# Index – Workshop

# Videos zum Thema: <https://www.linkedin.com/learning/search?keywords=sql%20index&u=2976210>

# Ziel: Indexe in Funktion, Ausprägung und Ausführung kennen und anwenden

# Methode: Selbstständiges erarbeiten des Themas mit Hilfe des Internets (Google) und Lehrerinput

**Tipps:**

<https://entwickler.de/online/datenbanken/spezielle-indexformen-241163.html>

<https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-2017>

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql?view=sql-server-2017>

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-xml-index-transact-sql?view=sql-server-2017>

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-spatial-index-transact-sql?view=sql-server-2017>

https://www.mssqltips.com/sqlservertutorial/9138/sql-server-spatial-indexes/

<https://docs.microsoft.com/de-de/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-2017>

**Theorie:**



# Auftrag: Lösen sie die Aufgaben ‘Einstieg’ und 1 bis 11 und halten sie ihre Ergebnisse in diesem Worddokument fest! Es dient dann später bei der Prüfungsvorbereitung als *Know-How*-Sammlung.

# Einstieg

Welche und wie viele Indexarten kennt der aktuelle SQL-Server von Microsoft und wozu dienen diese?  
Notieren sie sich je Indexart zwei bis drei Zeilen, die den Index erklären.

Tipp: 8 verschiedene gibt es! (<https://docs.microsoft.com/de-de/sql/relational-databases/indexes/indexes?view=sql-server-2017> )  
 ‚Rechtsklick‘ im SSMS auf einem Tabellenobjekt im Bereich ‚Indizes‘ zeigt sieben davon!

# Aufgabe 1

Sie benötigen für die folgenden Aufgaben eine ‚grosse‘ Tabelle (ca. 10 Mio. Datensätze):

Das untenstehende SQL-Skript erzeugt in einem Schritt die Tabelle ‘*TestData2*’ und fügt 106 Datensätze ein.  
Die Lehrperson erklärt im Detail wie das SQL-Skript funktioniert!

create Database Test

go

use Test

SELECT TOP 10000000 -- 106 <<<LOOK! CHANGE THIS NUMBER TO CHANGE THE NUMBER OF ROWS!

RowNum = IDENTITY(INT, 1, 1),

SomeID = ABS(CHECKSUM(NEWID()))%999000+1, --<<<LOOK! CHANGE THIS NUMBER TO 1/400th THE ROW COUNT

SomeCode = CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)

INTO TestData2

FROM Master.dbo.SysColumns t1, Master.dbo.SysColumns t2

* Führen sie eine Restart des SQL-Server-Dienstes durch damit gecachte Datensätze die Messung nicht verfälschen!
* Wie viele Datensätze hat die Tabelle TestData2? ……………………
* Wie gross in MByte ist die mdf-Datei der Datenbank? 34MB
* Wie gross in MByte ist die ldf-Datei der Datenbank? 8MB

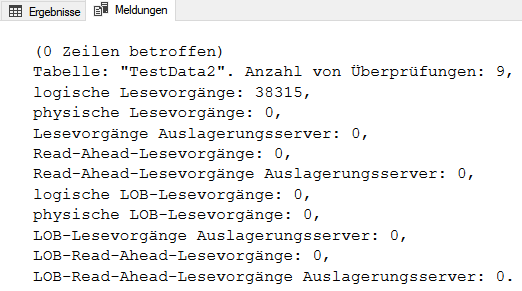
**Messen sie jetzt die Dauer der SQL-Abfrage** (nach Restart des SQL-Server-Dienstes):  
 SET STATISTICS IO, TIME ON;

-- Clean buffers so cold start performed

DBCC DROPCLEANBUFFERS

select \* from TestData2 where SomeCode='WQTLLGLLF'

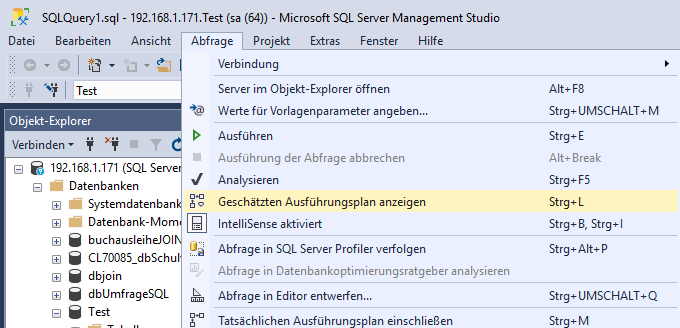
SET STATISTICS IO OFF

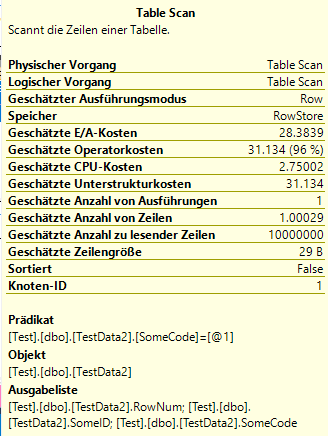
  
  
siehe dazu: <https://docs.microsoft.com/de-de/sql/t-sql/statements/set-statistics-io-transact-sql?view=sql-server-ver15>

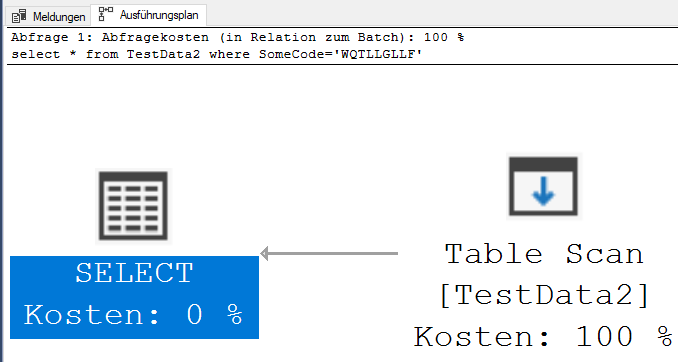
large object (LOB)

LOB-Speicher (Large Object)

|  |  |
| --- | --- |
| Ausgabeelement | Bedeutung |
| **Table** | Der Name der Tabelle. |
| **Scananzahl** | Die Anzahl von Suchen oder Scans, die nach Erreichen der Blattebene in beliebiger Richtung gestartet wurden, um alle Werte zum Erstellen des letzten Datasets für die Ausgabe abzurufen.  Die Scananzahl beträgt „0“ (null), wenn der verwendete Index ein eindeutiger Index oder ein gruppierter Index für eine Primärschlüsselspalte ist und Sie nur einen Wert suchen.  Beispiel: WHERE Primary\_Key\_Column = <value>.  Die Scananzahl beträgt „1“, wenn Sie anhand eines nicht eindeutig gruppierten Index, der für eine Nicht-Primärschlüsselspalte definiert ist, nach einem Wert suchen. Auf diese Weise wird nach doppelten Werten eines Schlüsselwerts gesucht, der als Suchwert verwendet wird. Beispiel: WHERE Clustered\_Index\_Key\_Column = <value>.  Die Scananzahl ist „N“, wenn „N“ der Anzahl unterschiedlicher Suchen oder Scans entspricht, die auf der Blattebene nach links oder rechts gestartet wurden, nachdem ein Schlüsselwert anhand des Indexschlüssels ermittelt wurde. |
| **Logische Lesevorgänge** | Anzahl der aus dem Datencache gelesenen Seiten |
| **Physische Lesevorgänge** | Anzahl der vom Datenträger gelesenen Seiten |
| **Read-Ahead-Lesevorgänge** | Anzahl der Seiten, die für die Abfrage im Cache platziert wurden |
| **Logische LOB-Lesevorgänge** | Anzahl der aus dem Datencache gelesenen Seiten Umfasst **text**, **ntext**, **image**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)** oder Columnstore-Indexseiten. |
| **Physische LOB-Lesevorgänge** | Anzahl der vom Datenträger gelesenen Seiten Umfasst **text**, **ntext**, **image**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)** oder Columnstore-Indexseiten. |
| **Read-Ahead-LOB-Lesevorgänge** | Anzahl der Seiten, die für die Abfrage im Cache platziert wurden Umfasst **text**, **ntext**, **image**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)** oder Columnstore-Indexseiten. |

Lassen sie sich die obige SQL-Abfrage vom SSMS ‚erklären‘: **CTRL-L im SSMS ausführen!****  
  
🡪**





Klick

Der **Ausführungsplan** weisst einen sogenannten *Table Scan* aus.

Klären sie via Google was das bedeutet! <https://use-the-index-luke.com/de/sql/ausfuehrungsplaene/sql-server/operationen-bedeutung>

**Zum Executenplan:**

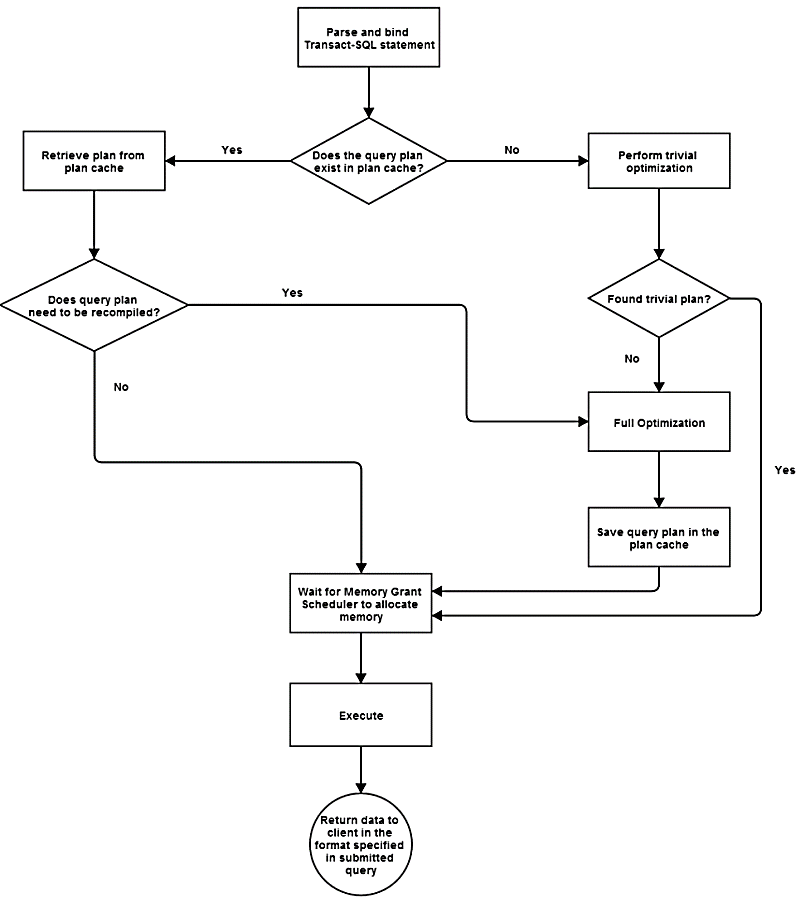
Ausgerüstet mit dem Ausführungsplan haben Sie einen einzigartigen Einblick in das, was im Hintergrund des SQL Server abläuft und eine Fülle von Informationen darüber, wie sich dewr SQL Server für eine Lösung entschieden hat

Sie können Dinge sehen wie:

* Die Reihenfolge, in der das Optimierungsprogramm auf die in der Abfrage referenzierten Tabellen zugreift.
* Welche Indizes es für jede Tabelle verwendete und wie die Daten von ihnen abgerufen wurden.
* Wie viele Zeilen der Optimierer basierend auf der Anzahl seiner Zeilen für einen Operator zurückgegeben hat

statistisches Verständnis der zugrunde liegenden Datenstrukturen und Daten und wie viele Zeilen fand es in der Realität.

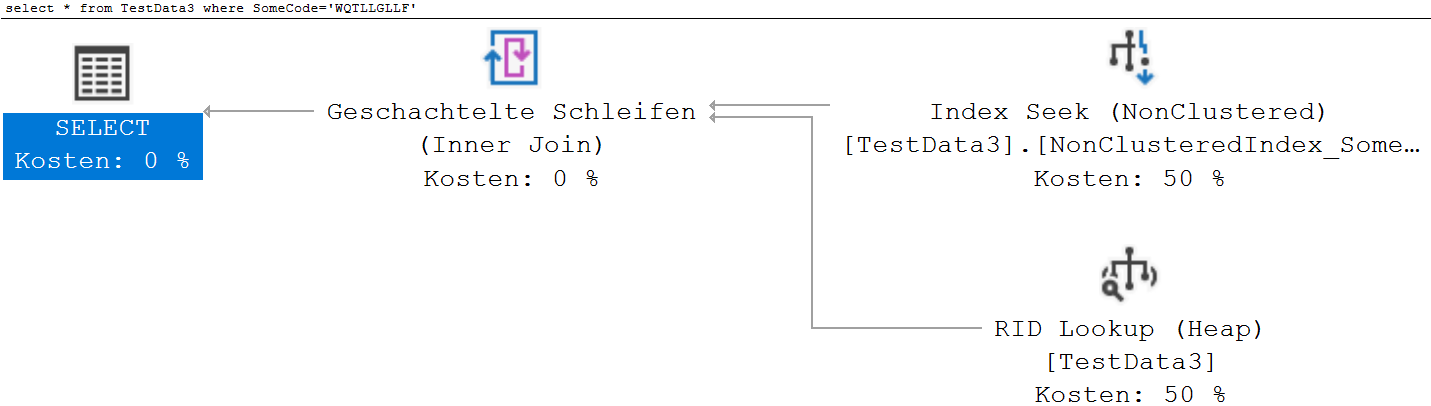
* Wie sich Schlüssel und referenzielle Einschränkungen auf das Verständnis des Optimierers von auswirken
* Daten und damit das Verhalten Ihrer Abfragen.
* Wie Daten zwischen den Tabellen in Ihrer Abfrage verknüpft werden.
* Bei der Filterung und Sortierung durchgeführte Berechnungen und Aggregationen.



**Erstellen sie jetzt einen NONCLUSTERED Index für das Attribut ‚*SomeCode*‘   
und messen sie die Zeit nochmals (**<https://www.mssqltips.com/sqlservertutorial/9133/sql-server-nonclustered-indexes/> **):**

Wie gross in MByte ist die mdf-Datei nach Erstellen des 1. Indexes? 98MB

Wie viel Disc-Platz in MByte wird pro Index benötigt? 72MB

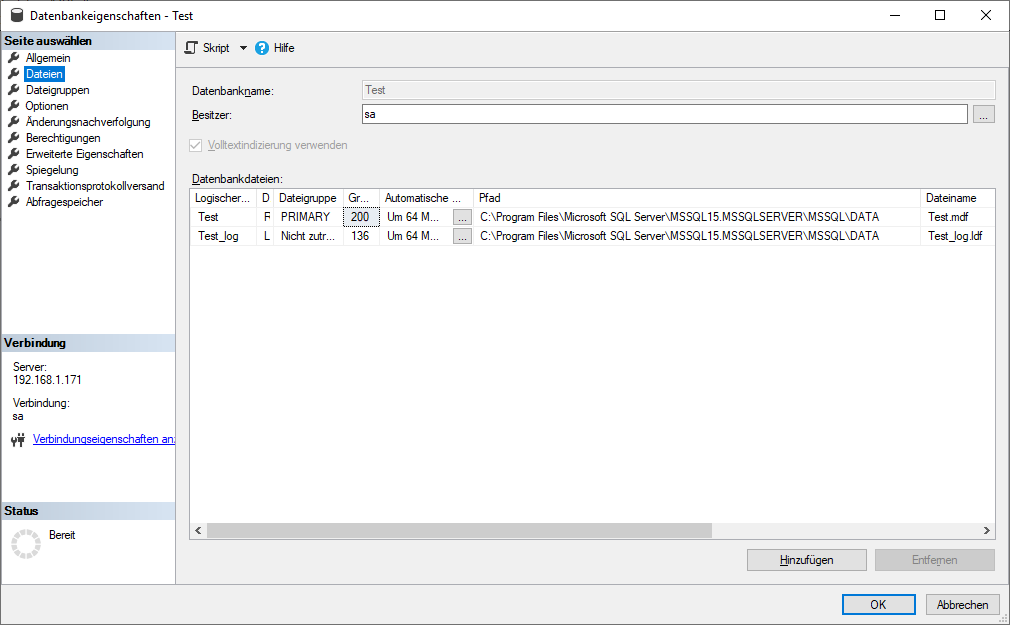
Zeigen sie sich den Ausführungsplan nochmals an!  


select \* from TestData2 where SomeCode='WQTLLGLLF'

Zeit in Sekunden 0 [sek]

Notieren sie hier das SQL für den ersten Index den sie erstellten:

create nonclustered index . . . . . . . . . . . . . . . . . . . . . .



Hier finden sie die mdf & lfd - Dateien

Der Ausführungsplan weisst jetzt neu einen sogenannten *Index-Seek* aus.   
Klären sie via Google was das bedeutet! <https://use-the-index-luke.com/de/sql/ausfuehrungsplaene/sql-server/operationen-bedeutung>

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql?view=sql-server-2017>

# Aufgabe 2

Wie lange dauert ein ‚Full-Table-Scan‘ wenn die Tabelle 12GByte Daten aufweist und sie einen Lesedurchsatz von 30MByte pro Sekunde garantieren können?

………………………..

# Jetzt ist der Lehrer dran! Theorie B-Tree!

# Aufgabe 3

Berechnen sie die Höhe z des B-Trees (siehe Theorie: ‚Indexe Schüler.docx ‘):

Formel: log2d+1(n+1) <= z <= 1+logd+1((n+1)/2)

d=4 (Ordnung)

n=20 (Anzahl der Blätter) ………………………..

und

d=3 (Ordnung)

n=100 (Anzahl der Blätter) ………………………..

# Aufgabe 4

Die Anzahl der Disczugriffe z ist proportional zur Höhe eines B-Trees (siehe Theorie: ‚Indexe Schüler.docx ‘):

Formel:log2d+1(n+1) <= z <= 1+logd+1((n+1)/2)

Berechnen sie die maximale Anzahl der Disczugriffe für folgende B-Trees:

d=100 (Ordnung)

n=20000 (Anzahl der Blätter) ………………………..

und

d=3000 (Ordnung)

n=1000000 (Anzahl der Blätter) ………………………..

# Aufgabe 5a

Die Programmiersprache Transact-SQL stellt DBCC-Anweisungen bereit, die als Datenbankkonsolenbefehle (Database Console Commands, DBCC) für SQL Server dienen.

DBCC-Anweisungen sind in die folgenden Kategorien eingeteilt.

**Befehlskategorie** **Funktion**

Verwaltung Aufgaben zur Verwaltung von Datenbanken, Indizes und Dateigruppen.

Sonstiges Verschiedene Aufgaben wie das Aktivieren von Ablaufverfolgungsflags oder   
 das Entfernen einer DLL aus dem Arbeitsspeicher.

Information Aufgaben zum Sammeln und Anzeigen verschiedener Arten von Informationen.

Überprüfung Überprüfungsvorgänge für Datenbanken, Tabellen, Indizes, Kataloge, Dateigruppen oder das Zuordnen von Datenbankseiten.

DBCC-Befehle akzeptieren Eingabeparameter und geben Werte zurück.

Alle Parameter für DBCC-Befehle nehmen sowohl Unicode- als auch DBCS-Literale (Double-Byte Character Set, Doppelbyte-Zeichensatz) an.

**Versuchen sie nun die folgende Analyse durchzuführen**:

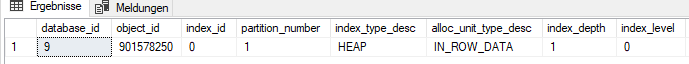
Wie viele Seiten (DB-Pages à 8kB) belegt die Tabelle ‚TestData2’?

Hinweis: DBCC (Google hilft): <https://docs.microsoft.com/de-de/sql/t-sql/database-console-commands/dbcc-transact-sql?view=sql-server-2017>

# Aufgabe 5b

Ermitteln sie die Indextiefe ihrer 10Mio-Tabelle (TestData2)!

Tipp: <https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-index-physical-stats-transact-sql?view=sql-server-2017>

Sieht dann etwa so aus!  


# Aufgabe 6

Erzeugen sie einen B-Tree. Fügen sie der Reihe nach folgende Werte in einen B-Tree der Ordnung 1 ein:

10, 22, 33, 44, 23

**Fertig?**

Löschen sie jetzt den Index 10!

# Aufgabe 7

Demonstrieren sie den Unterschied:

Wie sieht der Ausführungsplan aus

Was wird anders abgearbeitet

zwischen einem ***gruppierten Index*** und einem ***nicht gruppierten Index*** an einem anschaulichen Beispiel.

Tipp: <https://docs.microsoft.com/de-de/sql/relational-databases/performance/display-the-estimated-execution-plan?view=sql-server-2017>

**Vorgehensvorschlag:**

1. Lassen sie sich den Ausführungsplan A einer SQL-Abfrage anzeigen, welche nach einem Attribut A filtert, für das ein NONCLUSTERED-Index erstellt wurde. Speichern sie den Ausführungsplan (rechtsklick im Plan).  
   (Tipp: siehe Aufgabe 1)
2. Löschen sie den NONCLUSTERED-Index für das Attribut A!
3. Erstellen sie einen CLUSTERED-Index für dasselbe Attribut A!
4. Lassen sie sich den Ausführungsplan B derselben SQL-Abfrage anzeigen, welche nach einem Attribut A filtert, für das ein CLUSTERED-Index erstellt wurde. Speichern sie den Ausführungsplan (rechtsklick im Plan).
5. Vergleichen sie die Ausführungspläne: Was ist anders, was fällt auf?

# Aufgabe 8

Können Indexe auf berechnete Tabellen-Attribute erstellt werden?

# Aufgabe 9

Können Indexe auf Sichten-Attribute (gespeicherte Abfragen) erstellt werden?

# Aufgabe 10

Ermitteln sie die Fragmentierung (nicht mehr optimale B-Trees) aller Indexe einer Datenbank ihrer Wahl!

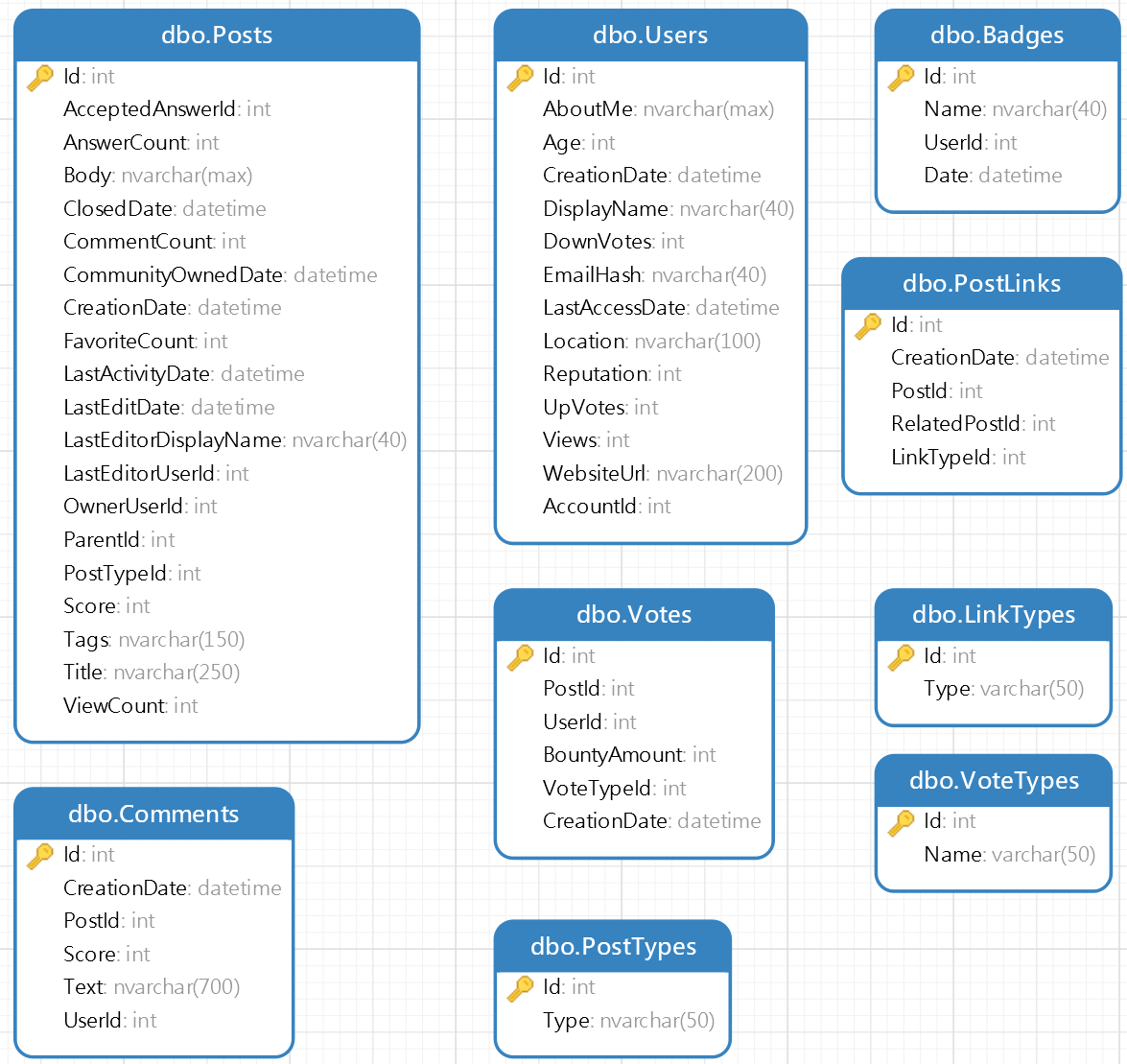
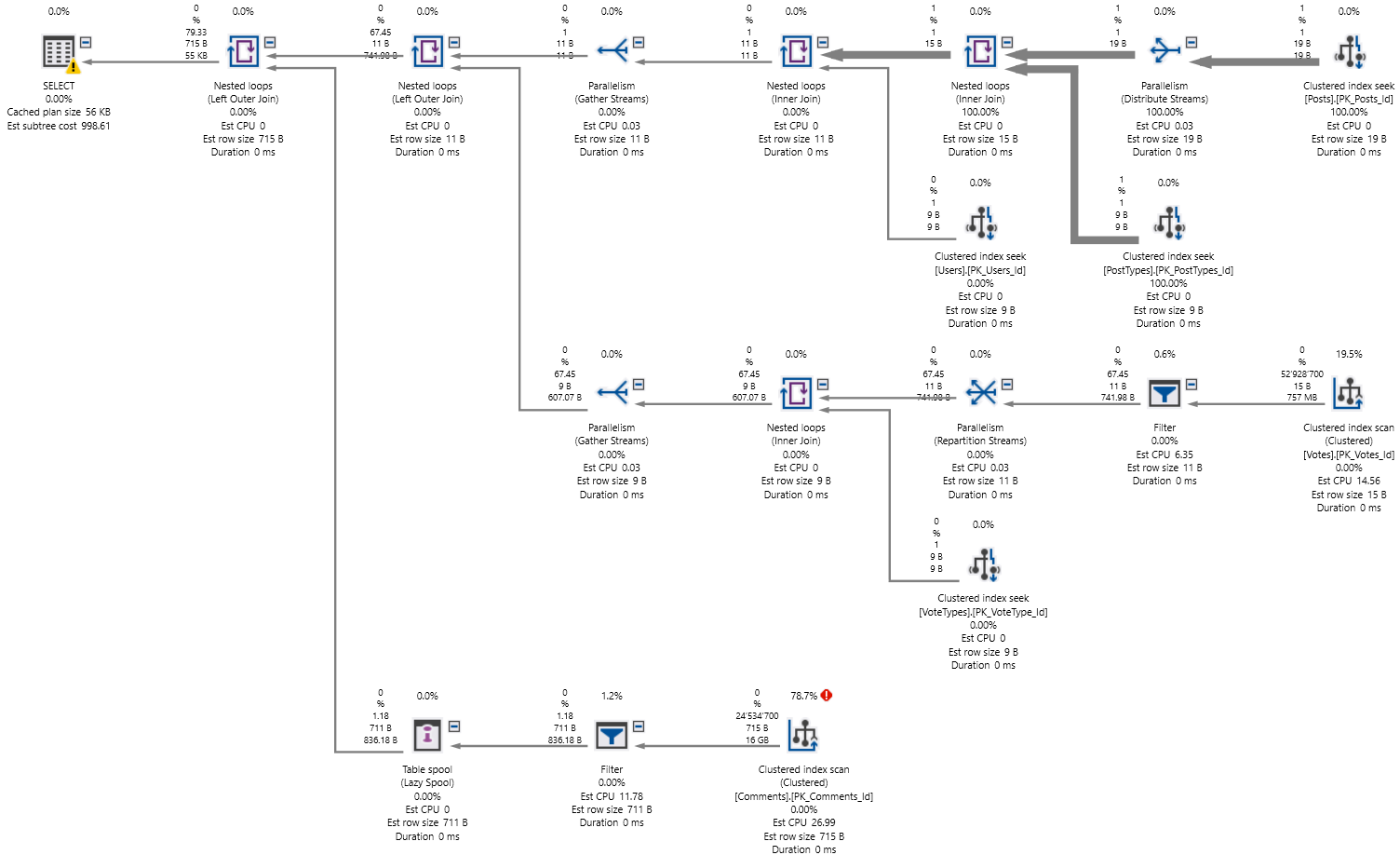
Tipp: Google hilft!

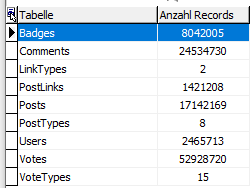
# Aufgabe 11

Defragmentieren sie alle Indexe einer Tabelle ihrer Wahl!

Tipp: Google hilft!

# Aufgabe 12



SELECT Posts.Id, Comments.Text

FROM Comments

RIGHT OUTER JOIN Posts

INNER JOIN PostTypes ON Posts.PostTypeId = PostTypes.Id

INNER JOIN Users ON Posts.OwnerUserId = Users.Id ON Comments.PostId = Posts.Id

LEFT OUTER JOIN VoteTypes

INNER JOIN Votes ON VoteTypes.Id = Votes.VoteTypeId ON Posts.Id = Votes.PostId

WHERE Posts.Id=1000000

Erstellen sie die nötigen Indexe um die obige SQL-Abfrage zu beschleunigen. Zurzeit haben alle Tabellen einzig den Primärschlüssel indexiert.

# Aufgabe 13 (Reserve Aufgabe)

Suchen sie ein Skript im Internet, das alle Indexe einer Datenbank ***reorganisiert***!

# Aufgabe 14 (Reserve Aufgabe)

Erstellen sie einen Spatial-Index und testen sie diesen an einem Beispiel

# 

# Aufgabe 15 (Reserve Aufgabe)

Erstellen sie eine XML-Index und testen sie diesen an einem Beispiel. Verwenden sie dazu ein Tutorial ihrer Wahl aus der untenstehenden Linkliste.

**Tipps/ Linkliste:**

# https://www.red-gate.com/simple-talk/sql/database-administration/getting-started-with-xml-indexes/

# <https://www.mssqltips.com/sqlservertutorial/9137/sql-server-xml-indexes/>

# <https://docs.microsoft.com/en-us/sql/relational-databases/xml/xml-indexes-sql-server?view=sql-server-2017>

<https://www.red-gate.com/simple-talk/sql/database-administration/getting-started-with-xml-indexes/>

# 

# Weiterführende Links

<https://www.sqlshack.com/sql-execution-plan-enhancements-in-ssms-18-0/>

**Freies Buch zum Thema:**

SQL Server Execution Plans by G.Fritchey

<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=2ahUKEwjIlOPAqMbhAhUWysQBHT_eCtIQFjAAegQIBBAC&url=https%3A%2F%2Fassets.red-gate.com%2Fcommunity%2Fbooks%2Fsql-server-execution-plans-3rd-edition.pdf&usg=AOvVaw3FgXAzxZiixdrBySYmIaJu>

# Anhang

‘Grosse’ Tabelle erzeugen:

create Database Test\_LB2

go

use Test\_LB2

SELECT TOP 5000000 -- 5\*106 <<<LOOK! CHANGE THIS NUMBER TO CHANGE THE NUMBER OF ROWS!

RowNum = IDENTITY(INT, 1, 1),

SomeID = ABS(CHECKSUM(NEWID()))%999999+1, --<<<LOOK! CHANGE THIS NUMBER TO 1/400th THE ROW COUNT

SomeCode = CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65),

SomeCodeB = CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)+

CHAR(ABS(CHECKSUM(NEWID()))%26+65)

INTO TestData2

FROM Master.dbo.SysColumns t1, Master.dbo.SysColumns t2

**Für Applikationsinformatiker: Mit Python und SQL-Server-Python**

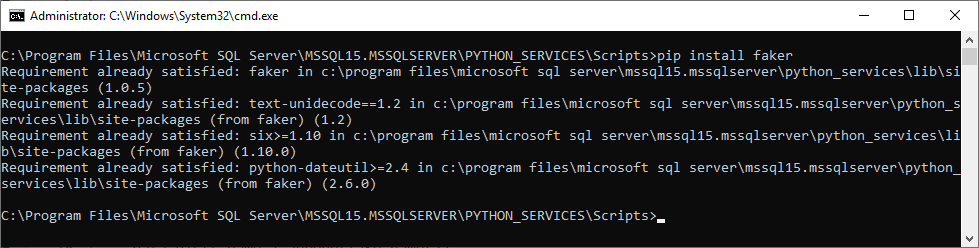
<https://richbenner.com/2019/02/test-data-python-faker/> <https://github.com/joke2k/faker>

<https://richbenner.com/2019/01/installing-pip-on-sql-server-with-python/>

<https://www.datacamp.com/community/tutorials/generate-data-trumania>

<https://github.com/tirthajyoti/pydbgen>

**C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\PYTHON\_SERVICES\Scripts\pip.exe**

pip install Faker  


DECLARE @sql nvarchar(max)

SET @sql =

N'#! usr/bin/env python3

import pandas as pd

import pydbgen

from pydbgen import pydbgen

myDB=pydbgen.pydb()

myDB2=pydbgen.pydb()

from faker import Faker

fake = Faker(''de\_DE'')

# data dictionaries is how python stores tables

data\_dict = {

"address":[],

"email":[],

"job\_title":[],

"name":[]

}

testdf=myDB2.gen\_dataframe(1000,[''email'',''job\_title''])

# we are inserting 1000 random names and addresses into this dictionary

for x in range(1000):

data\_dict["address"].append(fake.address())

data\_dict["email"].append(testdf.iat[x,0])

data\_dict["job\_title"].append(testdf.iat[x,1])

data\_dict["name"].append(fake.name())

# using pandas to create a data frame makes it into a more presentable format

output\_data = pd.DataFrame(data\_dict)

# this is the data we''re going to export to SQL Server

OutputDataSet = output\_data'

IF OBJECT\_ID('tempdb..#FakeData') IS NOT NULL DROP TABLE #FakeData

BEGIN

CREATE TABLE #FakeData

(

[address] varchar(50)

,email varchar(100)

,job\_title varchar(100)

,[name] varchar(30)

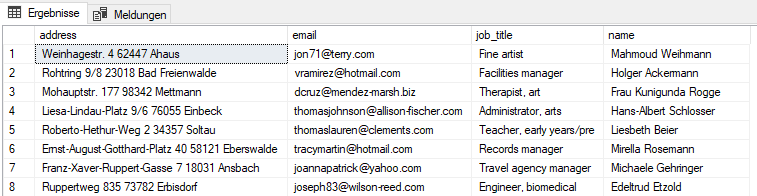
)

END

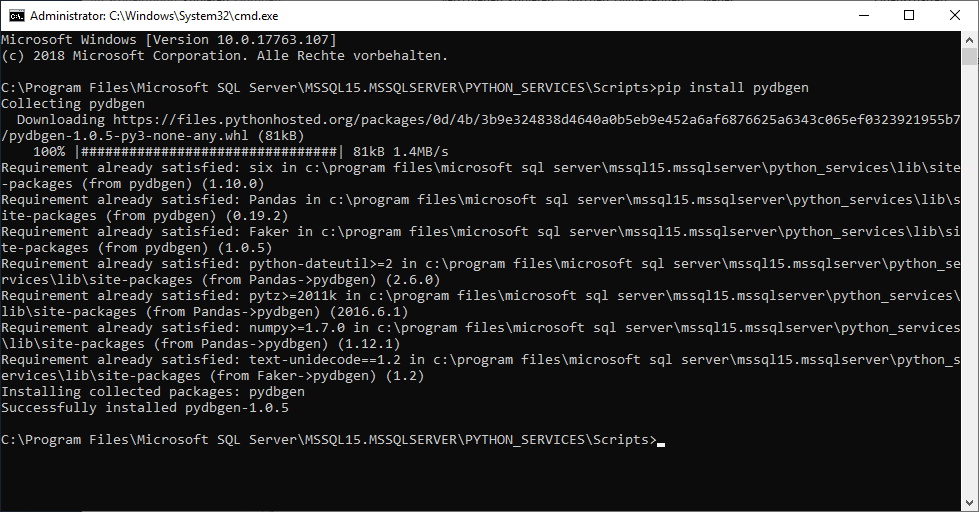
INSERT INTO #FakeData (address,email,job\_title,name) /\* data comes out by column name alphabetically \*/

EXECUTE sp\_execute\_external\_script @language = N'Python', @script = @sql

SELECT \* FROM #FakeData





pip install pydbgen (C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\PYTHON\_SERVICES\Scripts)

**Für Applikationsinformatiker: Andere Python-Variante um eine Exceldatei zu erstellen**

**ACHTUNG: Pyton IDE benötigt! Z. B: Anaconda**

import pydbgen

from pydbgen import pydbgen

myDB=pydbgen.pydb()

#myDB.city\_real()

#for \_ in range(10):

# print(myDB.license\_plate())

#print(myDB.gen\_data\_series(num=8,data\_type='city'))

#testdf=myDB.gen\_dataframe(5,['name','city','phone','email','date'])

#print(testdf)

myDB.gen\_excel(num=20,fields=['name','phone','year','time','country','city','company','job\_title','zipcode','email'],

phone\_simple=False,filename='c:\Temp\TestExcel.xlsx',real\_email=False)

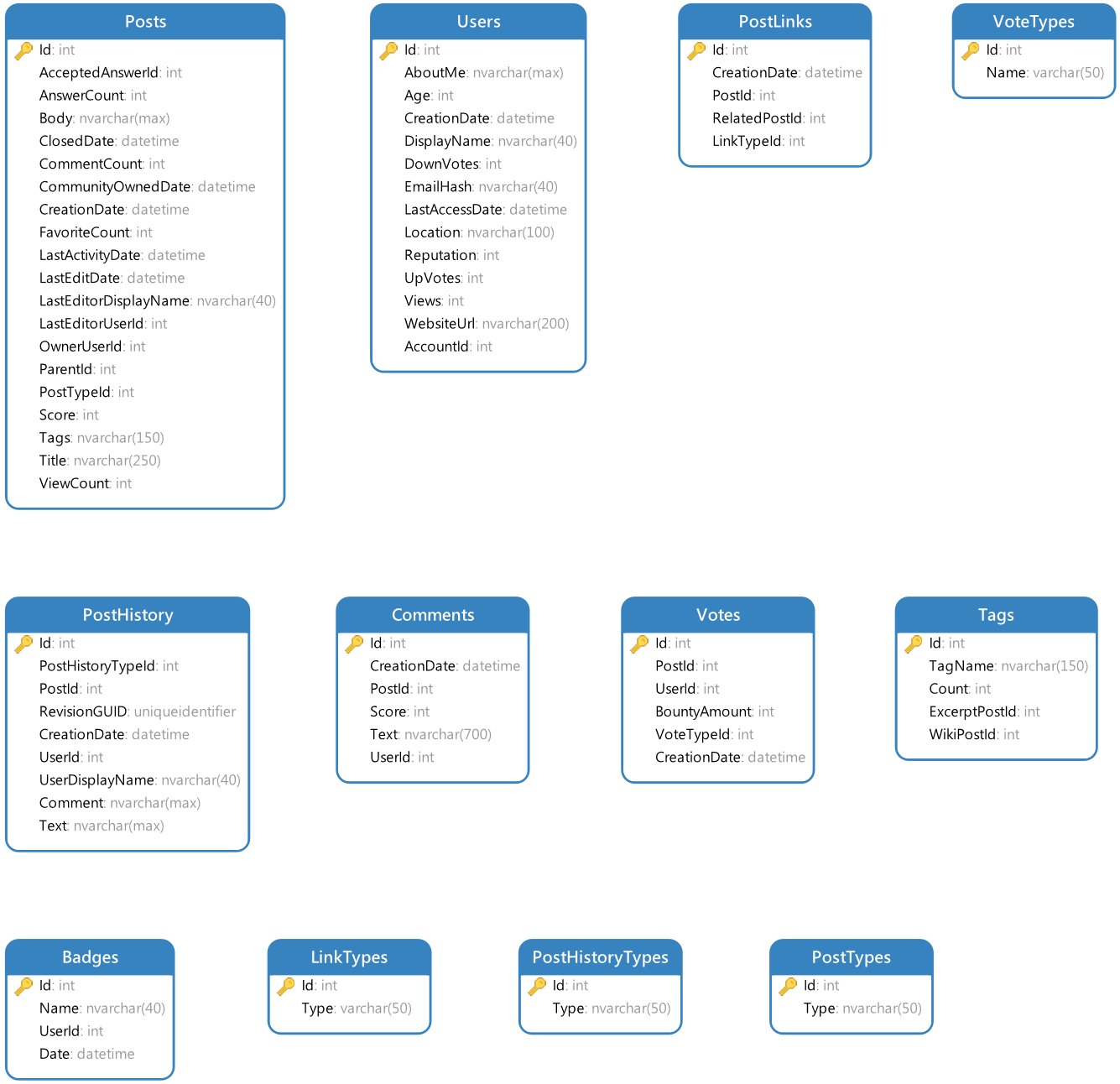


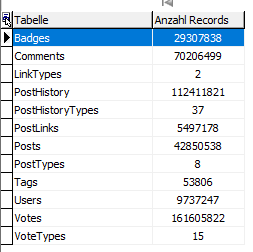
**Andere ‘grosse’ Datenbank zum Üben mit Indexen und Abfragen**

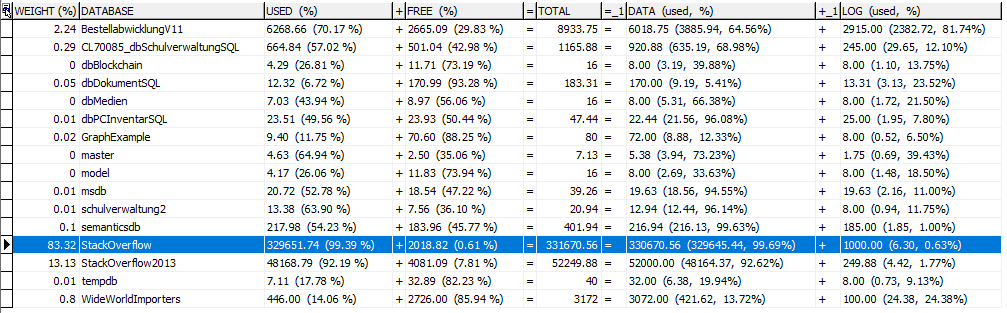
Zum Download (42GByte expandiert nach 324GByte): NUR AUF SERVER-HARDWARE UMSETZEN!!

<https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/>

<https://meta.stackexchange.com/questions/2677/database-schema-documentation-for-the-public-data-dump-and-sede>







Die grosse Datenbank Online schalten:

USE master

GO

CREATE DATABASE StackOverflow ON

( FILENAME = 'F:\BigDatabase\StackOverflow\_1.mdf' ),

( FILENAME = 'F:\BigDatabase\StackOverflow\_log.ldf' ),

( FILENAME = 'F:\BigDatabase\StackOverflow\_2.ndf' ),

( FILENAME = 'F:\BigDatabase\StackOverflow\_3.ndf' ),

( FILENAME = 'F:\BigDatabase\StackOverflow\_4.ndf' )

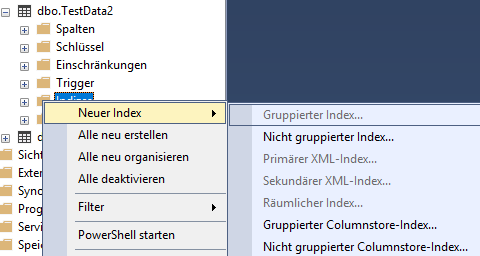
FOR ATTACH

# Lösungen

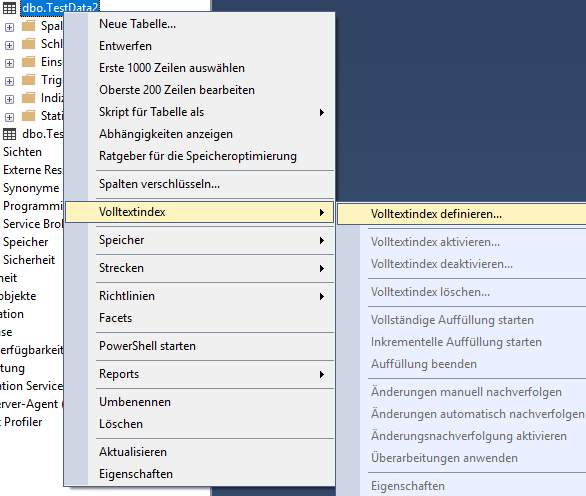
# Einstieg

Welche und wie viele Indexarten kennt der aktuelle SQL-Server von Microsoft?

Tipp: 8 verschiedene gibt es!

 sieben hier

Auf ‚Tabellenebene‘:

 und einer da!

**Gruppiert/Clustered**

Clustered-Indizes sortieren und speichern die Datenzeilen in der Tabelle oder Ansicht basierend auf ihren Schlüsselwerten. Dies sind die in der Indexdefinition enthaltenen Spalten. Es kann nur einen Clustered-Index pro Tabelle geben, da die Datenzeilen selbst nur in einer Reihenfolge gespeichert werden können.

Die Datenzeilen in einer Tabelle werden nur in sortierter Reihenfolge gespeichert, wenn die Tabelle einen gruppierten Index enthält. Wenn eine Tabelle einen gruppierten Index hat, wird sie als gruppierte Tabelle bezeichnet. Wenn eine Tabelle keinen Clustered-Index hat, werden ihre Datenzeilen in einer ungeordneten Struktur gespeichert, die als Heap bezeichnet wird.

**Nicht gruppiert**

Nicht gruppierte Indizes haben eine Struktur, die von den Datenzeilen getrennt ist. Ein nicht gruppierter Index enthält die nicht gruppierten Indexschlüsselwerte, und jeder Schlüsselwerteintrag weist einen Zeiger auf die Datenzeile auf, die den Schlüsselwert enthält.

Der Zeiger von einer Indexzeile in einem nicht gruppierten Index auf eine Datenzeile wird als Zeilenlokator bezeichnet. Die Struktur des Zeilenlokators hängt davon ab, ob die Datenseiten in einem Heap oder in einer Clustertabelle gespeichert sind. Bei einem Heap ist ein Zeilenlokator ein Zeiger auf die Zeile. Bei einer gruppierten Tabelle ist der Zeilenlokator der gruppierte Indexschlüssel.

Sie können der Blattebene des nicht gruppierten Index Nonkey-Spalten hinzufügen, um vorhandene Indexschlüsselgrenzwerte zu umgehen und vollständig abgedeckte, indizierte Abfragen auszuführen. Weitere Informationen finden Sie unter Erstellen von Indizes mit eingeschlossenen Spalten.

**XML-Indizes**

Zur Abfrage von XML-Spalten unterstützt SQL Server die XML Query Language (oder kurz: XQuery).   
Diese verwendet verschiedene Methoden, um auf XML-Strukturen zuzugreifen.

query: Abfrage von XML-Inhalten über XQuery

value: Abfrage von Skalarwerten aus einem XML-Dokument

exist: Prüfen, ob ein Element/Knoten in einem XML-Dokument existiert

modify: Inhalte eines XML-Dokuments verändern

nodes: XML-Daten in relationale Daten umformen

**Volltextindizes**

Eine Einschränkung der allgemeinen Indexformen liegt darin, dass sich damit keine Spalten der Datentypen varchar(max), nvarchar(max) (bzw. ebenso deren Vorgänger text und ntext) indizieren lassen. Das ist mit der Verwendung von Volltextindizes möglich. Doch Volltextindizes bieten noch eine Fülle an weiteren Möglichkeiten, die sich auch in Zusammenhang mit anderen alphanumerischen Datentypen einsetzen lassen.

Im Gegensatz zu den klassischen Indexformen werden Volltextindizes in separaten Volltextkatalogen angelegt und verwaltet, die in der Datenbank definiert werden. Bei kleinen Datenbanken reicht es aus, einen einzigen Katalog anzulegen, bei grösseren Datenbanken ist es bewährte Praxis, pro Datenbankschema einen separaten Volltextkatalog zu definieren.

**Räumlicher Index**

Ein räumlicher Index ist ein weiterer spezieller Indextyp, der das Hinzufügen von Indizes zu Spalten ermöglicht, die mit den räumlichen Datentypen Geografie und Geometrie erstellt wurden. Wie bei einem XML-Index erfordern räumliche Indizes auch, dass für die Datenbanktabelle, für die Sie den räumlichen Index erstellen, auch ein gruppierter Primärschlüsselindex definiert ist.

**Gruppierter Columnstore Index**

Der SQL Server in-Memory-Columnstore-Index speichert und verwaltet Daten mithilfe von spaltenbasiertem Datenspeicher und spaltenbasierten Abfragen. Columnstore-Indizes sind optimal für Data Warehousing-Arbeitsauslastungen geeignet, die hauptsächlich Massenladevorgänge und schreibgeschützte Abfragen ausführen. Verwenden Sie den Columnstore-Index, um eine bis zu zehnfache Abfrageleistung gegenüber der herkömmlichen zeilenorientierten Speicherung und eine bis zu siebenfache Datenkomprimierung im Vergleich zur unkomprimierten Datengrösse zu erzielen.

# Lösung 1

CREATE NONCLUSTERED INDEX NonClusteredIndex\_SomeID ON TestData2 (SomeID ASC)

CREATE NONCLUSTERED INDEX NonClusteredIndex\_RowNum ON TestData2 (RowNum ASC)

CREATE NONCLUSTERED INDEX NonClusteredIndex\_SomeCode ON TestData2 (SomeCode ASC)

# Lösung 2

Wie lange dauert ein Full-Table-Scan wenn die Tabelle 12GByte Daten aufweist und sie einen Lesedurchsatz von 30MByte pro Sekunde garantieren können?

12000/30 = 400 [sek]

# Lösung 3

Berechnen sie die Höhe des B-Trees:

d=4 (Ordnung)

n=20 (Anzahl der Blätter) 🡺 2.46 / 1.38

d=3 (Ordnung)

n=100 (Anzahl der Blätter) 🡺 4.89 / 2.37

# Lösung 4

log2d+1(n+1) <= z <= 1+logd+1((n+1)/2)

d=100 (Ordnung)

n=20‘000 (Anzahl der Blätter) 1.87 <= z <= 2.996 🡺d.h. 2 bis 3 Zugriffe

d=3‘000 (Ordnung)

n=1‘000‘000 (Anzahl der Blätter) 1.59 <= z <= 2.64 🡺d.h. 2 bis 3 Zugriffe

# Aufgabe 5a

DBCC CHECKTABLE ('TestData2')

🡺

DBCC-Ergebnis für 'TestData2'.

Es sind 250 Zeilen in 7 Seiten für das TestData2-Objekt vorhanden.

DBCC-Ausführung abgeschlossen.  
Falls DBCC Fehlermeldungen ausgegeben hat, wenden Sie sich an den Systemadministrator.

# DBCC <https://docs.microsoft.com/de-de/sql/t-sql/database-console-commands/dbcc-transact-sql?view=sql-server-2017>

Die Programmiersprache Transact-SQL bietet DBCC-Anweisungen, die als Datenbankkonsolen-Befehle für Microsoft SQL Server fungieren. Diese Anweisungen überprüfen die physische und logische Konsistenz einer Datenbank. Viele DBCC-Anweisungen können erkannte Probleme auch beheben.

DBCC-Anweisungen sind in die folgenden Kategorien eingeteilt.

|  |  |
| --- | --- |
| **Anweisungskategorie** | **Funktion** |
| Verwaltungsanweisungen | Aufgaben zur Verwaltung von Datenbanken, Indizes und Dateigruppen. |
| Verschiedene Anweisungen | Verschiedene Aufgaben wie das Aktivieren von Sperren auf Zeilenebene oder das Entfernen einer Dynamic Link Library (DLL) aus dem Arbeitsspeicher. |
| Statusanweisungen | Statusüberprüfungen |
| Überprüfungsanweisungen | Überprüfungsvorgänge für Datenbanken, Tabellen, Indizes, Kataloge, Dateigruppen, Systemtabellen oder das Reservieren von Datenbankseiten. |

Die DBCC-Anweisungen von SQL Server 2000 besitzen Eingabeparameter und Rückgabewerte. Alle Parameter für DBCC-Anweisungen nehmen sowohl Unicode- als auch DBCS-Literale (Double-Byte Character Set, Doppelbyte-Zeichensatz) an.

##### **Verwenden von DBCC-Resultsetausgaben**

Bei vielen DBCC-Befehlen kann die Ausgabe im Tabellenformat erfolgen (mit der Option WITH TABLERESULTS). Diese Informationen können zur weiteren Verwendung in eine Tabelle geladen werden. Ein entsprechendes Beispielskript wird im Folgenden gezeigt:

-- Create the table to accept the results

CREATE TABLE #tracestatus (

TraceFlag INT,

Status INT

)

-- Execute the command, putting the results in the table

INSERT INTO #tracestatus

EXEC ('DBCC TRACESTATUS (-1) WITH NO\_INFOMSGS')

-- Display the results

SELECT \*

FROM #tracestatus

# Lösung 5b

<https://www.sqlskills.com/blogs/paul/on-index-key-size-index-depth-and-performance/>

SELECT index\_type\_desc,index\_depth,index\_level, fragment\_count, page\_count, record\_count FROM sys.dm\_db\_index\_physical\_stats

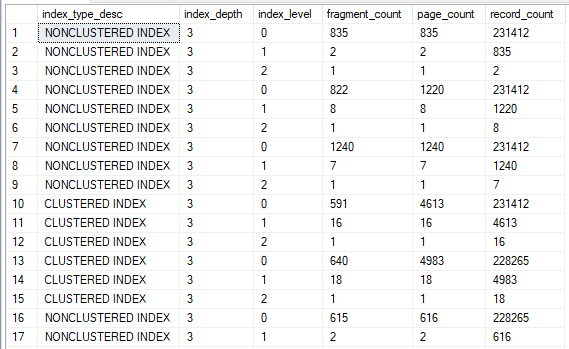
(DB\_ID ('WideWorldImporters'),

OBJECT\_ID ('Orders'),

-1,

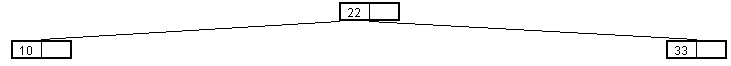
NULL,

'DETAILED') ORDER BY index\_depth DESC;

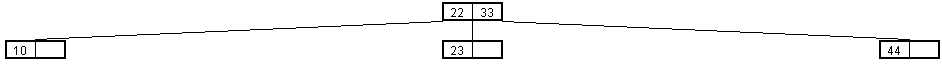


# Lösung 6

1. **22**

****

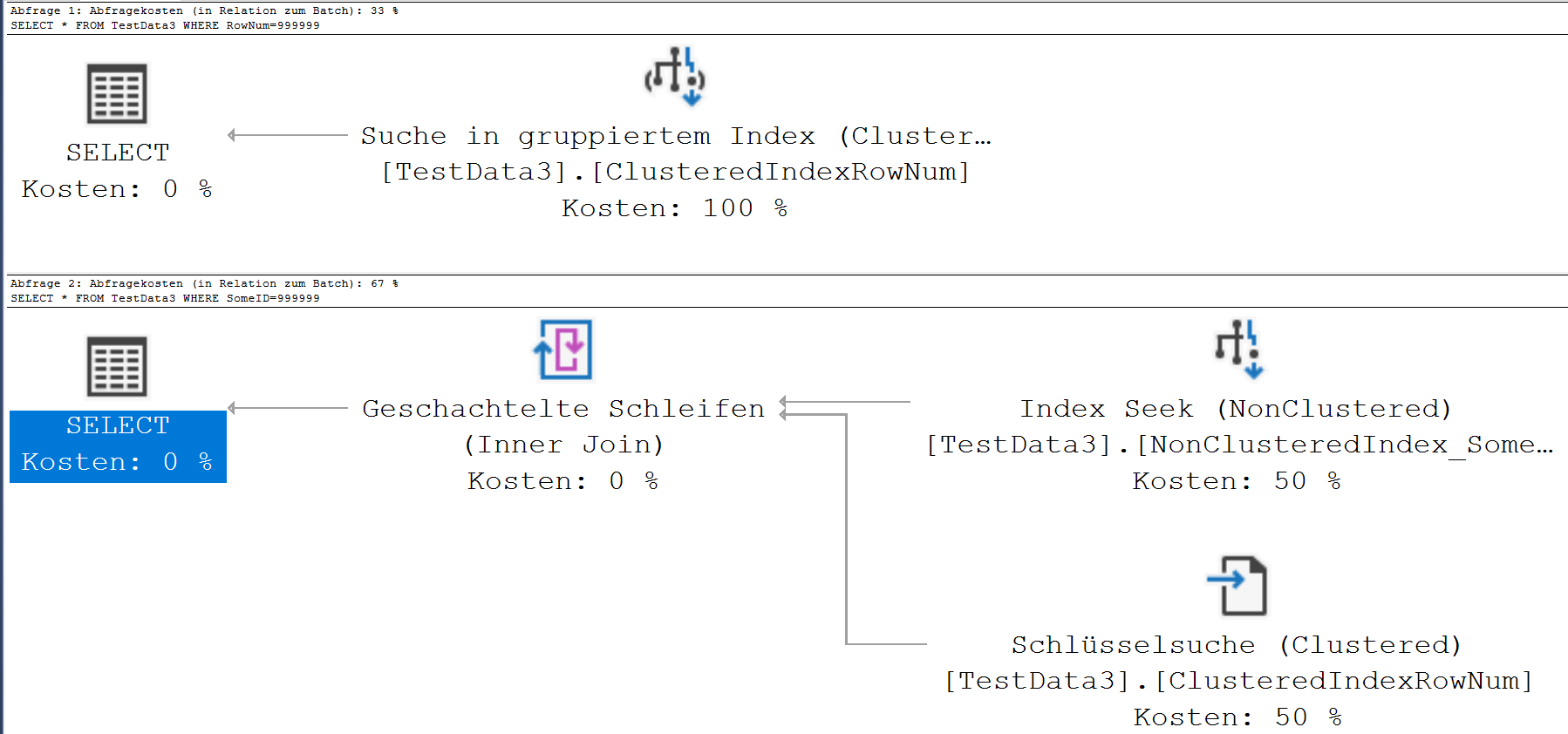
****

****



# Lösung 7

CREATE UNIQUE CLUSTERED INDEX ClusteredIndexRowNum ON TestData2 (RowNum ASC)



# Lösung 8

In SQL Server können Sie Indizes für berechnete Spalten erstellen. Um einen Index für die berechnete Spalte zu erstellen, muss die berechnete Spalte deterministisch und präzise sein und darf keinen Text-, Ntext- oder Bilddatentyp enthalten.

# Lösung 9

<http://www.sqlteam.com>  
<https://docs.microsoft.com/de-de/sql/relational-databases/views/create-indexed-views?view=sql-server-2017>

* **Ja**

There are several requirements that you must take into consideration when using Indexed views.

1. View definition must always return the same results from the same underlying data.
2. Views cannot use non-deterministic functions.
3. The first index on a View must be a clustered, UNIQUE index.
4. If you use Group By, you must include the new COUNT\_BIG(\*) in the select list.
5. View definition cannot contain the following
   1. TOP
   2. Text, ntext or image columns
   3. DISTINCT
   4. MIN, MAX, COUNT, STDEV, VARIANCE, AVG
   5. SUM on a nullable expression
   6. A derived table
   7. Rowset function
   8. Another view
   9. UNION
   10. Subqueries, outer joins, self joins
   11. Full-text predicates like CONTAIN or FREETEXT
   12. COMPUTE or COMPUTE BY
   13. Cannot include order by in view definition

Notice that Indexed Views change the very essence of what a view was before this version of Sql Server. First, the data represented by the view is actually stored in the database. Secondly, the view definition must always return the same results for the same underlying data and all functions and expressions must be deterministic no matter what the current session settings.

To make sure that you can meet this requirement, the following session options must be set when you create an index view, when you modify any of the tables included in an indexed view or when the optimizer decides to use the indexed view as part of a query plan. Session Options that must be on

ANSI\_NULLS

ANSI\_PADDING

ANSI\_WARNINGS

ARITHABORT

CONCAT\_NULL\_YEILDS\_NULL

QUOTED\_IDENTIFIERS

Session options that must be off

NUMERIC\_ROUNDABORT

Functions like GetDate(), rand() are non-deterministic because of different session settings can return different values and the settings for one user may not be the same as for another. The list of deterministic and non-deterministic functions will be included in the final version of Books Online for SQL Server 2000. (Look for topic Deterministic and Nondeterministic Functions in the Books Online)

Besides these restrictions, the underlying tables that make up the view must be protected from schema changes. Part of the syntax of the create view command is the “with SCHEMABINDING” phrase. This is required to create a View Index and this will prevent the dropping or altering of tables participating in an Indexed View. Note that dropping the clustered index of an Indexed View will return it to the standard SQL view as it was as described above in the section Before SQL Server 2000, what was a View?

How do I create an Indexed View?

1. Make sure that session properties are properly set.
2. Create a deterministic view with new SCHEMABINDING syntax.
3. Create unique Clustered Index.
4. Optionally, create additional nonclustered indexes.

Below you will find the code that you can paste into the Sql Server Query Analyzer to test this yourself. This example is based on the Northwind sample database.  
  
-- Use the northwind database

USE NORTHWIND

GO

-- Make sure that all of the session settings are set properly

IF sessionproperty('ARITHABORT') = 0 SET ARITHABORT ON

IF sessionproperty('CONCAT\_NULL\_YIELDS\_NULL') = 0 SET CONCAT\_NULL\_YIELDS\_NULL ON

IF sessionproperty('QUOTED\_IDENTIFIER') = 0 SET QUOTED\_IDENTIFIER ON

IF sessionproperty('ANSI\_NULLS') = 0 SET ANSI\_NULLS ON

IF sessionproperty('ANSI\_PADDING') = 0 SET ANSI\_PADDING ON

IF sessionproperty('ANSI\_WARNINGS') = 0 SET ANSI\_WARNINGS ON

IF sessionproperty('NUMERIC\_ROUNDABORT') = 1 SET NUMERIC\_ROUNDABORT OFF

go

-- Create the view, it must comply with the rules (deterministic)

CREATE VIEW PRODUCTS\_BY\_CUSTOMER WITH SCHEMABINDING AS

select customers.companyname,

products.productname,

sum(odetail.unitprice\*odetail.quantity) as TotalPurchase,

count\_big(\*) as cnt

from "order details" as odetail

inner join orders as omain

on omain.orderid = odetail.orderid

INNER join customers as customers

on customers.customerid = omain.customerid

INNER join products as products

on products.productid = odetail.productid

group by

customers.companyname,

products.productname

go

-- the following statement will cause an error if the view has not been

-- indexed

--EXEC SP\_SPACEUSED 'PRODUCTS\_BY\_CUSTOMER'

--Server: Msg 15235, Level 16, State 1, Procedure sp\_spaceused, Line 91

--Views do not have space allocated.

-- Check to see if the indexes can be created

if ObjectProperty(object\_id('products\_by\_customer'),'IsIndexable') = 1

BEGIN

-- Create a clustered index, it MUST be unique

CREATE UNIQUE CLUSTERED INDEX PRODUCTS\_BY\_CUSTOMER\_UNIQUE ON

PRODUCTS\_BY\_CUSTOMER(COMPANYNAME, PRODUCTNAME)

EXEC SP\_SPACEUSED 'PRODUCTS\_BY\_CUSTOMER'

-- Create NonClustered Indexes

CREATE INDEX PRODUCTS\_BY\_CUSTOMER\_1 ON

PRODUCTS\_BY\_CUSTOMER(COMPANYNAME)

EXEC SP\_SPACEUSED 'PRODUCTS\_BY\_CUSTOMER'

-- Create NonClustered Indexes

CREATE INDEX PRODUCTS\_BY\_CUSTOMER\_2 ON

PRODUCTS\_BY\_CUSTOMER(PRODUCTNAME)

EXEC SP\_SPACEUSED 'PRODUCTS\_BY\_CUSTOMER'

END

Please note the ObjectProperty(object\_id('products\_by\_customer'),'IsIndexable') = 1 in the above code listing. This command will tell you if all of the requirements for indexing a view have been met so that you can programmatically determine if a view can be indexed or not.  
  
Also note that no space is allocated in the database for this view until the clustered index is created. If you try to use the SP\_SPACEUSED stored procedure on a view that is not indexed, you get an error. The results of the SP\_SPACEUSED commands that are sprinkled throughout the above code listing gives the following results on my test machine.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | # of Rows | Data | Index | Total Used |
| After Clustered Index Created | 1685 | 168 kb | 16 kb | 184 kb |
| After NonClustered Index 1 | 1685 | 168 kb | 168 kb | 336 kb |
| After NonClustered Index 2 | 1685 | 168 kb | 320 kb | 488 kb |

#### How do I use the Indexed View?

You can use the view like you would any other view. Also, the SQL Server query optimizer will attempt to use a View Index even if the view is not referenced in the from clause, although you can override this behavior with the Expand Views hint.  
  
From the sample created in the above code example, you could use the view as follows

Example 1: select \* from products\_by\_customer

Example 1 above lets the query optimizer determine whether or not to use the view and its indexes or to use the base tables. Surprising on my test machine, this example uses the base tables, not the Indexed View. The query optimizer is a complex piece of technology but it isn’t always perfect. Based on my testing with this sample data in the Northwind database, I had to use the (noexpand) hint seen in the next example to force the optimizer to use the View Index. The speed of this on my test machine was about 3 times faster with 1685 records. By increasing the number of records in the base tables (orders 3000 records and order details 224,696 records), I found that the query optimizer did use the View Index without specifying the hint and the resulting query speeds where approximately 50 times faster. The # of records in the view, after adding all of these records in the base tables, was 1880 records. I conclude that the query optimizer with a small number of records in the base table (Orders had about 830 and order details had about 2155 records when I started) lean towards using the base tables instead of the View index. More testing would be needed to nail down the break even point but this just points out why the hints are still around and how much faster performance can be when the View Indexes are used.

Example 2: select \* from products\_by\_customer with (noexpand)

Example 2 uses a hint to force the query optimizer to consider only the view and its indexes in the execution plan.

Example 3: select \* from products\_by\_customer option (Expand Views)

Example 3 uses a hint to force the query optimizer to expand all indexed views into their underlying Select statements so the optimizer won’t consider any View Indexes in the execution plan.

#### When would I want to use a View Index?

If you have an application that is a Data-Mart, Data-Mining or decision-support type application, you can improve performance with View Indexes. Applications that do any of the following may benefit as well:

* Joins and aggregations of big tables
* Repeated patterns of queries
* Repeated aggregations on the same or overlapping sets of columns
* Repeated joins of the same tables on the same keys

Also, situations where you might consider de-normalizing a set of tables by storing aggregate information in the parent table may be good situations to consider creating an aggregate view on the child table and creating the appropriate View Index. In essence, the Indexed View replaces your de-normalized fields and all the work of keeping this de-normalized aggregate field up to date is taken care of by the database engine.

#### When would I NOT want to use a View Index?

You obviously cannot use a View Index if you need to include syntax in the view definition that is not allowed. It seems to me that Top, Min, Max, Count, using another view, union, subqueiries and outer joins are serious restrictions that would disqualify a large number of views that I might want to optimize using the View Index.

Storage may also be a major consideration as the data in the view is physically and permanently stored not only in its base table, but in the clustered index of the View Index. Effectively, the data is stored twice, once in its base table and once in the clustered index of the Indexed View.

Also, On-Line Transaction Processing systems (OLTP) will actually suffer performance loss if you try to use View Indexes. Databases with frequent inserts, updates, deletes suffer as these commands must not only process the table, but any associated Indexed Views that the base table is participating in. Also, views that are simply subsets of rows or columns with no aggregation or computation provide no benefit over a standard SQL view or a T-SQL command directly against the base tables. This additional overhead to update the data in the clustered index of the view I believe is the reason that the clustered index must be unique for an Indexed view. It uses this unique value to quickly update the appropriate record in the Indexed View. Without a unique index, the processing for updating the Indexed View records could be unacceptably long.

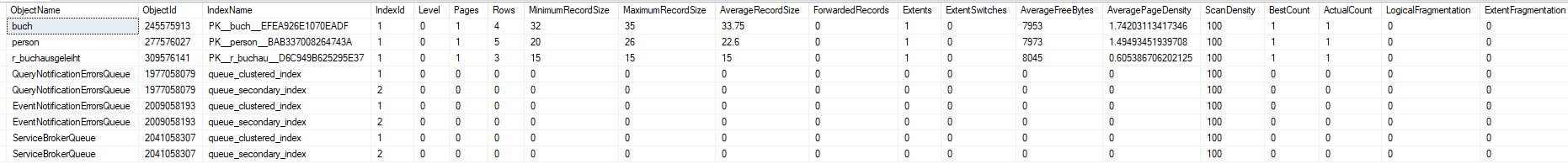
#### What are the performance benefits? As I indicated earlier, I experienced query times 3 times quicker using the Indexed Views over the same query not using the Indexed Views on the sample data in the NorthWind database. With a much bigger data set and with the same database objects defined, I got query times as much as 50 times faster. Microsoft has reported performance improvements of 10 to 100 times with applications that access indexed views instead of base tables. I also experimented with using other queries that did not directly reference the Indexed View and got similar performance gains when the optimizer selected the Indexed View over other possible execution plans.

#### Summary As you can see, even with its restrictions, the View Index is a powerful new tool in the SQL Server Developer’s toolbox. Because the optimizer can use a View Index, you won’t even have to change your existing T-SQL to take advantage of the performance benefits of the View Index. So take into consideration the information above when evaluating whether a View Index is right for your application.

# Lösung 10

Ermitteln sie die Fragmentierung aller Indexe einer Datenbank!

SELECT \* FROM sys.dm\_db\_index\_physical\_stats (NULL, NULL, NULL, NULL,'DETAILED');



# Lösung 11

DBCC DBREINDEX (Klasse, '', 70)

# Lösung 12

Datenbank: ‘StackOverflow2013’

SELECT Posts.Id, Comments.Text

FROM Comments

RIGHT OUTER JOIN Posts

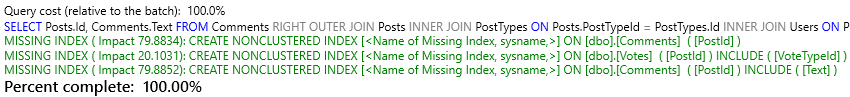
INNER JOIN PostTypes ON Posts.PostTypeId = PostTypes.Id

INNER JOIN Users ON Posts.OwnerUserId = Users.Id ON Comments.PostId = Posts.Id

LEFT OUTER JOIN VoteTypes

INNER JOIN Votes ON VoteTypes.Id = Votes.VoteTypeId ON Posts.Id = Votes.PostId

WHERE Posts.Id=1000000



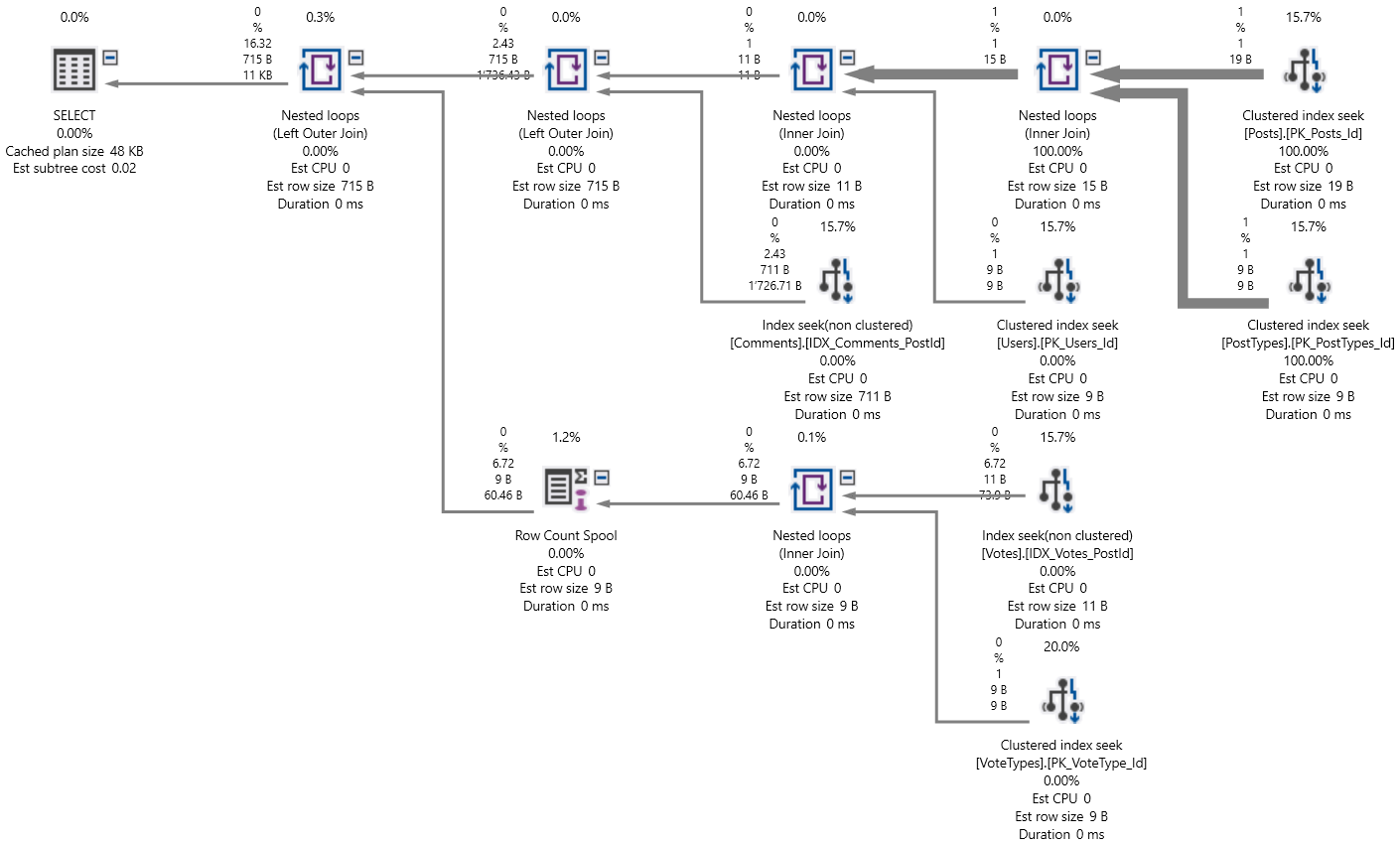
Die zwei zu erstellenden Indexe:

CREATE NONCLUSTERED INDEX IDX\_Comments\_PostId ON Comments

(PostId) INCLUDE (Text)

CREATE NONCLUSTERED INDEX IDX\_Votes\_PostId ON Votes

(PostId) INCLUDE (VoteTypeId)



  
mit AZUR Data Studio ermittelt

# Index Scan

# Da ein Scan jede Zeile in der Tabelle berührt, unabhängig davon, ob er qualifiziert ist oder nicht, sind die Kosten proportional zur Gesamtzahl der Zeilen in der Tabelle. Ein Scan ist daher eine effiziente Strategie, wenn die Tabelle klein ist oder wenn sich die meisten Zeilen für das Prädikat eignen.

# Index-Scan ist nichts anderes als das Scannen der Datenseiten von der Ersten bis zur letzten Seite. Wenn für eine Tabelle ein Index vorhanden ist und die Abfrage eine grössere Datenmenge berührt, bedeutet dies, dass die Abfrage mehr als 50 Prozent oder 90 Prozent der Daten abruft und der Optimierer dann alle Datenseiten scannt Rufen Sie die Datenzeilen ab. Wenn kein Index vorhanden ist, wird im Ausführungsplan möglicherweise ein Table-Scan (Index-Scan) angezeigt.

# Index Seek

# Indexsuchen werden im Allgemeinen für die hochselektiven Abfragen bevorzugt. Das bedeutet, dass die Abfrage nur eine geringere Anzahl von Zeilen anfordert oder nur die anderen 10 Prozent der Zeilen der Tabelle abruft.

# Im Allgemeinen versucht das Abfrageoptimierungsprogramm, einen Index Seek zu verwenden. Das bedeutet, dass das Optimierungsprogramm einen nützlichen Index zum Abrufen des Recordset gefunden hat. Wenn dies nicht möglich ist, weil entweder kein Index oder keine nützlichen Indizes für die Tabelle vorhanden sind, muss der SQL Server alle Datensätze überprüfen, die die Abfragebedingung erfüllen.

# Index-Spool

# Die Index-Spool-Funktion wird verwendet, um die Leseleistung einer Tabelle zu verbessern, die nicht indiziert ist. Wie bei anderen Spool-Operatortypen kann sie auch "Lazy" oder "Eager" verwendet werden. Wenn SQL Server eine Tabelle lesen muss, die nicht indiziert ist, kann SQL Server mit dem Spool einen "temporären Index" erstellen, der zu einer erheblichen Leistungsverbesserung Ihrer Abfragen führen kann.

# Eager Spool

# Die Rolle der Eager Spool besteht darin, alle von einem anderen Operator empfangenen Zeilen abzufangen und diese Zeilen in TempDB zu speichern. Das Wort "eifrig" bedeutet, dass der Operator ALLE Zeilen des vorherigen Operators gleichzeitig liest. Mit anderen Worten, es wird die gesamte Eingabe benötigt, wobei jede empfangene Zeile gespeichert wird.

# Lazy Spool

# Die Lazy Spool ist dem Eager Spool sehr ähnlich: Der Unterschied besteht lediglich darin, dass Lazy Spool Daten nur dann liest, wenn einzelne Zeilen erforderlich sind. Es erstellt eine temporäre Tabelle und erstellt diese Tabelle "faul"; Das heisst, es liest und speichert die Zeilen nur in einer temporären Tabelle, wenn der übergeordnete Operator tatsächlich nach einer Zeile fragt, im Gegensatz zu Eager Spool, der alle Zeilen gleichzeitig liest. Um auf etwas zu verweisen, auf das ich in der Eager Spool-Erläuterung eingegangen bin, ist die Lazy Spool ein nicht blockierender Operator, während die Eager Spool ein Blocking-Operator ist.

# Row count spool

# Von allen Spool-Operatoren ist dies der einfachste. Der Zeilenanzahl-Spool-Operator scannt nur eine Eingabe, zählt die Anzahl der vorhandenen Zeilen und gibt die Anzahl der Zeilen ohne die darin enthaltenen Daten zurück. Dieser Operator wird verwendet, wenn es wichtig ist, das Vorhandensein von Zeilen zu überprüfen, jedoch nicht, welche Daten sie enthalten. Wenn zum Beispiel ein Nested-Loops-Operator eine linke Anti-Semi-Join-Operation ausführt und das Join-Prädikat auf die innere Eingabe angewendet wird, kann eine Zeilenanzahl-Spool am Anfang dieser Eingabe platziert werden, um die Anzahl der Zeilen zu zwischenspeichern, die das Argument erfüllen. Dann kann der Operator für verschachtelte Schleifen diese Zeilenanzahlinformationen einfach verwenden (da die tatsächlichen Daten der inneren Eingabe nicht benötigt werden), um zu bestimmen, ob die äussere Zeile zurückgegeben werden soll oder nicht (bzw. wie viele Zeilen zurückgegeben werden sollen).

# Lösung 13

 4900/8300 Zeilen SQL-Code****

SELECT 'DBCC DBREINDEX (' + OBJECT\_NAME(id) + ', ' + name + ')'

FROM sysindexes

WHERE OBJECTPROPERTY(id, 'IsMSShipped') = 0

AND indid NOT IN (0, 255)

AND INDEXPROPERTY(id, name, 'IsStatistics') = 0

AND INDEXPROPERTY(id, name, 'IsHypothetical') = 0

AND INDEXPROPERTY(id, name, 'IsAutoStatistics') = 0

AND INDEXPROPERTY(id, name, 'IsFulltextKey') = 0

**🡺**

DBCC DBREINDEX (Aufgabe, aaaaaAufgabe\_PK)

DBCC DBREINDEX (Beruf, aaaaaBeruf\_PK)

DBCC DBREINDEX (Firma, aaaaaFirma\_PK)

DBCC DBREINDEX (Gewicht, AnstellungAnstellung)

DBCC DBREINDEX (Gewicht, aaaaaGewicht\_PK)

DBCC DBREINDEX (Hilfe, Reg)

DBCC DBREINDEX (Hilfe, aaaaaHilfe\_PK)

DBCC DBREINDEX (Einstellung, aaaaaEinstellung\_PK)

DBCC DBREINDEX (idEingeloggtePerson, BerufNr)

DBCC DBREINDEX (idEingeloggtePerson, idEingeloggtePerson)

DBCC DBREINDEX (idEingeloggtePerson, idLektion)

DBCC DBREINDEX (idEingeloggtePerson, LektionidEingeloggtePerson)

DBCC DBREINDEX (idEingeloggtePerson, PersonidEingeloggtePerson)

DBCC DBREINDEX (idEingeloggtePerson, aaaaaidEingeloggtePerson\_PK)

OBJECTPROPERTY(id, 'IsMSShipped') = 0:  Leaves out indexes on system tables (DBCC INDEXDEFRAG is not supported on system tables)

indid NOT IN (0, 255): Leaves out tables without an index (indid = 0) and indexes for text, ntext and image columns (indid = 255)

INDEXPROPERTY(id, name, 'IsStatistics') = 0: leaves out entries in sysindexes that are actually statistics and not indexes. Statistics can be updated with UPDATE STATISTICS.

INDEXPROPERTY(id, name, 'IsHypothetical') = 0: leaves out (temporary) indexes created by the Index Tuning Wizard to aid in performance tuning

INDEXPROPERTY(id, name, 'IsAutoStatistics') = 0: not necessary in this query. IsStatistics covers these as well.

INDEXPROPERTY(id, name, 'IsFulltextKey') : leaves out full text indexes, they are handled indepently of normal SQL Server indexes

# Aufgabe 14 (Reserve Aufgabe)

Erstellen sie einen Spatial-Index und testen sie diesen an einem Beispiel

# 

# Aufgabe 15 (Reserve Aufgabe) https://www.red-gate.com/simple-talk/sql/database-administration/getting-started-with-xml-indexes/

# <https://www.mssqltips.com/sqlservertutorial/9137/sql-server-xml-indexes/>

# <https://docs.microsoft.com/en-us/sql/relational-databases/xml/xml-indexes-sql-server?view=sql-server-2017>

<https://www.red-gate.com/simple-talk/sql/database-administration/getting-started-with-xml-indexes/>

Erstellen sie eine XML-Index und testen sie diesen an einem Beispiel.

CREATE TABLE T\_Person(

idPerson int NOT NULL,

Demographics xml NULL,

CONSTRAINT PK\_T\_Person PRIMARY KEY CLUSTERED

(idPerson ))

)

CREATE PRIMARY XML INDEX PXML\_Person\_Demographics ON T\_Person(Demographics);

CREATE XML INDEX XMLPATH\_Person\_Demographics ON T\_Person (Demographics)

USING XML INDEX PXML\_Person\_Demographics FOR PATH;

CREATE XML INDEX XMLPROPERTY\_Person\_Demographics ON T\_Person (Demographics)

USING XML INDEX PXML\_Person\_Demographics FOR PROPERTY;

CREATE XML INDEX XMLVALUE\_Person\_Demographics ON T\_Person (Demographics)

USING XML INDEX PXML\_Person\_Demographics FOR VALUE;

Testen:

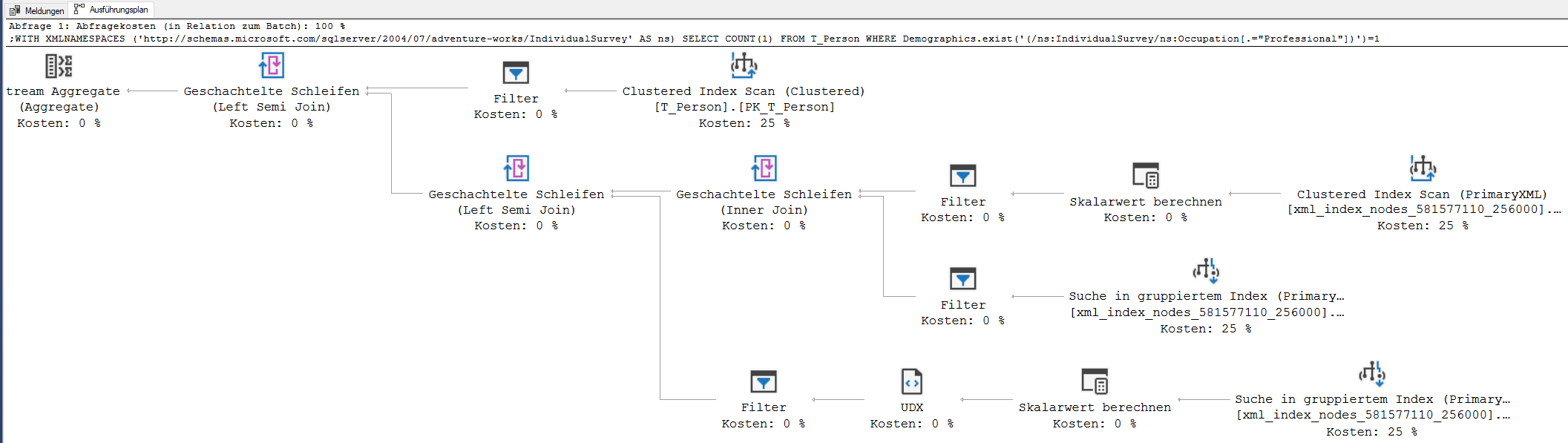
;WITH XMLNAMESPACES

('http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey' AS ns)

SELECT COUNT(1)

FROM T\_Person

WHERE Demographics.exist('(/ns:IndividualSurvey/ns:Occupation[.="Professional"])')=1;

Beweis:  




**Variante 2**

<https://www.red-gate.com/simple-talk/sql/database-administration/getting-started-with-xml-indexes/>

USE AdventureWorks2017

GO

CREATE TABLE EMP\_XML

(

ID INT PRIMARY KEY,

EMP\_DETAILS XML

)

GO

--run script to convert employee records to xml data

DECLARE @counter INT = 1

DECLARE @row VARCHAR(MAX)

WHILE @counter <= 19614

BEGIN

SET @row = (

SELECT

AddressLine1 AS 'AddressLine1',

AddressLine2 AS 'AddressLine2',

City AS 'City',

PostalCode AS 'PostalCode'

FROM [Person].[Address]

WHERE [AddressID]= @counter

FOR XML PATH ('Employee')

)

INSERT EMP\_XML VALUES(@counter,@row)

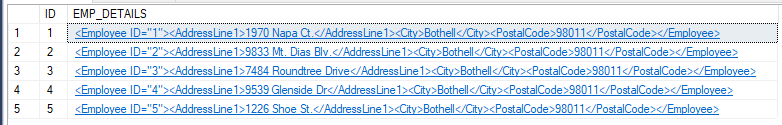
SET @counter += 1

END

--get sampling of new xml records

SELECT TOP 5 \* FROM EMP\_XML

GO



<Employee ID="1">

<AddressLine1>1970 Napa Ct.</AddressLine1>

<City>Bothell</City>

<PostalCode>98011</PostalCode>

</Employee>

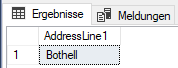
--find

SELECT EMP\_DETAILS.value('(/Employee/City)[1]','varchar(50)') AS AddressLine1

FROM EMP\_XML

WHERE EMP\_DETAILS.exist('/Employee[@ID=12]') = 1

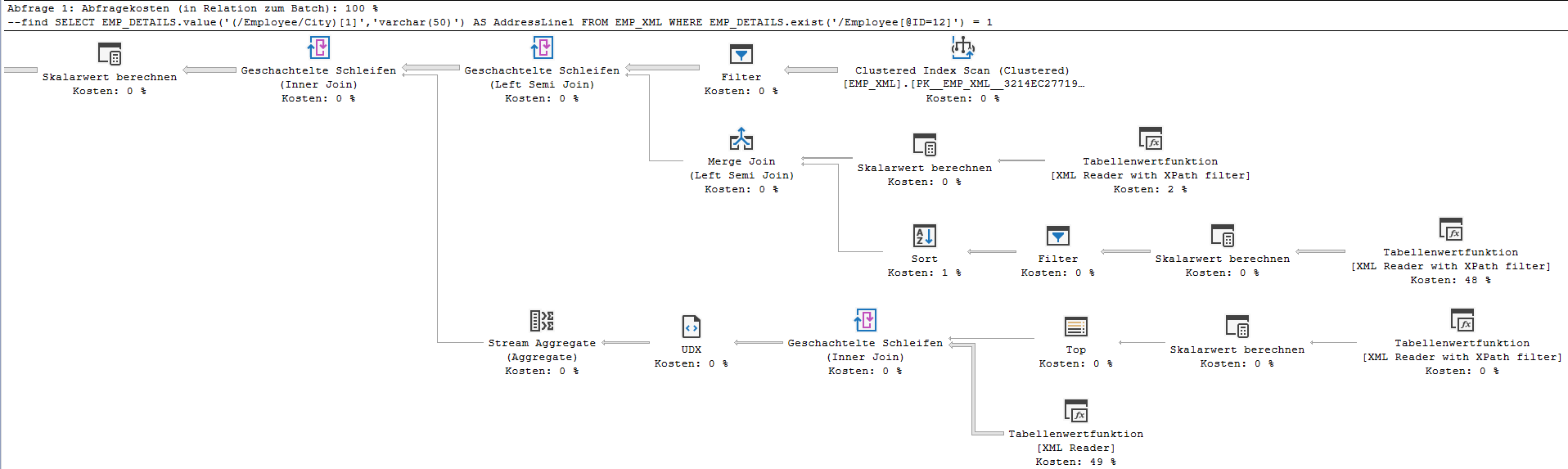
GO

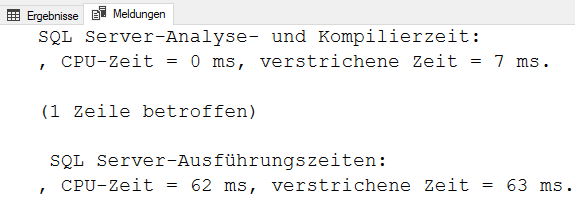


--turn on statistics time

SET STATISTICS TIME ON

GO





--create primary xml index DROP INDEX [IX\_EMP\_DETAILSx] ON EMP\_XML

CREATE PRIMARY XML INDEX [IX\_EMP\_DETAILSx]

ON EMP\_XML(EMP\_DETAILS)

GO

