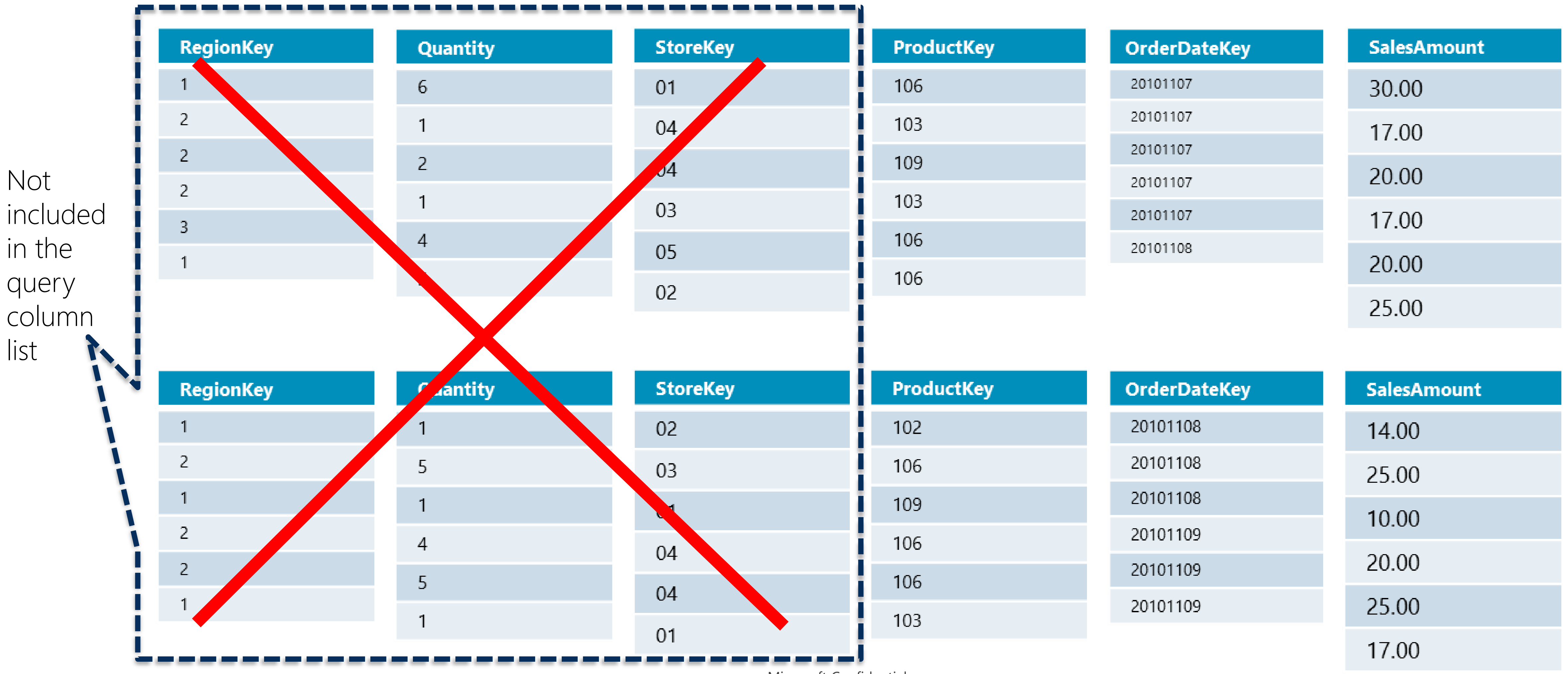
Segment Elimination

1. Fetches only needed columns

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



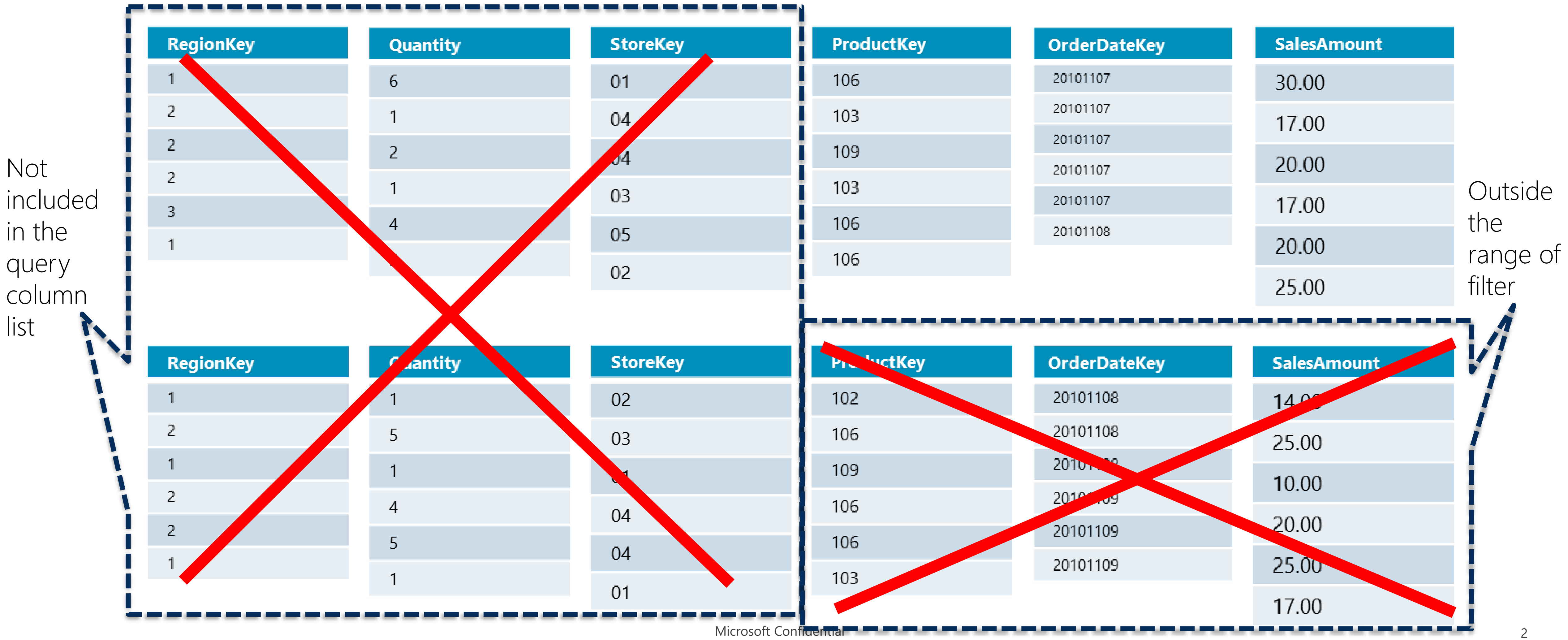
RegionKey	Quantity	StoreKey	ProductKey	OrderDateKey	SalesAmount
1	6	01	106	20101107	30.00
2	1	04	103	20101107	17.00
2	2	04	109	20101107	20.00
2	1	03	103	20101107	17.00
3	4	05	106	20101107	20.00
1	[illegible]	02	106	20101108	25.00

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

Fetch Only Needed Segments

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

2. Fetches only needed segments



Not included in the query column list
Outside the range of filter

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

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

Minimum and Maximum Value

sys.column_store_segments

Important columns for segment elimination

Column name	Data type	Description
column_id	Int	ID of the columnstore column
segment_id	Int	ID of the column segment
min_data_id	Bigint	Minimum data id in the column segment
max_data_id	Bigint	Maximum data id in the column segment

Limitation

Segment elimination works for the data types:

- Number
- Date
- Time

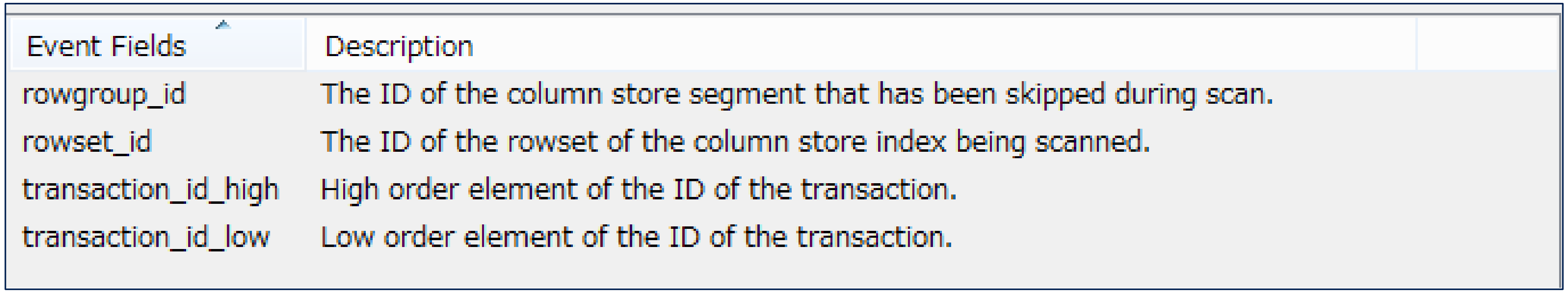
String data types aren't supported

Tracking Segment Elimination

xEvent

name	object_type	description
column_store_segment_eliminate	event	Occurs when a filter eliminates a column store segment during scan

Event Fields:

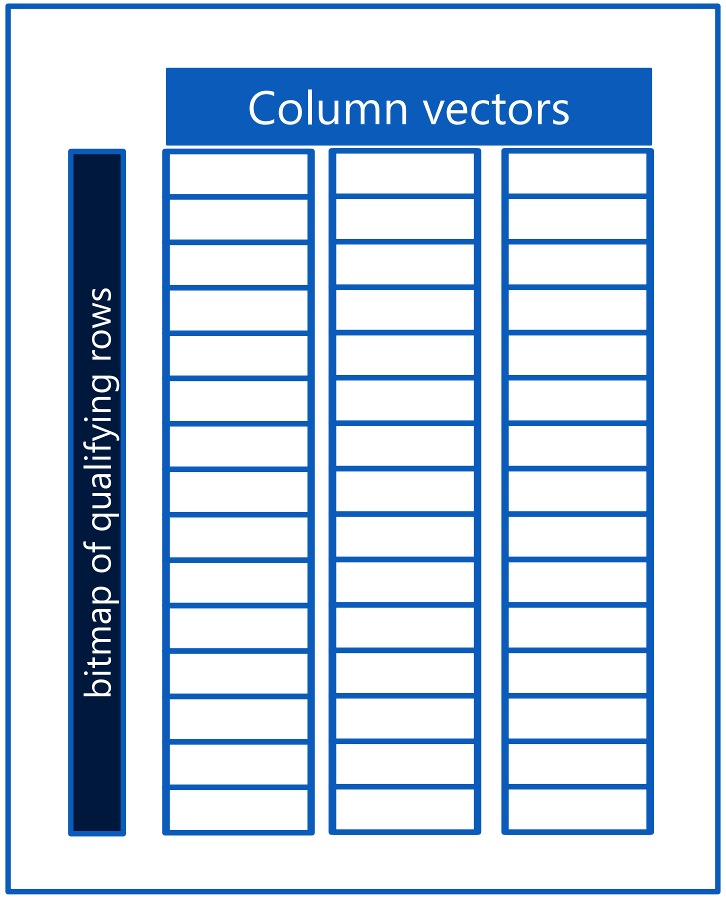


Event Fields	Description
rowgroup_id	The ID of the column store segment that has been skipped during scan.
rowset_id	The ID of the rowset of the column store index being scanned.
transaction_id_high	High order element of the ID of the transaction.
transaction_id_low	Low order element of the ID of the transaction.

Batch Mode Processing QP Vector Operators

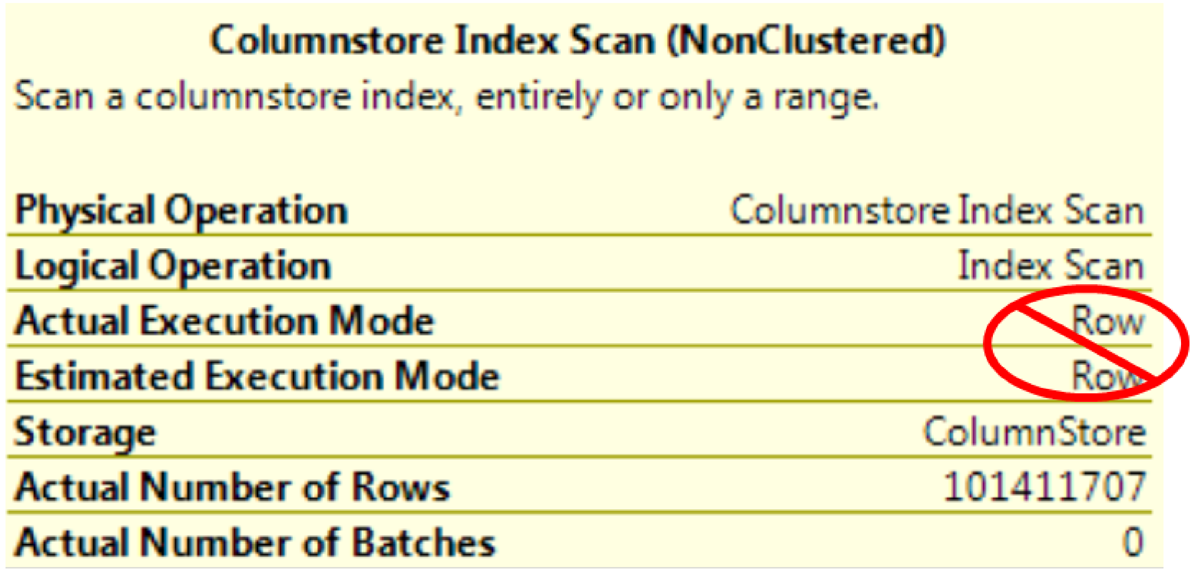
- Process ~1000 rows at a time
- Batch stored in vector form
  - Optimized to fit in cache
- Vector operators implemented
  - Filter, hash join, hash aggregation, …
- Greatly reduced CPU time (7 to 40X)

Batch object



Batch Mode on SQL Server 2012

Several engine limitations that can cause queries to run in row mode instead of batch mode….



Columnstore Index Scan (NonClustered)

Scan a columnstore index, entirely or only a range.

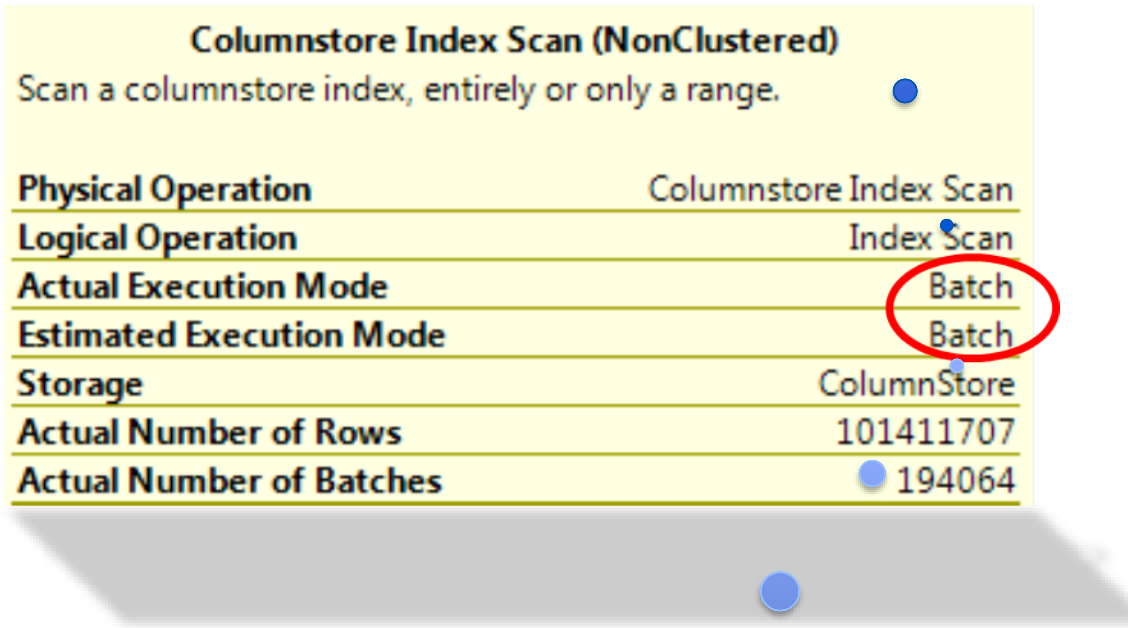
Physical Operation	Columnstore Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	ColumnStore
Actual Number of Rows	101411707
Actual Number of Batches	0

Use Outer Join and Still Get the Benefit of Batch Processing
Work Around Inability to get Batch Processing with IN and EXISTS
Perform NOT IN and Still Get the Benefit of Batch Processing
Perform UNION ALL and Still Get the Benefit of Batch Processing
Perform Scalar Aggregates and Still get the Benefit of Batch Processing
Maintaining Batch Processing with Multiple Aggregates Including one or More DISTINCT Aggregates

Improvements of SQL Server 2014

DOP > = 2

- Support for all flavors of JOINs
- UNION ALL
- Scalar aggregates
- Mixed mode plans



Columnstore Index Scan (NonClustered)

Scan a columnstore index, entirely or only a range.

Physical Operation	Columnstore Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Batch
Estimated Execution Mode	Batch
Storage	ColumnStore
Actual Number of Rows	101411707
Actual Number of Batches	194064

Segment Elimination controls DOP

Newly Supported Joins

Outer join

Semi-join

- IN
- NOT IN (Fullouter join, RightOuter join, Rightsemi join, Rightantisemi join, LeftOuter join, Leftsemi join, Leftantisemi join)

Global Batch Aggregation

SQL 2012

Each local node will feed data to global node

Global node processes data in row mode

SQL 2014

Global node runs in batch mode processing

Improves scenarios with large aggregation output

- Process the same data with less memory than local batch aggregation
- Better performance than local batch aggregation, example big hash tables
- Removes the need for row mode aggregation in mostly batch query plans, resulting in less data conversion and better management of granted memory

Mixed Execution

Transition between batch and row mode

SQL Server 2012

- Only at prescribed points in the plan

SQL Server 2014

- Any point in an execution plan

Statistics for Columnstore Index

The needs for statistics

- Histogram of statistics is required for query plan generation for Columnstore indexes used by the optimizer

Best Practices

- Keep statistics up to date
- Create multicolumn statistics on correlated columns

Sampling for Statistics of Columnstore

SQL Server 2012 (non-clustered columnstore)

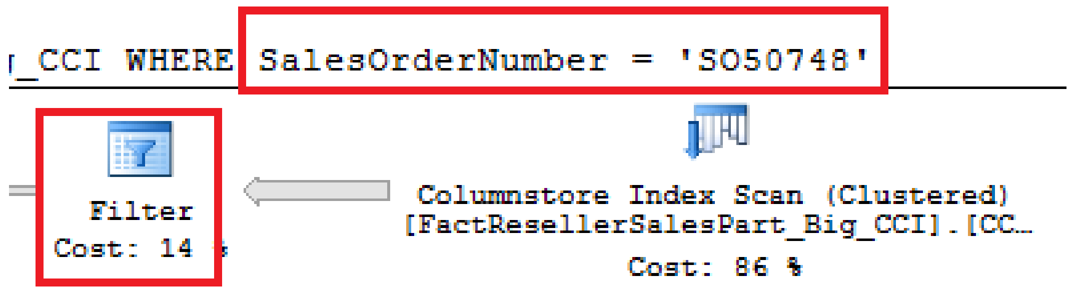
- Statistics are computed from the base data
- Page sampling from B-tree/heap

SQL Server 2014

- Cluster sampling
  - For dictionary creation
  - Sampling from limited number of segments and row groups
- Truly random sampling
  - For creation of histogram for query optimization
  - Row level sampling from all segments
  - More accurate than cluster sampling and B-tree page level sampling

Design Out Strings from ColumnStores

- Joining on Strings can be slow
- Factor strings out to dimensions
- It’s generally good DW design practice anyway



```
_CCI WHERE SalesOrderNumber = 'SO50748'
```

Filter
Cost: 14 %

Columnstore Index Scan (Clustered)
[FactResellerSalesPart_Big_CCI].[CC…
Cost: 86 %

Dimensions and Fact Tables

Date	LicenseNum	Measure
20120301	XYZ123	100
20120302	ABC777	200

Date	LicenseId	Measure
20120301	1	100
20120302	2	200

LicenseId	LicenseNum
1	XYZ123
2	ABC777

Update Effect on Query Execution

Delta store	• Parallel scan
Delete Bitmap	• Deleted row is just skipped while scanning
Large volumes of delete rows	• Rebuild index is required
Segment Elimination	• Minimum and maximum values aren't modified by Columnstore; index is rebuilt

Best Practices

- Create columnstore index on "large" fact tables
- Leverage "star joins"
- Joins on integer keys
- Leverage Parallelism
- Provide sufficient memory
- Use in conjunction with partitioned tables

Non-Clustered Columnstore indexes

Do we still need them?

Yes, if you need constraints or triggers on the table

- Creating a CCI will fail if there is a B-tree enforcing a key constraint
- However, you won't be able to update the table

No, if constraints are not needed

- Create table and add a clustered columnstore index
- No other indexes to worry about
- Can insert / update / delete in the table
- Consistent fast query performance

Updating Non-Clustered Columnstore

Disable index, update data, rebuild

- or -

Use partition switching

- or-

Use delta table and UNION ALL