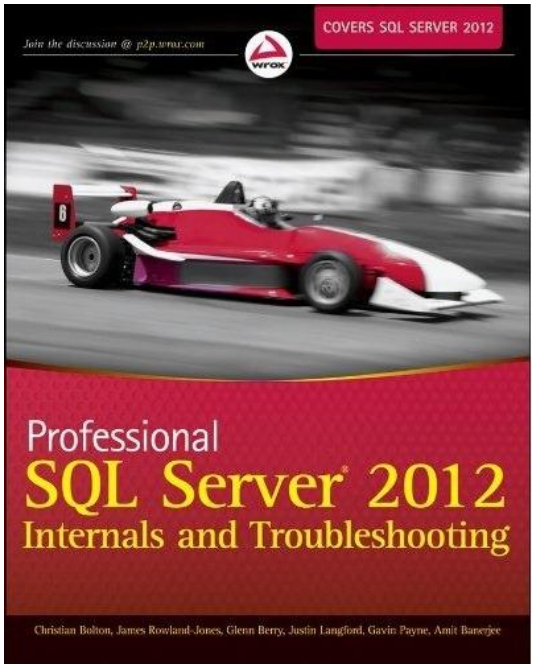
The SQL Server Experts
Discover. Optimise. Mentor.

Accelerating the data warehouse

Gavin Payne

gavin@coeo.com
@GavinPayneUK



- Author of two chapters
- SQL Saturday - Exeter

Acknowledgements

- **Thomas Kejser**
  CTO, Fusion-IO EMEA
- TechNet and MSDN articles
- But very little real-world documentation

Making SQL Server go faster

Transactional databases

- Add more, or faster, server hardware resources
- Tune scans into seeks using non-clustered indexes
- The database engine is designed for row-by-row query execution and storage

Data warehouses

- Tune the storage for long running sequential scans
- The cost of universal index tuning is too high to be feasible
- Biggest bottleneck becomes the row-by-row nature of the database engine

SQL Server 2012 Enhancements

Columnstore indexes

- Store data on a column basis
- Reduce the amount of data a query has to process
- Benefit from internal data compression

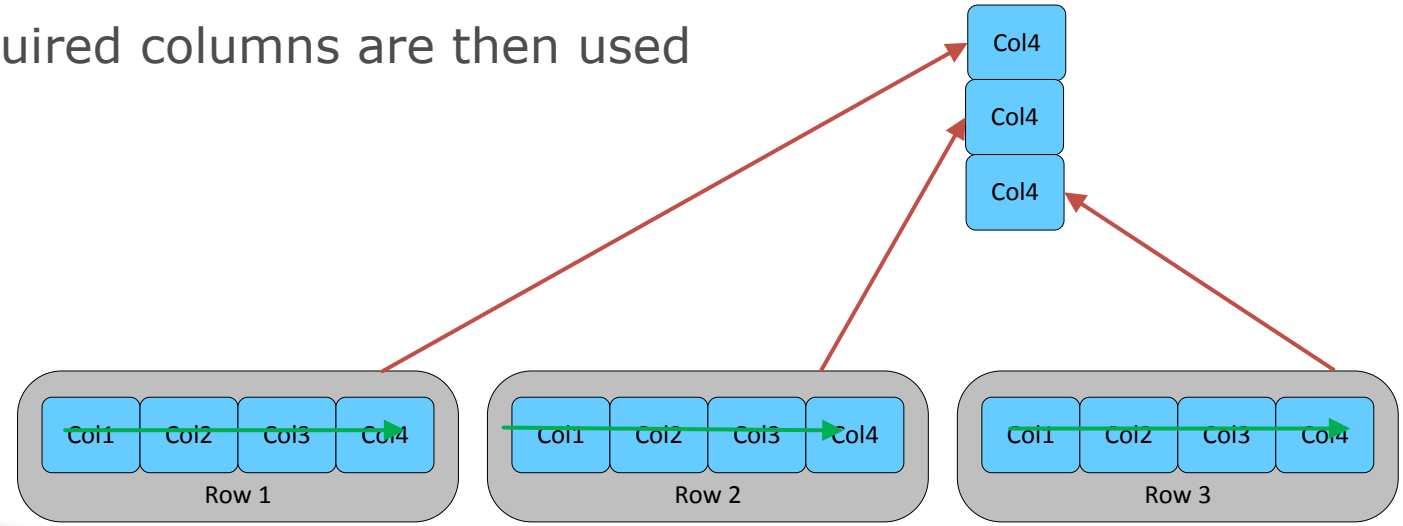
Batch-mode query processing

- Reduce CPU load for queries
- Execute using batches of data, not row-by-row
- Exclusively for Columnstore indexes

Row-based Data Storage

- How SQL Server has always stored data
- Using B-trees and heaps

- Complete rows are read into memory
- Required columns are then used



Row-based Data Storage

What transactional systems have always had

- Values for each column are physically stored contiguously
- Fast to retrieve multiple columns for a specific row

A bottleneck for data warehouses – part 1

- Pointer traversal and page parsing inefficient to find just one column
- Physical data distribution limits potential for compression

Row-based Data Storage

A bottleneck for data warehouses – part 2

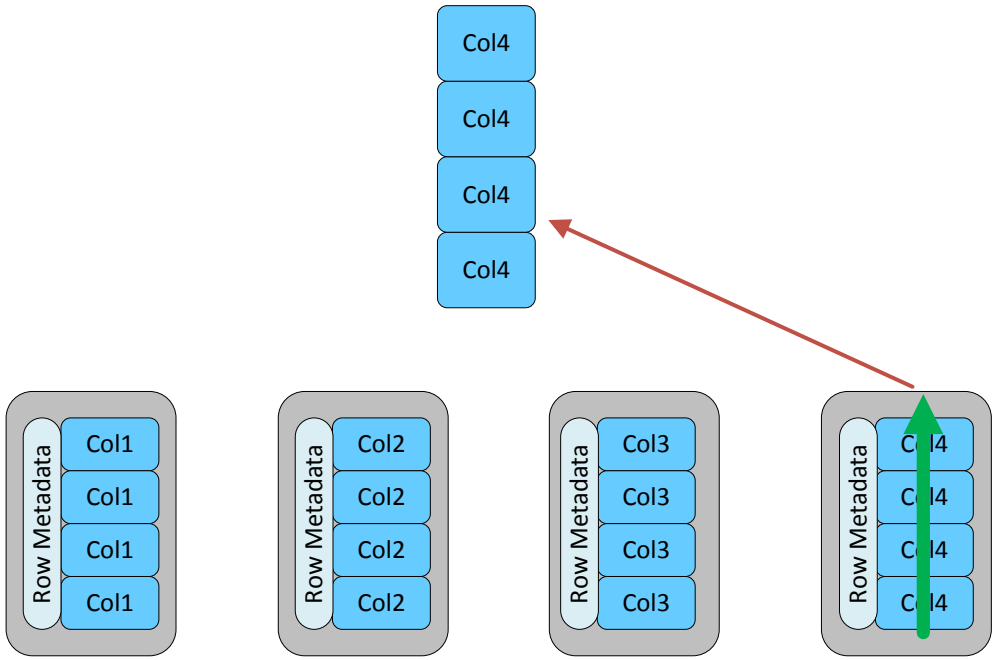
- DW queries can perform large fact seeks looking for column intersects
- Or long sequential scans

A bottleneck for data warehouses – part 3

- Seeks are Random IO while Scans are Sequential IO
- But why are we always worried about IO?

Column-based Data Storage

- Used within Columnstore indexes
- Non-clustered index today



Column-based Data Storage

Very efficient for fact table storage

- Values for each column are physically stored together
- High repetition of values allows very high compression ratios

Very efficient for business intelligence queries

- Fact tables often wide and long, but queries only use 10-15% of columns
- Fast to retrieve all the values for a particular column in a table
- Reads reduced by requiring fewer compressed data pages

Columnstore Details

Compatibilities

- Works with standard string and most numeric data types
- Works with table partitioning
- Uses segment based storage
- Stored in varbinary(max)

Limitations

- SQL Server 2012 Enterprise Edition only
- Table becomes read-only
- Only 1 per table

Columnstore Index Properties

Index design

- Up to 1024 columns
- No penalty for indexing the whole table
- Index key order not relevant

Design limitations

- Cannot be unique
- No include columns
- No sparse columns
- No computed columns
- No filtering
- Troublesome statistics

Updatable Columnstore Tables?

- Table partitioning can be a workaround for the read-only limitation

1. Partition the table
2. Switch out a partition to a staging table

3. Drop the Columnstore index
4. Update/insert data in the staging table
5. Create a Columnstore index

6. Switch in the staging table

- Next SQL Server version will have read-write Columnstore support

Batch-mode Query Processing

- Reducing query IO activity moves the bottleneck to the CPU
- Row-by-row computation is not efficient for long running data warehouse queries
- Takes advantage of modern CPU designs
- Batch-mode query processing mode created
- Designed to reduce the CPU utilisation during long running data warehouse queries

Batch-mode Query Processing

Data retrieval

- Batches of ≈1000 rows used
- Vectors map to internal Columnstore structures

Query execution

- Used automatically
- Designed for large fact tables with star schema joins
- Reduces CPU utilisation by 7-40x

Batch-mode Considerations

Server resource intensive

- Queries require more memory than row-mode equivalents
- Operations must run as parallel operators

Query usage considerations

- Designed to do the heavy lifting in the early parts of the query
- Aggregations and groupings required to benefit from the technology
- No OUTER JOIN or UNION ALL support

Summary

Columnstore indexes

- Column-based storage reduces the amount of data a query has to read
- Physical storage format then allows for internal data compression

Batch-mode query processing

- Compliments Columnstore indexes by processing batches instead of rows
- Designed to speed up large aggregates on fact tables