CONSOLIDATION OF TABLE INDEX STRUCTURE

Understanding of table index structure is often omitted skill by the tuner.

Michal Tinthofer

CEO at Woodler/ Trainer / IT Consultant / Admin / Developer

MCSE since 2009 | MCT | No MCM I was too late ☹

Michal.Tinthofer@woodler.eu

http://tinyurl.com/W-linked

AGENDA

- Introduction to the concept of index consolidation
- Analyzing table usage patterns
- Analyzing query usage patterns
- Actual index consolidation recommendations
- BONUS - Clustered index consolidation on big tables

MICHAL TINTHOFER

- CEO at Woodler company (www.Woodler.eu) – Performance tuning focused SQL Support team
  - Architect – Consultant
    - Consulting industry leading companies in various countries in Europe.
    - Designing new SQL Server solutions, Evaluation of existing SQL servers (SQL Audit),Workload impact analysis, Migration and consolidation services
  - Performance tuner
    - Feature tuning, Complex Server tuning, Query Refactoring
  - Trainer
    - Coaching company admins and teaching at Gopas (Bratislava/Brno).
  - Admin
    - Performance tuner, External 3rd line support for company clients.
  - Developer
    - Development of SMT (SQL Server Monitoring Tool)

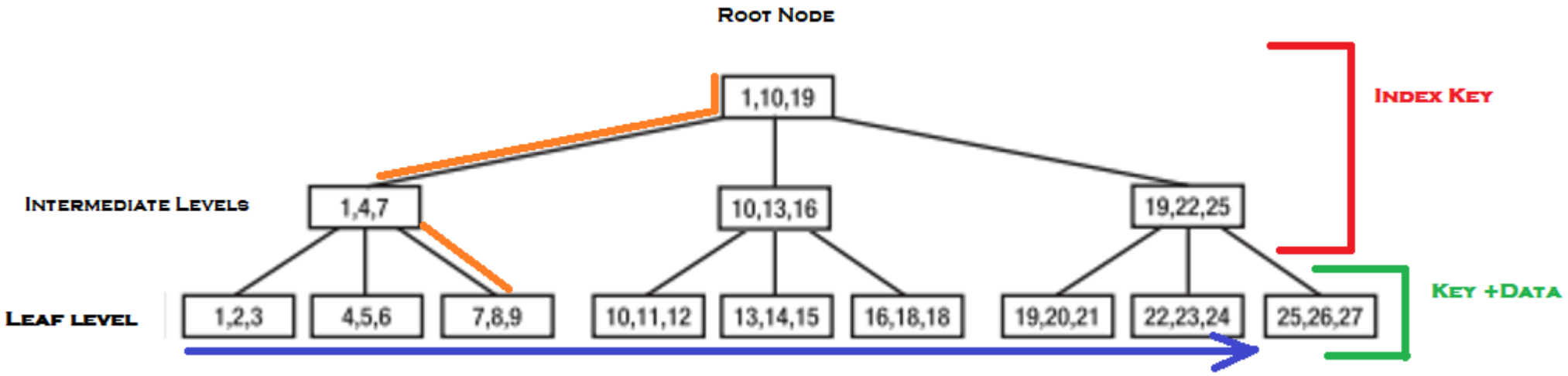
References: http://www.woodler.eu/references

Introduction to the concept of index consolidation

Woodler www.Woodler.eu

INDEX ADVANTAGES

- Reduce memory read operations
- Reduce CPU & Storage IO time requirements for the queries
- Save RAM capacity
- Server automatically create composite column stats on index key



INDEX DISADVANTAGES

- Can be tricky to correctly configure
  - Partitioning, Filtering, Key order (for columnstore)
- Requires extra disk space (often cumulatively more than table data itself)
- Increase maintenance times and sometimes extra maintenance (Filtered Indexes / Partitioned indexes)
- Can be disqualified from query tuning over time (due to fragmentation – index tipping point)

```
<relational_index_option> ::=
{
    PAD_INDEX = { ON | OFF }
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB = { ON | OFF }
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
  | STATISTICS_INCREMENTAL = { ON | OFF }
  | DROP_EXISTING = { ON | OFF }
  | ONLINE = { ON | OFF }
  | RESUMABLE = { ON | OFF }
  | MAX_DURATION = <time> [MINUTES]
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | OPTIMIZE_FOR_SEQUENTIAL_KEY = { ON | OFF }
  | MAXDOP = max_degree_of_parallelism
  | DATA_COMPRESSION = { NONE | ROW | PAGE }
    [ ON PARTITIONS ( { <partition_number_expression> | <range> }
    [ , ...n ] ) ]
}
```

	Reserved (KB)	Data (KB)	Indexes (KB)	Unused (KB)
97 7	896 746 056	896 594 080	5 768	146 208
29 5	384 369 032	295 425 608	88 896 640	46 784
31	299 768 856	20 966 880	278 642 168	159 808
30	141 044 200	137 661 392	3 343 576	39 232
91	104 863 720	104 336 936	479 528	47 256
19	93 264 256	86 741 856	4 792 624	1 729 776
50	90 854 816	90 469 808	376 784	8 224
02	84 847 864	50 200 320	34 409 216	238 328
51	80 992 176	72 706 632	8 133 432	152 112
78	49 725 568	26 593 416	23 100 344	31 808
08 5	48 356 672	15 842 352	32 510 752	3 568

INDEX BAD HABITS

- Making of indexes without evaluation of existing index structure on table
- Dumping Index recommendations without evaluation of table state
  - sys.dm_db_missing_index_details table
  - Query plan > Missing index

- To prevent this, we need to work on index consolidation procedure regularly

Analyzing table usage patterns

Woodler www.Woodler.eu

TABLE USAGE PATTERNS

- High read table
  - Variety of more indexes doesn't have a bad performance impact
  - Be careful with selectivity of your queries, server favor index joins only for very selective one, otherwise it drop query execution to clustered index scans
- High write table
  - Careful consideration is needed during index creation
  - Need to identify type of operations (updates or inserts?) on table
- Mixture of both
  - Usual situation for OLTP workload
  - We need to evaluate amount of reads vs updates/inserts on the table
  - Then amount of useful indexes might be setup after

WHERE TO FIND RELEVANT DATA?

```
SELECT * FROM SYS.dm_db_index_usage_stats AS a WHERE a.object_id =
Object_id('sales.salesorderdetail')
```

Results | Messages | Execution plan

database_id	object_id	index_id	user_seeks	user_scans	user_lookups	user_updates
6	1810105489	3	50179	1240	0	63448
6	1810105489	1	71979	1596	49262	63448

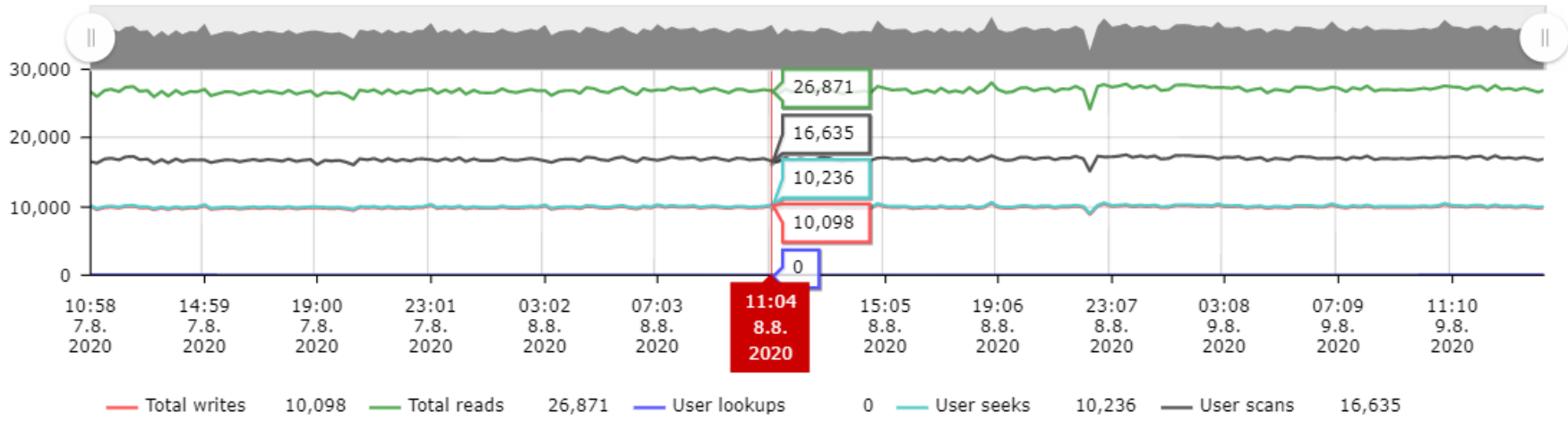
last_user_seek	last_user_scan	last_user_lookup	last_user_update	system_seeks
2020-08-09 14:36:51.807	2020-08-09 14:36:50.770	NULL	2020-08-09 14:36:54.550	0
2020-08-09 14:36:54.547	2020-08-09 14:36:50.940	2020-08-09 14:36:51.807	2020-08-09 14:36:54.550	0

THROUGH SMT

Operation Summary

Index Name	Total Reads	Total Writes	Reads vs. Writes Percentage			Seeks [%]	Scans [%]	Lookups [%]	Fill Factor	Disabled	Hypotetical	Filtered
PK_Subscription2Check	861939255	534423971	61.7%		38.3%	3.9	96.1	.0	0	0	0	0
NC_Subscription2Check_IsEnabled_Processing	438153946	533688911	45.1%		54.9%	.0	100.0	.0	0	0	0	0

Operations over time



Analyzing query usage patterns

UNDERSTANDING QUERY PATTERNS

- We need to know how queries are using our table
- Read Patterns
  - On which columns are we searching table?
  - What values are used during table searches?
  - Which other columns are needed for the queries?
- Write Patterns
  - Not every write is insert, It could also be an update
    - Update could be for specific column thus affects only specific set of indexes
    - Insert affects all indexes on the table

READ PATTERNS

- So we need to somehow understand our queries on specific table. Collection Time!
- You can use several tools to do so.
  - Query Plan Cache
  - SQL Audit Feature (It was enterprise edition only)
  - Extended events
  - Missing index rcmd..
- Then we need to analyze usage patterns.

Actual index consolidation recommendations

REVIEW INDEX KEY DEFINITION

- Index Key order matters
  - (ColA,ColB <> ColB,ColA)
  - Key Column selectivity is important but not always trustworthy
  - Application usage patterns matter
    - Key filtering
    - Ordering
    - Query importance
- You may always make an educated guess, but prepare your self to analyze & hotfix after deployment of change

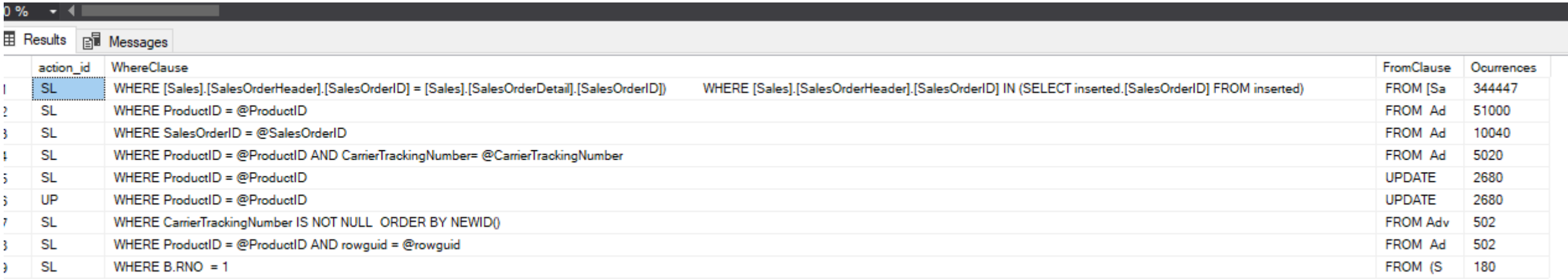
START WITH EASY OPTIONS

- Simple consolidation
  - Look for duplicate key index
  - Remove low or not read indexes
  - Favor equality indexes instead of inequality based one

KeyColumns	IncludedColumns	total_reads	total_writes	pct_reads_seek	pct_reads_total
[CarrierTrackingNumber] Asc	[ProductID]	234	86416	19.2	0.3
[CarrierTrackingNumber] Asc[ProductID] Asc	NULL	2001	86416	100.0	2.3
[CarrierTrackingNumber] Asc[ProductID] Asc	[OrderQty][LineTotal][rowguid]	0	86416	0.0	0.0
[ProductID] Asc	NULL	99522	417785	97.2	19.2
[ProductID] Asc	[CarrierTrackingNumber][OrderQty][LineTotal][rowguid]	0	86416	0.0	0.0
[ProductID] Asc[rowguid] Asc	NULL	4720	86416	95.0	5.2
[rowguid] Asc	[CarrierTrackingNumber][OrderQty][ProductID][SpecialOfferID][UnitPrice][UnitPriceDiscount][Line...	17	86416	100.0	0.0
[rowguid] Asc	NULL	21	85553	4.8	0.0
[SalesOrderID] Asc[SalesOrderDetailID] Asc	NULL	538546	417785	80.8	56.3

ADVANCED OPTIMIZATION

- Check for key value distribution
- How are your queries using the table
- Which parameters are used and what is possible from application perspective
- Adapt indexes to most resource intensive queries found (Covering index)



0 %

Results | Messages

	action_id	WhereClause		FromClause	Ocurrences
1	SL	WHERE [Sales].[SalesOrderHeader].[SalesOrderID] = [Sales].[SalesOrderDetail].[SalesOrderID])	WHERE [Sales].[SalesOrderHeader].[SalesOrderID] IN (SELECT inserted.[SalesOrderID] FROM inserted)	FROM [Sa	344447
2	SL	WHERE ProductID = @ProductID		FROM Ad	51000
3	SL	WHERE SalesOrderID = @SalesOrderID		FROM Ad	10040
4	SL	WHERE ProductID = @ProductID AND CarrierTrackingNumber= @CarrierTrackingNumber		FROM Ad	5020
5	SL	WHERE ProductID = @ProductID		UPDATE	2680
6	UP	WHERE ProductID = @ProductID		UPDATE	2680
7	SL	WHERE CarrierTrackingNumber IS NOT NULL ORDER BY NEWID()		FROM Adv	502
8	SL	WHERE ProductID = @ProductID AND rowguid = @rowguid		FROM Ad	502
9	SL	WHERE B.RNO = 1		FROM (S	180

OUTRO

- Don't forget to regularly review your index usage
- Monitor your SQL Server continuously to allow you to make the right decision on table indexes.
  - We often see the same application used by different customer having different usage statistics, so single recommendation approach to all is not possible
- Review your query usage on tables, it could really change your mind regarding the final index design
- Maintain your databases regularly

Q&A

Thank You!

Woodler Hiring!

- Contact: office@Woodler.eu

- Do you love SQL Server?
- Like a personal challenge?
- Happy to learn new stuff?
- Analytical?
- Like to technically own critical customer systems?
- We welcome you in our team!

- Woodler is only dedicated company in CZ/SK specializing directly on performance tuning and root cause analysis for SQL Servers.

BONUS

Clustered index consolidation on big tables

Woodler www.Woodler.eu

CCI

- Decrease costly scans on tables by using CCI
  - It reduce Reads of the queries
  - But first understand implications
    - No seek on CCI
    - Costly lookups if any – Partially fixed by CUs
- Use covering NC indexes
  - It is must for OLTP workload!
- Preorder CCI by creating standard CL index on table
  - Will result in better compressed index

NC INDEX TIPPING POINT

- Tipping point (when QO choose to scan table instead nonclustered index)
  - Counted only for non-covering non-clustered indexes
  - Rough point is between 25-33% of table data pages
  - However, parallelism, some server settings (processor affinity and I/O affinity), memory and table size – all can have an impact.
  - Compensate by
    - WITH (INDEX ('indexname'))
    - WITH(FORCESEEK)
    - ideally use COVERING INDEX

```
SELECT in_row_data_page_count
FROM sys.dm_db_partition_stats
WHERE object_id = OBJECT_ID('sales.SalesOrderDetail')
                  AND index_id = 1
```