

Delivering a Relational Data Warehouse

Week 3 – Optimizing a Data Warehouse for Scale and Performance

Module 08

Improving Query Performance



Module Outline

08 | Improving Query Performance

	Topic
▶	Table Indexing
▶	Data Warehouse Indexing Strategies
▶	Demo: Optimizing Table Query Performance



©2016 Microsoft Corporation. All rights reserved. Microsoft, Windows, Office, Azure, System Center, Dynamics and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

Module Outline

08 | Improving Query Performance

Topic
Table Indexing
Data Warehouse Indexing Strategies
Demo: Optimizing Table Query Performance

Table Indexing

Without table indexes, SQL Server will retrieve data by using a table scan

- **Table scan:** SQL Server reads all table pages
- **Index:** SQL Server uses index pages to find rows



1110 1110 1110 1110 1110
 1010 1010 1010 1010 1010
 1010 1010 1010 1010 1010

Table Indexing

Indexing Possibilities

- Heap (no index)
- Clustered index
- Nonclustered index
- Columnstore index



100
 10101010
 1011100010
 10101010



100
 10101010
 1011100010
 10101010

Table Indexing

Tree Structures

- Indexes are commonly based on tree structures
 - Top node is called the **root node**
 - Bottom level nodes are called **leaf nodes**

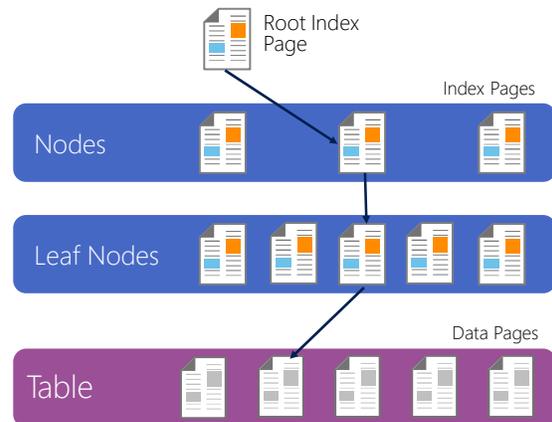


Table Indexing

Heaps

- A table with no indexes is called a **heap**
 - No specified order for pages within the table
 - No specified order for data within each page
- Inserted or updated rows can be placed anywhere within the table
 - Updates can result in rows moving to another page, and can leave forwarding pointers
 - This can impact on query performance



Table Indexing

Clustered Indexes

- A **clustered index** sorts and stores the data rows of the table in order of the clustering key
 - Table pages are stored in a logical order
 - Rows are stored in a logical order within table pages
 - There can only be one clustered index per table
 - Clustered indexes can be unique or non-unique
- A table is structured as either a heap or clustered index

Table Indexing

Clustered Indexes (Continued)



- Limits on clustering keys:
 - 16 columns
 - 900 bytes

Table Indexing

Clustered Indexes ► Mechanics

- Inserted rows must be placed into the correct logical position
 - This may involve splitting pages of the table
- Updated rows can either remain in the same place if they still fit and if the clustering key value is still the same
- Deleted rows free up space by flagging the data as unused

Table Indexing

Clustered Indexes ► Data Persistence

- Creating a clustered index on a view persists the data set that the query returns
- Creating indexes on computed columns avoids the need to calculate values at the time of execution
- Queries run faster because the data set—including aggregations, joins, and calculations—is stored



Table Indexing

Clustered Indexes ► Querying

- Queries related to the clustering key can seek
- Queries related to the clustering key can scan and avoid sorts



Table Indexing

Nonclustered Indexes

- A **nonclustered index** can be created on a table, whether structured as a heap or clustered index
 - They are also based on tree structures
 - Leaf levels point to base table structure rather than containing data
 - They can improve the performance of frequently used queries
 - The impact on data modification performance needs to be considered



Table Indexing

Nonclustered Indexes ► On a Heap

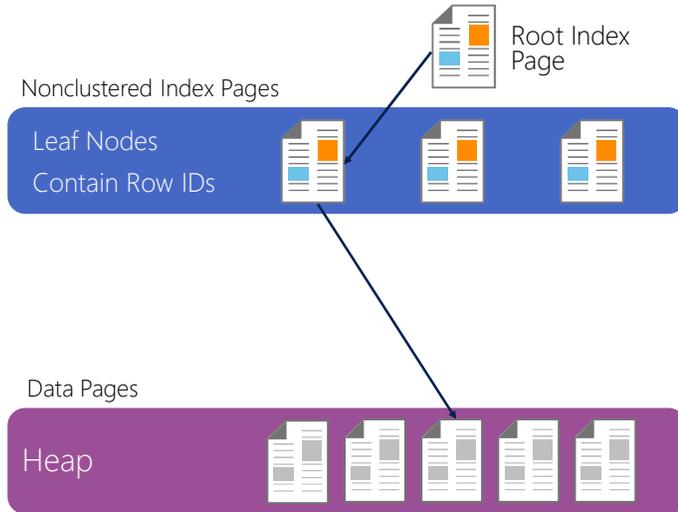


Table Indexing

Nonclustered Indexes ► On a Clustered Index

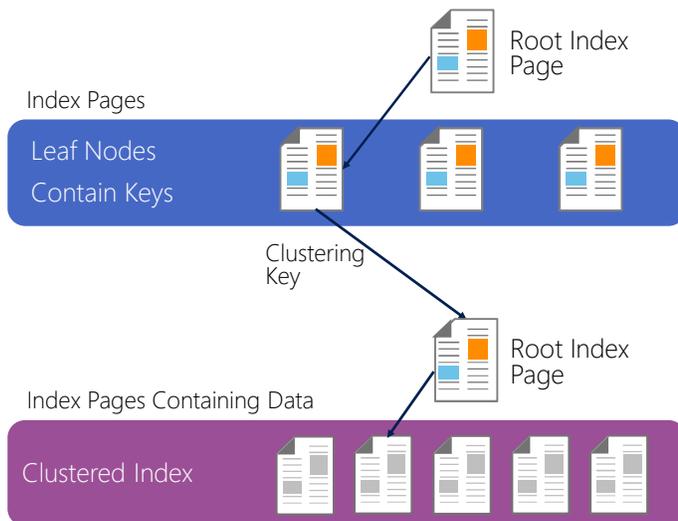


Table Indexing

Nonclustered Indexes ► Covering Indexes



- **Covering indexes** can greatly increase performance of queries
- Nonclustered indexes can use the INCLUDE clause
- The INCLUDE clause enables storage of selected data columns at the leaf level of a nonclustered index

Table Indexing

Columnstore Indexes

- A **columnstore index** resides in-memory, as compressed data in pages based on columns instead of rows
 - Can achieve 10x data compression and up to 100x speed up in analytics query processing



Table Indexing

Columnstore Indexes ► Example

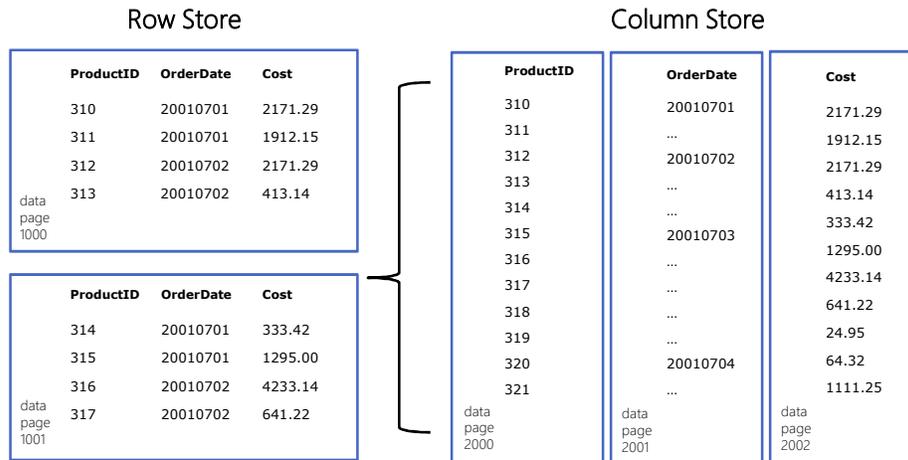


Table Indexing

Columnstore Indexes (Continued)

- Columnstore indexes are most suitable for:
 - Databases that have star or snowflake schemas
 - Tables that have large numbers of rows
 - Tables that contain data that responds well to compression
- Two types:
 - Clustered
 - Nonclustered

Table Indexing

Columnstore Indexes ► Types

- Clustered columnstore indexes:
 - Include all columns in the table
 - The only index on the table
 - Updatable
- Nonclustered columnstore indexes:
 - Include some or all columns in the table
 - Can be combined with other indexes
 - Read-only

1110 1110
 1110 1110
 1010 1010
 1010 1010 1010

Table Indexing

Recommended Practices

- Carefully consider how the table will be queried
- Good candidates for clustering keys:
 - Short (integers preferred)
 - Static
 - Increasing (not necessarily monotonically)
 - Unique
- Always specify indexes as unique if the data they contain is unique





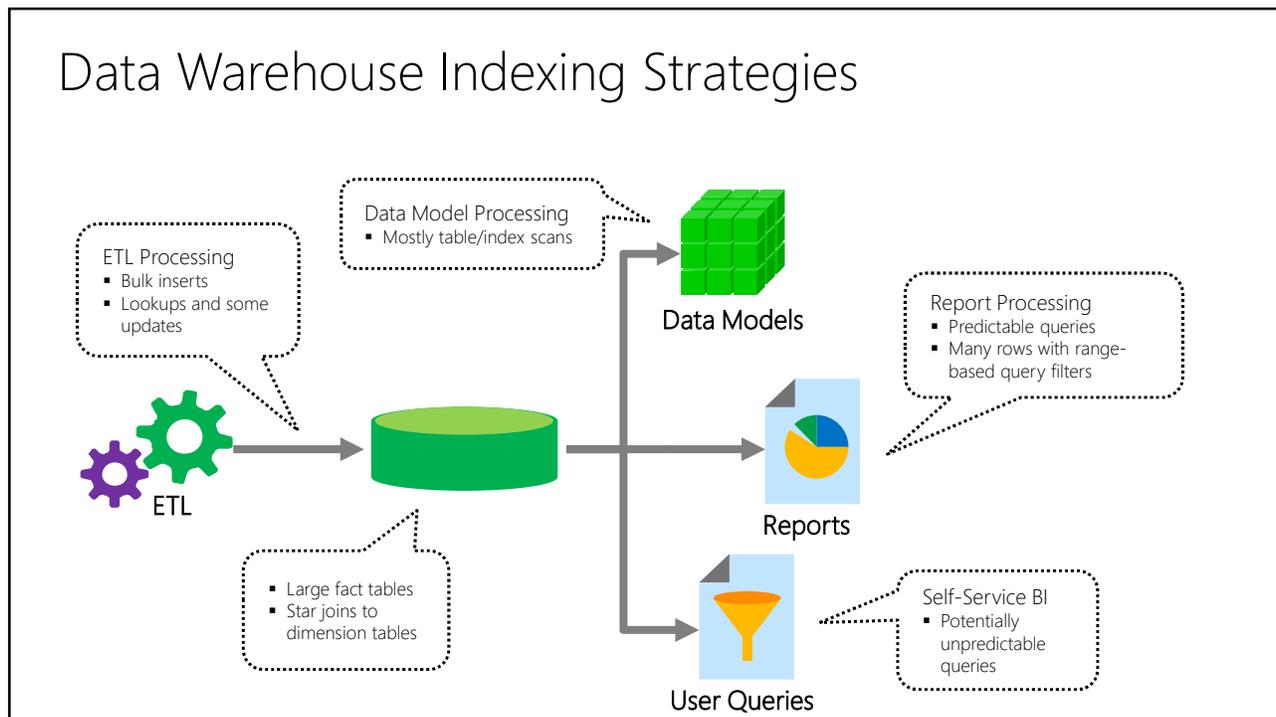
©2016 Microsoft Corporation. All rights reserved. Microsoft, Windows, Office, Azure, System Center, Dynamics and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

Module Outline

08 | Improving Query Performance

Topic
Table Indexing
Data Warehouse Indexing Strategies
Demo: Optimizing Table Query Performance

Data Warehouse Indexing Strategies



Data Warehouse Indexing Strategies

Workloads

- ETL processing
 - Staging
 - Data warehouse loading
- Data model processing
- Reporting
- Self-service BI, including ad hoc analysis



Data Warehouse Indexing Strategies

Workloads ► ETL Processing

- ETL loads need to be fast, and minimize impact on source systems
- Consider:
 - Creating non-indexed (heaps) staging tables



Data Warehouse Indexing Strategies

Workloads ► Data Warehouse Loading

- Different consideration should be given to dimension and fact table index designs



Data Warehouse Indexing Strategies

Workloads ► Data Warehouse Loading ► Dimension Tables

- Consider:
 - Creating a clustered index on the surrogate key for each dimension table
 - Creating a nonclustered index on the business key(s) to support surrogate key lookups during loads
 - If Type 2 changes are supported, combine an appropriate change tracking column
 - Creating nonclustered indexes on other frequently searched dimension columns
- Avoid partitioning dimension tables

Data Warehouse Indexing Strategies

Workloads ► Data Warehouse Loading ► Fact Tables

- Consider:
 - Creating a clustered index on the date key of the fact table
 - Creating nonclustered indexes for each foreign key
 - If possible create columnstore indexes
- For periodic or accumulative snapshot tables, consider a clustering key on the date key and also an appropriate dimension key

Data Warehouse Indexing Strategies

Workloads ► Data Warehouse Loading ► Fact Tables (Continued)

- For partitioned table:
 - Align the cluster key with the partition key
 - Ensure nonclustered indexes are aligned



Data Warehouse Indexing Strategies

Workloads ► Data Model Processing

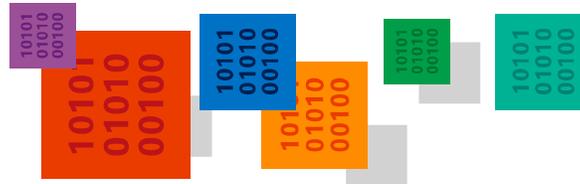
- Analysis Services data models can be developed to process and cache data, or passthrough
- Cache storage:
 - Multidimensional models (cubes): MOLAP
 - Tabular models: xVelocity (or Vertipaq)
- Passthrough:
 - Cubes: ROLAP
 - Tabular models: DirectQuery



Data Warehouse Indexing Strategies

Workloads ► Data Model Processing ► Caching

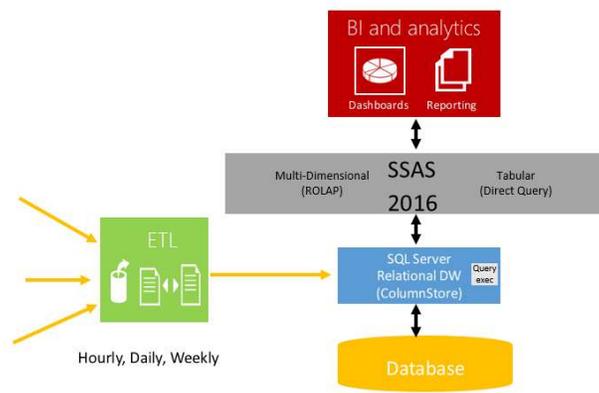
- Cached data models:
 - Large data models are usually partitioned and processed by date ranges
 - A clustered index on a date key of a fact table will optimize the data processing fact retrieval



Data Warehouse Indexing Strategies

Workloads ► Data Model Processing ► Passthrough

- Passthrough data models:
 - Clustered columnstore indexes provide excellent performance



Data Warehouse Indexing Strategies

Workloads ► Reporting

- Reporting, including self-service BI, workloads will require an understanding of queries and frequencies
- Consider:
 - Columnstore indexes for fact tables
 - Non-clustered indexes, and also covering indexes, for dimension keys



Data Warehouse Indexing Strategies

Index Maintenance

- Indexes need to be maintained
 - Post-ETL
 - Scheduled maintenance
- Indexes should also be periodically reviewed





©2016 Microsoft Corporation. All rights reserved. Microsoft, Windows, Office, Azure, System Center, Dynamics and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

Module Outline

08 | Improving Query Performance

Topic
Table Indexing
Data Warehouse Indexing Strategies
Demo: Optimizing Table Query Performance

Demo

Optimizing Table Query Performance

Demo objectives:

1. Configure table data compression
2. Create table indexes



©2016 Microsoft Corporation. All rights reserved. Microsoft, Windows, Office, Azure, System Center, Dynamics and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.