SQL Server 2012:
Nonclustered Columnstore Indexes

Module 2: Nonclustered Columnstore Index Fundamentals

Joe Sack

Joe@SQLskills.com

pluralsight
hardcore developer training

Module Introduction

- **Nonclustered columnstore indexes can significantly improve the performance of relational data warehouse workloads against very large data sets**

- **This performance is achieved through a combination of new SQL Server features**

- **This module will review the fundamentals behind the nonclustered columnstore index feature so that you can decide if it will meet your specific new and existing data-tier requirements**

Nonclustered Columnstore Indexes Defined

- **Storage**
  - Data is grouped and stored by column

- **Compression**
  - Column storage can benefit significantly from compression

- **New execution mode**
  - Queries against columnstore indexes can leverage an optimized "batch" execution mode

Targeted Workloads and Environments

- Star and snowflake schemas
- Relational data warehouse queries
- Very large fact and/or dimension tables
- Queries that only reference a subset of table columns
- High read-to-write ratio
- Queries with grouping, filtering, aggregation
- "Reluctant" OLAP users
- Indexed views
- Pre-computed aggregate tables

Non-Ideal Candidates

- **Organizations**
  - Already using BI stack effectively (and happily)
  - Investment in in-house SSAS skills
  - Leveraging features not available in the Database Engine

- **Workloads**
  - Singleton operations
  - Small row range operations
  - Frequent data modifications, real-time selects
  - Touching each column
  - Small tables (< 1 million rows)

B-Tree Row Storage

SQL Server Data Page

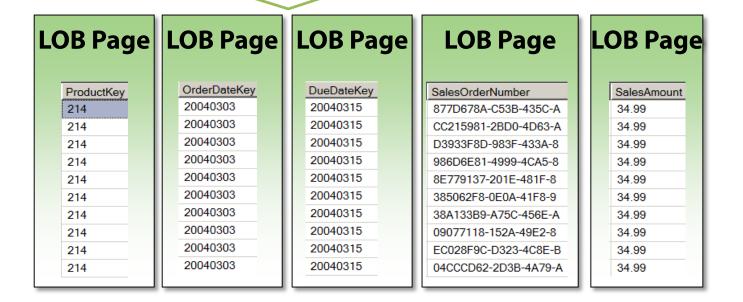|	ProductKey	OrderDateKey	DueDateKey	SalesOrderNumber	SalesAmount	TaxAmt	Freight
1	214	20040303	20040315	877D678A-C53B-435C-A	34.99	2.7992	0.8748
2	214	20040303	20040315	CC215981-2BD0-4D63-A	34.99	2.7992	0.8748
3	214	20040303	20040315	D3933F8D-983F-433A-8	34.99	2.7992	0.8748
4	214	20040303	20040315	986D6E81-4999-4CA5-8	34.99	2.7992	0.8748
5	214	20040303	20040315	8E779137-201E-481F-8	34.99	2.7992	0.8748
6	214	20040303	20040315	385062F8-0E0A-41F8-9	34.99	2.7992	0.8748
7	214	20040303	20040315	38A133B9-A75C-456E-A	34.99	2.7992	0.8748
8	214	20040303	20040315	09077118-152A-49E2-8	34.99	2.7992	0.8748
9	214	20040303	20040315	EC028F9C-D323-4C8E-B	34.99	2.7992	0.8748
10	214	20040303	20040315	04CCCD62-2D3B-4A79-A	34.99	2.7992	0.8748

All columns per row

Just Accessing a Few Columns? It Doesn't Matter…

SQL Server Data Page

	ProductKey	OrderDateKey	DueDateKey	SalesOrderNumber	SalesAmount	TaxAmt	Freight
1	214	20040303	20040315	877D678A-C53B-435C-A	34.99	2.7992	0.8748
2	214	20040303	20040315	CC215981-2BD0-4D63-A	34.99	2.7992	0.8748
3	214	20040303	20040315	D3933F8D-983F-433A-8	34.99	2.7992	0.8748
4	214	20040303	20040315	986D6E81-4999-4CA5-8	34.99	2.7992	0.8748
5	214	20040303	20040315	8E779137-201E-481F-8	34.99	2.7992	0.8748
6	214	20040303	20040315	385062F8-0E0A-41F8-9	34.99	2.7992	0.8748
7	214	20040303	20040315	38A133B9-A75C-456E-A	34.99	2.7992	0.8748
8	214	20040303	20040315	09077118-152A-49E2-8	34.99	2.7992	0.8748
9	214	20040303	20040315	EC028F9C-D323-4C8E-B	34.99	2.7992	0.8748
10	214	20040303	20040315	04CCCD62-2D3B-4A79-A	34.99	2.7992	0.8748

Columnar Data Storage

Only required columns are accessed, reducing I/O and memory required

LOB Page	LOB Page	LOB Page	LOB Page	LOB Page
ProductKey	**OrderDateKey**	**DueDateKey**	**SalesOrderNumber**	**SalesAmount**
214	20040303	20040315	877D678A-C53B-435C-A	34.99
214	20040303	20040315	CC215981-2BD0-4D63-A	34.99
214	20040303	20040315	D3933F8D-983F-433A-8	34.99
214	20040303	20040315	986D6E81-4999-4CA5-8	34.99
214	20040303	20040315	8E779137-201E-481F-8	34.99
214	20040303	20040315	385062F8-0E0A-41F8-9	34.99
214	20040303	20040315	38A133B9-A75C-456E-A	34.99
214	20040303	20040315	09077118-152A-49E2-8	34.99
214	20040303	20040315	EC028F9C-D323-4C8E-B	34.99
214	20040303	20040315	04CCCD62-2D3B-4A79-A	34.99

Columnar data is compressed
when possible

Segments

- **A segment is a unit of transfer for columnstore index queries**
- **One segment contains values for one column + partition combination**

LOB Page(s)	LOB Page(s)	LOB Page(s)	LOB Page(s)	LOB Page(s)
ProductKey	**OrderDateKey**	**DueDateKey**	**SalesOrderNumber**	**SalesAmount**
214	20040303	20040315	877D678A-C53B-435C-A	34.99
214	20040303	20040315	CC215981-2BD0-4D63-A	34.99
214	20040303	20040315	D3933F8D-983F-433A-8	34.99
214	20040303	20040315	986D6E81-4999-4CA5-8	34.99
214	20040303	20040315	8E779137-201E-481F-8	34.99
214	20040303	20040315	385062F8-0E0A-41F8-9	34.99
214	20040303	20040315	38A133B9-A75C-456E-A	34.99
214	20040303	20040315	09077118-152A-49E2-8	34.99
214	20040303	20040315	EC028F9C-D323-4C8E-B	34.99
214	20040303	20040315	04CCCD62-2D3B-4A79-A	34.99
Segment	Segment	Segment	Segment	Segment

Row Groups

Full row group = 2^{20} (1,048,576)

Each segment has the same number of rows

Syntax (1)

```
CREATE [ NONCLUSTERED ] COLUMNSTORE INDEX index_name
ON <object> ( column [ ,...n ] )
[ WITH ( <column_index_option> [ ,...n ] ) ]
[ ON {
{ partition_scheme_name ( column_name ) }
| filegroup_name
| "default"          }
]
[ ; ]
```

Syntax (2)

```
<object> ::=
{
[database_name. [schema_name ] . | schema_name . ]
table_name


{ <column_index_option> ::= {
DROP_EXISTING = { ON | OFF }
| MAXDOP = max_degree_of_parallelism
}
```

This is NOT a B-Tree Index

- Column order doesn't matter
- Choose up to 1,024 columns that you anticipate might be used
- Only a nonclustered option today, so plan for additional storage
- No concept of UNIQUE, PRIMARY KEY, FOREIGN KEY
- No INCLUDE
- Cannot apply to an Indexed View
- No filtered columnstore index
- One nonclustered columnstore index allowed
- ASC and DESC don't apply

Supported Data Types

char	varchar	nchar (NOT nvarchar(max))	nvarchar (NOT varchar(max))
decimal	numeric	int	bigint
smallint	tinyint	float	real
bit	money	smallmoney	date and time data types

Unsupported Data Types

sparse columns	binary	varbinary	varchar(max)
nvarchar(max)	uniqueidentifier	rowversion	timestamp
sql_variant	decimal w/precision > 18	numeric w/precision > 18	datetimeoffset w/scale > 2
CLR types	hierarchyid	spatial	xml

How Performance is Achieved

- **Columnar storage**
  - Only columns needed are read
  - Reduces I/O and memory usage
  - Data warehouse data tends to be "compression-friendly"

- **Batch execution mode**
  - New mode of query processing
  - Significant area for query execution time reduction

- **Segment elimination**
  - Segments can be bypassed based on predicates
  - Similar to "partition elimination" concept

Dictionary Encoding

- **Dictionaries are used to encode values**
  - Values stored in the segments will be pointers to a dictionary
  - For high repetition values, can yield significant compression
  - For highly unique data, can provide no benefit at all

- **There are different types of dictionaries**
  - Primary
  - Secondary

- **Dictionaries are also stored as LOB allocation units**

Batch Execution Mode

- **Batch mode allows for optimized execution**
  - Allows several rows at a time (batch) to fit in L2 cache
  - Instruction costs spread over batch instead of single-row-at-a-time

- **Reduced CPU consumption**

- **Only available today in conjunction with columnstore indexes**

- **Requires degree of parallelism greater than or equal to 2**

- **We'll cover this in more detail in the next module…**