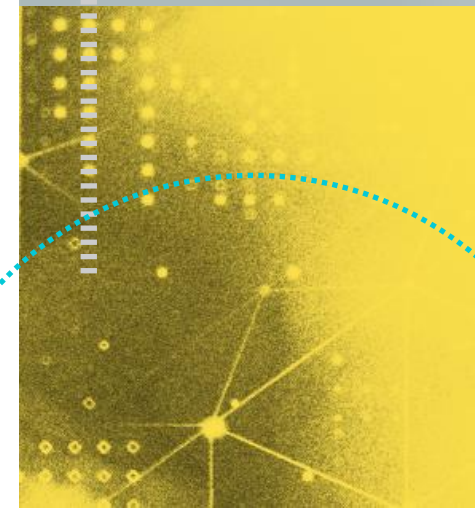




Index Strategies for SQL Server Performance

Kevin Kline, Head Geek, SolarWinds

Thomas LaRock, Head Geek, SolarWinds

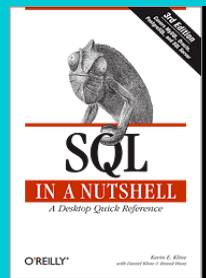








Kevin Kline

Head Geek™

Kevin has over 30 years experience in roles including programmer, data scientist, DBA, enterprise architect, and manager. He enjoys traveling, writing, teaching, and indie rock.



-  facebook.com/kekline
-  twitter.com/kekline
-  linkedin.com/in/kekline/
-  instagram.com/kevin_e_kline





Thomas LaRock

Head Geek

Thomas has over 20 years experience in roles including programmer, developer, analyst, and DBA. He enjoys working with data, probably too much to be healthy, really.



 facebook.com/thomas.larock

 twitter.com/SQLRockstar

 linkedin.com/in/sqlrockstar/

 instagram.com/sqlrockstar



Agenda



- Index structures and fundamentals
- Strategies for choosing good indexes
- Best practices for high-performance indexes
- Summary
- Q&A



POLL

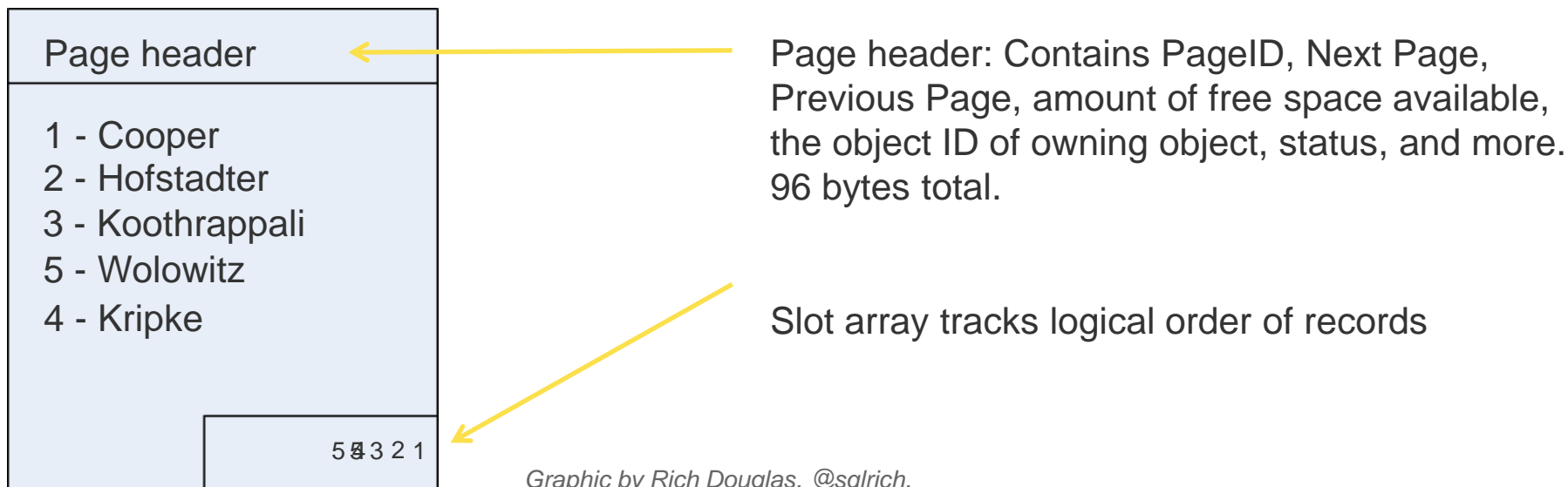
In your organization, who creates database indexes?





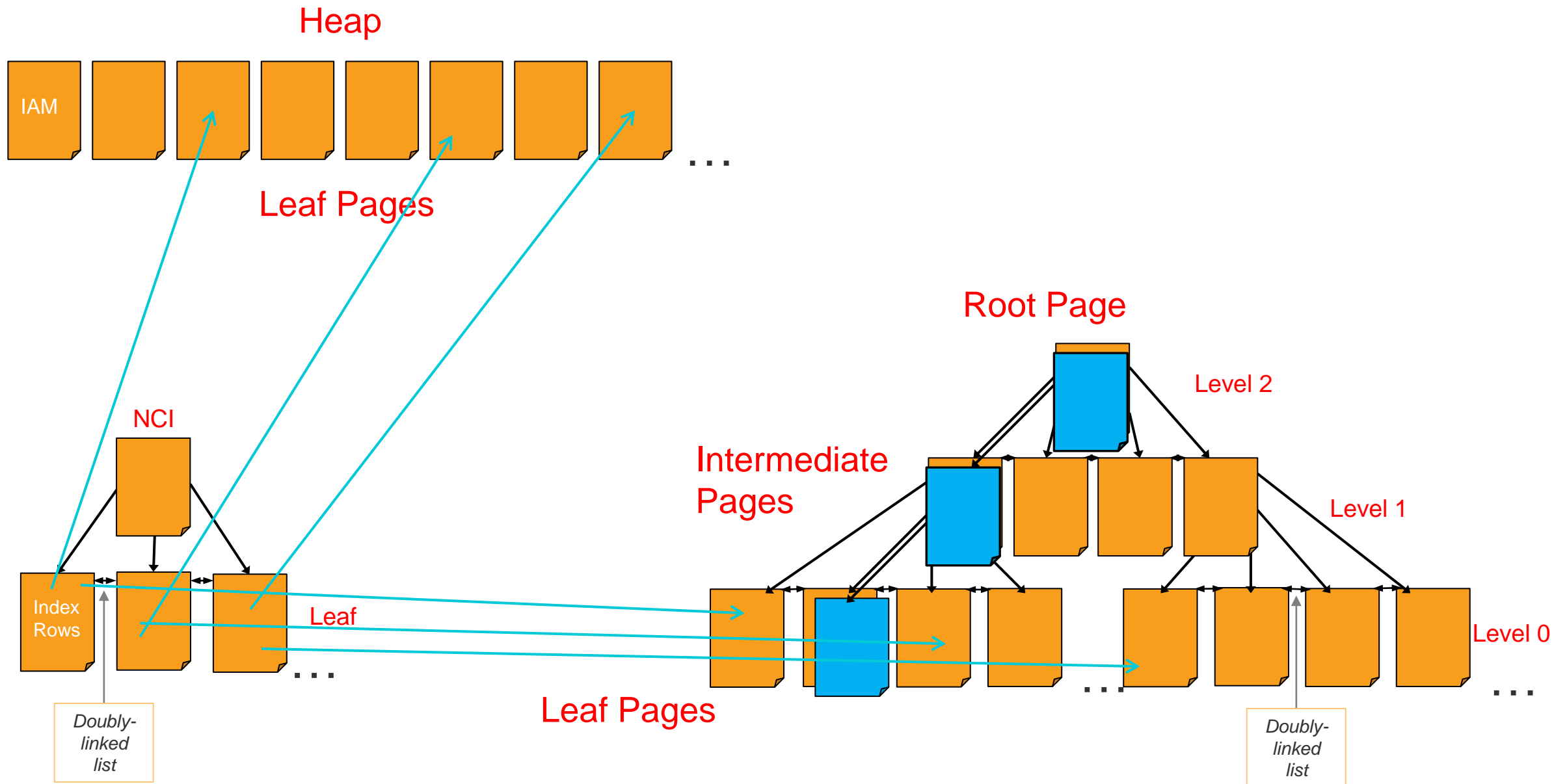
Overview of Tables and Indexes

- Three primary structures to remember – tables without a clustered index (Heap), tables with a clustered index, (CI) and non-clustered indexes (NCI).
- All three structures use an 8Kb data page (8192b total/ 8096 available) for storage:



Graphic by Rich Douglas, @sqlrich,
rdouglas@sentryone.com

Heaps, Clustered, and Non-Clustered





Clustered vs. Non

Clustered index *is* the data, sorted based on key

Non-clustered index has its own key

- Points to the row (RID) for a heap, to the key for a clustered index
- Extents (sets of eight 8kb pages) may not be physically sorted

Like a clustered index key, narrower index is better

Should every table have a clustered index? In most systems, yes!



Heap Pros and Cons

Pros:

- Great for bulk loading (create indexes after)
- Freed space can be reused on delete + subsequent insert

Cons:

- Insert performance can be poor if space is reclaimed
- Updates lead to forwarding pointers
- 8-byte RID assigned to every row (RID is file:page:slot)

Considerations for Clustered



Narrow, static, unique, ever-increasing

- Narrow and static because key is repeated in every non-clustered index
- Unique because otherwise a unique identifier needs to be added to each row
- Ever-increasing to avoid “bad” page splits and fragmentation

Some will argue a GUID is better – spread out I/O

- This is great for high-end, write-heavy workloads, until you have to read

On modern hardware, these factors are less important

- More on this later



Considerations for Non-Clustered

Consider an index a skinnier, sorted version of your table

Leading key column should support some balance of:

- Most selective
- Most likely to be used to sort
- Most likely to be used to filter
- Most likely to be used to join
- Less likely to change often (or at least less often than queried)



Benefits of Non-Clustered

Can have multiple (only one clustered)

Can support ordering not supported by clustered index

Can contain fewer columns and still “cover” many queries, i.e., fewer I/O reads.

Can contain fewer rows (filtered) and still satisfy queries



Seek vs. Scan

Seeks are used for single-row or range scans

- Seek really means “where does the scan start”
- SQL Server chooses seeks versus scans based on *statistics*

Scans are generally used to look at the whole index / table

- Scan count in stats I/O does not necessarily mean full scan count
- E.g., a parallel query or partitioned query has multiple partial scans

Generally, a seek is better, but not universally true

- Depending on index structure, a scan can be better
- And a NC index scan can be better than a CIX/table scan
- Don't blindly optimize for seeks, or assume a scan is bad news



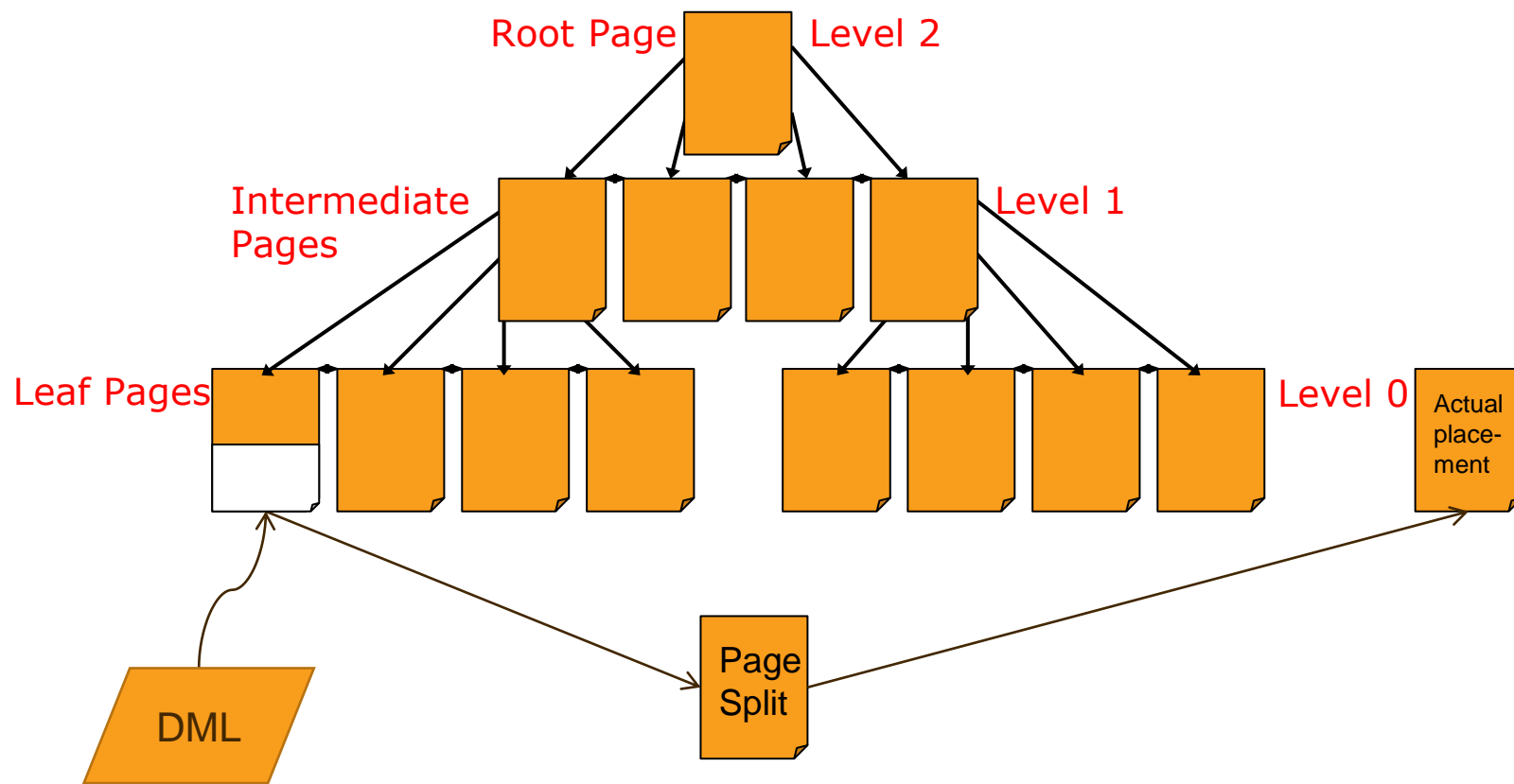
Full Pages vs. Read Performance

- During design, try to maximize the number of rows per page
 - Determine row size = data type consumption + overhead bytes
 - Determine rows per page = $8096 / \text{row size}$
 - Full pages are best for read-heavy workloads. Less pages to read equals less I/O work
- Avoid:
 - MAX data types, where possible, especially on heaps
 - ONE BIG TABLE: Too many columns hanging on one primary key, usually 1st or 2nd normal form
 - Redundant data or malformed data, as evidenced by lots of calculations, datepart function calls, isnull function calls, concatenations, and shredding upon retrieval
 - Column stuffing:
 - Storing arrays (CSVs, XML, JSON to stuff multiple values into a single field)
 - Parsing and validation are very cumbersome.
- Clustered index best practices:
 - Narrow, static, unique (but not GUID), and ever-increasing columns
 - Remember the clustered index is stored in every non-clustered index!



Full Pages vs. Write Performance

- Upon creation of the clustered index every page is 100% full leaving no empty space on the leaf or intermediate pages for added data from INSERT or UPDATE statements.
- Promotes logical table fragmentation, if there are writes.
- Default fill factor of 0/100 can cause costly *page splits* on some tables for write-heavy workloads.
- **Ways to fix:**
 - Rebuild or Reorg indexes
 - Specify FILL FACTOR option to leave open space in leaf pages
 - Specify PAD_INDEX option pushes fill factor up into intermediate pages



Remember:

Each write to leaf pages requires all non-clustered index structures to be updated!

Index Analogy – Full Pages vs. an Old Time Bath



Power move? Implement Data Compression for huge performance improvements



DEMO

T-SQL Demo of Index Performance



Performance Penalty for Concatenated Index

They can help. They can hurt.

- Don't forget the clustered index is stored in EVERY non-clustered index! The shorter the clustered index, the less the storage and fewer I/Os.
- Remember composite keys are most useful from the leftmost column to the rightmost column, in the order they appeared in the CREATE INDEX statement.

Example:

- Now, consider this index:
 - ALTER TABLE yellow_pages ADD CONSTRAINT [UPKCL_sales] PRIMARY KEY CLUSTERED ([lname], [fname], [street])

LName	FName	Street
<ul style="list-style-type: none">• Austin• Hanso• Linus	<ul style="list-style-type: none">• Kate• Magnus• Benjamin	<ul style="list-style-type: none">• 1010 Madison• 315 Ajira• 815 Oceanic



What Happens When Querying a Concatenated Index

Might not be what you expect

Works well with:

- WHERE lname = @a AND fname = @b AND street = @c

Also works well with:

- WHERE lname = @a AND fname = @b

How does the index work with this?

- WHERE fname = @b OR title_id = @c

Best practice? Use a surrogate key for the primary key and then a natural key as an additional non-clustered index.

Covering Indexes – The Flipside of Concatenated Indexes



An index covering the entire data requested by the query

A covering index answers a query entirely from its own intermediate pages, rather than going all the way down the tree to leaf pages.

All columns referenced in the query need to be in the index. For example:

- `SELECT Iname, fname`
- `FROM employee`
- `WHERE Iname LIKE '%ttlieb%'`

All columns in the index must be referenced in the query:

- `CREATE INDEX employee_name_lvl ON employee (Iname, fname)`

Don't confuse a covering index with an included columns index



Include Columns

The foil to the dreaded key/RID lookup

- Included columns “come along for the ride”
 - Copied into NC index structure, again to avoid lookups
 - Can exceed 900b key length, can be types not allowed in key (e.g., MAX)
- Should additional columns be in the key or included?
- Generally:
 - Only add columns that are small and queried often
 - Don't change an index to satisfy one query that does 12 LOB lookups
 - If they are just in the SELECT list, INCLUDE
 - If they are used in JOIN/WHERE/GROUP BY/ORDER BY, key if possible



“Tipping Point”

You can't always get what you want

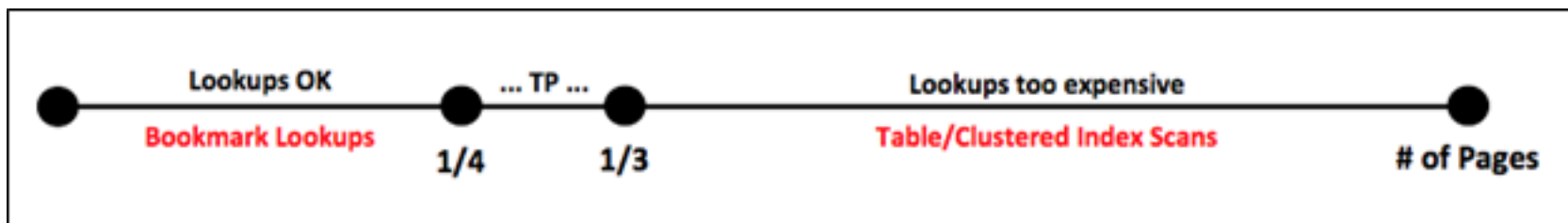
Generally, if seek estimated to hit > 25-30% of pages, scan is better

Lookups needed with seek on non-covering indexes

- A lookup is executed for every row

Again, generally, tipping point looks like this:

- Decision can be guided by index coverage, row size, cost of lookups, hardware
- Often the tipping point will be much lower



<http://www.sqlpassion.at/archive/2013/06/12/sql-server-tipping-games-why-non-clustered-indexes-are-just-ignored/>



Sargability

Surprising reasons why indexes might be unused

The ability to actually *use* the index

Beware of applying functions against JOIN/WHERE columns

- WHERE DATEDIFF(DAY, Column, GETDATE()) = 0
 - Not sargable – will scan – converting to a string is even worse
- WHERE Column >= @today AND Column < DATEADD(DAY, 1, @today)
 - Sargable

WHERE CONVERT(DATE, Column) = @today

- Can be sargable, but there's a catch – bad estimates – see link in notes

ORDER BY key column(s) won't use index if it doesn't cover



Creating Indexes: Rules of Thumb for CIs

- Clustered indexes are the actual physically written records
- A `SELECT` statement with no `ORDER BY` clause will return data in the clustered index order
- 1 clustered index per table, 249 non-clustered indexes per table
- Highly (almost) recommended for every table!
- Very useful for columns sorted on `GROUP BY` and `ORDER BY` clauses, as well as those filtered by `WHERE` clauses
- Place clustered indexes on the PK on high-intensity OLTP tables

Creating Indexes: Rules of Thumb for NCIs



Non-clustered indexes

- Useful for retrieving a single record or a range of records
- Maintained in a separate structure and maintained as changes are made to the base table
- Tend to be much narrower than the base table, so they can locate the exact record(s) with much less I/O
- Has at least one more intermediate level than the clustered index but are much less valuable if table doesn't have a clustered index

Creating Indexes: Short vs. Long Keys



- A “concatenated key” is an index with more than one column:
 - Up to 16 columns
 - Up to 900 bytes
- Short keys usually perform better, from fewer I/Os.
- Concatenated keys, also called composite keys, are evaluated from leftmost column to right (more on that later). So be sure the columns are indexed in order from most frequently used to least frequently used.

GUID vs. INT / BIGINT as Clustered Key



Constant debate – use GUID or INT for clustering key?

	INT / BIGINT	GUID	SEQUENTIAL GUID
Pros	<ul style="list-style-type: none">• Small• Compresses well• Fewer “bad” page splits	<ul style="list-style-type: none">• Globally unique*• Minimal hotspots	<ul style="list-style-type: none">• Globally unique*• Fewer “bad” page splits
Cons	<ul style="list-style-type: none">• Can cause hotspots• IDENTITY/SEQUENCE issues on 2012+• Ascending key problem	<ul style="list-style-type: none">• Wide• Compresses poorly• Many “bad” page splits• Heavy fragmentation• Tough to troubleshoot	<ul style="list-style-type: none">• Wide• Compresses poorly• Causes hotspots• Tough to troubleshoot

** I'm not convinced – we'll run out eventually, right?*



GUID vs. INT / BIGINT Issues

I'm still a fan of minimalism

- Prefer INT / BIGINT mostly for memory footprint

Which would you rather troubleshoot?

- OrderID = 42
- OrderID = '2C0C06E1-28A0-46BF-8CC6-63BC437AB42F'

On modern hardware, performance differences can be minimal

- “Disk space is cheap” – as long as you have lots of memory too
- Modern CPUs and SSDs are fast



DEMO

DPA Index Recommendations in Action

Optimize Index I/O



- Minimize the number of pages you need to read
 - Narrower indexes, narrower columns, compression, columnstore
- Minimize output columns to hopefully hit non-clustered and avoid lookups
- Keep statistics up to date (so memory grants are right)
 - Otherwise tempdb may be required to help sorts and joins
- Avoid sorting by columns not supported by indexes
 - Adding memory can keep bigger indexes in cache, but can't help sort
- Minimize the number of indexes, especially in write-heavy workloads
 - One wide index may be better than two skinny indexes

Summary



- Understand the Primary Usage of Table (read vs read/write). This determines lots of things such as fill factor, number of indexes, and which is column clustered index.
- (Almost) Always create primary key, clustering key.
- Manually add (non-clustered) indexes to foreign key constraints and other important columns such as join columns.
- Most important of all is to test, analyze, and retest. Don't be afraid to experiment!



solarwinds.com/DPA





Q&A





THANK
YOU



The SolarWinds, SolarWinds & Design, Orion, and THWACK trademarks are the exclusive property of SolarWinds Worldwide, LLC or its affiliates, are registered with the U.S. Patent and Trademark Office, and may be registered or pending registration in other countries. All other SolarWinds trademarks, service marks, and logos may be common law marks or are registered or pending registration. All other trademarks mentioned herein are used for identification purposes only and are trademarks of (and may be registered trademarks) of their respective companies.