

Indexe im Praxis

Indexe

- Ein Index = eine Struktur auf Platte, die einer Tabelle oder Sicht zugeordnet ist, um die Tupeln in der Tabelle oder Sicht schneller abzurufen
- Gute Indexierung → schnelle Applikation
- Schlechte Indexierung → verlangsamt das ganze SQL Server

Syntax

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]  
INDEX index_name  
ON <object> (column [ ASC | DESC ] [ ,...n ] )  
[ INCLUDE (column_name [ ,...n ] ) ]  
[ WHERE <filter_predicate> ]  
[ WITH ( <relational_index_option> [ ,...n ] ) ]
```

Eigenschaften eines Index

- Geclustert vs. Nicht-geclustert
- Unique(eindeutig) vs. Nonunique
- Mit einfachem vs. zusammengesetztem Schlüssel
- Aufsteigender oder absteigender Reihenfolge auf die Spalte des Index
- Für die ganze Tabelle vs. Gefiltert für nicht-geclusterte Indexe

Geclusterte vs. Nicht-geclusterte Indexe

- Geclusterte Indexe: die Tupeln in der Tabelle werden nach dem Suchschlüssel sortiert und gespeichert

```
CREATE CLUSTERED INDEX Index_Name  
ON Schema.TableName(Column);
```

- Nicht-geclusterte Indexe: Suchschlüsselwerte und Zeiger zu den Tupeln in der Datei (die ein Haufendatei oder sogar ein geclusterter Index sein kann)

```
CREATE INDEX Index_Name  
ON Schema.TableName(Column);
```

Geclusterte vs. Nicht-geclusterte Indexe

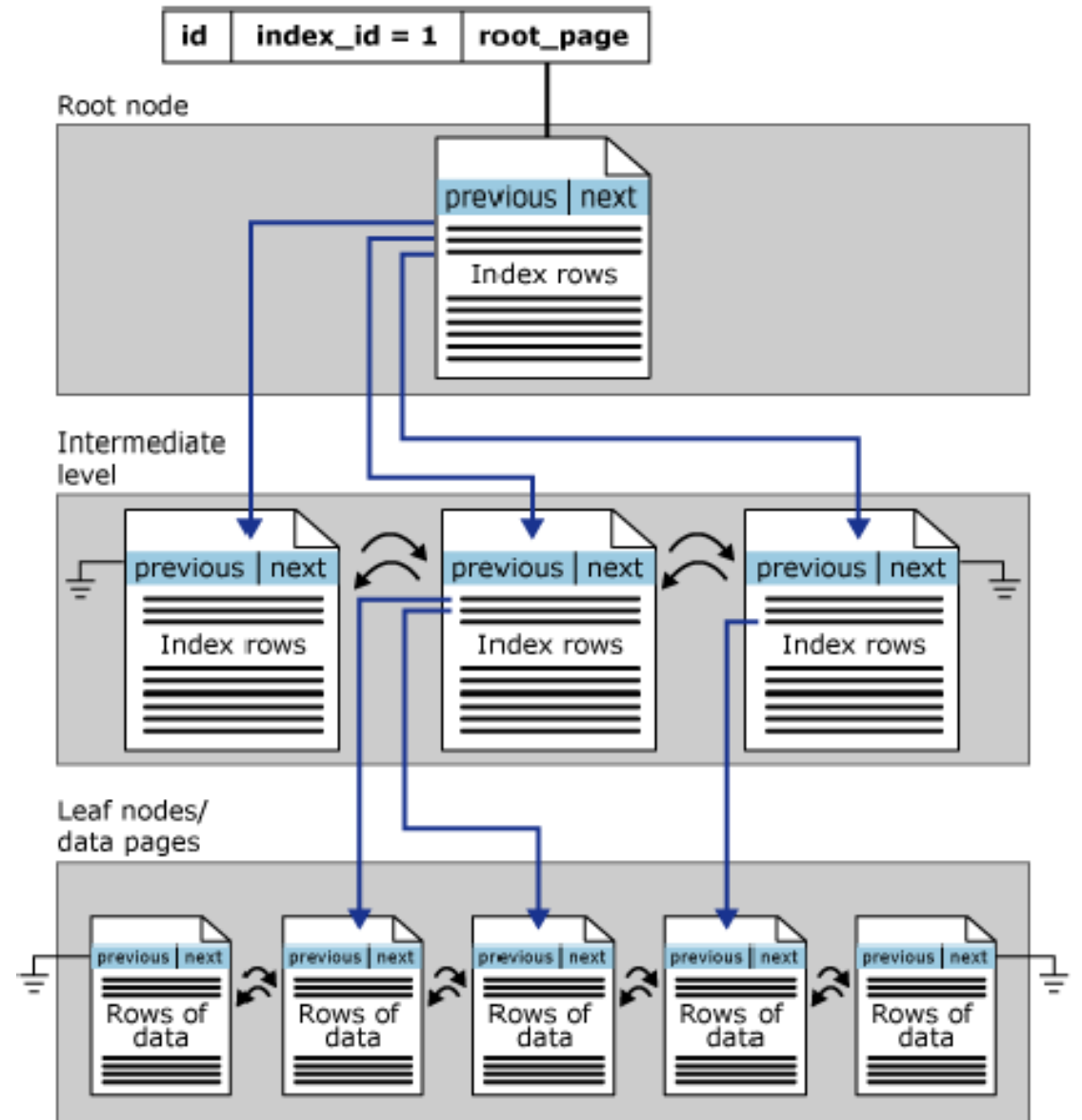
- Die Datenseiten eines geclusterten Index werden immer alle Spalten in der Tabelle enthalten
- Es gibt nur ein geclusterter Index per Tabelle
- SQL Server unterstützt bis zu 999 nicht-geclusterte Indexe für eine Tabelle
- Ein Suchschlüssel für ein Index (geclustert oder nicht-geclustert) kann maximal 16 Spalten und 900 Bytes enthalten

Geclusterte Indexe

- Können für Anfragen, die oft vorkommen, benutzt werden
- Sind effizient für Bereichsanfragen
- Geclusterte Indexe sind nicht gut für:
 - Suchschlüssel, die oft geändert werden
 - Große Suchschlüssel

Geclusterte Indexe

- Werden als B-Bäume organisiert



Geclusterte vs. Nicht-geclusterte Indexe

- Wenn man ein Primärschlüssel für eine Tabelle erstellt und:
 - kein geclusterter Index wurde definiert
 - kein nicht-geclusterter Index wurde erstellt→ ein eindeutiger geclusterter Index wird erstellt
- Wenn der Index die gesamte Abfrage deckt (alle nötige Spalten enthält), dann bezeichnet man den Index auch als *Covering Index*

Schlüssel vs. Nichtschlüssel Indexspalten

- Schlüssel Indexspalten: die Spalten, die in dem Suchschlüssel des Index enthalten sind
- Nichtschlüssel Indexspalten: Spalten, die in dem INCLUDE Klausel enthalten sind
 - Diese sind Spalten, die nicht Teil des Suchschlüssels sind, aber die wir vielleicht in Abfragen brauchen werden

```
CREATE INDEX Index_Name  
ON Schema.TableName (Column)  
INCLUDE (ColumnA, ColumnB);
```

Schlüssel vs. Nichtschlüssel Indexspalten

- Vorteile für das Benutzen der Nichtschlüssel Indexspalten:
 - Man kann auf diese Spalten mit einem Indexscan zugreifen (obwohl die Spalten nicht Teil des Suchschlüssels sind) ; d.h. man kann ein *Key Lookup* vermeiden
 - Hier werden auch Datentypen erlaubt, die bei dem Suchschlüssel nicht erlaubt sind (z.B. text, ntext, image)
 - Nichtschlüssel Indexspalten zählen nicht für die 900 Byte Limit (in SQL Server) für den Suchschlüssel

Eindeutige Indexe

- Ein eindeutiger Index versichert, dass der Suchschlüssel des Index keine Duplikate enthält
- Ein eindeutiger Index zu erstellen macht nur Sinn, wenn wir wissen, dass die Werte der Suchschlüssel eindeutig sind
- Die Eindeutigkeit ist eine wichtige Information für den Anfrageoptimierer

Gefilterte Indexe

- Gefilterte Indexe: ein optimierter nicht-geclusterter Index, besonders geeignet für Anfragen, die Daten aus einer gut definierte Teilmenge von Daten selektiert

```
CREATE NONCLUSTERED INDEX FI_EndDate ON  
Products (ProductID, EndDate)  
WHERE EndDate IS NOT NULL ;  
GO
```

- Verbesserte Anfrage Performanz
- Reduziert die Kosten für die Erhaltung des Index (Index maintenance)
- Reduziert die Speicherkosten des Index

Gefilterte Indexe

- Um sicher zu sein, dass der gefilterte Index benutzt wird:

```
SELECT ProductID, EndDate  
FROM Products  
WITH ( INDEX (FI_EndDate ) )  
WHERE EndDate IN ( '20000825', '20000908' );
```

Index Entwurf

- Man muss die Eigenschaften der Datenbank verstehen
 - OLTP -Online Transaction Processing
 - Bei OLTP wiederholen sich die Datenbankprozesse ständig, sind strukturiert und bestehen aus isolierten, atomaren Transaktionen
 - Anwendungsbereich: Administrationssysteme
 - OLAP -Online Analytical Processing
 - Analyseprozesse auf Unternehmensdaten interaktiv („Online“) durchzuführen
 - Die historische, aggregierte Information steht im Vordergrund
 - Zugriff erfolgt meist nur lesend
 - Anwendungsbereich: Data Warehouse Systeme
- Man muss die Eigenschaften der häufigsten Anfragen verstehen
- Man muss die Eigenschaften der Spalten verstehen, die in den Anfragen benutzt werden
- Man muss den optimalen Speicherort bestimmen

Allgemeine Index Entwurf Guidelines

- Datenbank Überlegungen:
 - Zu viele Indexe auf eine Tabelle beeinflussen die Performance folgender Anweisungen: INSERT, UPDATE, DELETE, MERGE
 - Die Indexierung der kleinen Tabellen könnte vielleicht nicht optimal sein
 - Die Indexierung der Sichten (Views) ist nützlich, wenn die Sichten Tabellen Joins enthalten

Allgemeine Index Entwurf Guidelines

- Anfragen Überlegungen:
 - Erstelle nicht-geclusterten Indexe für Spalten, die oft in WHERE und JOIN Klauseln benutzt werden
 - Covering Indexe können die Effizienz verbessern
 - Schreibe Anfragen, die so viele wie möglich Tupeln in derselben Anweisung einfügt oder ändert

Allgemeine Index Entwurf Guidelines

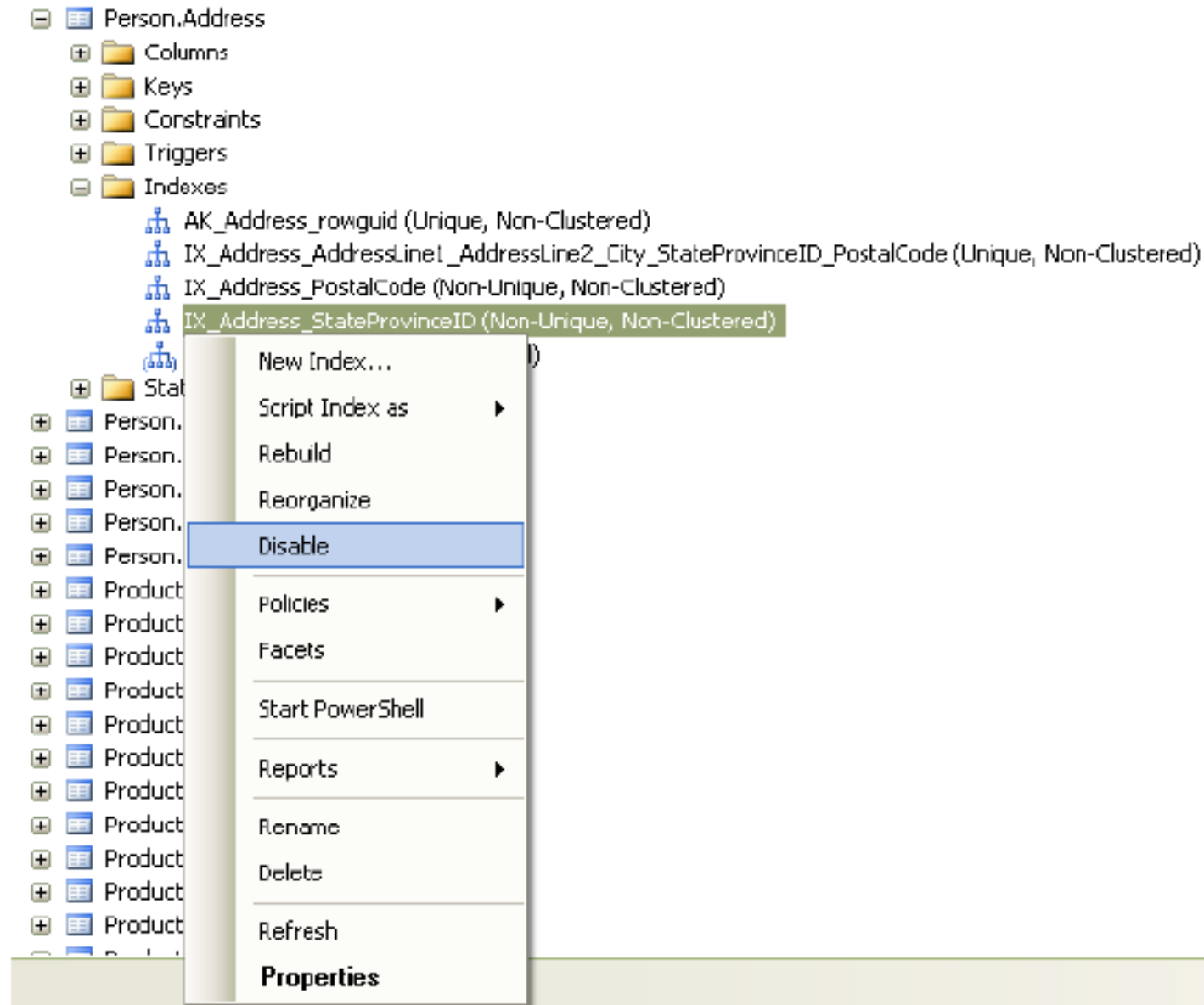
- Spalten Überlegungen:
 - Die Länge des Suchschlüssels für einen geclusterten Index soll „klein“ sein
 - Es ist besser geclusterte Indexe auf eindeutige/ nicht-NULL Spalten zu erstellen
 - Spalten mit dem Datentyp **ntext**, **text**, **image**, **varchar(MAX)**, **nvarchar(MAX)**, **varbinary(MAX)** können nicht in dem Suchschlüssel des Index enthalten sein
 - Bestimme ob die Spalte eindeutig ist oder nicht
 - Bestimme die Verteilung der Werte in der Spalte (vermeide Indexe auf Spalten mit einem sehr kleinen Wertebereich) – benutze gefilterte Indexe
 - Bestimme die Reihenfolge der Spalten für einen Index mit zusammengesetztem Suchschlüssel

Spalten, die in Bedingungen mit “=”, “<”, “>” oder BETWEEN vorkommen, sollen als erste positioniert werden. Der Rest der Spalten soll man folgendermaße ordnen: von dem größten Wertebereich zu dem kleinsten Wertebereich

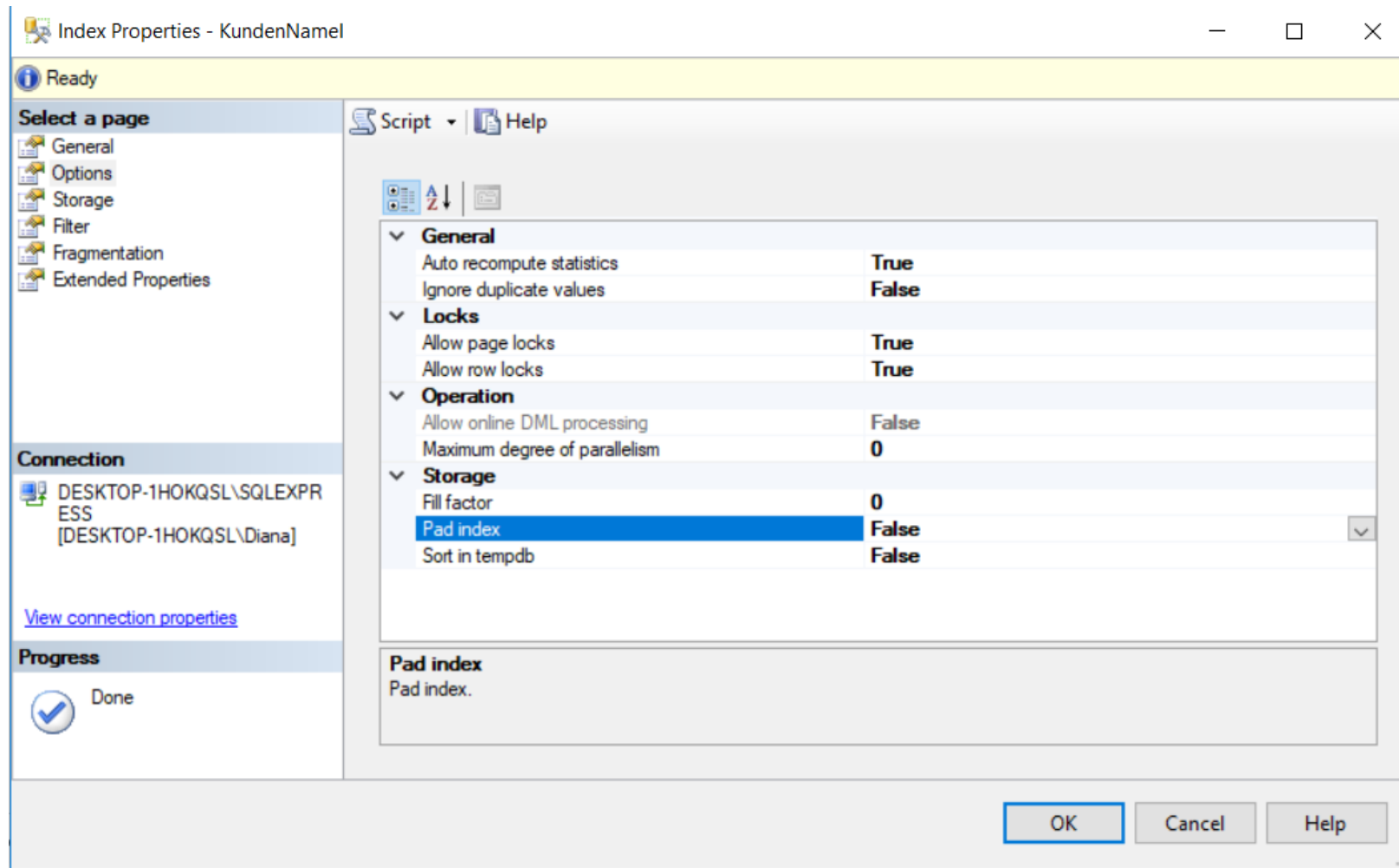
Index Deaktivieren

```
ALTER INDEX IX_Address_StateProvinceID  
ON Person DISABLE
```

Index Deaktivieren



Index Properties



- Mehrere Informationen: [https://technet.microsoft.com/en-us/library/ms186872\(v=sql.105\).aspx](https://technet.microsoft.com/en-us/library/ms186872(v=sql.105).aspx)

Index Aktivieren

```
ALTER INDEX IX_Address_StateProvinceID  
ON Person REBUILD
```

Indexe für Löschen (DELETE)

- Beim DELETE:
 - SQL Server sucht nach abhängigen Zeilen, indem alle Fremdschlüssel überprüft werden
 - Er sucht also in allen zusammenhängenden Tabellen:
 - Falls es einen Index gibt, dann benutzt SQL Server den Index um in diesen Tabellen Daten zu suchen
 - Falls es keinen Index gibt, dann muss SQL Server die ganze Tabelle scannen um Daten zu suchen
 - DELETEs können sehr langsam sein, wenn es keinen Index gibt für die Fremdschlüssel

Indizierten Sichten

- Anforderungen:
 - Die SELECT Klausel darf nicht andere Sichten benutzen
 - Die Sichtsdefinition muss *deterministisch* sein (Eine Sicht ist deterministisch, wenn alle Ausdrücke in der Auswahlliste sowie die WHERE-Klausel und die GROUP BY-Klausel deterministisch sind – geben stets dasselbe Ergebnis zurück, für dasselbe Input; z.B. GETDATE ist nicht deterministisch)
 - AVG, MIN, MAX, STDEV, STDEVP, VAR (die letzten drei sind statistische Funktionen)
 - Der Index für die Sicht muss eindeutig und geclustert sein
 - SELECT Klausel muss keine Unterabfragen, Outer Joins, EXCEPT, INTERSECT, TOP, UNION, ORDER BY, DISTINCT, ... enthalten

Columnstore Indexe

- Stellt den Standard für das Speichern und Abfragen großer Faktentabellen im Data Warehousing dar (read-only Tabellen)
- Verwendet spaltenbasierte Datenspeicherung und Abfragenverarbeitung
- Bis zu 10x höhere Abfrageleistung im Vergleich zu der zeilenorientierter Speicherung
- Bis zu 10x Datenkomprimierung im Vergleich zu der unkomprimierten Datenvolumen
- Dieselbe Tabelle kann einen zeilenorientierten und einen spaltenorientierten Index haben → der Abfrageoptimierer entscheidet welchen Index er braucht
- Für mehrere Unterschiede zwischen columnstore und rowstore Indexe:
<http://www.sqlservercentral.com/articles/ColumnStore+Index/125264/>

Row store for B-Tree or Heap

Row 1	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 2	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 3	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 4	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 5	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10

Page 1

Row 6	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 7	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 8	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
.....	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row n	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10

Page 2

Column Store Index

Row 1	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 2	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 3	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 4	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 5	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 6	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 7	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 8	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
.....	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row n	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
	Page 1	Page 2	Page 3	Page 4	Page 5	Page 6	Page 7	Page 8	Page 9	Page 10

Regeln für Indexierung

- Jede Tabelle sollte einen geclusterten Index haben mit folgenden Eigenschaften:
 - (idealerweise) klein,
 - selektiv (eindeutig oder „fast“ eindeutig),
 - immer aufsteigend (wähle ein Suchschlüssel, der für neue Tupeln immer größer wird – z.B. Automatisch inkrementierter Id) und
 - Statisch – ändert sich nicht viel

→ eine Tabelle ohne einen geclusterten Index heißt Haufendatei (heap)
- Erstelle nicht-geclusterte Indexe für Fremdschlüsseln
- Erstelle nicht-geclusterte Indexe on Spalten, die oft in WHERE Klauseln vorkommen

Regeln für Indexierung

- Erstelle kein Index mit einfachem Schlüssel für jede Spalte in der Tabelle → dann wird es zu viele Indexe geben, nicht effizient
- Bei Indexen mit zusammengesetzten Schlüsseln, ordne die Spalten so, dass die am meisten selektiven (→ am nächsten zu Eindeutigkeit) die ersten sind
- Für die Abfragen, die am meisten vorkommen, erstelle covering, nicht-geclusterte Indexe

Indexfragmentierung

- Interne Fragmentierung: wenn es freien Platz gibt zwischen Datensätze in einer Seite
 - Der freie Platz verursacht schlechte Cache Ausnutzung und dadurch mehrere I/O Operationen
- Externe Fragmentierung: wenn die Seiten auf die Platte nicht in aufeinanderfolgende Reihenfolge gespeichert sind
- Logische Fragmentierung: wenn die Blattseiten eines Heaps/Index nicht ordnungsgemäß sortiert sind

Indexfragmentierung

- `sys.dm_db_index_physical_stats`
 - `avg_fragmentation_in_percent`: durchschnittlicher Prozentsatz der logischen Fragmentierung (falsche Reihenfolge der Seiten in einem Index).
 - `avg_page_space_used_in_percent`: durchschnittlicher Prozentsatz des auf allen Seiten verwendeten verfügbaren Datenspeicherplatzes.

Beheben der Fragmentierung

- Für Haufendateien :
 - Erstelle einen geclusterten Index: sortiere die Tupeln in einer bestimmten Reihenfolge und speichere die Seiten aufeinanderfolgend auf die Platte
- Für Indexe:
 - Durch Neuorganisieren oder Neuerstellen des Index
 - `avg_fragmentation_in_percent`:
 - `>5% und < 30%` → `ALTER INDEX REORGANIZE`
 - Die Blätterseiten des Index werden in einer logischen Reihenfolge sortiert und reorganisiert
 - `> 30%` → `ALTER INDEX REBUILD`
 - In diesem Fall können wir den Index auch löschen und neu erstellen