OPTION (RECOMPILE) on the statement that needs to leverage the filtered index**

© SQLskills, All rights reserved.
https://www.SQLskills.com

Filtered Indexes: COVER MORE!

- **They're small so add more!**
- **More effective for covering when sets are clearly defined**
  - Incredibly powerful
  - Hard to manage without knowledgeable database developer/admin/architect – really need to have someone that's dedicated/managing the server
- **When the sets are equality-based consider NOT including that attribute (unless the query returns that column)**
- **When the sets are NOT equality-based consider including the attribute as part of the key for effective seeking**
- **In summary, don't go wild with this feature – it has powerful but has specific uses!**

© SQLskills, All rights reserved.
https://www.SQLskills.com

Other Less Obvious Index Access Patterns

- Scanning nonclustered indexes to cover the query
- Nonclustered index intersection to join multiple indexes to cover the query
- Hash aggregates to scan and build aggregates out of order – still better to scan a covering index rather than a clustered
- Certainly, the BEST case is when you cover ONLY the necessary data AND it's seekable!
- But, you can't cover everything…

© SQLskills, All rights reserved.
https://www.SQLskills.com

Many Ways to Cover Queries

- Clustered index: always covers (only one way to seek or partially scan, full scan is the most expensive here as it's the entire row/set)
- Nonclustered indexes: cover the query with narrower rows = "just the data you need"
- Nonclustered indexes with INCLUDE: can allow you to cover ANY query and can even include LOB types in the leaf level of the index
- Indexed views: can cover wide queries (as does include) but these can include joins, aggregates, computations, deterministic functions, etc.

- Do you need to cover EVERY query? NO!

© SQLskills, All rights reserved.
https://www.SQLskills.com

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 like…

© SQLskills, All rights reserved.
https://www.SQLskills.com

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!**
- ***I'm surprised at how often this is overlooked/missed!***

© SQLskills, All rights reserved.
https://www.SQLskills.com

Review

- **Data access patterns**
- **Covering**
  - Understanding selectivity
  - Understanding the "tipping point"
- **What methods exist for covering?**
  - Nonclustered indexes (all releases)
  - Using indexed views (SQL Server 2000+)
  - Using INCLUDE (SQL Server 2005+)
  - Using filtered indexes (SQL Server 2008+)
  - Using filtered statistics (SQL Server 2008+)
- **Too many cooks in the kitchen…**
- **Index consolidation**

© SQLskills, All rights reserved.
https://www.SQLskills.com

Questions!