Sepand Gojgini

PASS SQLSATURDAY

ColumnStore Index Primer

SQLSaturday Sponsors!

Titanium & Global Partner	Microsoft
Gold	COZYROC™ Go to the next level; amazon web services
Silver	XGURU; Datrium; Quest
Bronze	sqlbi; BORNSQL

Without the generosity of these sponsors, this event would not be possible! Please, stop by the vendor booths and thank them.

Background

- Product Director at Datateam
- 10+ years of experience designing data model and ETL engines used globally
- Worked on world biggest relational Data Warehouse at Amazon
- Professional Dodgeball player (paid in Beer and Nachos)

Agenda

Overview

ColumnStore: What?

It is data logically organized by rows and column but physically stored in columnar data format

- Data Is compressed, managed and stored as collection partial columns

ColumnStore: Why?

Designed to optimize access to large DWs

Optimized for join on integer keys, aggregations, scans, reporting

Faster query response time

Transparent to the application or reports

Details

- Highly compressed – Allows more data remain in RAM
- Aggressive Read Ahead
- Processes data in units called “batches”

Non-Clustered ColumnStore

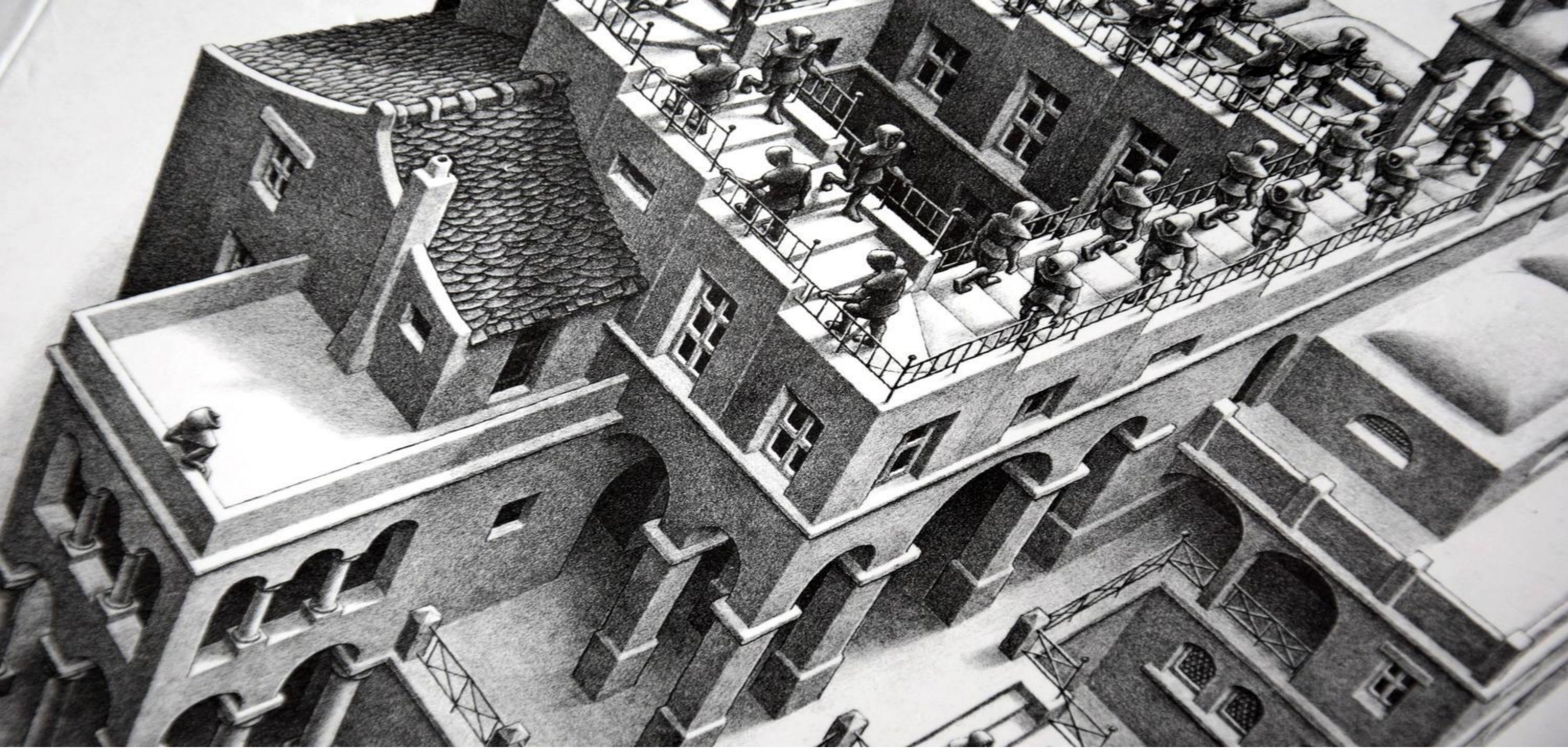
- Does not need to include all of the columns in a table
- Can be combined with other indexes on the table
- After being added to table it becomes Read-Only (2012/2014)
- Requires additional storage to store a copy of column in the index

```
CREATE NONCLUSTERED COLUMNSTORE INDEX <index name>
ON <table> (<columns list>);

CREATE NONCLUSTERED COLUMNSTORE INDEX myCSIndex
ON Customers (CustomerID, CompanyName, ContactName);
```

Clustered ColumnStore

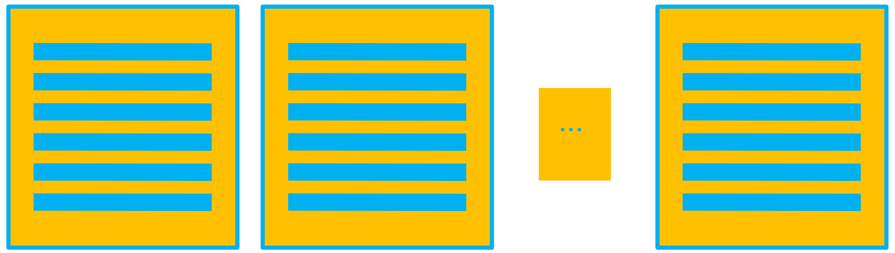
- Available on Enterprise edition starting with SQL Server 2014/2016
  - SQL Server 2016 SP1 All Editions
- Include ALL the column in table
- Is the only index on the table
- Replaces B+ tree for storage with Columnar technology
- Table remain updatable*

```
CREATE CLUSTERED COLUMNSTORE INDEX <index name>
ON <table>;

CREATE CLUSTERED COLUMNSTORE INDEX myCSIndex
ON Customers;
```

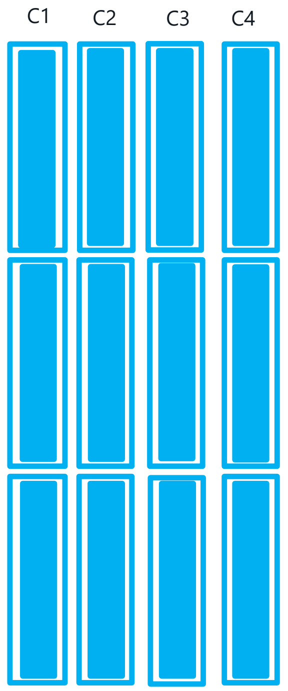


Architecture

ColumnStore Vs. RowStore



❑ Heaps, B-trees store data row-wise

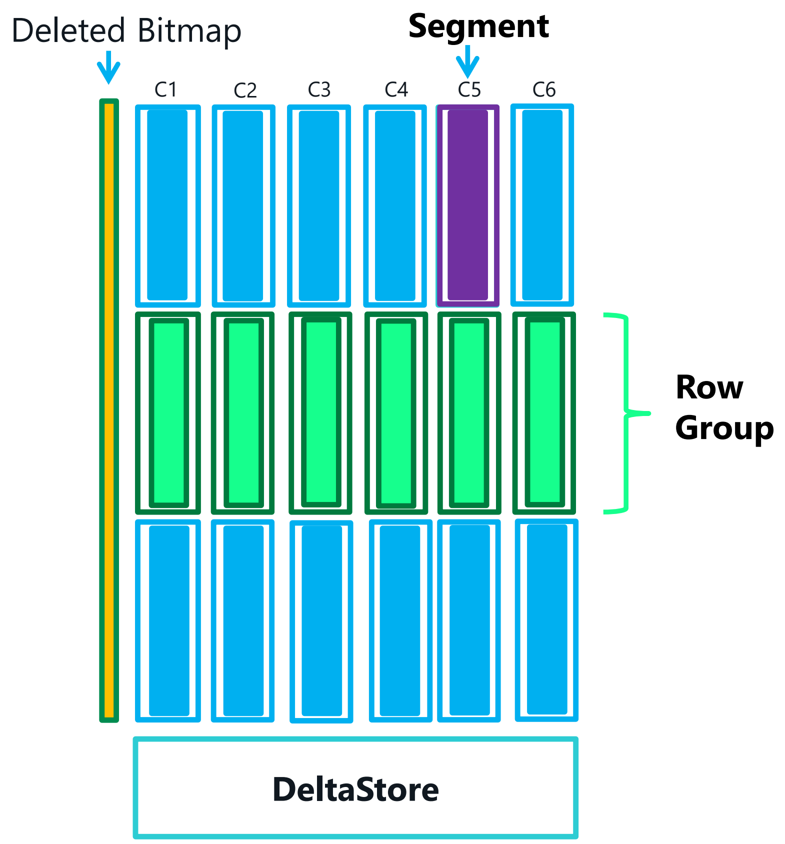


❑ Columnstore indexes store data column-wise
  - Each page stores data from a single column

❑ Highly compressed
  - About 10x better than Row Store
  - More data fits in memory

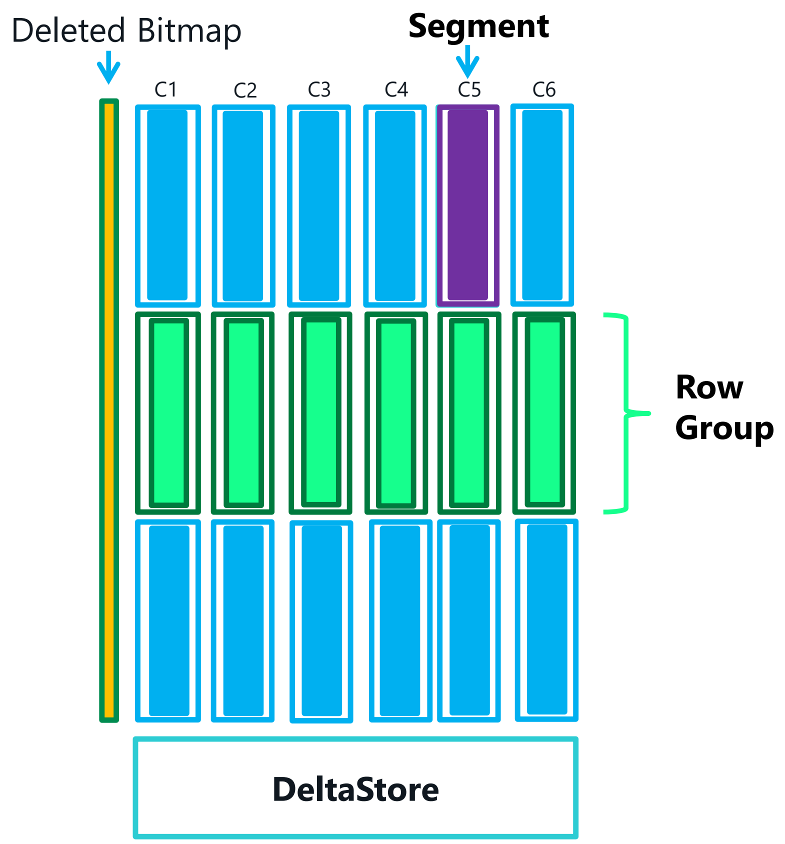
❑ Each column accessed independently
  - Fetch only needed columns
  - Can dramatically decrease I/O

Terminology – Part I



- 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 independently
  - Segment is unit of transfer between disk and memory
  - Each segment stored in a separate LOB
- Row Group
  - Up to 1 million logically contiguous rows
  - Collection of column segments

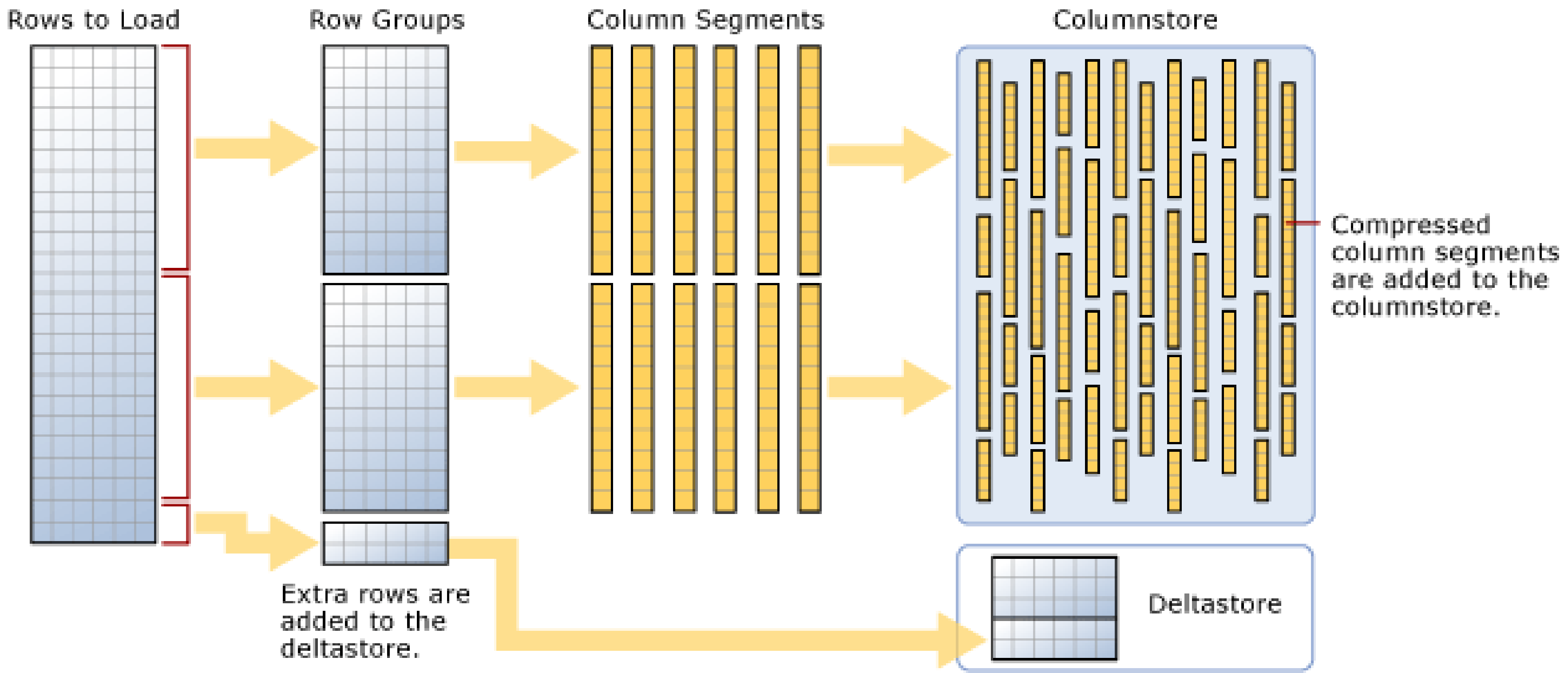
Terminology – Part II



- DeltaStore
  - Introduced with Clustered ColumnStore in SQL Server 2014
  - Is a RowStore table that holds rows until the number of rows is large enough to move into ColumnStore
  - Uses traditional B-Tree for storage
  - There could be multiple DeltaStores per ColumnStore
  - Holds Maximum of 1,048,576 or 2^20 rows before new one is created
- Deleted Bitmap
  - Identifies records deleted from Row Group

Terminology at Glance

Data is highly compressed. Dramatically reduced IO. More Data can fit into memory.



Rows to Load
Row Groups
Column Segments
Columnstore
Compressed column segments are added to the columnstore.
Extra rows are added to the deltastore.
Deltastore

Query Execution Walkthrough

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

Up to 1M rows

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

Represents Up to 1M rows

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

Some segments will compress more than others

*Encoding is explained in future slide

Fetch only needed columns

```
SELECT ProductKey, SUM (SalesAmount)
FROM SalesTable
WHERE OrderDateKey < 20101108
GROUP BY ProductKey
```

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

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

Under the covers

Basic Operations

Inserts[1]:

Added to one of currently open **DeltaStores**

Deletes:

If deleted row is found in **RowGroup** then the Deleted Bitmap information is updated with RowId of respective row

If deleted row is still in **DeltaStore** then it is simply removed from B-Tree

Updates:

They are handled as combination of delete and insert using workflow above

1 Bulk Insert API skips DeltaStore

Creation of new Row Group

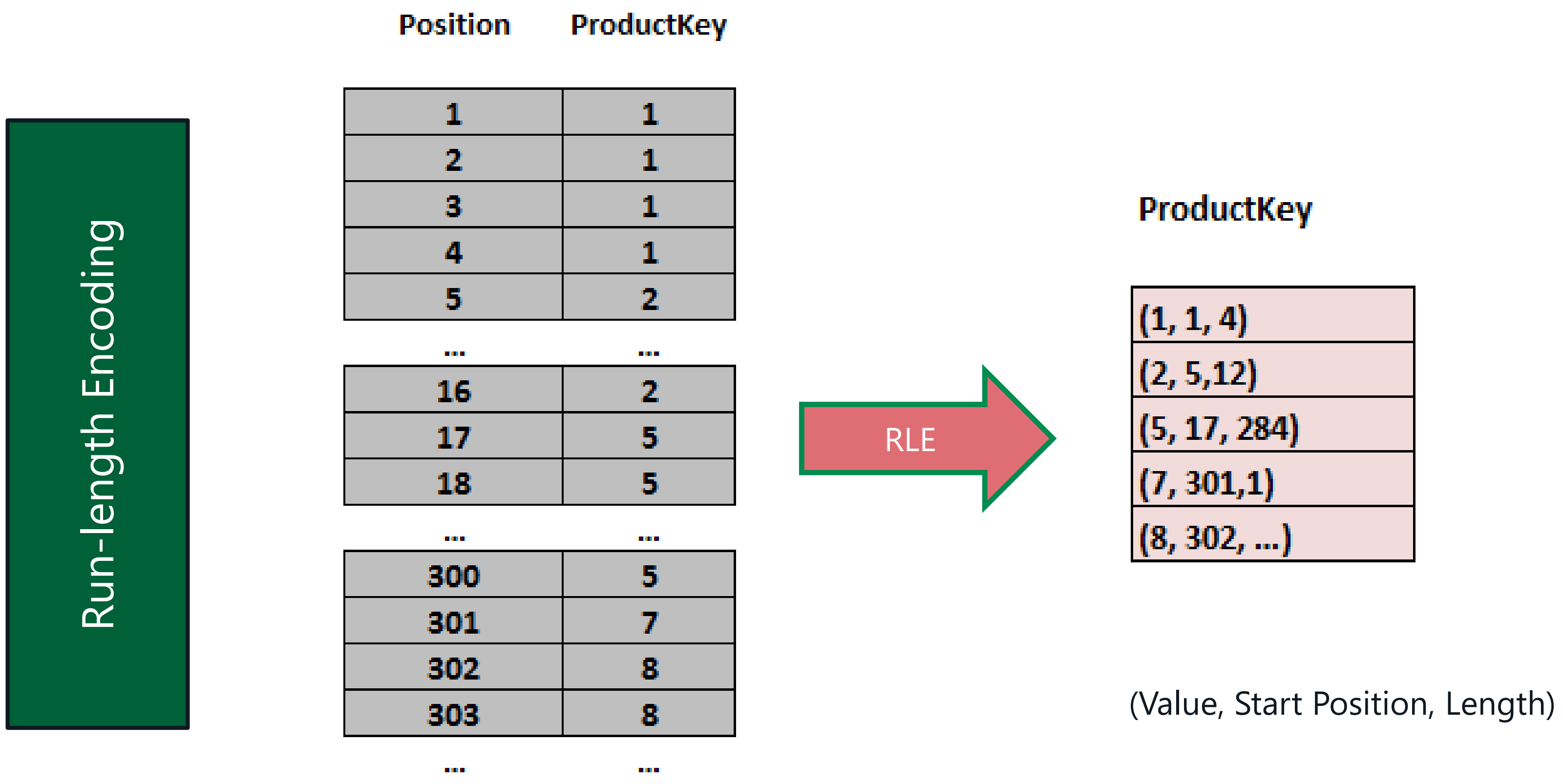
Tuple Mover:

Once a DeltaStore reaches 1,048,576 it is closed
Tuple Mover runs every 5 minutes and converts any closed **DeltaStore** into **Row Group**
Added to one of currently Open **DeltaStores**
There is single Tuple Mover per instance

Bulk Insert API:

If there are more than 102,400 records inserted into table in single operation it is triggered
Skip insertion of rows into **DeltaStore** and create a new **Row Group** for that operation

Data Encoding - Integer

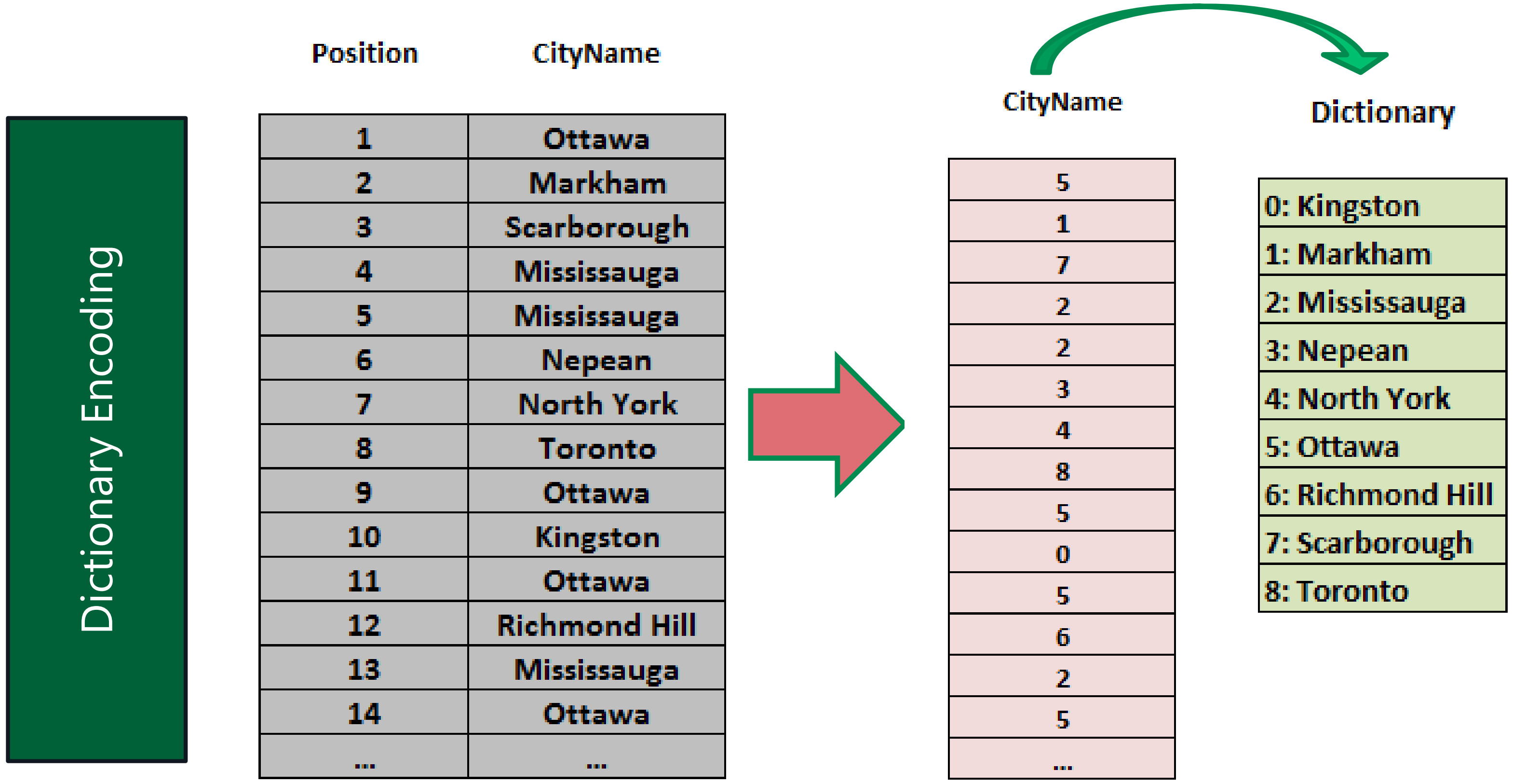


Position	ProductKey
1	1
2	1
3	1
4	1
5	2
...	...
16	2
17	5
18	5
...	...
300	5
301	7
302	8
303	8
...	...

ProductKey
(1, 1, 4)
(2, 5,12)
(5, 17, 284)
(7, 301,1)
(8, 302, ...)

(Value, Start Position, Length)

Data Encoding - VarChar



Position	CityName
1	Ottawa
2	Markham
3	Scarborough
4	Mississauga
5	Mississauga
6	Nepean
7	North York
8	Toronto
9	Ottawa
10	Kingston
11	Ottawa
12	Richmond Hill
13	Mississauga
14	Ottawa
...	...

CityName
5
1
7
2
2
3
4
8
5
0
5
6
2
5
...

Dictionary

0: Kingston
1: Markham
2: Mississauga
3: Nepean
4: North York
5: Ottawa
6: Richmond Hill
7: Scarborough
8: Toronto

Summary

SQL Server 2012 Limitation

- 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 updatable
- Not compatible with indexed views, filtered indexes, sparse columns, computed columns
- Only Inner Join operation is supported
- Datatype support significant but not complete
    - int, real, string, money, datetime, decimal <= 18 digits

SQL Server 2014 Improvement

- Clustered ColumnStore – Underlying storage is Columnar
- Query rewrite is no longer necessary in order to use ColumnStore
- Clustered ColumnStore is now updatable
  - Non-Clustered ColumnStore is still ready only
- Datatype support has been expanded
  - Everything except: blobs, CLR, NVarChar(max), VarBinary(max), XML, Spatial
- Improvement to Query Engine
  - Support for all flavors of JOINs
  - OUTER JOIN
  - Semi-join: IN, NOT IN
  - UNION ALL
  - Scalar aggregates
  - Mixed mode plans
  - Improvements in bitmaps, spill support, …
- Archival Compression was introduced
  - approximately 27% space saving using second compression at page level

SQL Server 2016 Improvements

- Additional B-Tree Indexes on Clustered ColumnStore index
- Updatable Non-Clustered ColumnStore
- Support for filtered Non-Clustered ColumnStore
- Non-Clustered ColumnStore support for In-Memory Tables
- Improvement to Query Engine
  - Expanded Support for Batch mode execution
  - Sort, Aggregates with multiple distinct functions
  - Window Aggregate functions
  - Window Aggregate Analytical functions
  - Single-Threaded queries can also run in batch mode

SQL Server 2016 SP1

- Now available in all editions
- ColumnStore Segment Cache
  - Standard Edition – 32 GB
  - Web Edition – 16 GB
  - Express Edition – 352 MB
- MAXDOP
  - Standard Edition – 2 Cores
  - All Other Editions – 1Core
- Non-Enterprise Limitations
  - Aggregate Pushdown = No
  - String Predicate Pushdown = No
  - Local Aggregation = No
  - Index Build/Rebuild = Limited to 1 Core
  - SIMD Support – No

Questions?

Resources

ColumnStore Overview – MSDN

https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview

Niko Neugebauer

http://www.nikoport.com/columnstore/

SQL Server 2016 SP1

https://docs.microsoft.com/en-us/sql/sql-server/editions-and-supported-features-for-sql-server-2016

SQL Server Tiger Team

https://blogs.msdn.microsoft.com/sql_server_team/tag/columnstore-index/