Columnstore Indexes in SQL Server 2014

Flipping the DW Faster Bit

Jimmy May, MCM

MSIT Principal Architect: Database

jimmymay@microsoft.com

http://blogs.msdn.com/jimmymay

Agenda

1. **Introduction & Preamble**
2. SQL Server 2012 & 2014: New! Improved! Features
3. Columnstore Indexes
    1. Overview
    2. Architecture
    3. SQL Server 2012 vs. 2014
    4. Implementation
    5. Scenarios
    6. Best Practices
4. More Info

Bio

Jimmy May, MCM
- MSIT Principal Architect: Database

Formerly:
- Sr. Program Manager, SQL CAT
- SQL Server Customer Advisory Team

Microsoft Certified Master: SQL Server (2009)
MS IT Gold Star Recipient (2008)
Microsoft Oracle Center of Excellence (2008)
SQL Server MVP Nominee (2006)
Indiana Windows User Group www.iwug.net
- Founder & Board of Directors

Indianapolis Professional Association for SQL Server www.indypass.org
- Founder & Member of Executive Committee

SQL Server Pros: Founder & Visionary-in-Chief
SQL Innovator Award Recipient (2006)
- Contest sponsored in part by Microsoft

Formerly Chief Database Architect for high-throughput OLTP VLDB at ExactTarget (recently IPO)
Senior Database Administrator for OpenGlobe/Escient

jimmymay@microsoft.com
www.twitter.com/aspiringgeek
http://blogs.msdn.com/jimmymay

Session Objectives And Takeaway

Session Objectives

1. Understand that columnstore indexes can increase SQL Server DW query performance by one, two, or three orders of magnitude.
2. Understand the advantages & limitations of columnstore indexes in SQL Server 2012 & the significant improvements available in SQL Server 2014.

Key Takeaway

Understand the circumstances under which columnstore indexes may be compelling reason to upgrade to SQL Server 2012 or 2014.

Note: Slides & scripts on SkyDrive: Columnstore Collateral

MICROSOFT CONFIDENTIAL – INTERNAL ONLY

C:\>sqlservr.exe /faster

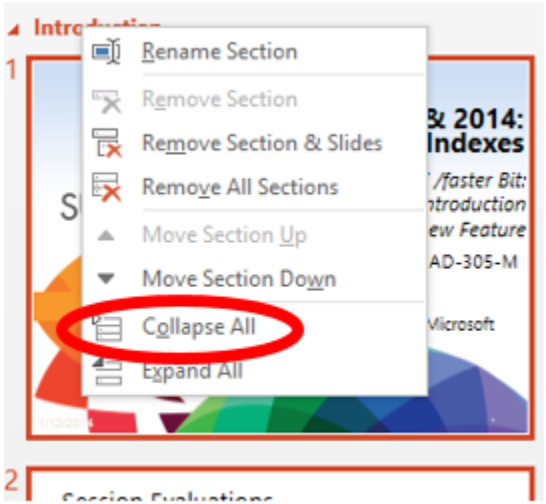
Acknowledgements

- Shahry Hashemi, Wingman, CapitalOne
- Thomas Kejser, Fusion-io CTO (former SQL CAT PM)
- Mike Ruthruff, Alpha Geek, Bungie Studios (former SQL CAT PM)
- Justin Lane, Principal PM
- Susan Price, SQL Server PM (former)
- Eric Hansen, SQL Server PM
- Gus Apostol, SQL Server PM
- Gavin Payne, MCA, Coeo Consultant
- Steve Fisher, MSIT Senior Service Engineer
- LeRoy Tuttle, Sr. SDE
- Mahesh Jambunathan, SDE
- Hrair Kerametlian, Principal PM
- Emanuel Rivera Aleman (EMAN), Sr. SE
- Bill Kan, Sr. SE
- Igor Stanko, Principal PM
- Srikumar Rangarajan, Principal Dev Lead
- Stanislav Novoseletskiy, Principal Platform Specialist

Navigating this deck

You may find the deck easier to navigate by collapsing the Sections:

1. Right-click on any *section name*
2. Click “Collapse All”)
3. Review the section names for desired content



Agenda

1. Introduction & Preamble
2. SQL Server 2012 & 2014: New! Improved! Features
3. **Columnstore Indexes**
    1. **Overview**
    2. Architecture
    3. SQL Server 2012 vs. 2014
    4. Implementation
    5. Scenarios
    6. Best Practices
4. More Info

Demo: Columnstore vs. Conventional Index Performance

What is a SQL Columnstore Index?

- Codename Apollo
- Part of SQL Server in-memory family of technologies
  - Common codepath with VertiPaq
  - PowerPivot, PowerView, SSAS
- Contrasted with traditional row stores in which data is physically stored row-by-row
- Columnstore stores values for all rows *for a given column*

Columnstore High-Level Characteristics

- Highly compressed
- Aggressive readahead
- In-memory/disk hybrid structures
- Processes data in units called "batches"
- Vector-based query execution
- Query Optimizer automatically considers columnstore indexes during compilation
- Numerous deep engine modifications
  - I/O, Memory, & Caching

Why Use Columnstore Indexes?

- Designed to optimize access to large DWs (vs. OLTP)
  - Star schema, large fact tables (esp. integer keys), aggregations, scans, reporting
- Faster, interactive query response time
- Transparent to the application
  - Most things "just work"
  - Backup and restore
  - Mirroring, log shipping
  - SSMS, etc.
  - There are some gotchas (stay tuned…)

Why Use Columnstore? (cont.)

- Reduced physical DB design effort
  - Fewer (or no!) conventional indexes
  - Reduced need for summary aggregates and indexed views
  - May eliminate need for OLAP cubes
- Lower TCO
  - Yes, I had to say this!

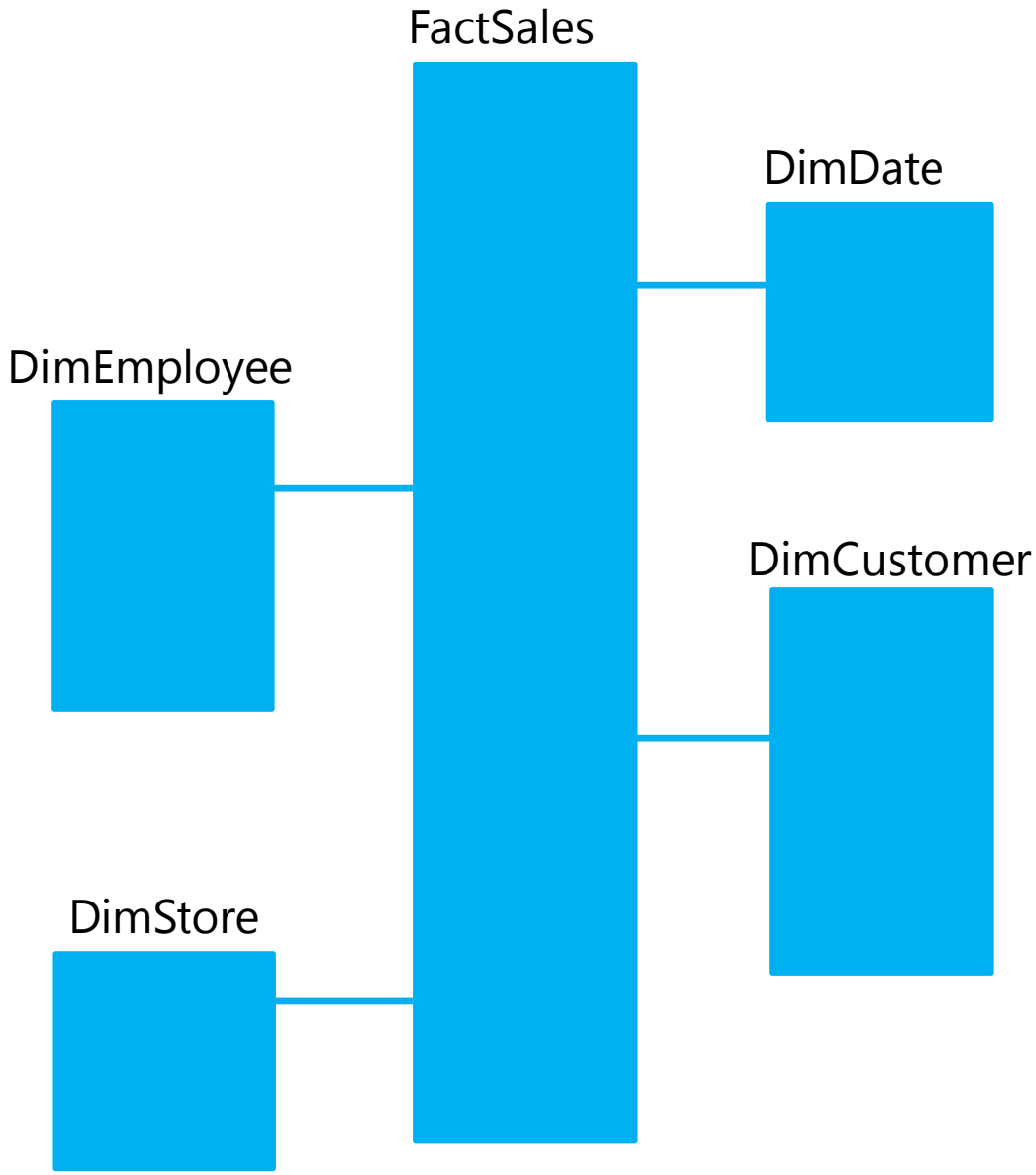
Agenda

1. Introduction & Preamble
2. SQL Server 2012 & 2014: New! Improved! Features
3. **Columnstore Indexes**
    1. Overview
    2. **Architecture**
    3. SQL Server 2012 vs. 2014
    4. Implementation
    5. Scenarios
    6. Best Practices
4. More Info

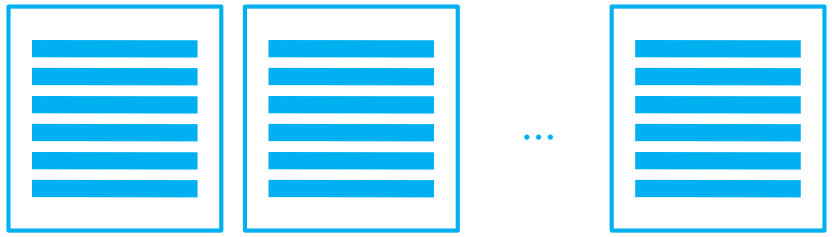
Star Schema

```
FactSales ( CustomerKey int
          , ProductKey int
          , EmployeeKey int
          , StoreKey int
          , OrderDateKey int
          , SalesAmount money
          )
--note: lots of ints in fact tables

DimCustomer ( CustomerKey int
            , FirstName nvarchar(50)
            , LastName nvarchar(50)
            , Birthdate date
            , EmailAddress nvarchar(50)
            )

DimProduct (…
```

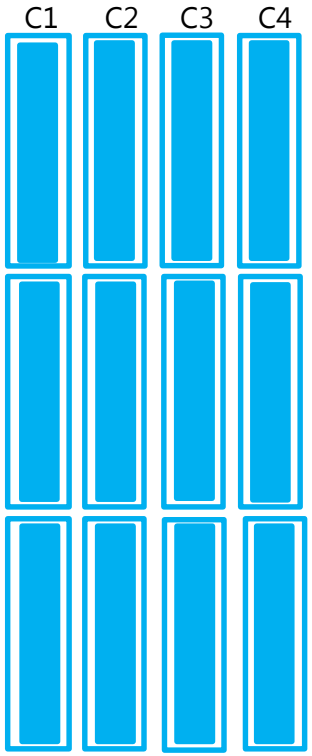
Best Practice: Integer keys!



How Do Columnstore Indexes Optimize Perf?

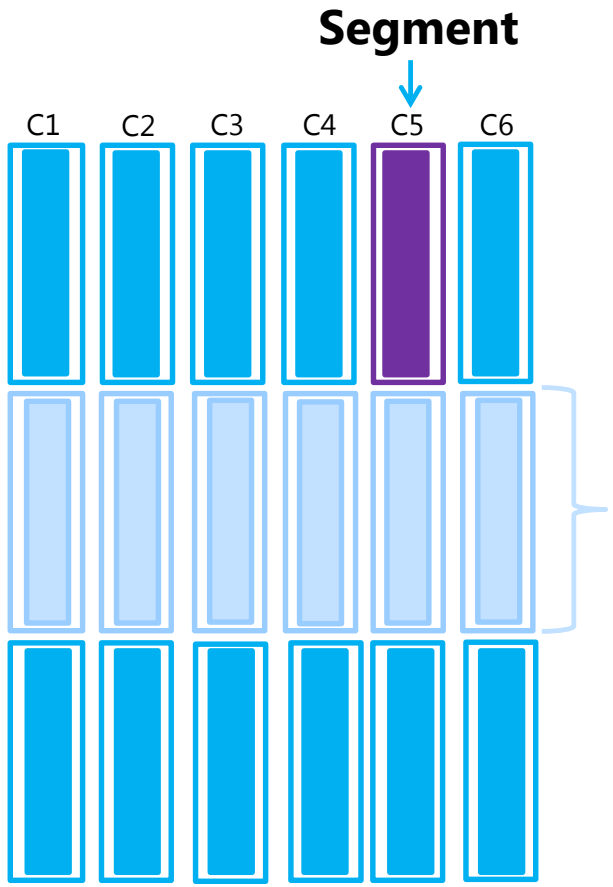


- Heaps, B-trees store data row-wise



- Columnstore indexes store data column-wise
  - Each page stores data from a single column
- Highly compressed
  - About 2x better than PAGE compression
  - More data fits in memory
- Each column accessed independently
  - Fetch only needed columns
  - Can dramatically decrease I/O

Columnstore Index Architecture



- Row Group
  - 1 million logically contiguous rows
- Column Segment
  - **Segment** contains values from one column for a set of rows
  - Segments for the same set of rows comprise a **row group**
  - Segments are compressed
  - Each segment stored in a separate LOB
  - Segment is unit of transfer between disk and memory

Columnstore Index Example

OrderDateKey	ProductKey	StoreKey	RegionKey	Quantity	SalesAmount
20101107	106	01	1	6	30.00
20101107	103	04	2	1	17.00
20101107	109	04	2	2	20.00
20101107	103	03	2	1	17.00
20101107	106	05	3	4	20.00
20101108	106	02	1	5	25.00
20101108	102	02	1	1	14.00
20101108	106	03	2	5	25.00
20101108	109	01	1	1	10.00
20101109	106	04	2	4	20.00
20101109	106	04	2	5	25.00
20101109	103	01	1	1	17.00

1. Horizontally Partition (Row Groups)

OrderDateKey	ProductKey	StoreKey	RegionKey	Quantity	SalesAmount
20101107	106	01	1	6	30.00
20101107	103	04	2	1	17.00
20101107	109	04	2	2	20.00
20101107	103	03	2	1	17.00
20101107	106	05	3	4	20.00
20101108	106	02	1	5	25.00

OrderDateKey	ProductKey	StoreKey	RegionKey	Quantity	SalesAmount
20101108	102	02	1	1	14.00
20101108	106	03	2	5	25.00
20101108	109	01	1	1	10.00
20101109	106	04	2	4	20.00
20101109	106	04	2	5	25.00
20101109	103	01	1	1	17.00

2. Vertically Partition via Columns (Segments)

OrderDateKey	ProductKey	StoreKey	RegionKey	Quantity	SalesAmount
20101107	106	01	1	6	30.00
20101107	103	04	2	1	17.00
20101107	109	04	2	2	20.00
20101107	103	03	2	1	17.00
20101107	106	05	3	4	20.00
20101108	106	02	1	5	25.00

OrderDateKey	ProductKey	StoreKey	RegionKey	Quantity	SalesAmount
20101108	102	02	1	1	14.00
20101108	106	03	2	5	25.00
20101108	109	01	1	1	10.00
20101109	106	04	2	4	20.00
20101109	106	04	2	5	25.00
20101109	103	01	1	1	17.00

3. Compress Each Segment*

OrderDateKey
20101107
20101108

ProductKey
106
103
109

StoreKey
01
04
03
05
02

RegionKey
1
2

Quantity
6
1
2
4
5

SalesAmount
30.00
17.00
20.00
25.00

OrderDateKey
20101108
20101109

ProductKey
102
106
109
103

StoreKey
02
03
01
04

RegionKey
1
2

Quantity
1
5
4

SalesAmount
14.00
25.00
10.00
20.00
25.00
17.00

Some segments will compress more than others

*Encoding and reordering not shown

Agenda

1. Introduction & Preamble
2. SQL Server 2012 & 2014: New! Improved! Features
3. **Columnstore Indexes**
    1. Overview
    2. Architecture
    3. **SQL Server 2012 vs. 2014**
    4. Implementation
    5. Scenarios
    6. Best Practices
4. More Info

SQL Server 2012 Caveats

- Non-clustered columnstore—underlying B-tree still required to support the columnstore
- Some queries, even the schema, might have to be modified to fully leverage columnstore
- Read only, not writable
  - *In SQL 2012*
  - Work-around: "Trickle-loading" (described later)
- Memory Required to create columnstore indexes may exceed that for conventional indexes
- Enterprise Edition only
- Not yet available in Azure
  - Except IaaS
- Not compatible with indexed views, filtered indexes, sparse columns, computed columns
- Datatype support significant but not complete

SQL Server 2014 Query Optimization

- Good design practices for dimensional models
  - Star schema
  - Columnstores on Fact tables
  - Integer surrogate keys for joins to Dimension tables
- The power of the platform
  - You just write queries!
  - Logically no difference between Clustered Columnstores and conventional tables

Columnstore in SQL 2014

- Fully Read/Write
  - More transparency to the app than ever
  - No need for “trickle-loading” or other workarounds
  - …Yet Partition switching & BULK INSERT remain best practices
- Data type support expanded:
  - All data types except: (n)varchar(max), varbinary(max), XML, Spatial, CLR
  - Basically, SQL14 columnstore is compatible with all non-blob datatypes

Columnstore in SQL Server 2014 (cont.)

- New: Clustered Columnstore
  - Dependency on conventional b-tree structures has been removed
  - Potential for significant disk space savings if workload is satisfied without conventional indexes
- Note: Non-clustered columnstore is still supported & is still a read-only structure
  - Required if:
    - Constraints are required
    - Workload requires b-tree non-clustered indexes

Columnstore in SQL Server 2014 (cont.)

- "Batch mode":
  - Query Processor vector-based (L1 cache resident) operations expanded, improved
  - New support for:
    - All joins (including OUTER, HASH, SEMI (NOT IN, IN)
    - UNION ALL
    - Scalar aggregates
    - "Mixed mode" plans
- Improvements to bitmap, spill support, etc.
- Hash join enhancements

Factor Strings from Columnstore Joins

- Columnstore joins on string columns is better in 2014 than 2012, yet still slower!
- Factor strings from Fact out to Dim tables
- Integer FKs is a DW best practice
  - See "Star Join" slide earlier
  - Demo later

Date	LicenseNum	Measure
20120301	XYZ123	100
20120302	ABC777	200

Date	LicenseId	Measure
20120301	1	100
20120302	2	200

LicenseId	LicenseNum
1	XYZ123
2	ABC777

Agenda

1. Introduction & Preamble
2. SQL Server 2012 & 2014: New! Improved! Features
3. **Columnstore Indexes**
    1. Overview
    2. Architecture
    3. SQL Server 2012 vs. 2014
    4. **Implementation**
    5. Scenarios
    6. Best Practices
4. More Info

Syntax Similar to Existing DDL

```
CREATE NONCLUSTERED COLUMNSTORE INDEX ix_cs_MyDWTable
   ON dbo.MyDWTable
   (col1 , col2 , ... , coln);

CREATE CLUSTERED COLUMNSTORE INDEX ix_cs_MyDWTable
   ON dbo.MyDWTable;
   --no column list!

DROP INDEX dbo.MyDWTable.ix_cs_MyTable;

ALTER INDEX dbo.MyDWTable.ix_cs_MyTable DISABLE;

ALTER INDEX dbo.MyDWTable.ix_cs_MyTable ENABLE;
```

Hints to Force / Prevent Usage

- Force the Optimizer to use:

```
…FROM dbo.MyDWTable
     WITH (INDEX (ix_cs_MyTable))…
```

- Prevent the Optimizer from using:

```
SELECT…
FROM dbo.MyDWTable
WHERE…
GROUP BY…
ORDER BY…
OPTION (IGNORE_NONCLUSTERED_COLUMNSTORE_INDEX);
```

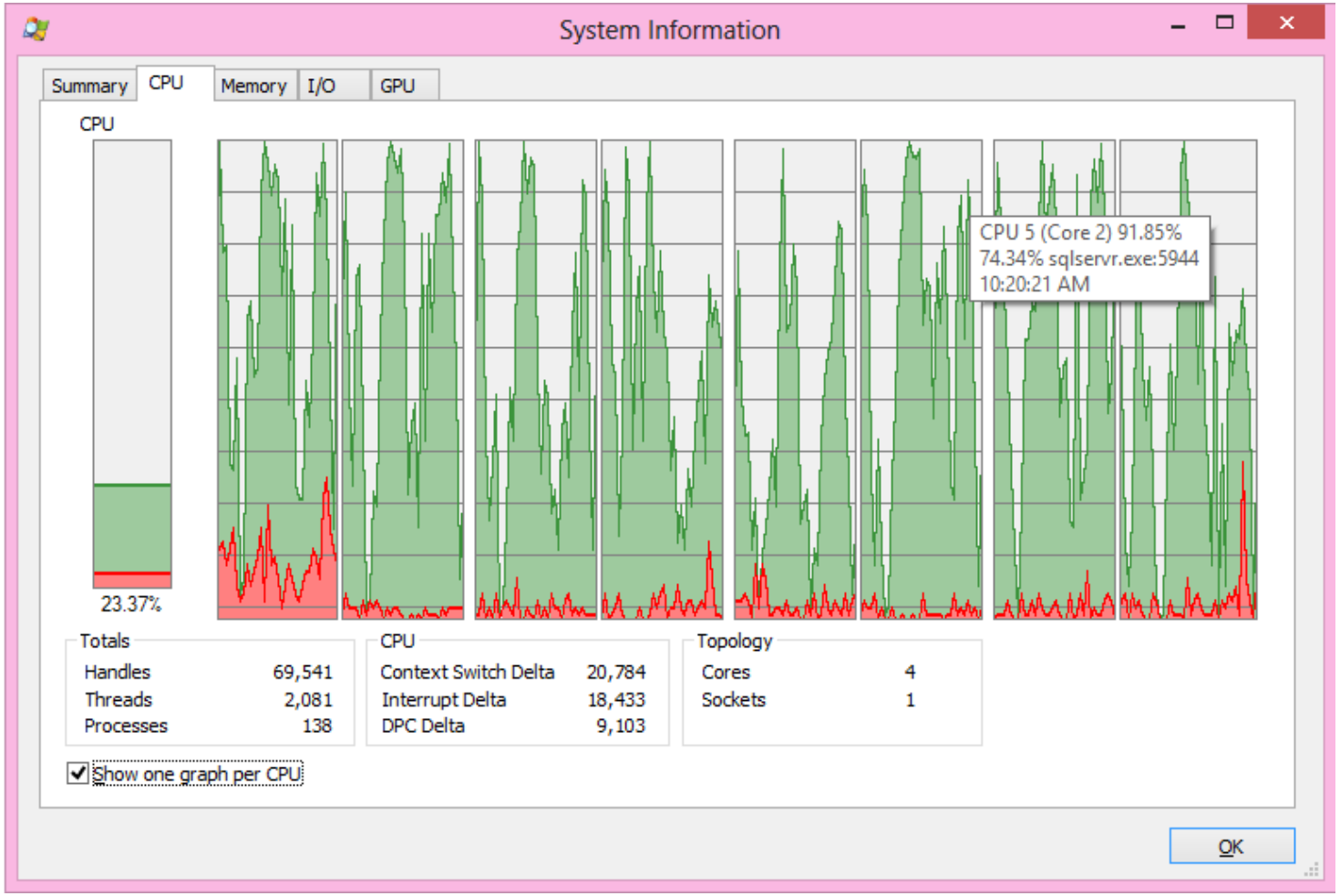
Demo: Columnstore Index DDL

Building Columnstore Indexes Fast!

- Memory resource intensive
  - Memory requirement related to number of columns, data, DOP
- Unit of parallelism is the segment
  - Lots of segments, lots of potential parallelism
- Low memory throttles parallelism
  - Increase the max server memory option
  - Set REQUEST_MAX_MEMORY_GRANT_PERCENT to 50
  - Add physical memory to the system
  - Adaptive index build process: adjust DOP and segment size

Columnstore ♥ Parallelism

- In general, SQL Server will leverage all available CPUs for creating indexes or issuing readahead for satisfying queries
  - True for columnstore or conventional indexes



Demo: Leveraging Resource Governor

Loading New Data

- Table with columnstore index can be read, but not updated
  - Partition switching is allowed (SQL Server 2012 & 2014)
  - INSERT, UPDATE, DELETE, and MERGE not allowed
- Methods for loading data
  - *Partition switching is a BEST PRACTICE for loading data in SQL Server 2012 & 2014*
  - Disable, update, rebuild
  - UNION ALL between large table with columnstore index and smaller updateable table

Loading New Data (2012): Trickle Loading

- Means by which to write data to SQL Server 2012
- Simple variation on classic partitioning
  - Large historical fact table with (read-only) columnstore index
  - Smaller table with contemporaneous data
    - Analogous to partition staging table
    - Same schema as historical table, except no columnstore index
    - New data is loaded in real-time in
  - Query via UNION ALL
  - Columnstore index is created on staging table during maintenance window, then switched in
  - New staging table created to host new data

Insert & Updating Data

- Bulk insert
  - Creates row groups of 1Million rows, last row group is probably not full
  - But if <100K rows, will be left in Row Store
- Insert/Update
  - Collects rows in Row Store
- Tuple Mover
  - When Row Store reaches 1Million rows, convert to a Columnstore Row Group
  - Runs every 5 minutes by default
  - Started explicitly by ALTER INDEX <name> ON <table> REORGANIZE

Batch Size Case Study

- Problem: Query Perf Degrading
  - Formerly minutes, worsening to 67 minutes
- Scenario
  - 1000 row inserts
  - 4,056 "OPEN" Row Groups
  - Join to Hekaton table
- Remedy
  1. REBUILD Columnstore Index
  2. Leverage 100,000 row inserts
- Outcome
  - 522 Row Groups
  - 23sec (x174!)
  - No loss of concurrency
- Recommendation
  - Leverage partitioning, result in fewer, larger Row Groups

Demo: Table Partitioning

Agenda

1. Introduction & Preamble
2. SQL Server 2012 & 2014: New! Improved! Features
3. **Columnstore Indexes**
    1. Overview
    2. Architecture
    3. SQL Server 2012 vs. 2014
    4. Implementation
    5. **Scenarios**
    6. Best Practices
4. More Info

Scenarios

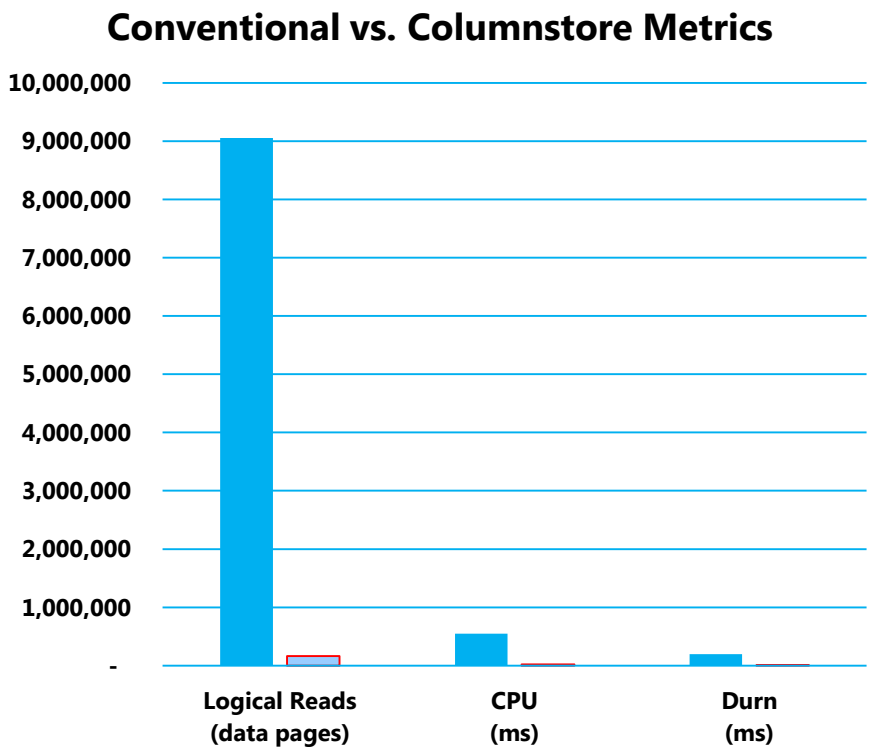
- Customer: Motricity: Case Study
- MSIT SONAR App: Aggregations
  - *booyah!*
- Customer: DevCon Security: Production App
  - *double-booyah!*
- Windows Telemetry—Watson
  - *triple-booyah!*
- MSIT Problem Management App
  - *booyah^{nth}*
- But, we still have room for improvement…

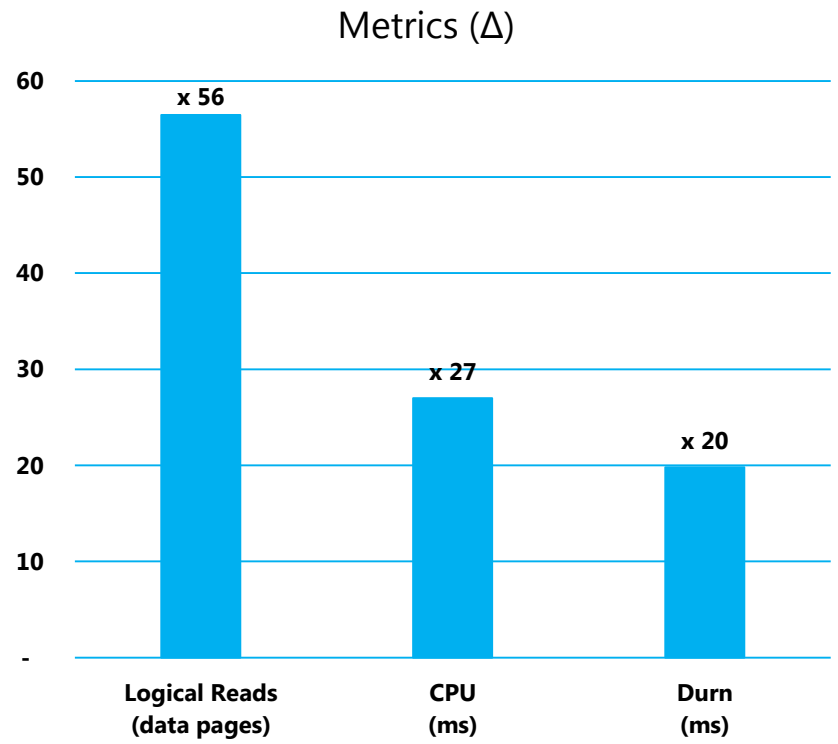
SQL CAT Customer: Motricity

- Sybase customer
- Demographic data & app required columnstore capabilities
- Proved out technology on Denali bits
- Numerous learnings discovered & incorporated into SQL Server 2012 RTM
- Production implementation leveraged partitioning & "trickle loading" (& AlwaysOn Availability Groups)
- Case Study: http://www.microsoft.com/casestudies/Microsoft-SQL-Server-2012-Enterprise/Motricity/Mobile-Advertiser-Makes-Gains-with-Easy-Migration-of-Sybase-Database-to-Microsoft/710000000170
- Customers: @armpitslinky, @SQLJackBurton,
  & the cuddliest customer ever in the SQL CAT Lab: @dsfnet

MSIT SONAR App PoC

	Logical Reads (8K pages)	CPU (ms)	Durn (ms)
Columnstore	**160,323**	**20,360**	**9,786**
Conventional Table & Indexes	**9,053,423**	**549,608**	**193,903**
Δ	**x56**	**x27**	**x20**





Customer: DevCon Security: Production App

- Revenue History Fact Table
- Table stats
  - 22M rows
  - 33GB data
  - 137 columns
- Indexing strategery challenging, especially for ad hoc queries

Customer: DevCon Security: Production App: SSRS

- Scenario 1: SSRS
  - Before: SSAS cube processing + batch process ran MDX to dump results into a separate table
  - "…hassle to maintain"
- Before: Queries against conventional table structures:
  - 10 - 12 seconds
- After: Queries against Columnstore:
  - 1 second
  - One order of magnitude faster

Customer: DevCon Security: Production App: Ad Hoc Queries

- Scenario 2: Ad Hoc Queries
- Before: Queries took 5 – 7 minutes (300 – 420 seconds)
- After: Columnstore: 1 – 2 seconds
  - Two orders of magnitude faster

Customer: DevCon Security: Production App: Outcomes

- Unsolicited feedback:
  - "What we found was pretty awesome."
  - "It has been a game changer for us in terms of the flexibility we can offer people that would like to get to the data in different ways."

  — Nathan Allphin, BI Team Lead
- Customer is deprecating this aspect of their SSAS infrastructure in favor of columnstore

Windows Telemetry—Watson WER Failure Event Hit Counting: Aggregate record of all occurrences of user mode failures on Windows machines worldwide

- **Data:** 61 billion rows, 6440 partitions, partitioned by date (12+ years), keyed on *Day, iBucket, LCID, OEM, OS Version*
- **ETL:** SSIS flat files into staging HEAP table, indexes built to match target, perform SWITCH (+300M rows daily)
- **Formerly hours, now 5-minute queries!** (50-100x gain)
  - Total hits by day, for all days
  - Filtered rollup on 3 dimensions


2DA8113A.sqlplan


C91A0610.sqlplan

Acknowledgements:
LeRoy Tuttle, Sr. SDE
Mahesh Jambunathan, SDE

- **Features/Tricks/Lessons:**
  - String values & SQLCLR UDT column located in dimension tables
  - Really bad perf hit if Foreign Keys are not trusted (is_not_trusted = 1)
  - Clustered index order can make huge difference in columnstore efficiency by grouping repeating values
  - MAXDOP configuration tradeoff; columnstore queries want it, other queries suck too much memory if MAXDOP is set too high
  - Lack of query partition elimination diminishes columnstore gains

MSIT Problem Management App

- *Interactive* app for which queries returned results in unpredictable time frame: "2 – 7 minutes"
- Wide table: 151 columns all commonly subject to interrogation
  - DB Dev troubleshooting time & expertise scarce
  - Indexing strategery difficult/impossible
- Table stats
  - 3.1M rows
  - 15GB data (*barely* fits in memory; lots of churn)
  - 1.7M data pages
- Columnstore index: 478 pages (memory resident)

MSIT Problem Management

❑ How to index a table with these columns?

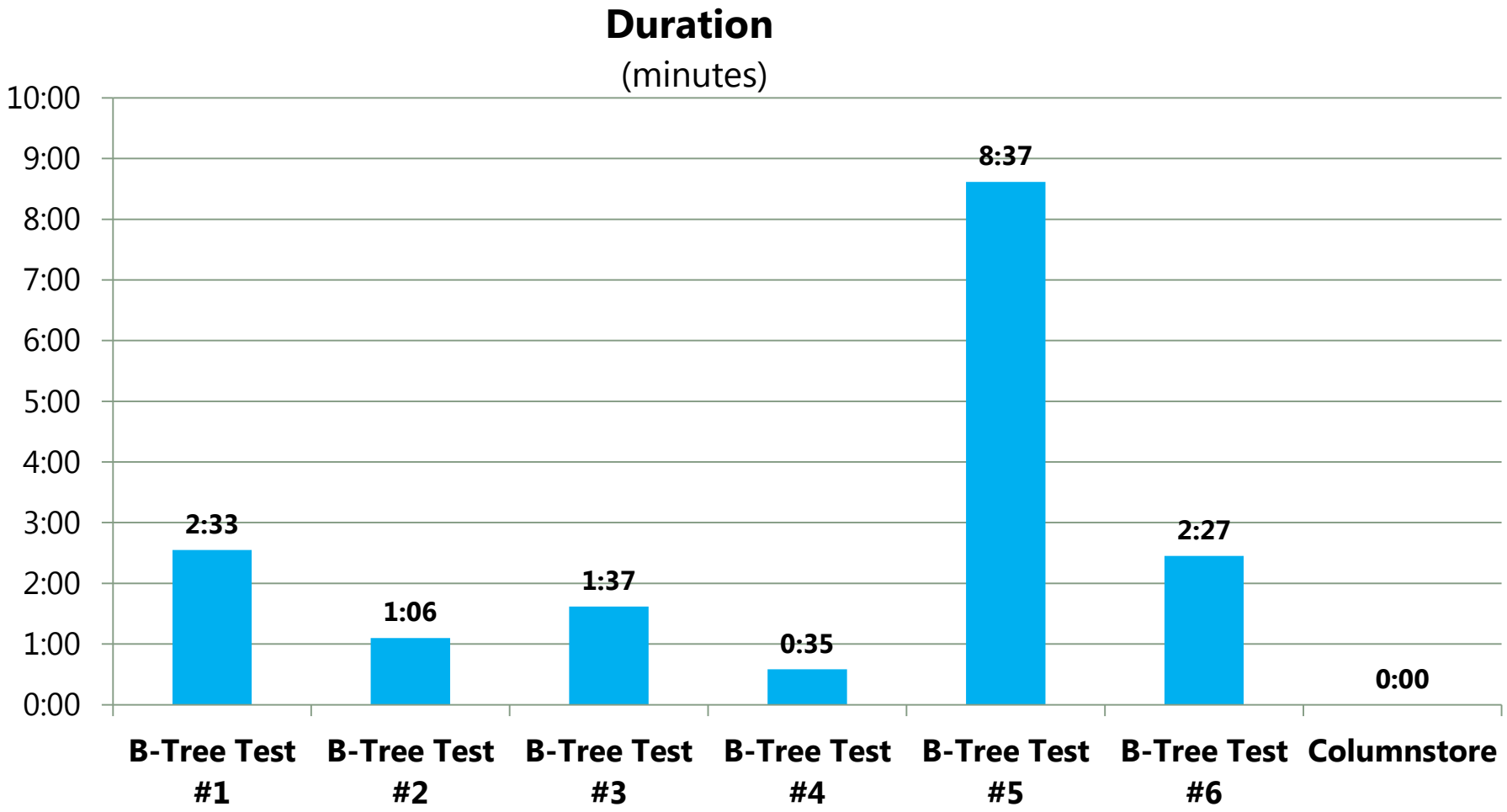
TicketFolderID, ModuleName, ModuleSubType, TicketID, CreatedDate, CreatedByAlias, CreatedByFullName, CreatedByLocationCountry, CreatedByLocationCity, CreatedByLocationBuilding, Source, ContactType, ContactAlias, FirstName, LastName, Email, BusinessCustomer, ContactLocationCountry, ContactLocationCity, ContactLocationBuilding, ContactLocationAltCountry, ContactLocationAltCity, ContactLocationAltBuilding, AssetType, AssetService, ServiceCategory, AssetName, AssetModel, AssetModel1, AssetDescription, AssetManufacturer, AssetMake1, State, Status, Summary, Impact, Urgency, Priority, ContactReason, Details, Environment, EnvironmentSpecific, ScheduledDate, CurrentL1, CurrentL2, CurrentL3, CurrentGroupID, CurrentGroup, CurrentSubGroupID, CurrentSubGroup, CurrentTeamID, CurrentTeam, CurrentIndividual, AssignedToName, SLADeadline, ResolvedAtDate, ResponseDeadline, RespondedAtDate, AffectedService, RequestedService, ResolutionCategory, CloseDetails, FinalStatus, FailureImpact, ServiceOutage, CIOutage, KBArticleAction, KBArticleTitle, KBArticleID, ResolvedByAlias, ResolvedByFullName, ResolvedByLocationCountry, ResolvedByLocationCity, ResolvedByLocationBuilding, ResolvedByEmployeeType, ResolvedByCompany, MajorIncidentReviewDate, MajorIncidentReviewOutcome, AddlContacts, OnBehalfOf, OriginalPriority, OriginalServiceCategory, OriginalAssetType, OriginalAssetService, OriginalAssetName, OriginalAssetModel, OriginalAssetDescription, OriginalAssetManufacturer, OriginalContactReason, OriginalL1, OriginalL2, OriginalL3, OriginalGroupID, OriginalGroup, OriginalSubgroupID, OriginalSubgroup, OriginalTeamID, OriginalTeam, OriginalIndividual, AltTelephone, AltLocation, LocationType, TicketLocation, ReactivationReason, ClosedDate, EmailSentToQueueDate, FirstNonAutoResponseDate, FirstNonAutoResponseActType, FirstEscalationL1toL2Flag, FirstEscalationL1toL2Date, L2EscalationGroupID, L2EscalationGroup, L2EscalationSubgroupID, L2EscalationSubgroup, L2EscalationTeamID, L2EscalationTeam, L2FirstNonAutoResponseDate, L2FirstNonAutoResponseActivity, FirstEscalationL2toL3Flag, FirstEscalationL2toL3Date, L3EscalationGroupID, L3EscalationGroup, L3EscalationSubgroupID, L3EscalationSubgroup, L3EscalationTeamID, L3EscalationTeam, L3FirstNonAutoResponseDate, L3FirstNonAutoResponseActivity, ThirdPartyPendStatusDuration, ThirdPartyReference, LastUpdatedDate, PendingClosure, ThirdPartyName, ThirdPartyOther, SubGroupTransferFlag, SubGroupResponseDeadline, SubGroupResponseDate, PrimaryConfigFolderID, ServiceConfigFolderID, KBArticleDetails, ClusterNodes, ResolvableByPreviousTeam, ResolveComments, ClientPendingStatusDurationMin, PartsPendingStatusDurationMin, ContactJobTitle, KBArticleCategory, KBType, KBNotes, MetCommunication, AvgCommunicationTime

Typical Query

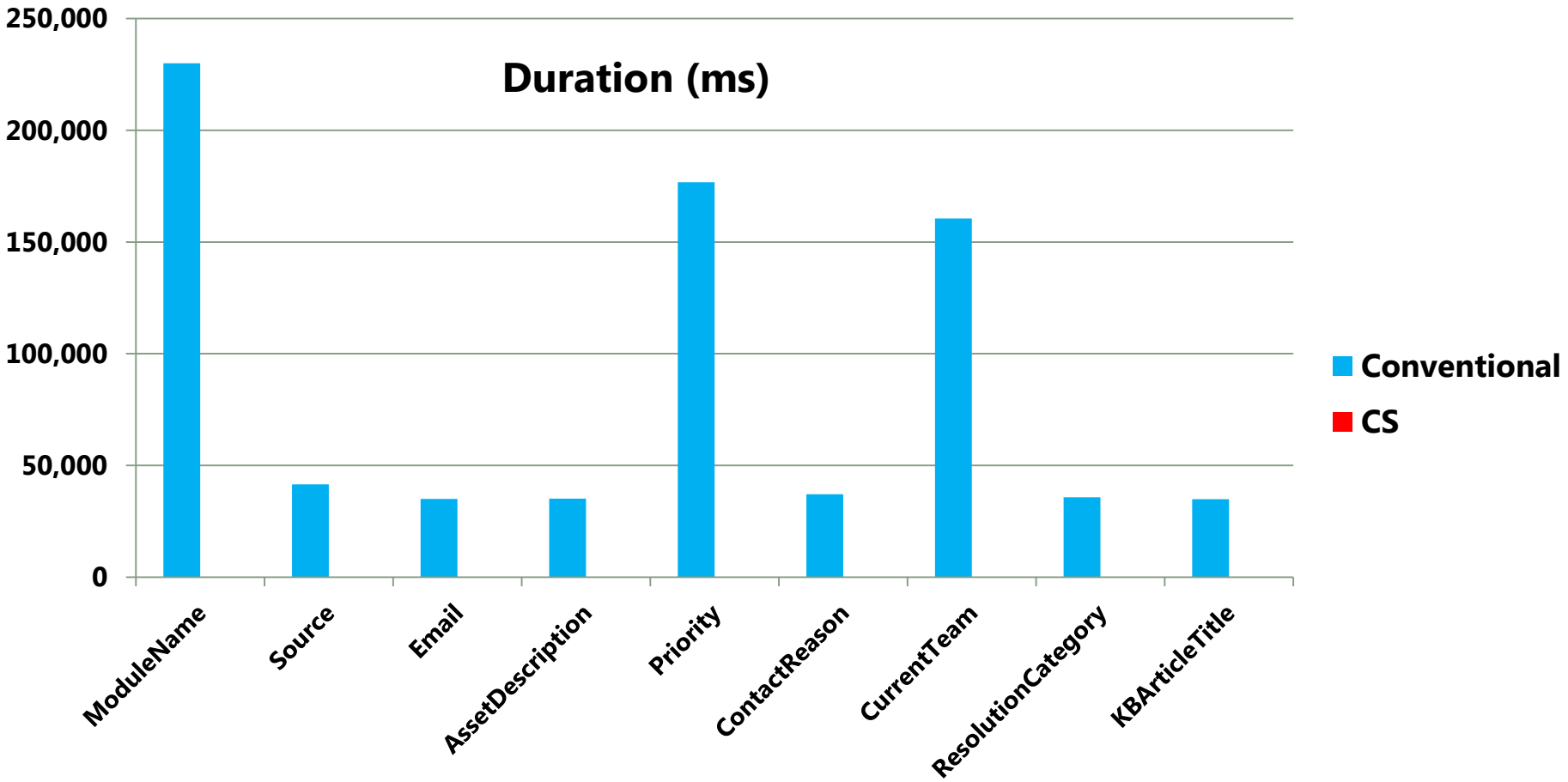
```
--CurrentGroup, CurrentSubGroup, ModuleName
SELECT TOP 10 CurrentGroup, CurrentSubGroup,
KBArticleTitle, COUNT(*) as RowCnt
    FROM dbo.ITSM01ISRM
    WHERE CurrentGroup = 'ExD Services'
       AND CreatedDate > dateadd(dd, -30, getdate())
       AND CurrentSubGroup = 'SAP'
       AND Status = 'Resolved'
    GROUP BY CurrentGroup, CurrentSubGroup, ModuleName
    ORDER BY COUNT(*) DESC
     --OPTION (IGNORE_NONCLUSTERED_COLUMNSTORE_INDEX);
```

Query Duration used to be Unpredictable

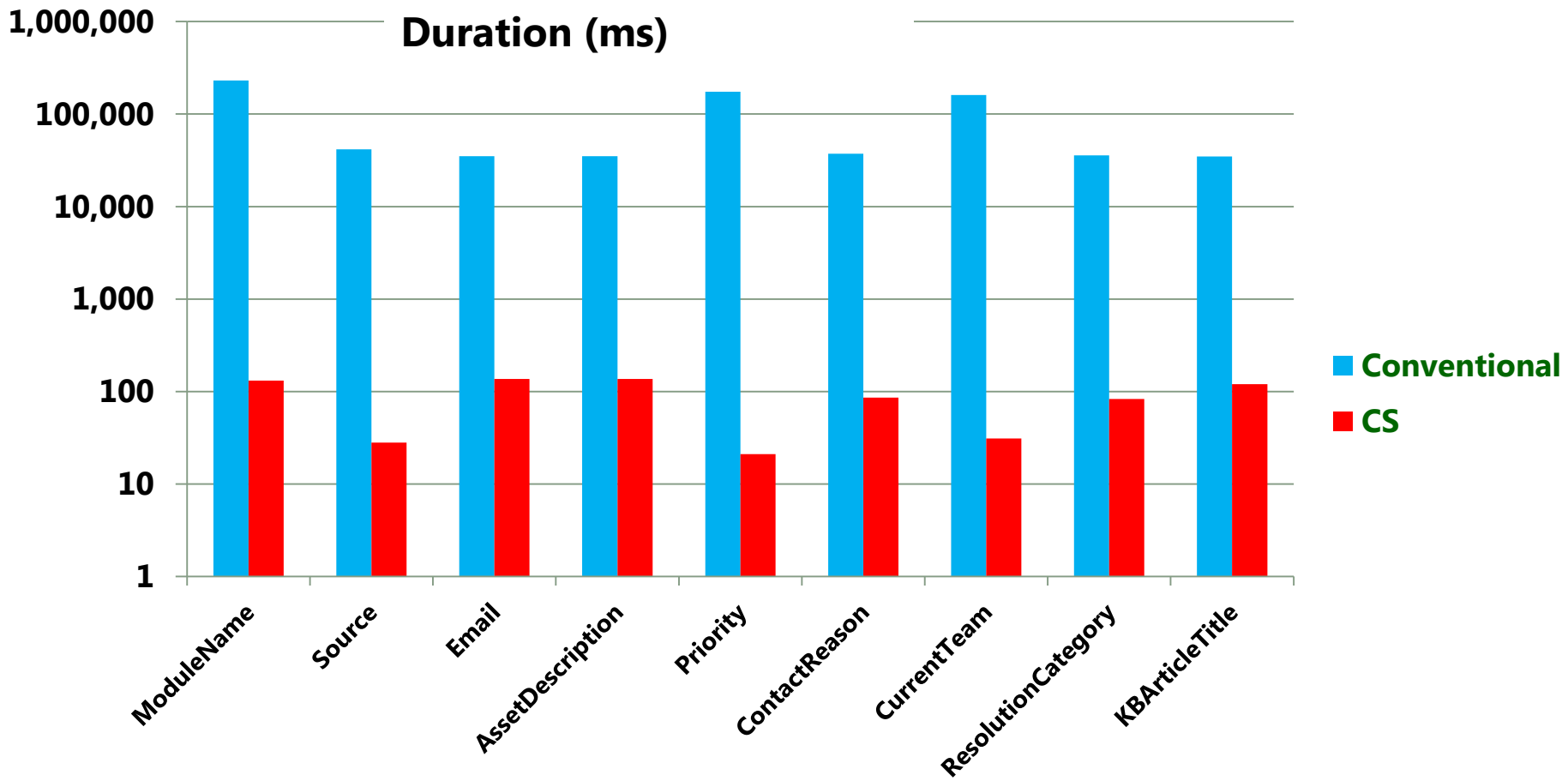
But Columnstore renders consistent subsecond results



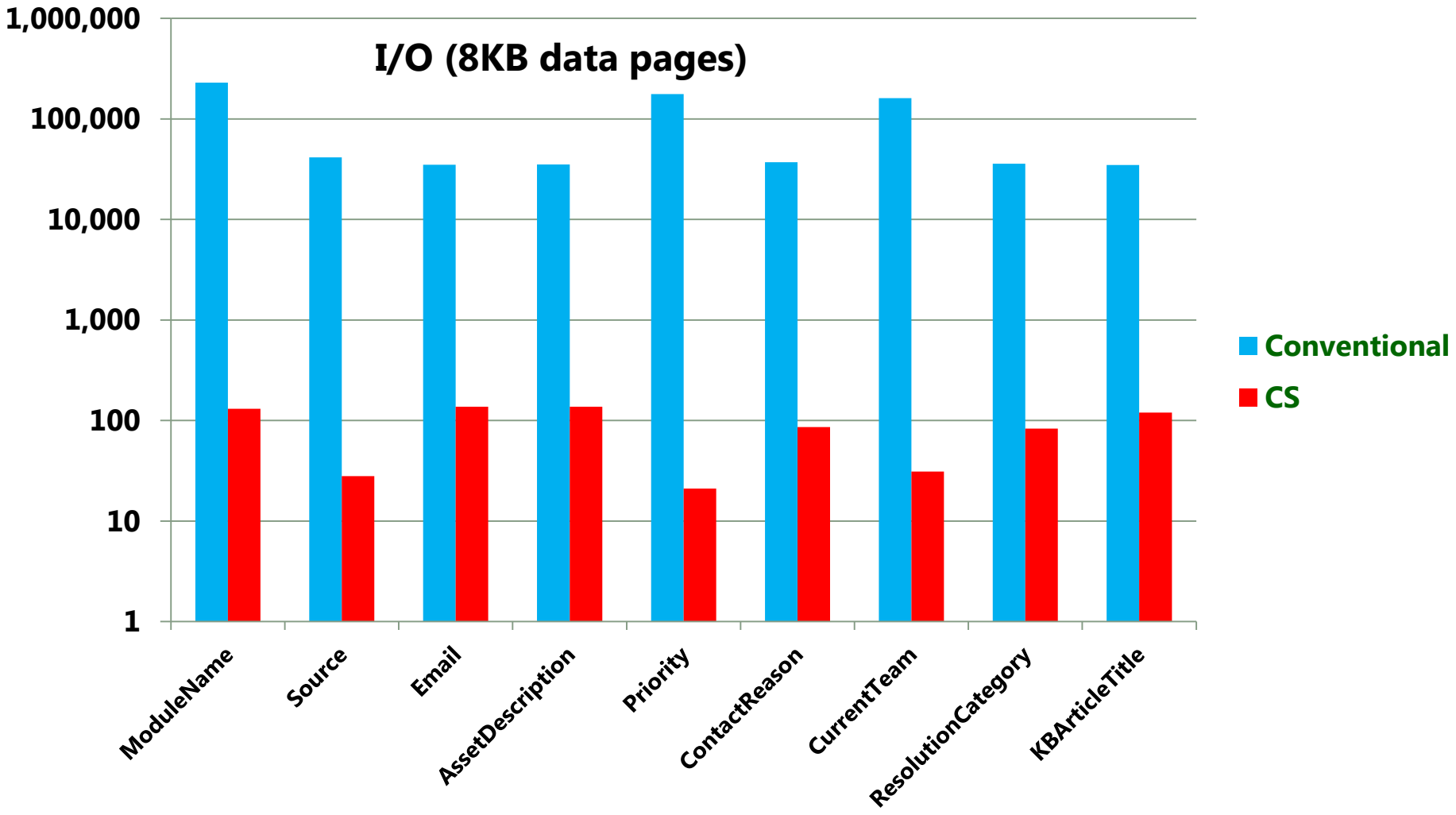
Conventional vs. CS Perf: Duration



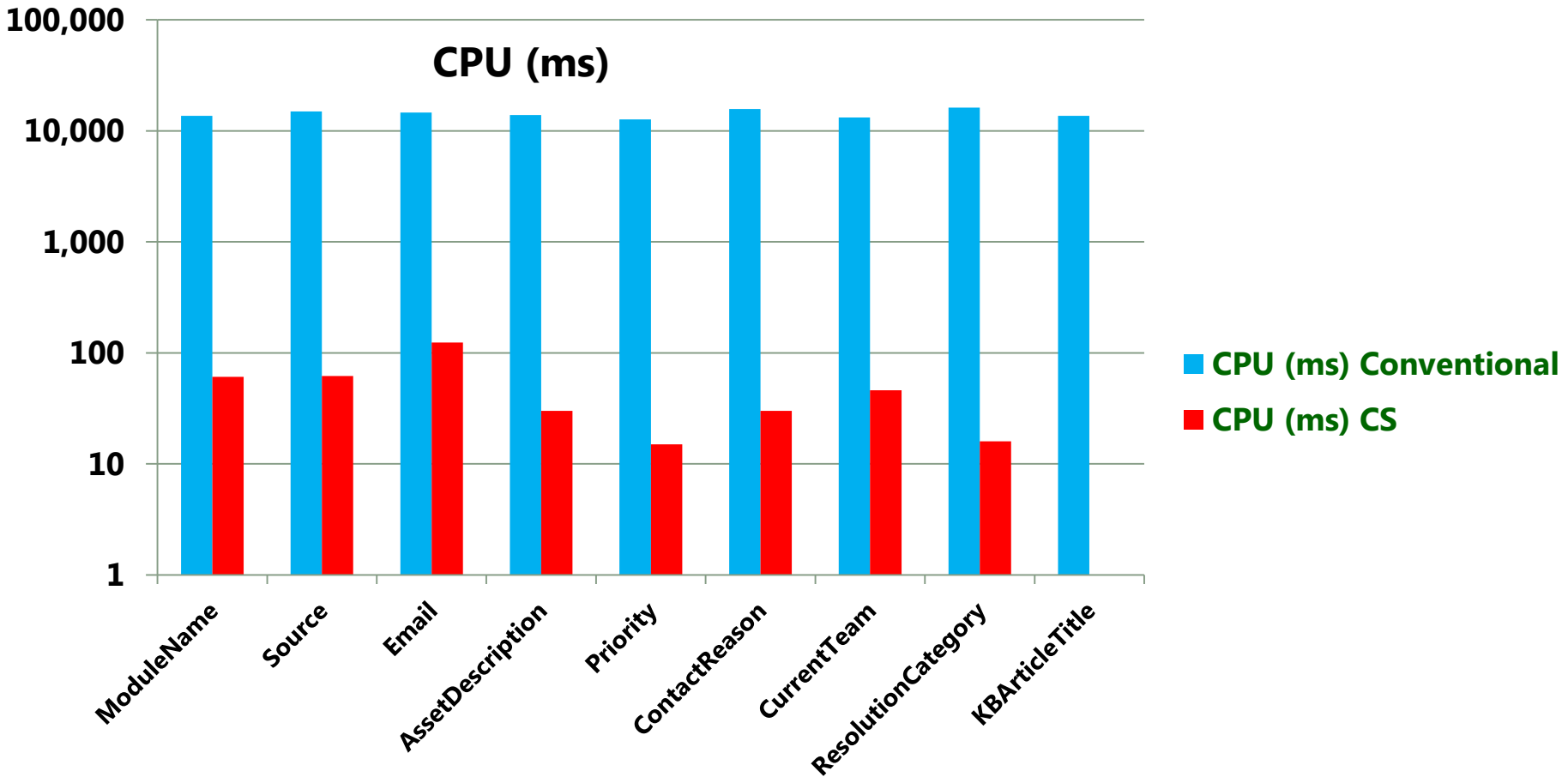
Conventional vs. CS Perf: Logarithmic



Conventional vs. CS Perf: I/O



Conventional vs. CS Perf: CPU



Columnstore Indexes:
Room for Improvement…

Agenda

1. Introduction & Preamble
2. SQL Server 2012 & 2014: New! Improved! Features
3. **Columnstore Indexes**
    1. Overview
    2. Architecture
    3. SQL Server 2012 vs. 2014
    4. Implementation
    5. Scenarios
    6. **Best Practices**
4. More Info

Best Practices

- Create CS indexes on “large” fact tables
- Leverage “star joins”
  - Joins on integer keys
- Include all columns of eligible datatypes
- Leverage Parallelism
- Provide sufficient memory
- Use in conjunction with partitioned tables

Query Optimization

- SQL Server 2012
  - We had a list of "gotchas" to watch out for, and workarounds
- SQL Server 2014
  - *Significant improvements to Optimizer*
- Good design practices for dimensional models
  - Star schema
  - Columnstores on Fact tables
  - Integer surrogate keys for joins to Dimension tables
- The power of the platform
  - You just write queries!
  - Logically no difference between Clustered Columnstores and regular tables

Key Learnings

- Some specific considerations in 2012 :
  - Read-only—*for now* (see references for Trickle Loading implementation)
  - Filtering optimizations in the storage engine are limited to numeric data types
  - Using OR statements in predicates results in less efficient plan vs. what is generated for row based indexes
  - Some OUTER JOINs
  - Large number of joins results in inability to use batch mode processing
  - Joining on string data types does not push bitmap filtering down into the storage engine
  - Optimizer may not optimize UNION ALL
  - Data type support is extensive but incomplete
- *AS NOTED: Significant improvements are implemented in SQL Server 2014*

Key Learnings (cont.)

- Query performance of columnstore indexes is good, even great—once you have the "*right*" plan & can leverage batch mode
- Able to meet/beat competitors in terms of performance; however query rewrite and schema changes may be required
- To get full parallelism of column store index builds may require significant memory grants. This can be problematic on wide or string heavy tables.
  - Parallelism on index build is not supported if table has < 1 million rows.
    - (Because it's superfluous)
- AlwaysOn availability groups work well for many DW workloads
  - Able to keep up with expected bulk load rates and columnstore index builds
  - Observed some upper limits on the "bytes per second" to the replica in the 40MB/s*. This caused the replica to fall behind for regular index build scenarios and more intensive loads.

*Under review with development team

Agenda

1. Introduction & Preamble
2. SQL Server 2012 & 2014: New! Improved! Features
3. Columnstore Indexes
    1. Overview
    2. Architecture
    3. SQL Server 2012 vs. 2014
    4. Implementation
    5. Scenarios
    6. Best Practices
4. **More Info**

Columnstore Indexes: More Info

The following is a must-read:

SQL Server Columnstore Index FAQ

Eric Hanson, Susan Price, etc.
http://social.technet.microsoft.com/wiki/contents/articles/3540.sql-server-columnstore-index-faq-en-us.aspx

Microsoft Internal:

SQL 14 What's new with AlwaysOn and Column Store Indexes

Other references:

Forthcoming: My blog http://blogs.msdn.com/jimmymay

Columnstore Indexes
http://msdn.microsoft.com/en-us/library/gg492088(SQL.110).aspx

SQL Server Columnstore Performance Tuning
http://social.technet.microsoft.com/wiki/contents/articles/4995.sql-server-columnstore-performance-tuning.aspx

Trickle Loading with Columnstore Indexes
http://social.technet.microsoft.com/wiki/contents/articles/trickle-loading-with-columnstore-indexes.aspx

How do Column Stores Work?
Thomas Kejser
http://blog.kejser.org/2012/07/04/how-do-column-stores-work

This is Columnstore – Part 1
Gavin Payne, MCM
http://gavinpayneuk.com/2012/07/22/this-is-columnstore-part-1

Saturday Keynote: Inside SQL Server 2012's Columnstore Index
http://sqlbits.com/Sessions/Event10/Saturday_Keynote

Questions?

Thank You for Attending