smartlearn
virtual learning. real success.

MODUL 153
DATENMODELLE ENTWICKELN

Reto Glarner

Vertiefung physisches Datenmodell

- Sie bilden das logische Datenmodell korrekt als physisches Datenmodell ab (HZ 5)
- Sie kennen die Möglichkeiten, wie die Zugriffszeiten auf eine Datenbank durch die Definition von Indizes verbessert werden können (Hanok 5.1)
- Sie kennen die Möglichkeiten, wie die Zugriffszeiten auf eine Datenbank durch Denormalisierung (kontrollierte Redundanz) von Tabellen verbessert werden können (Hanok 5.1)
- Sie kennen die Datentypen und können diese im physischen Datenmodell korrekt verwenden (Teil von HZ 3)
- Sie können einen Prototypen mit Daten befüllen um die Anforderungen zu überprüfen (Hanok 4.2)

Was gehört zum physischen Datenmodell?

Das physische Datenmodell entspricht der Organisation der Speicherung der Tabellen und deren Beziehungen (logisches Datenmodell) auf einem Speichermedium (Laufwerk, Arbeitsspeicher[1]). Dieses Datenmodell definiert in welcher Form die Daten(sätze) abgespeichert werden.

Nicht nur die eigentliche Speicherung gehört zum physischen Datenmodell, sondern auch der Prozess wie diese auf dem RDBM definiert wird. Dieser Prozess ist stark von der eingesetzten Technologie bzw. des Produkts abhängig. Der verwendete SQL Server von Microsoft wird über die SQL Sprache (T-SQL, Transact SQL) mittels DDL (Data Definition Language) gesteuert.

→ Scripts zur Erstellung der Datenbank, Tabellen, Attributen, Schlüssel

Bei Abfragen (Read) müssen die Daten durchsucht, gefiltert, sortiert und gruppiert werden, dies nimmt je nach Komplexität der Abfrage und Menge der vorhandenen Daten sehr viel Zeit in Anspruch. Damit der Benutzer nicht lange Wartezeiten in Kauf nehmen muss, sollten die Daten in ihrer Speicherform optimiert abgelegt werden auf dem Laufwerk. Dies geschieht mittels zusätzlichen „Nachschlagekatalogen“ genannt Indizes. (Einzahl: Index) Jeder Index verlangsamt allerdings die Modifikation (Create, Update, Delete), da diese auch nachgeführt werden müssen. Es liegt also am Datenbankarchitekt die Indizes optimal zu setzen:

→ Verwendung von Indizes zur Erreichung der Optimalen Performance. Dies wird mit einem Prototypen der Datenbank erreicht auf welchem Zugriffe simuliert werden. Die Aufzeichnung gibt Aufschluss darüber wie die physische Speicherung optimiert werden könnte.

[1] IMDB, Beispiele: Oracle Exalytics, SAP HANA usw.

Übergang vom logischen ins physische Datenmodell mittels Scripts

Hinweis: Die folgenden Manipulationen können alle entweder mittels Script oder mittels GUI Tools (Bsp.: SQL Server Management Studio) durchgeführt werden. Der Vorteil der Scripts liegt darin, dass diese (automatisch) wiederholt werden können. Die Manipulationen mittels GUI sind „verloren“. Aus diesem Grund werden hier nur die Scripts behandelt. Als Beispiel wird die Datenbank von „Tante Emma“ vom Arbeitsblatt 2 verwendet, wobei nun zusätzliche Optionen besprochen werden.

Die Realisierung des physischen Datenmodells beginnt mit dem Anlegen der Datenbank

```
USE master
CREATE DATABASE TanteEmma
GO
USE TanteEmma
```

Das sieht einfach aus – zu einfach leider.

USE wechselt den Ausführungskontext auf eine bestimmte Datenbank. In diesem Fall wird zur Master DB gewechselt, in welcher alle Metadaten abgespeichert sind. (Datenbankendefinitionen, Tabellendefinitionen, Attributdefinitionen, Benutzer usw.)

GO wird verwendet, um die Ausführung zu erzwingen. Der SQL Server kann selbständig mehrere Befehle zu einem Batch zusammenfassen, mittels GO wird sichergestellt, dass alle vorherigen Befehle ausgeführt wurden bevor der nächste Befehl ausgeführt wird.

CREATE DATABASE legt eine neue Datenbank an. Der CREATE Befehl kann mit einer riesigen Menge an Optionen verwendet werden, dass es sich lohnen würde ihm ein ganzes Arbeitsblatt zu widmen.

Erklärung aller Optionen von CREATE DATABASE

https://msdn.microsoft.com/en-us/library/ms176061.aspx

oder 1_CREATE DATABASE (SQL Server Transact-SQL).pdf

Die wichtigsten Optionen, welche Einfluss auf die Performance haben:

ON gibt an, wo die Daten physisch gespeichert werden. Dabei kommen Dateigruppen zum Einsatz, welche wiederum auf Datenträger verteilt werden können. Prinzipiell gilt, dass Daten auf dem gleichen Datenträger sequentiell gelesen werden können. Dadurch wird die Performance eingeschränkt da keine Parallelität genutzt werden kann. Eine Datenbank legt Daten in mindestens zwei Dateien ab.

Die MDB Datei enthält die eigentlichen Datensätze der Tabellen.

Die LDB Datei enthält (je nach LOG-Level) alle Zugriffe auf die Daten. Sie kommt bei Transaktionen zum Einsatz und garantiert zudem die Persistenz. D.h. bei einem Ausfall eines Datenträgers oder des Systems sind in der Logdatei die letzten Zugriffe gespeichert, welche noch nicht in der Datensicherung vorhanden sind.

Wird die Option ON nicht verwendet, legt der SQL Server die beiden Dateien am Standardspeicherort ab, welcher in den Eigenschaften des Servers definiert sind.

Lesen Sie die Tipps zur Verbesserung der Performance mittels Speicherstrategie:

https://technet.microsoft.com/de-de/library/ms187087(v=sql.105).aspx

oder 2_Verwenden von Dateien und Dateigruppen.pdf

Innerhalb der ON Option wird auch die Grösse der Datenbank- und Logdatei angegeben:

- Startgrösse
- Wachstumsstrategie (in Prozent oder absoluter Wert)
- Maximale Grösse

Wenn Sie eine grössere Datenbank (z.B. durch Import) verwenden, setzten Sie die Anfangsgrösse korrekt, da sonst laufend die Grösse angepasst werden muss welches die Performance natürlich verringert.

Kontrollieren Sie von Zeit zu Zeit die Grösse der Datenbankdateien von produktiven Datenbanken. Falls viele Daten gelöscht wurden, kann die Datei manuell verkleinert werden (shrinking) Nach Datensicherungen kann die Logdatei praktisch auf 0 verkleinert werden, da alle Daten in der Sicherung und im Original vorhanden sind.

Anlegen von Tabellen

```
USE Laden
GO


CREATE TABLE ARTIKEL (
      ArtikelID int IDENTITY(1,1) NOT NULL,
      SteuerID smallint NOT NULL,
      Name varchar (50) NOT NULL,
      Preis decimal (10, 2) NOT NULL,
 CONSTRAINT PK_ArtikelID PRIMARY KEY (
      ArtikelID)
GO
```

Schlüsselbegriffe:

- Tabellennamen
- Attributnamen
- Datentypen
- NULL Werte
- Primärschlüssel (Varianten möglich)
- Automatische Nummerierung
- Fremdschlüsselbeziehungen
- Referentielle Integritätsbedingungen

Verbesserung der Performance mittels Index

Was verstehen wir unter dem Begriff „Performance“ bei einer Datenbank? Unabhängig von der eingesetzten Software können folgende Kriterien herangezogen werden:

1. Reaktionszeit
   Zeitliche Differenz vom Eingang der Anfrage bis zum Erhalte der Antwort. Die Reaktionszeit wird durch datenbankspezifische Faktoren (physische Organisation, Datenmenge, Komplexität der Anfrage, Caching usw.) sowie externe Faktoren (Arbeitsspeicher, eingesetzte Hardware, Systemlast, Priorisierung der Prozesse usw.) beeinflusst.
2. Speicherplatznutzung
   Die Menge des verwendeten (nichtflüchtigen) Speichers zur Ablage der Daten- und Logdateien. Dazu gehören auch die Indizes und Metadaten.
3. Transaktionsdurchsatz
   Wie viele Transaktionen können pro Zeiteinheit (Sekunde, Minute, ..) verarbeitet werden? Dieser Wert gibt die Obergrenze der Auslastung im Dauerbetrieb an.

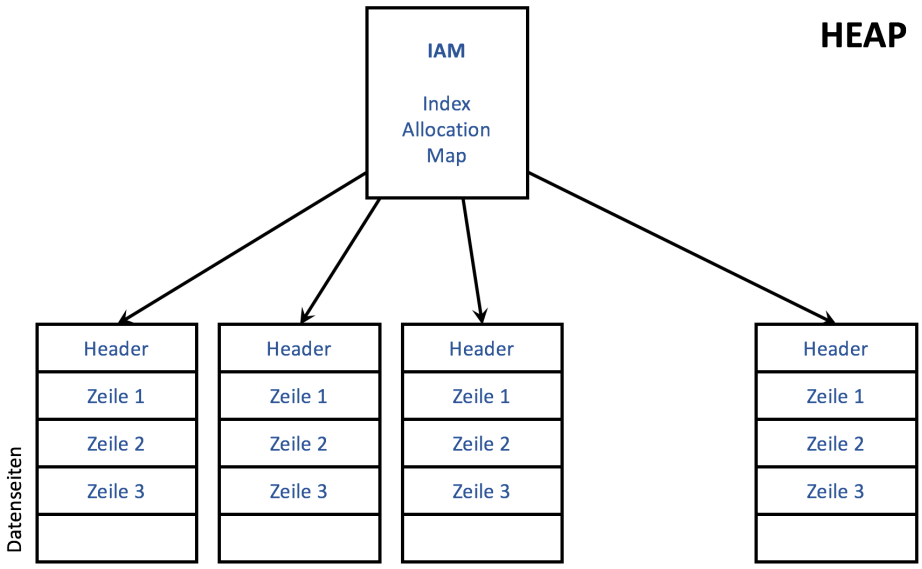
Speicherung von Daten in einem Heap (Kein Index)

Wie werden die Daten einer Tabelle eigentlich auf dem Datenträger gespeichert? Als Textdatei? Sind sie komprimiert? Sind sie sequentiell oder parallel lesbar?

Die einfachste Art der Speicherung ist der Heap, dieser entspricht einer fortlaufenden Nummerierung von Speicherorten (Seiten) innerhalb einer Datei. Innerhalb dieser Seiten sind die Daten in Zeilen abgelegt. (SQL Server verwendet Seiten von maximal 8KB Grösse)

Die Organisation der Seiten ist nicht geordnet, weil auch Seiten gelöscht werden können ergibt dies Löcher. Für die Daten bedeutet dies, dass sie irgendwo liegen können. Es gibt keine Logik, es muss immer in der Seite mit dem Verzeichnis (IAM = Index Allocation Map) nachgeschlagen werden. Jede Seite wird direkt adressiert (Pfeile).

Wir stellen uns eine Ablage mit folgender interner Struktur vor:



Vorteile:

› Schnelle INSERT (einfach hinten anhängen)
› Schnelle DELETE (1. Schritt Referenz löschen in IAM, 2. Schritt Platz in Datenseiten freigeben)

Nachteile:

› Langsames SELECT mit WHERE / ORDER / JOIN / GROUP (es müssen alle Datensätze durchsucht werden)

Syntax in SQL: (Die Beispieltabellen werden in der Aufgabe 2 verwendet)

```
CREATE TABLE KUNDE1(
ID int NOT NULL,
Vorname varchar(50) NOT NULL,
Nachname varchar(50) NOT NULL
)
```

Speicherung von Daten in einem Heap (mit Index)

Um die Nachteile des Heap (langsame SELECT) auszugleichen, kann zusätzlich zu den Datenseiten ein Index angelegt werden. Wie funktioniert dieser Index?

Ein Index ist in einer Baumstruktur abgelegt. Der SQL Server verwendet einen „Balanced Tree“ (B-Tree), welcher an allen Verästelungen links und rechts gleich viele Blätter trägt. Ein Index auf Zeichenkette ist unten abgebildet. Im obersten (root) Element wird nicht fix das Alphabet halbiert, sondern es kommt auf die Anzahl der Einträge pro jeweiligem Anfangsbuchstaben an!

Es gibt B-Trees, welche in den Blättern mehrere Elemente zulassen (siehe Bild unten A-D) und es gibt B-Trees, welche auch in der untersten Hierarchie (Blätter) nur maximal 2 Elemente zulassen. Die letzteren sind einfacher zu berechnen, haben aber die grössere Anzahl Hierarchiestufen.

Ersparnis:

Je nach Aufbau des B-Trees lässt sich die Suchzeit enorm verkürzen. Stellen wir uns einen B-Tree vor mit Zahlen zwischen 1-1024, welche keine Duplikate enthält (wie ein Primärschlüssel).

Root Element (1. Hierarchie) teilt 512 Elemente links und rechts auf.

... 2. Hierarchie 256 …

3. Hierarchie 128 …

… …

9 Hierarchie 2 Elemente …

10 Hierarchie 1 Element

Das bedeutet, dass in maximal 10 Schritten jedes Element aufgefunden werden kann!

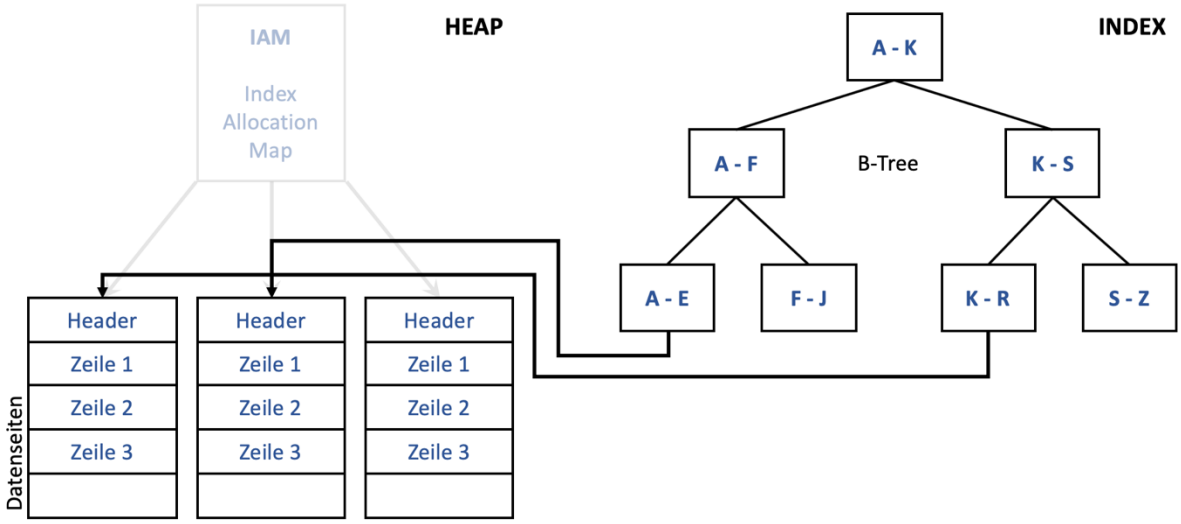
Wären die Daten in einem Heap (ohne Index) gespeichert worden, wären folgende Suchen notwendig:

› Wenn Duplikate vorkommen ist eine **komplette Suche** durch alle 1024 Elemente („Full-Row-Scan“) nötig
› Wenn keine Duplikate vorkommen wird im Durchschnitt nach **der Hälfte** der Elemente das gesuchte gefunden. Es werden 512 Elemente durchsucht.

Mathematisch lässt sich die Ersparnis durch die folgende Formel darstellen.

$$maximale\ Schritte = \frac{\log_{10}(Anzahl\ Elemente)}{\log_{10}(Anzahl\ Verzweigungen)} \Rightarrow \frac{log_{10}(1024)}{log_{10}(2)} = 10$$

Vertiefung Balanced Tree → Aufgabe 1

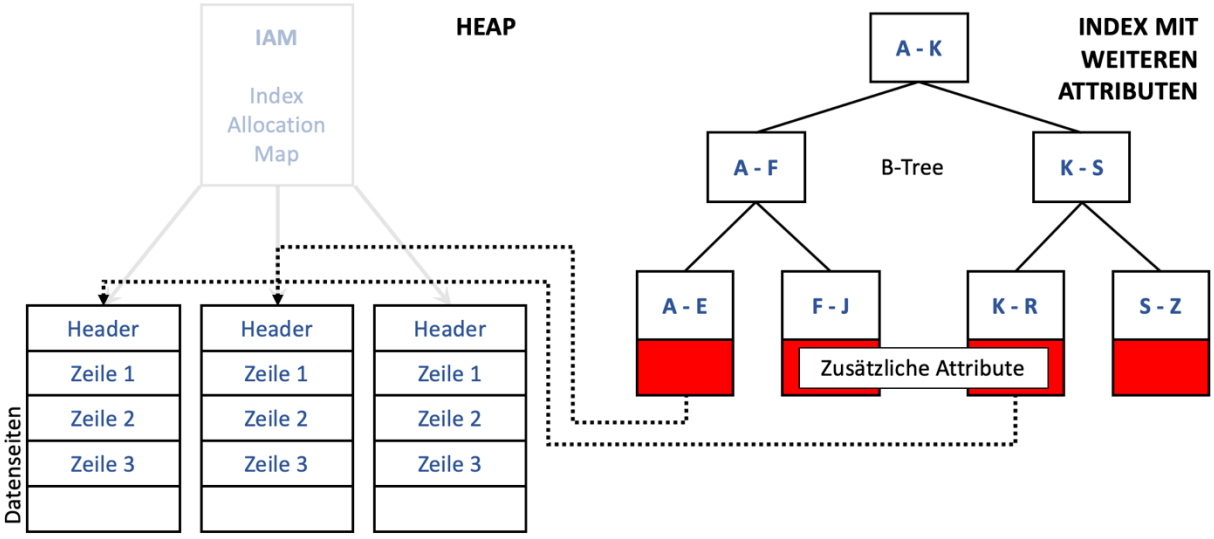


Der SQL Server zeigt in den Blättern des B-Trees direkt auf die Speicherseiten des Heap’s, dabei wird Zeit gespart, da die IAM nicht auch noch durchsucht werden muss. (Nachteil: Der Index muss zusätzlich zur IAM aktualisiert werden)

```
CREATE TABLE KUNDE2(
ID int NOT NULL,
Vorname varchar(50) NOT NULL,
Nachname varchar(50) NOT NULL
)
GO
CREATE NONCLUSTERED INDEX idx_nc_Nachname ON KUNDE2
(Nachname)
GO
```

Speicherung von Daten in einem Heap (mit Index mit Zusatzattributen)

In manchen Fällen wird immer nach den gleichen Attributen gesucht und nicht nach dem ganzen Datensatz. Wenn dem so ist, kann dieses gesuchte Attribut direkt im Index abgelegt werden (rot) und die Suche im Heap ist nicht mehr notwendig:



Beispiel: Die Telefonanlage sucht bei einem Anruf nach der Telefonnummer (Index) und zeigt immer den Namen des Anrufers (Zusatzattribut) an.

Vorteile:

› Immer wieder abgefragte Attribute können ohne Zusatzschritte **im** Index abgerufen werden
› Alle anderen Attribute können über Zeiger auf den Heap abgerufen werden

Nachteile:

› Bei INSERT UPDATE DELETE muss der Index nachgeführt werden
› Erhöhter Speicherbedarf

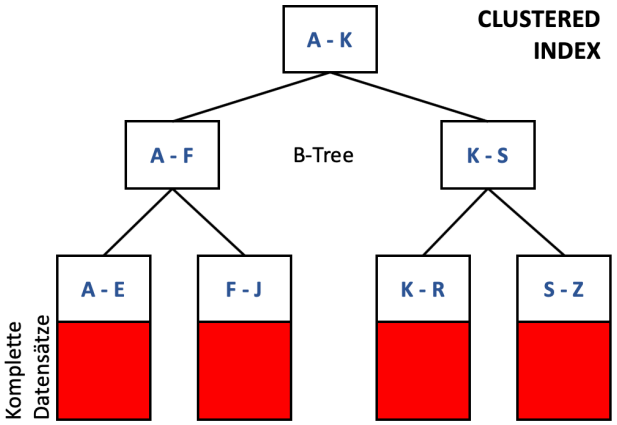
● ● ● Wenn Daten mehrfach abgelegt werden nennt sich das Redundanz. Hier handelt es sich um **gewollte Redundanz** (Denormalisierung). Da diese komplett durch das RDBMS verwaltet wird, ergibt sich für den Benutzer keinen Zusatzaufwand.

```
CREATE TABLE KUNDE2(
ID int NOT NULL,
Vorname varchar(50) NOT NULL,
Nachname varchar(50) NOT NULL
)
GO
CREATE NONCLUSTERED INDEX idx_nc_Nachname ON KUNDE2
(Nachname)
INCLUDE (Vorname);
GO
```

Speicherung von Daten in einem clustered Index (Primärspeicher)

Ein clustered Index ist nichts anderes als ein Index, bei welchem alle Attribute einer Tabelle in den Blättern abgespeichert werden. (wie ein include *). Da der clustered Index alle Daten im Original vorliegen – es gibt kein Heap mehr daneben – nennt sich dieser auch Primärspeicher.

In einer Tabelle kann es höchstens einen clustered Index geben, da dieser die Rolle des Heap übernimmt:



Vorteile:

› Daten werden schnell gefunden wie bei einem „normalen“ Index
› Keine weiteren Sprünge auf Heap notwendig
› Keine Redundanz

Nachteile:

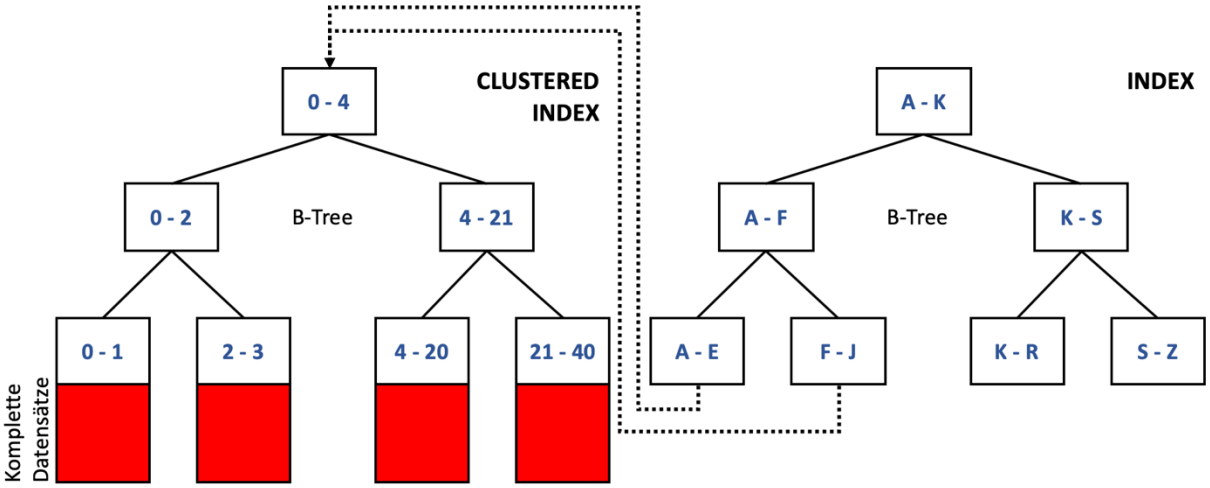
› Nur für ein Attribut (oder zusammengesetzter Schlüssel) möglich

Syntax in SQL:

```
CREATE TABLE KUNDE3(
ID int NOT NULL,
Vorname varchar(50) NOT NULL,
Nachname varchar(50) NOT NULL
)
GO
CREATE CLUSTERED INDEX idx_cl_Nachname ON KUNDE3 (Nachname)
GO
```

Speicherung von Daten in einem clustered Index plus Zusatzindex

Der scheinbare Nachteil des Primärspeichers kann durch zusätzliche Indizes eliminiert werden:



Dabei zeigt die Adressierung des Zusatzindex' auf den Primärschlüssel des clustered Index und nicht wie beim Heap direkt auf die Speicherseite. Dies kann dazu führen, dass der Zugriff langsamer ist als beim normalen nonclustered Index.

Syntax in SQL:

```
CREATE TABLE KUNDE5(
ID int NOT NULL PRIMARY KEY,
Vorname varchar(50) NOT NULL,
Nachname varchar(50) NOT NULL
)
GO
CREATE NONCLUSTERED INDEX idx_nc2_Nachname ON KUNDE5
(Nachname)
GO
```

Im obigen Beispiel wird der Begriff „clustered Index" nicht verwendet. Dies soll unterstreichen, dass der SQL Server bei der Verwendung des Schlüsselworts „Primary Key" automatisch einen clustered Index erstellt!

Vorteile:

› Mehrere Indizes möglich

Nachteile:

› Zugriff über nonclustered Index minimal langsamer als über Heap, da beide Bäume durchlaufen werden müssen

Lesen Sie sich durch die Artikel im Unterordner auf dem Share. Besonders empfohlen: Artikel „4_Brads Sure Guide to Indexes“ als zusätzliche Quelle ab Seite 8.

Nicht behandelte Themen:

- Mehrere unterschiedliche Indexe pro Tabelle mit Heap
- Zusammengesetzte Indexe (wenn immer in gleicher Reihenfolge gesucht)
  „1_Die Richtige Spaltenreihenfolge im Index.pdf“
- Fragmentierung von Indizes
  „2_SQL Server Solutions with Practical SQL DBA- SQL Server - Index Fragmentation - Understanding Fragmentation.pdf“

Es gäbe noch eine letzte Kombination von Indizes, wie könnte diese aussehen? Skizzieren Sie!

Aufgabenteil

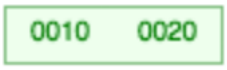
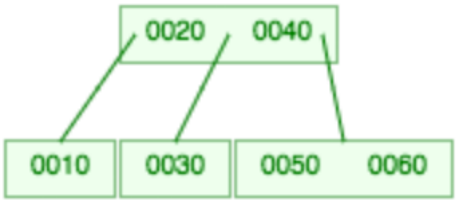
1) Der Balanced Tree 10 Minuten

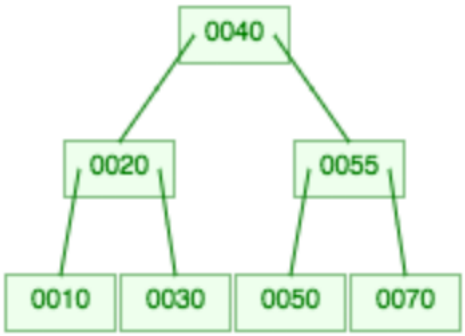
Arbeiten Sie mit der Simulation unter:
https://www.cs.usfca.edu/~galles/visualization/BTree.html

Die Standardeinstellungen reichen aus, um die Funktionsweise eines B-Trees zu analysieren und zu verstehen. Fügen Sie nach und nach Daten auf wie unten beschrieben! Beobachten Sie nach welcher Logik sich der Baum aufbaut und welche Werte jeweils im Root-Element gespeichert sind.

A) 10 Werte zwischen 1-99 (schauen Sie wie sich der Baum aufbaut) Dann wieder clear drücken!
B) 10 Werte zwischen 1-50, anschliessend 3 Werte zwischen 80-99. Dann wieder clear drücken!
C) 10 Buchstaben

Sagen Sie den nächsten Schritt voraus! (ohne das online Tool zu nutzen)

 30 wird eingefügt	
 70 wird eingefügt	



70 wird gelöscht

2) Performanceanalyse mittels Prototyp (6er Gruppen) 60 Minuten

Nun wollen wir die Theorie mit der Praxis vergleichen, damit wir aussagekräftige Daten erhalten, müssen wir mit grossen Datenmengen arbeiten. Alle Dateien liegen bereits vorgefertigt auf dem Share. Damit wir die erhaltenen Werte vergleichen können, füllen wir die Performancewerte in eine Exceltabelle ein.

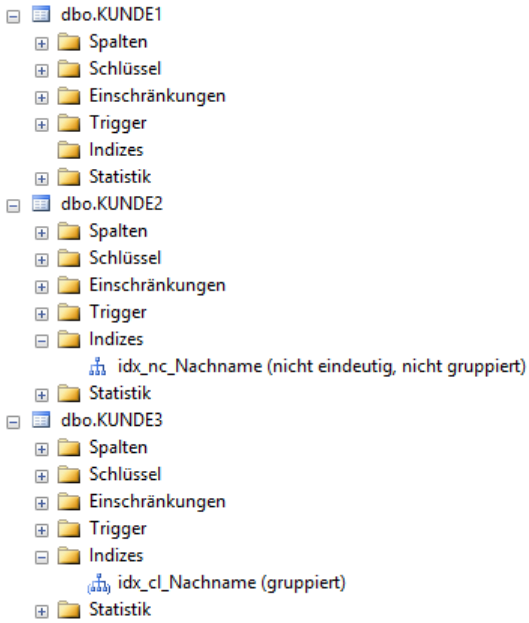
Pro Gruppe übernimmt jeweils ein Mitglied eine Variante.

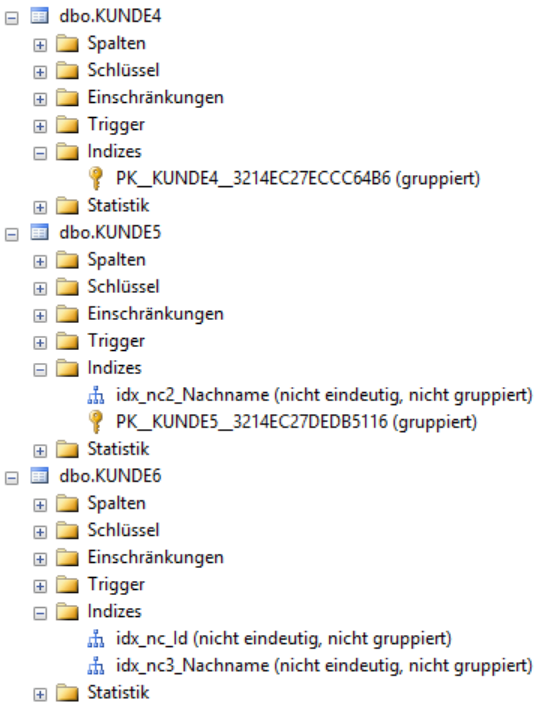
Es gibt 6 Varianten:

- Kunde1 reiner HEAP
- Kunde2 Heap mit Index auf Nachname
- Kunde3 clustered Index auf Nachname
- Kunde4 clustered Index auf ID
- Kunde5 clustered Index auf ID mit nonclustered auf Name
- Kunde6 nonclustered Index auf ID, nonclustered Index auf Name

Erstellen Sie eine neue Datenbank „AB04“ (Script „0_Create Database.sql“) und kontrollieren Sie die Anfangsgrösse, diese sollte auf 200 MB eingestellt sein. (Eigenschaften der DB, Abschnitt Dateien -> Anfangsgrösse)

Lassen Sie die Tabellen erstellen (Script „1_Create Kunden mit Varianten.sql“) und analysieren Sie die unterschiedlichen Varianten. Kontrollieren Sie ebenfalls im Objekt-Explorer ob die



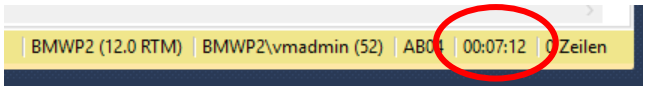


Tabellen sowie die Indizes korrekt angelegt wurden!

Nun werden die 500'000 Datensätze importiert (2_INSERT KUNDE1 mit Nachname.sql). Diese Operation bringt das Management Studio an seine Grenzen, es ist mit Fehlermeldungen zu rechnen. Deshalb kontrollieren Sie vor und nach jedem Import die Menge der Datensätze in der jeweiligen Tabelle mittels COUNT.

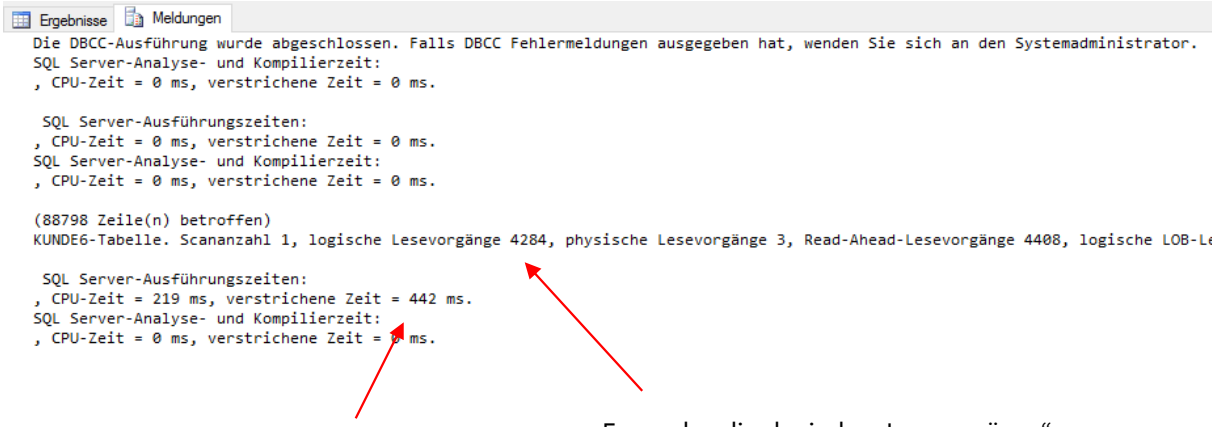
Tipp: führen Sie in der gleichen Datei wie die INSERT Statements stehen zuerst eine belanglose Abfrage aus, anschliessend wird erst die ganze Datei verarbeitet.

Notieren Sie in der Auswertung, wie lange der Import gedauert hat. Dies können Sie unten rechts im Management Studio ablesen:



Nach erfolgreichem Import können nun die Abfragen durchgeführt werden. **Machen Sie für jede Abfrage 5 Durchläufe** damit die Resultate besser gesichert sind und tragen Sie diese in die Auswertung ein. (Auswertung.xlsx sowie 3_Div Abfragen.sql)

Wie werden die Statistiken richtig gelesen? Wechseln Sie nach der Ausführung auf das Register „Meldungen“:



Es wird die Zeit gemessen, um die Abfrage komplett auszuführen.

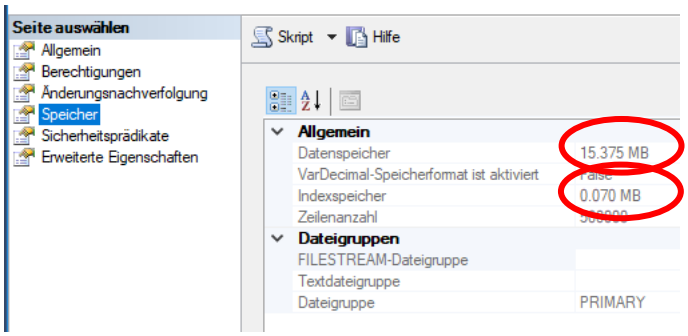
Es werden die „logischen Lesevorgänge“ gemessen welche durchgeführt werden, um die Anfrage komplett auszuführen.

In Excel:

Variante	Aktivität	# log Reads	Zeit				
			Lauf 1	Lauf 2	Lauf 3	Lauf 4	Lauf 5
Variante 1	INSERT File (Sekunden)						
HEAP	SELECT Nachname FROM KUNDE1 WHERE N						
	SELECT Nachname FROM KUNDE1 WHERE N						
	SELECT * FROM KUNDE1 WHERE Nachname						

Nach erfolgter Durchführung werden alle Resultate pro Team in eine Excel Datei kopiert.

Notieren Sie auch die **Dateigrösse** ganz unten. Diese wird auf der Tabelle → Eigenschaften → Speicher aus der Summe der Daten aus «Datenspeicher» und «Indexspeicher» berechnet.



Anschliessend wird jede Gruppe im Plenum ihr Resultat vorstellen.

→ Vergleichen Sie die verschiedenen Varianten und machen Sie innerhalb des Teams Schlussfolgerungen!

→ Welche Empfehlung geben Sie nun ab für die produktive DB? Begründen Sie!

3) Verständnisfragen (Einzelarbeit) 15 Minuten

Was sind die Auswirkungen, wenn nach einem Attribut gesucht wird, für welches kein Index besteht?

In welcher Reihenfolge stehen die Daten in einem Heap?

Wieso geht ein INSERT in einem clustered Index schneller als in einem Heap mit Index?