Module 6

Statistics and indexes

Module Overview

- Statistics and cardinality estimation
- Indexes
- Columnstore indexes

Lesson: Statistics and cardinality estimation

- Cost-based optimization
- Predicate selectivity
- Inspecting statistics
- Cardinality estimation
- Optimizer fixes
- Optimizer enhancements
- Controlling CE, OF and OE
- Creating statistics
- Updating statistics
- Filtered statistics

Cost-based optimization

- Cost-based optimization selects the query execution plan with the lowest estimated cost for execution
- Statistics are a critical part of cost-based optimization
  - Statistics provide information about data distribution in a column or group of columns
  - Columns in tables and indexes might have statistics

What Is a Cost-Based Optimizer?
https://www.brentozar.com/archive/2021/09/what-is-a-cost-based-optimizer/

Predicate selectivity

- Predicates
  - Expressions which evaluate to true or false
  - Found in joins, WHERE and HAVING clauses
- Predicate Selectivity
  - The number of rows from a table that meet a predicate
  - High selectivity = small percentage of rows returned
  - Low selectivity = large percentage of rows returned
- Predicate Selectivity in Query Optimization
  - Selectivity used to sequence join operations and choose between scan/seek

Query Tuning Fundamentals: Density, Predicates, Selectivity, and Cardinality
https://sqlserverdbknowledge.wordpress.com/2014/02/24/query-tuning-fundamentals-density-predicates-selectivity-and-cardinality/

Inspecting statistics

- Statistics header
  - Statistics metadata
- Density vector
  - Measure of uniqueness
- Histogram
  - Data distribution, up to 200 steps
- You can investigate statistics using DBCC SHOW_STATISTICS

Demo Show statistics

`Name` – Namnet på statistikobjektet.
- **`Updated`** – Senaste gången statistiken uppdaterades.
- **`Rows`** – Totalt antal rader i tabellen.
- **`Rows Sampled`** – Antal rader som användes för att bygga statistiken.
- **`Steps`** – Antal steg i histogrammet (max 200).
- **`Density`** – Omvända värdet av antalet unika värden (`1 / DISTINCT COUNT`).
- **`Average key length`** – Genomsnittlig längd (i byte) för kolumnens värden.
- **`String Index`** – Om SQL Server har skapat ett strängindex (`YES/NO`).
- **`Filter Expression`** – Om statistiken är filtrerad (`NULL` om ej).
- **`Unfiltered Rows`** – Antal rader i tabellen innan eventuella filter.
- **`Persisted Sample Percent`** – Om en specifik sampling har sparats (`0` = ingen).

HISTOGRAM
- **`RANGE_HI_KEY`** – Högsta nyckelvärdet i intervallet (t.ex. `A.` betyder förnamn som börjar på "A").
- **`RANGE_ROWS`** – Antal rader mellan föregående och detta nyckelvärde, exklusive exakt träff.
- **`EQ_ROWS`** – Antal rader som exakt matchar `RANGE_HI_KEY`.
- **`DISTINCT_RANGE_ROWS`** – Antal unika värden i intervallet mellan föregående och nuvarande nyckelvärde.
- **`AVG_RANGE_ROWS`** – Genomsnittligt antal rader per distinkt värde i intervallet.

Statistics

https://docs.microsoft.com/en-us/sql/relational-databases/statistics/statistics

What are SQL Server Statistics and Where are they Stored?
https://www.sqlnethub.com/blog/what-are-sql-server-statistics-and-where-are-they-stored/

DBCC SHOW_STATISTICS (Transact-SQL)
https://docs.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-show-statistics-transact-sql

Cardinality estimation

- In SQL Server, cardinality estimation attempts to predict the number of rows returned by a query or query operator
- SQL Server 2014 and SQL Server 2016 include rewritten cardinality estimation logic
  - Controlled by database compatibility level
  - Minor enhancements in subsequent releases
- Several factors can cause poor cardinality estimation:
  - Out-of-date or missing statistics
  - Functions in predicates
  - Table variables

Cardinality Estimation (SQL Server)
https://docs.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server

Compatibility Levels and Cardinality Estimation Primer
https://sqlperformance.com/2019/01/sql-performance/compatibility-levels-and-cardinality-estimation-primer

Optimizer fixes

- A weakness in the optimizer can be fixed
- Prior to 2016 we had to opt in:
  - Using the trace flag for that particular fix
  - Or 4199 whcih enables all fixes
- As of 2016 the fixes of RTM are enabled
  - Assuming 2016 compatibility level
  - Patching SQL server will not add fixed
  - Unless you use trace flag 4199

Enabling SQL Server Optimizer Hotfixes
https://sqlrus.com/2020/07/understanding-how-to-enable-sql-server-optimizer-hotfixes/

Självklart, Robert! Från och med SQL Server 2016 ändrades hanteringen av optimeringsrelaterade fixar i frågeoptimeraren på ett viktigt sätt. Låt mig förklara det lite mer detaljerat baserat på din beskrivning:

Innan SQL Server 2016
Före 2016, om Microsoft släppte en fix till frågeoptimeraren som ändrade hur frågor optimerades och kördes, var denna fix som standard inte aktiverad. För att dra nytta av specifika optimeringsfixar, behövde användarna aktivt välja att använda dem. Detta kunde göras på två sätt:

1. **Använda en spårningsflagga (trace flag) för den specifika fixen**: Detta aktiverade endast den specifika ändringen.
2. **Använda spårningsflagga 4199**: Detta aktiverade alla optimeringsrelaterade fixar som släppts upp till den punkten.

Denna metod tillät användare att vara försiktiga och endast välja de förändringar som de hade testat och visste att de skulle gynna deras specifika användningsfall eller miljö, utan att oväntat påverka prestanda genom att introducera nya beteenden i optimeraren.

Från och med SQL Server 2016
Med lanseringen av SQL Server 2016 ändrades denna modell:

- **Fixar i RTM (Release to Manufacturing) är aktiverade som standard**: Detta innebär att alla optimeringsförbättringar som finns i den ursprungliga versionen av SQL Server 2016 automatiskt är aktiverade när du använder kompatibilitetsnivå 130 (som motsvarar SQL Server 2016). Användarna behöver inte längre aktivt välja att använda dessa fixar; de är en integrerad del av optimeringsprocessen från start.

- **Patchning lägger inte till nya fixar automatiskt**: Efterföljande optimeringsfixar som introduceras i patchar eller service packs för SQL Server 2016 läggs inte automatiskt till om du bara patchar servern. Dessa senare fixar kräver att du använder spårningsflagga 4199 för att aktivera dem. Detta ger en balans mellan att direkt dra nytta av viktiga förbättringar i SQL Server 2016 och ha möjlighet att kontrollera införandet av nya optimeringsbeteenden som kan påverka existerande frågeprestanda.

Denna förändring i hantering och distribution av optimeringsfixar gör att baslinjeprestandan för SQL Server kan förbättras för alla användare med den senaste versionen, samtidigt som flexibilitet ges för de som behöver finjustera sitt system baserat på specifika prestandakrav eller beteenden.

Optimizer enhancements

- New functionality is added to the optimizer
- One example is ”Batchmode over rowstore”
  - Added in 2019
- Enabled by the database compatibility level
  - Same as when enhancement was released
  - Or higher

```
SE filen Robert Batchmode böand demofilerna!
CREATE DATABASE BatchModeDemo;
GO
USE BatchModeDemo;
GO
ALTER DATABASE BatchModeDemo SET
COMPATIBILITY_LEVEL = 150;

USE BatchModeDemo;
GO
DROP TABLE IF EXISTS Sales;
CREATE TABLE Sales (
    ID BIGINT IDENTITY PRIMARY KEY,
    Product NVARCHAR(50),
    Quantity INT,
    Price DECIMAL(10,2),
```

```
    SaleDate DATETIME
);
GO

-- Infoga 10 miljoner rader
INSERT INTO Sales (Product, Quantity,
Price, SaleDate)
SELECT
    'Product ' +
CAST(ABS(CHECKSUM(NEWID())) % 1000 AS
NVARCHAR),
    ABS(CHECKSUM(NEWID())) % 100,
    RAND() * 100,
    DATEADD(DAY, -ABS(CHECKSUM(NEWID())) %
365, GETDATE())
FROM master.dbo.spt_values t1
CROSS JOIN master.dbo.spt_values t2
WHERE t1.type = 'P' AND t2.type = 'P';
GO

GO

--2. Kontrollera om Batch Mode används
SELECT Product, SUM(Quantity) AS
TotalQuantity, AVG(Price) AS AvgPrice
FROM Sales
GROUP BY Product
```

```
--SQL Server 2019+ kan nu använda Batch
Mode för frågan även utan columnstore-
index.
--Tidigare var Batch Mode endast
tillgängligt för Columnstore-index.
```

Intelligent query processing in SQL databases
https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing

SQL SERVER 2022 Features for Performance Optimization
https://blog.sqlauthority.com/2021/11/08/sql-server-2022-features-for-performance-optimization/

Controlling CE, OF and OE

Setting	Type	CE	OF	OE
Db compat level	DB	Y	Y	Y
QUERY_OPTIMIZER_COMPATIBILITY_LEVEL_xxx	H	N	Y	Y
QUERY_OPTIMIZER_HOTFIXES	DB	N	Y	N
ENABLE_QUERY_OPTIMIZER_HOTFIXES	H	N	Y	N
FORCE_LEGACY_CARDINALTY_ESTIMATION	H	Y	N	N
LEGACY_CARDINALITY_ESTIMATION	DB	Y	N	N
9481 (force legacy CE)	TF	Y	N	N
2312 (force current CE with db compat 2012)	TF	Y	N	N

DB: Database setting
H: Hint
TF: Trace flag, can also be turned on at query level with QUERYTRACEFLAG hint

Demo CE and optimizer fixes

CE=Cardinality Estimate
OF=Optimizer Fixes
OE=Optimizer Enhancements

Creating statistics

- Automatic creation:
  - When AUTO_CREATE_STATS = ON
  - Single column statistics only
  - Names start _WA…
- Manual creation:
  - CREATE STATISTICS

Create Statistics
https://docs.microsoft.com/en-us/sql/relational-databases/statistics/create-statistics

Updating statistics

- Automatic update:
  - AUTO_UPDATE_STATISTICS = ON
  - Automatic update threshold
  - AUTO_UPDATE_STATS_ASYNC option
- Manual update:
  - UPDATE STATISTICS command
  - sp_updatestats
- Updating statistics causes query plans to be recompiled
- Database maintenance:
  - Maintenance plans will update statistics all statistics
    - Even if we have modified 0 rows since last time – waste of resources
    - Ola Hallengren's IndexOptimize focuses on defragmentation
      - Will only update statistics if it happens to do a rebuild on the index
      - Create your own job that calls IndexOptimize with parameters to update statistics only

Demo Update statistics

Ja, här är en sammanfattning av de nämnda delarna:

Automatic update

•AUTO_UPDATE_STATISTICS = ON

- När denna inställning är aktiverad i SQL Server, uppdateras statistik automatiskt när frågeoptimeraren bedömer att den är föråldrad. Detta förbättrar prestanda genom att säkerställa att frågeplaner baseras på aktuell datafördelning.

•Automatic update threshold

- SQL Server bestämmer automatiskt när statistik behöver uppdateras baserat på en intern tröskel.
  - För små tabeller (mindre än 500 rader) uppdateras statistik vid varje förändring.
  - För större tabeller används en dynamisk tröskel: 500+20% av totala rader500 + 20\% \text{ av totala rader}
  - I nyare versioner (SQL Server 2016+) kan denna tröskel justeras med **incremental statistics**.

•AUTO_UPDATE_STATS_ASYNC option

- Om denna inställning är aktiverad uppdateras statistik **asynkront**, vilket betyder att frågorna fortsätter att köras med den gamla statistiken tills den nya har beräknats.
- Detta är särskilt användbart för stora tabeller där synkron statistikuppdatering annars kan orsaka långa körningstider och blockeringar.

Manual update

•UPDATE STATISTICS command

- Används för att manuellt uppdatera statistik på en specifik tabell eller index.
  - Exempel: UPDATE STATISTICS MyTable;
  - Kan också specificeras för enskilda kolumner eller index.

•**sp_updatestats**

- En systemlagrad procedur som uppdaterar alla statistikobjekt i en databas där någon data har ändrats.
    - Exempel: EXEC sp_updatestats;
    - Användbar vid schemalagda jobb där man vill uppdatera statistik utan att behöva specificera varje tabell manuellt.

Både automatisk och manuell statistikuppdatering är viktiga för att SQL Server ska generera effektiva frågeplaner och förbättra prestanda.

SQL Server Statistics and how to perform Update Statistics in SQL
https://www.sqlshack.com/sql-server-statistics-and-how-to-perform-update-statistics-in-sql/

UPDATE STATISTICS (Transact-SQL)
https://docs.microsoft.com/en-us/sql/t-sql/statements/update-statistics-transact-sql

Update Statistics
https://docs.microsoft.com/en-us/sql/relational-databases/statistics/update-statistics

Filtered statistics

- CREATE STATISTICS using a WHERE clause
- Some limitations:
  - AUTO_UPDATE_STATISTICS threshold is based on all rows in table
    - Not the rows that the filter specifies
  - Limited to simple comparison logic
  - Cannot be created on all data/column types
  - Cannot be created on indexed views
- Histogram has finer resolution
  - When you combine several statistics

```
-- ============================================
-- 1. Skapa en testdatabas och använd den
-- ============================================
CREATE DATABASE FilteredStatsDemo;
GO
USE FilteredStatsDemo;
GO

-- ============================================
-- 2. Skapa en tabell för ordrar
-- ============================================
CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,   -- Unikt order-ID (Primärnyckel)
   Customer NVARCHAR(50),     -- Kundens namn
   OrderDate DATE,            -- Orderdatum
   Status NVARCHAR(20)        -- Orderstatus ('Pending', 'Shipped', 'Cancelled' etc.)
);
GO

-- ============================================
-- 3. Infoga testdata i tabellen
-- ============================================
-- Infoga 100 000 rader för att göra statistiken relevant
INSERT INTO Orders (Customer, OrderDate, Status)
SELECT
```

```
    'Customer' + CAST(ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS
NVARCHAR),
    DATEADD(DAY, -ABS(CHECKSUM(NEWID())) % 365, GETDATE()),
    CASE WHEN ABS(CHECKSUM(NEWID())) % 3 = 0 THEN 'Pending'
        WHEN ABS(CHECKSUM(NEWID())) % 3 = 1 THEN 'Shipped'
        ELSE 'Cancelled' END
FROM master.dbo.spt_values t1
CROSS JOIN master.dbo.spt_values t2
WHERE t1.type = 'P' AND t2.type = 'P';
GO

-- Kontrollera att vi har data
SELECT TOP 10 * FROM Orders;
GO

-- ============================================
-- 4. Skapa standardstatistik för hela kolumnen
-- ============================================
-- Detta gör att SQL Server skapar statistik för hela Status-kolumnen
CREATE STATISTICS stat_Orders_Status ON Orders (Status);
GO

-- Visa statistik
DBCC SHOW_STATISTICS ('Orders', 'stat_Orders_Status');
GO

-- ============================================
-- 5. Skapa Filtered Statistics för 'Pending' ordrar
-- ============================================
CREATE STATISTICS stat_Orders_Pending ON Orders (Status)
WHERE Status = 'Pending';
GO

-- Visa statistik för enbart 'Pending'
DBCC SHOW_STATISTICS ('Orders', 'stat_Orders_Pending');
GO

-- ============================================
-- 6. Testa hur statistik påverkar exekveringsplanen
-- ============================================
-- Aktivera statistikmätning
SET STATISTICS IO ON;
```

```
SET STATISTICS TIME ON;

-- Fråga som kan använda standardstatistik
SELECT COUNT(*) FROM Orders WHERE Status = 'Shipped';
GO

-- Fråga som kan använda vår Filtered Statistics
SELECT COUNT(*) FROM Orders WHERE Status = 'Pending';
GO

-- Kolla om SQL Server använde vår filtered statistics i exekveringsplanen!
-- Du kan aktivera "Actual Execution Plan" i SSMS (CTRL + M) innan du kör frågorna

-- ============================================
-- 7. Rensa upp (Kör detta om du vill ta bort allt)
-- ============================================
DROP STATISTICS Orders.stat_Orders_Status;
DROP STATISTICS Orders.stat_Orders_Pending;
DROP TABLE Orders;
DROP DATABASE FilteredStatsDemo;
GO
```

Filtered Statistics Follow-up
https://www.brentozar.com/archive/2016/12/filtered-statistics-follow/

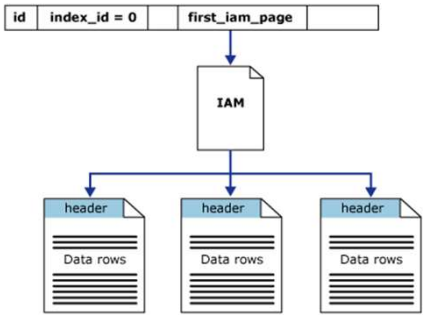
CREATE STATISTICS (Transact-SQL)
https://docs.microsoft.com/en-us/sql/t-sql/statements/create-statistics-transact-sql

Lesson: Indexes

- Heap internals
- Index structure
- Picking the right index key
- Single column and multicolumn indexes
- Filtered indexes
- The query optimizer's choice of indexes
- Data modification internals
- Index fragmentation
- Index column order
- Identify and create missing indexes

Heap internals

- A table without a clustered index is a heap
- IAM pages contain pointers to extents containing heap data
  - The IAM is the only link between pages in a heap; row data is unordered
- Heap rows are identified by a row identifier (RID)
  - Nonclustered indexes on a heap contain RID pointers



Heaps (Tables without Clustered Indexes)
https://docs.microsoft.com/en-us/sql/relational-databases/indexes/heaps-tables-without-clustered-indexes

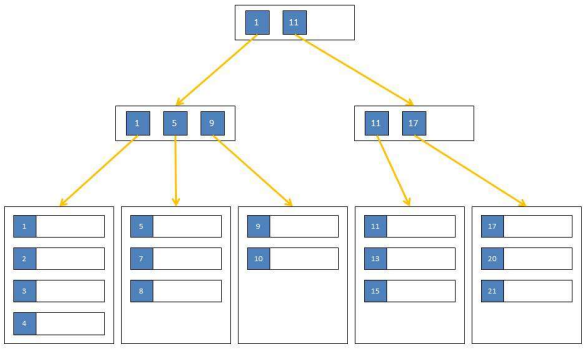
Tables Without Clustered Indexes
https://www.brentozar.com/blitz/heaps-tables-without-primary-key-clustered-index/

Heaps in SQL Server: Part 1 The Basics
https://www.red-gate.com/simple-talk/databases/sql-server/t-sql-programming-sql-server/heaps-in-sql-server-part-1-the-basics/

Clustered Index vs. Heap in SQL Server
https://www.sqlshack.com/clustered-index-vs-heap/

Index structure

- B-trees:
  - Self-balancing tree data structure
  - One root node; many non-leaf nodes; many leaf nodes
  - Clustered and nonclustered indexes are b-trees
- Index key:
  - Defines the order of data in an index
- Clustered index:
  - Leaf note ***is*** the data, contains all columns
  - Higher levels has index key
  - Index order = table data order
- Nonclustered index:
  - All nodes contain index pages
- There are other index types than B-Trees:
  - XML, full-text, hash, columnstore and spatial indexes



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

Clustered and nonclustered indexes described
https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described

Picking the right index key

- Clustered index key criteria:
  - Unique
  - Non-nullable
  - Narrow
  - Static
  - Ever-increasing
- Nonclustered index criteria:
  - Frequently used predicates
  - Join columns
  - Aggregate queries
  - Avoid:
    - Redundant/duplicate indexes
    - Wide keys
    - Indexes serving only one query for systems with a lot of data change

Designing effective SQL Server clustered indexes
https://www.sqlshack.com/designing-effective-sql-server-clustered-indexes/

Clustered and Non Clustered Index: 7 Top Points Explained
https://codingsight.com/clustered-and-non-clustered-index-7-top-points-explained/

Single column and multicolumn indexes

- Single column index:
  - Each predicate column has its own index
  - Less performance gain, but more reusable
- Multicolumn index:
  - All predicate columns are included in the key of the index
  - Greatest performance gain, but limited reusability

Demo Index several columns

Filtered indexes

- Nonclustered index with a filter predicate:
  - Filter predicate defined with a WHERE clause in the index definition
- Better performance than a whole-table index:
  - Finer-grained statistics
  - Smaller size
- Suitable when table data is queried in clearly identifiable subsets
- Some restrictions apply

Demo Filtered index

```
-- ============================================
-- 1. Skapa en testdatabas och använd den
-- ============================================
CREATE DATABASE FilteredIndexDemo;
GO
USE FilteredIndexDemo;
GO

-- ============================================
-- 2. Skapa en tabell för ordrar
-- ============================================
CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,   -- Unikt order-ID (Primärnyckel)
   Customer NVARCHAR(50),     -- Kundens namn
   OrderDate DATE,            -- Orderdatum
   Status NVARCHAR(20)        -- Orderstatus ('Pending', 'Shipped', 'Cancelled' etc.)
);
GO

-- ============================================
-- 3. Infoga testdata i tabellen
-- ============================================
INSERT INTO Orders VALUES
(1, 'Alice', '2024-01-10', 'Pending'),
(2, 'Bob', '2024-01-11', 'Shipped'),
```

```
(3, 'Charlie', '2024-01-12', 'Cancelled'),
(4, 'David', '2024-01-13', 'Pending'),
(5, 'Emma', '2024-01-14', 'Shipped');
GO

-- Kontrollera att data har lagts in korrekt
SELECT * FROM Orders;
GO

-- ============================================
-- 4. Skapa ett Filtered Index
-- ============================================
-- Detta index gör sökningar på "Pending"-ordrar snabbare
CREATE NONCLUSTERED INDEX idx_Orders_Pending
ON Orders (OrderDate)
WHERE Status = 'Pending';
GO

-- ============================================
-- 5. Testa indexet - denna fråga bör använda det
-- ============================================
SET STATISTICS IO ON;  -- Aktivera prestandamätning
SET STATISTICS TIME ON; -- Visa exekveringstid

SELECT OrderID, OrderDate FROM Orders WHERE Status = 'Pending';
GO

-- Kolla Execution Plan (CTRL + M i SSMS innan du kör frågan)
-- Om indexet används bör du se "Index Seek" på idx_Orders_Pending

-- ============================================
-- 6. Vad händer om vi söker på något annat än 'Pending'?
-- ============================================
SELECT OrderID, OrderDate FROM Orders WHERE Status = 'Shipped';
GO

-- Denna fråga använder INTE vårt Filtered Index, eftersom indexet bara täcker
'Pending'

-- ============================================
-- 7. Rensa upp (Kör detta om du vill ta bort allt)
-- ============================================
```

```
DROP INDEX idx_Orders_Pending ON Orders;
DROP TABLE Orders;
DROP DATABASE FilteredIndexDemo;
GO
```

Create filtered indexes
https://docs.microsoft.com/en-us/sql/relational-databases/indexes/create-filtered-indexes

What You Can (and Can't) Do With Filtered Indexes
https://www.brentozar.com/archive/2013/11/what-you-can-and-cant-do-with-filtered-indexes/

The query optimizer's choice of indexes

- Predicate SARGability (*Search ARGument ABLE)*:
  - WHERE <column> <operator> <value>
  - <column> must appear alone
  - Leading string wildcards prevent index seek

Demo SARG

Sargable
https://en.wikipedia.org/wiki/Sargable

The Two Ways to Fix Non-Sargable Queries
https://www.brentozar.com/archive/2019/12/the-two-ways-to-fix-non-sargable-queries/

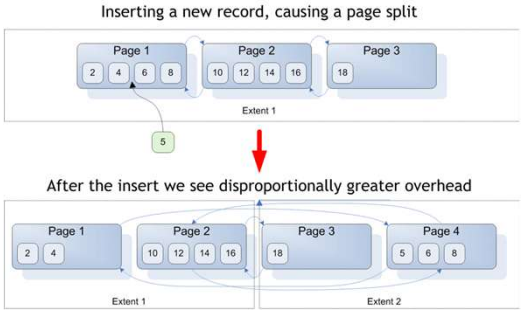
How to use sargable expressions in T-SQL queries; performance advantages and examples
https://www.sqlshack.com/how-to-use-sargable-expressions-in-t-sql-queries-performance-advantages-and-examples/

Data modification internals

- Delete:
  - Key reference removed from leaf-level page
  - A page with no references is removed from the b-tree
- Insert:
  - New key reference added to either:
    - Existing free space on a page
    - New page
- Update:
  - A delete followed by an insert

Index fragmentation

- External fragmentation
  - Linked list jumps back and forth in the database file
  - Less relevant unless you have single magnetic disks
- Internal fragmentation
  - Pages are less than 100% full
  - Watch out for
    - Rebuilding and re-applying a lower fill-factor
    - Rebuilding and ***not*** re-applying a lower fill-factor



Demo Fragmentation

Optimize index maintenance to improve query performance and reduce resource consumption
https://docs.microsoft.com/en-us/sql/relational-databases/indexes/reorganize-and-rebuild-indexes

Index fragmentation revisited
https://sqlblog.karaszi.com/index-fragmentation-revisited/

Stop Worrying About SQL Server Fragmentation
https://www.brentozar.com/archive/2012/08/sql-server-index-fragmentation/

Rebuild all fragmented heaps
https://karaszi.com/rebuild-all-fragmented-heaps

Index column order

- SELECT performance:
  - Only the first column has a histogram
  - Use the most selective column as the first column, but consider how the index will be used
- INSERT performance:
  - Consider the effect column order will have on INSERT performance for a clustered multicolumn index

How to Think Like the Engine: Index Column Order Matters a LOT.
https://www.brentozar.com/archive/2019/11/how-to-think-like-the-engine-index-column-order-matters-a-lot/

Identify and create missing indexes

- Query plans
- Query store
- Database Engine Tuning Advisor
- Missing index DMVs

- Tools will not:
  - Suggest clustered indexes
  - Suggest modifications to existing indexes
  - Analyze column order in multicolumn indexes
  - Suggest filtered indexes

Demo Missing index views

Index Related Dynamic Management Views and Functions (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/index-related-dynamic-management-views-and-functions-transact-sql

Diagnosis: Index-a-phobia
https://www.brentozar.com/blitzindex/sp_blitzindex-missing-indexes/

Glenn Berry's SQL Server Diagnostic Queries
https://glennsqlperformance.com/resources/

Lesson: Columnstore indexes

- Columnstore indexes
- Columnstore index features
- Columnstore index recommendations
- Maintaining Columnstore indexes

```
-- ============================================
-- 1. Skapa en ny testdatabas
-- ============================================
USE master
DROP DATABASE IF EXISTS ColumnstoreDemo
CREATE DATABASE ColumnstoreDemo;
GO
USE ColumnstoreDemo;
GO

-- ============================================
-- 2. Skapa en stor tabell med 10 miljoner
rader
-- ============================================
DROP TABLE IF EXISTS SalesData;
GO
```

```
CREATE TABLE SalesData (
    ID BIGINT IDENTITY(1,1) PRIMARY KEY
NONCLUSTERED,
    ProductID INT,
    CustomerID INT,
    OrderDate DATE,
    Quantity INT,
    Price DECIMAL(10,2),
    Status NVARCHAR(20) -- 'Pending',
'Shipped', 'Cancelled'
);
GO


--
==========================================
==
-- 3. Infoga 10 miljoner rader i tabellen
--
==========================================
==
INSERT INTO SalesData (ProductID,
CustomerID, OrderDate, Quantity, Price,
Status)
SELECT
    ABS(CHECKSUM(NEWID())) % 10000,  --
Slumpmässiga ProductID
```

```
    ABS(CHECKSUM(NEWID())) % 500000, --
Slumpmässiga CustomerID
    DATEADD(DAY, -ABS(CHECKSUM(NEWID())) %
365, GETDATE()), -- Datum senaste året
    ABS(CHECKSUM(NEWID())) % 10 + 1, --
Antal mellan 1-10
    RAND() * 100,                        --
Pris mellan 0 och 100
    CASE WHEN ABS(CHECKSUM(NEWID())) % 3 =
0 THEN 'Pending'
         WHEN ABS(CHECKSUM(NEWID())) % 3 =
1 THEN 'Shipped'
         ELSE 'Cancelled' END --
Slumpmässig orderstatus
FROM master.dbo.spt_values t1
CROSS JOIN master.dbo.spt_values t2
WHERE t1.type = 'P' AND t2.type = 'P';
GO

-- Kontrollera antal rader (ska vara runt
10 miljoner)
SELECT COUNT(*) AS AntalRader FROM
SalesData;
GO

--
==========================================
==
```

```
-- 4. Skapa ett Clustered Columnstore
Index (CCI)
--
==========================================
==
-- Detta konverterar hela tabellen till
Columnstore-format
CREATE CLUSTERED COLUMNSTORE INDEX
CCI_SalesData ON SalesData


--
==========================================
==
-- 5. Skapa ett Nonclustered Columnstore
Index (NCCI) på vissa kolumner
--
==========================================
==
-- Detta gör att tabellen fortfarande
fungerar som en "vanlig rowstore-tabell"
-- men vissa kolumner kan dra nytta av
Columnstore för analytiska frågor.
CREATE NONCLUSTERED COLUMNSTORE INDEX
NCCI_SalesData
ON SalesData (Quantity,  Status);
GO
```

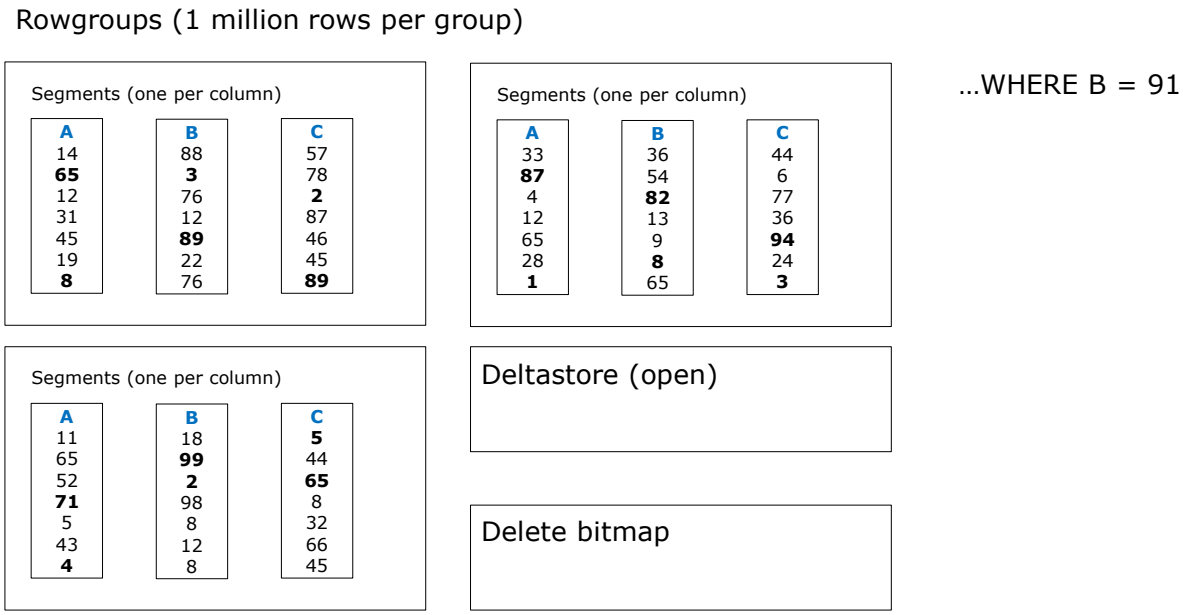
```
--
==========================================
==
-- 6. Testa prestandan för analytiska
frågor
--
==========================================
==
-- Denna fråga bör använda Clustered
Columnstore Index (CCI)
SELECT Status, SUM(Quantity) AS
TotalSales, COUNT(*) AS Orders
FROM SalesData
GROUP BY Status;
GO

-- Denna fråga bör använda Nonclustered
Columnstore Index (NCCI)
SELECT SUM(Quantity) AS TotalUnitsSold
FROM SalesData
WHERE Status = 'Shipped';
GO

-- Kontrollera Execution Plan i SSMS (CTRL
+ M innan du kör)
-- Leta efter "Columnstore Index Scan" i
exekveringsplanen!
```

```
--
============================================
==
-- 7. Rensa upp om du vill ta bort allt
--
============================================
==
DROP INDEX CCI_SalesData ON SalesData;
DROP INDEX NCCI_SalesData ON SalesData;
DROP TABLE SalesData;
DROP DATABASE ColumnstoreDemo;
GO
```

Columnstore indexes



Demo Columnstore

```
--Visa row_groups
SELECT *
FROM sys.column_store_row_groups
WHERE object_id = OBJECT_ID('SalesData');

--Vilka segment lagrar vilken data:
SELECT * FROM
sys.dm_db_column_store_row_group_physical_stats
WHERE object_id = OBJECT_ID('SalesData');
```

Columnstore indexes: Overview
https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview

What are Columnstore Indexes?
https://www.red-gate.com/simple-talk/databases/sql-server/t-sql-programming-sql-server/what-are-columnstore-indexes/

Columnstore Index Scan
https://sqlserverfast.com/epr/columnstore-index-scan/

Niko Neugebauer, Columnstore
https://www.nikoport.com/columnstore/

Columnstore index features

- Clustered or nonclustered
- We can have both row and column indexes on the same table
  - Only one columnstore index
- Columnstore index options
  - Filter
  - Compression delay
- Data is highly compressed
  - Inserts and after-image for update are initially reflected in row-based delta-store
    - After 1 million rows, such open group is compressed
  - Delete and before-image of update are reflected in delete bitmap

CREATE COLUMNSTORE INDEX (Transact-SQL)
https://docs.microsoft.com/en-us/sql/t-sql/statements/create-columnstore-index-transact-sql

Columnstore indexes - what's new
https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-what-s-new

Stairway to Columnstore Indexes
https://www.sqlservercentral.com/stairways/stairway-to-columnstore-indexes

Ordered clustered columnstore index (2022)

- ORDER clause for clustered Columnstore index (2022)
  - Data is potentially sorted when the index is built
  - Data is sorted (per batch) when data is added
- Rebuild becomes OFFLINE
  - Unless data hasn't been modified in a partition
    - That partition won't be rebuilt
- Sort done in tempdb
  - If not enough space, it is a *soft sort*
    - Stop the sort when tempdb is full
    - Essentially, data won't be sorted
- Make sure you verify the segment alignment

ORDER for columnstore index
https://sqlblog.karaszi.com/order-for-columnstore-index/

Ordered Columnstore Indexes in SQL Server 2022
https://www.red-gate.com/simple-talk/databases/sql-server/t-sql-programming-sql-server/ordered-columnstore-indexes-in-sql-server-2022/

Columnstore index recommendations

- Clustered Columnstore indexes
  - Fact tables
  - Large dimensional tables
  - Large log tables (typically append-only, potentially huge)
- Non-clustered Columnstore indexes
  - OLTP system where you also perform analysis
  - Not uncommon that many indexes has been added over time
  - Indexes added for analysis queries can potentially be replaced with:
    - One (1) Columnstore index, vastly smaller in size than the b-tree indexes combined
- Do not remove b-tree indexes created for high selectivity queries
  - No SEEKs in a Columnstore index

Sammanfattning av Columnstore Index-rekommendationer
•**Clustered Columnstore Index (CCI)** är bäst för **stora tabeller** som används i analys, exempelvis **faktatabeller och loggtabeller** som växer snabbt.
•**Non-Clustered Columnstore Index (NCCI)** är bra om du har en **OLTP-databas** där vissa frågor kräver **snabb analys** utan att påverka den dagliga transaktionshanteringen.
•**Ett Columnstore Index kan ersätta många vanliga B-tree-index** och ta upp mycket mindre lagringsutrymme.
•**MEN! Behåll vanliga B-tree index för frågor som kräver snabba, exakta uppslag (SEEKs)**, eftersom Columnstore Index **inte är bra på punktuppslag**.
Kort sagt: **Använd Columnstore för analys, men behåll vanliga index för transaktioner och snabba sökningar!** 🚀

Varför är Non-Clustered Columnstore Index (NCCI) bra i en OLTP-databas?
I en **OLTP-databas** (Online Transaction Processing) körs många **små och snabba transaktioner** (INSERT, UPDATE, DELETE, SELECT). Samtidigt kan vissa **stora analysfrågor** behöva köras utan att störa transaktionshanteringen. Här kommer **Non-Clustered Columnstore Index (NCCI)** in i bilden.
Fördelar med NCCI i en OLTP-databas
✅ **1. Förbättrar prestandan på analytiska frågor**
•Ett **NCCI lagrar endast vissa kolumner i columnstore-format**.
•Detta gör att SQL Server kan använda **Batch Mode** för att snabbt bearbeta analyser på stora datamängder.
✅ **2. Påverkar inte vanliga transaktioner**
•Eftersom NCCI är ett **extra index**, kan den underliggande tabellen **fortsätta fungera som en vanlig rowstore-tabell**.

•**INSERT/UPDATE/DELETE använder fortfarande rowstore**, medan **stora SELECT-frågor** kan dra nytta av columnstore.

✅ **3. Minskar behovet av många B-tree index**

•Många OLTP-databaser har **flera b-tree index** för att snabba upp olika SELECT-frågor.

•Ett **enda NCCI kan ersätta flera index**, vilket sparar **lagringsutrymme och uppdateringskostnader**.

✅ **4. Passar bra för "hybrid-databaser"**

•Om en databas behöver både **snabba transaktioner (OLTP)** och **snabb analys (OLAP)**, kan NCCI göra att man slipper flytta data till en separat analysmiljö.

Exempel på när NCCI är användbart

Tänk dig en e-handelsdatabas där:

•**Kunder köper produkter (INSERT, UPDATE, DELETE - OLTP)**.

•**Ledningen vill analysera försäljningen i realtid (SELECT - OLAP)**.

Utan NCCI:

•Stora SUM(), COUNT() eller GROUP BY-frågor kan **sakta ner transaktionerna**.

•Många index kan behövas för att optimera olika SELECT-frågor.

Med NCCI:

•Vanliga köptransaktioner fortsätter i rowstore-tabellen.

•**Analytiska frågor använder NCCI och påverkar inte OLTP-trafiken**.

Sammanfattning

✅ **NCCI gör det möjligt att kombinera OLTP och OLAP i samma databas**.

✅ **Snabba SELECT-frågor för analys utan att påverka vanliga transaktioner**.

✅ **Minskar behovet av många B-tree index och förbättrar prestanda**.

Perfekt för system som behöver både snabba transaktioner och effektiva rapporter i realtid! 🚀

Columnstore indexes - what's new
https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-what-s-new

Niko Neugebauer, Columnstore
https://www.nikoport.com/columnstore/

Maintaining Columnstore indexes

- What is the purpose of the index?
  - Compression only
  - Used for queries, where you also expect potential segment elimination
- What modifications are performed?
  - If data was removed (DELETE or UPDATE)
    - Consider getting rid of the delete bitmap
  - If data was added (INSERT or UPDATE)
    - And you expect segment alignment
    - …then you want to re-process the index
    - ALTER INDEX REBUILD of an ORDERed clustered Columnstore index
      - Might be soft sort, which doesn't re-align all data
    - Consider converting to a row-store and then back again to Columnstore
      - Using CREATE INDEX … WITH DROP_EXISTING

Demo Columnstore ordered

Enkel förklaring av "Maintaining Columnstore Indexes"
Den här bilden handlar om **hur man underhåller Columnstore Index** i SQL Server, speciellt när data ändras.
1. Vad är syftet med Columnstore Index?
•**Huvudsyftet är kompression → sparar lagringsutrymme och förbättrar prestanda**.
•Används i **frågor där segment elimination är viktig**, dvs. SQL Server kan hoppa över stora delar av datan och läsa bara relevanta segment.
2. Vad händer när data ändras?
Om rader tas bort (DELETE eller UPDATE)
•**Rader tas inte bort direkt** från Columnstore Index.
•Istället markeras de som raderade i en **Delete Bitmap**.
•**Lösning:** Om många rader har raderats, **överväg att "städa bort" delete bitmap** genom att **rebygga indexet**.
Om rader läggs till (INSERT eller UPDATE)
•Nya rader hamnar i **Delta Store**, som är en vanlig rowstore-tabell.
•Om många rader läggs till kan **segmenten bli ooptimerade**.
•**Lösning:**

- **Rebuild Index** med ALTER INDEX REBUILD för att optimera lagringen.
- Om indexet är **ORDNAT (ORDERED)** kan en "soft sort" ske → **inte all data blir perfekt omorganiserad**.
- För **bästa resultat**:
  - **Konvertera först till Rowstore** (DROP_EXISTING)
  - **Skapa om Columnstore Indexet från scratch**

3. Hur kan man fixa problemen?
•**ALTER INDEX REBUILD** – Förbättrar prestanda, men kanske inte alltid sorterar perfekt.

•**CREATE INDEX ... WITH DROP_EXISTING** – Konverterar först till rowstore och sedan tillbaka till Columnstore för **perfekt optimering**.

Sammanfattning

✅ **Columnstore Index sparar plats och gör analyser snabbare**.

✅ **DELETE/UPDATE lämnar en Delete Bitmap → Rebuild behövs ibland**.

✅ **INSERT går till Delta Store → kan göra indexet ineffektivt**.

✅ **Bästa sättet att fixa allt** är att **skapa om indexet helt (DROP_EXISTING)** om det blivit för rörigt.

TL;DR: Om din Columnstore-tabell blir långsam, rebuilda indexet eller skapa om det! 🚀 😀

ORDER for columnstore index
https://sqlblog.karaszi.com/order-for-columnstore-index/

Performance tuning with ordered clustered columnstore index
https://learn.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/performance-tuning-ordered-cci

Columnstore indexes - Data loading guidance
https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-data-loading-guidance

Lab 6: Statistics and indexes

- Ex 1: Improve a query, the easy way
- Ex 2: Improve a query, the traditional way
- Ex 3: Improve a query, by query re-write

Estimated Time: 30 minutes