

Teil XI

Spalten-orientierte DBMSs

Spalten-orientierte Datenbankmanagementsysteme

- 1 Motivation
- 2 Funktionsweise
- 3 Erweiterungen
- 4 Literatur

On-line Analytical Processing (OLAP)

- Anfragen dienen Analyse von Daten (lange Transaktionen)
- Datenbestand stabil, d.h. wenige/keine Updates
- Import von Daten (oft) über ETL-Prozess
- Einzelwerte oft uninteressant (vgl. Anwendungsfelder DWH)
- Häufig erstellen und verarbeiten aggregierter Werte
 - ▶ AVG(), SUM(), COUNT()
 - ▶ GROUP BY (CUBE)
 - ▶ CUBE-Operationen
- Für Aggregatfunktionen (z.B. AVG()) einzelne Spalten interessant
- Auch Gruppierungen (und CUBE) intuitive spaltenweise Bearbeitung

Datenexplosion

- Datenbestände im OLAP-Bereich wachsen ständig \leftrightarrow Daten in zentralisierten Systemen kaum noch verwaltbar
- Historisierung der Daten (z.B. im DWH) erhöht Datenvolumen zusätzlich
- OLAP-Anfragen sehr speicher- und rechenintensiv \leftrightarrow Verteilung der Last notwendig
- Für Aggregationen (OLAP) ist eine vertikale Partitionierung/ Fragmentierung sinnvoll \leftrightarrow bereits bestehende Partitionierung von Column Stores ausnutzen
- Aktuelle Systeme setzen Kompressionstechniken zur Datenvolumenreduktion ein

OLAP: Row Store

- Historisch: Anwendung in On-line Transactional Processing (OLTP) mit kurzen Transaktionen, z.B. Banktransaktionen
- Abbildung von Tupeln in DBMS, d.h. Tupel sequentiell gespeichert
- Verarbeitung ganzer Tupel für Aggregatsfunktionen \leftrightarrow I/O Overhead
- **Insgesamt:** Tupel-orientierte physische Speicherung ungünstig für OLAP

Last Name	First Name	E-mail	Phone #	Street Address

Abb. aus "Query Execution in Column-Oriented Database Systems" by D. Abadi at SIGMOD Jim Gray Doctoral Dissertation Award.

OLAP: Column Store

- Neue Architektur für OLAP notwendig
↳ Column Stores
- Tupel spaltenweise partitioniert, d.h. Werte einer Column sequentiell gespeichert (und sortiert)
- Aggregatsfunktionen arbeiten direkt auf Spalten ↳ nur benötigte Daten eingelesen
- **Insgesamt:** Column Stores können Aggregationen viel effektiver bearbeiten

Last Name	First Name	E-mail	Phone #	Street Address

Abb. aus "Query Execution in Column-Oriented Database Systems" by D. Abadi at SIGMOD Jim Gray Doctoral Dissertation Award.

Kompression

- Row Stores
 - ▶ Kompression relations- oder partitionsweise
 - ▶ Verschiedene Datentypen \leftrightarrow Trade off bei Kompressionstechnikauswahl notwendig
 - ▶ Übliche Kompressionsraten 2:1 bis 5:1
- Column Stores
 - ▶ Kompression je Column möglich
 - ▶ Nutzung verschiedenster Techniken, z.B. Run Length Encoding (RLE), Wörterbuchkodierung (Lempel-Ziv)
 - ▶ Auswahl bester Kompressionstechnik je Spalte, d.h. für jeden Datentyp
 - ▶ Übliche Kompressionsraten 10:1 bis 40:1
- Column Stores reduzieren I/O Overhead **und** verringern Datenvolumen z.T. erheblich \leftrightarrow bessere Ausnutzung des Hauptspeichers

Anfragebearbeitung

- Column Stores basieren auch auf relationalem Datenmodell \leftrightarrow Verwendung der relationalen Algebra und derer Operationen¹
- Logischer Anfrageplan wie bei Row Stores
- Architekturspezifische Ausführung transparent
- Bitoperationen von Natur aus unterstützt (vgl. Bitmap-Join-Index)
- Spaltenweise Kompression erlaubt Bearbeitung ohne Dekompression
 - ▶ Verlustfreie Kompressionstechniken (Bekannteste: Lempel-Ziv und Derivate) \leftrightarrow gleiche (unkomprimierte) Werte haben gleiche komprimierte Darstellung
 - ▶ D.h. Vergleichswert wird falls nötig vor Anfrageausführung in komprimierte Darstellung überführt \leftrightarrow gut geeignet für nicht vektorbasierte Joins
 - ▶ Ordnungserhaltende Techniken wie RLE \leftrightarrow Werte ggf. direkt vergleichbar oder wie bei verlustfreier Kompression
 - ▶ D.h. Aggregatfunktionen wie MIN/MAX oder SUM können Daten komprimiert verarbeiten

¹Nicht relationale oder noSQL DBMS werden in dieser Vorlesung nicht

Anfrageplanausführung Column vs. Row Store

```
SELECT AVG(Attr. 1), Attr. 2 FROM Tabelle
WHERE Attr. 2=X;
```

Row Store

Seite

111	104	121	...		T1:
...	T2:	Attr. 1	Attr. 2 = X	...	T3:	Attr. 1	...
Attr. 2	...	T4:	Attr. 1	Attr. 2 = X	...	T5:	Attr.
1	Attr. 2	...					

↓ Selektion Attr. 2

T2:	Attr. 1	Attr. 2 = X	...
T4:	Attr. 1	Attr. 2 = X	...

↓ AVG(Attr. 1)
+ Projektion

Ergebnis

Column Store

Seite

211	199	222	...	
	T1:	Attr. 1	T2:	Attr.
1	T3:	Attr. 1	T4:	...

Seite

45	32	101	...	
	T1:	Attr. 2	T2:	
Attr. 2 = X	T3:	Attr. 2		
	T4:	Attr. 2 = X		

↓ AVG(Attr. 1)

AVG(Attr. 1)

↓ Selektion

T2:	Attr. 2 = X
T4:	Attr. 2 = X

Merge Columns

Ergebnis

Tupelrekonstruktion

- Operator heißt **SPC** (Scan, Predicate, Construct) für volle Tupelrekonstruktion
- Merge** ist ein k-tuplige Rekonstruktion (mit Spalten $VAL_1 \dots VAL_k$)

Relation ABCD

A	B	C	D

Row Store

Header						
<table border="1"> <tr> <td style="background-color: blue;"></td> <td style="background-color: yellow;"></td> </tr> <tr> <td style="background-color: yellow;"></td> <td style="background-color: red;"></td> </tr> <tr> <td style="background-color: red;"></td> <td></td> </tr> </table>						

Column Store

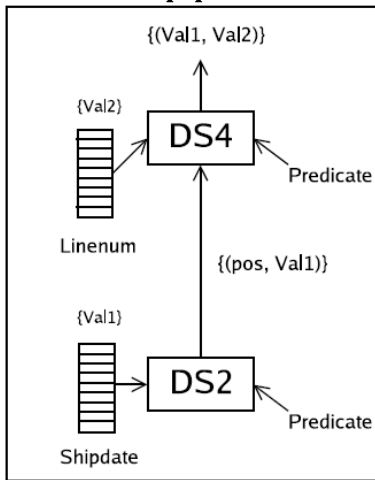
Header												
<table border="1"> <tr> <td style="background-color: blue;">A</td> <td style="background-color: yellow;">A</td> <td style="background-color: red;">A</td> <td>...</td> </tr> <tr> <td style="background-color: blue;">B</td> <td style="background-color: yellow;">B</td> <td style="background-color: red;">B</td> <td>...</td> </tr> <tr> <td style="background-color: red;">C</td> <td>...</td> <td style="background-color: blue;">C</td> <td style="background-color: yellow;">C</td> </tr> </table>	A	A	A	...	B	B	B	...	C	...	C	C
A	A	A	...									
B	B	B	...									
C	...	C	C									

Materialisierungszeitpunkt

- Frühe Materialisierung (EM)
 - ▶ Anfragebearbeitung sehr nah an Row Stores
 - ▶ Aggregatfunktionen auf einzelnen Columns
 - ▶ Tupelrekonstruktion sobald Tupel verwendet
 - ▶ Zumeist Verwendung bei tupel-orientierter Anfragebearbeitung
- Späte Materialisierung (LM)
 - ▶ So lang wie möglich auf Columns arbeiten
 - ▶ Mehrfacher Zugriff auf Basistabellen und/oder Zwischenergebnisse
 - ▶ Folge: Anfrageplan kein Baum mehr
 - ▶ Aber: Gleichzeitige Bearbeitung auf komprimierten und unkomprimierten Daten möglich
 - ▶ Notwendig für (effektive) spalten-orientierte Anfragebearbeitung

Frühe Materialisierung (EM)

EM-pipelined



EM-parallel

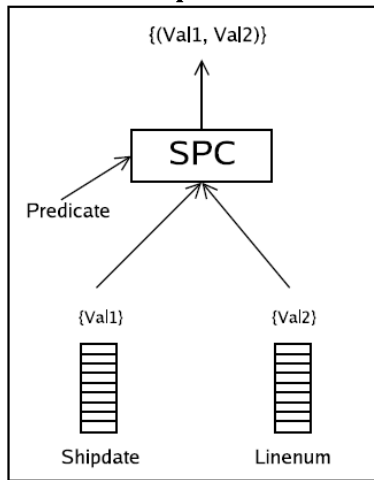


Abb. aus "Query execution in column-oriented database systems", PhD thesis by D. Abadi

Späte Materialisierung (LM)

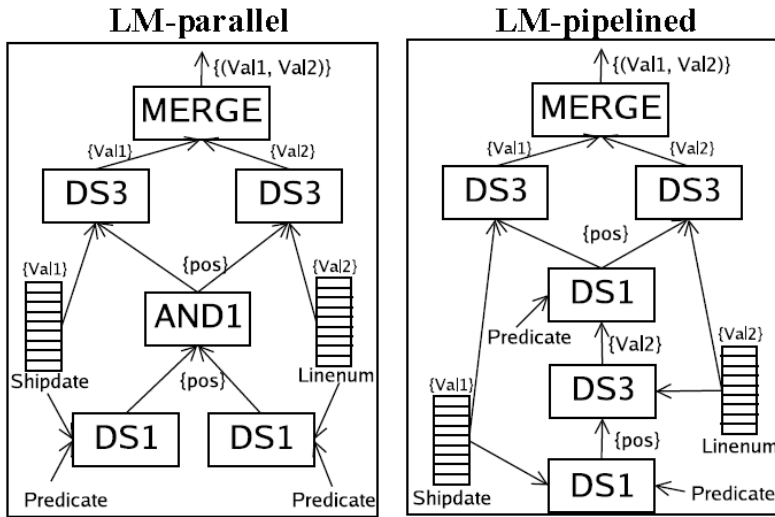


Abb. aus "Query execution in column-oriented database systems", PhD thesis by D. Abadi

Nachteile

- Tupelrekonstruktion verursacht Kosten
- Kosten für Einfügeoperation durch Partitionierung der Tupel
- Updates benötigen Tupelrekonstruktion
- Folge: Insert- und Update-in-place nicht möglich
- **Aber:** Updates und Inserts in OLAP/DWH Anwendung selten oder nur durch ETL

Lösungsansätze

- Tupel-orientierte Anfragebearbeitung und frühe Materialisierung
- C-Store/Vertica
 - ▶ Read-optimized (RS) and write-optimized Storage (WS)
 - ▶ Verschiedene sich überlappende Projektionen im RS
 - ▶ Inserts und Updates nur im WS
 - ▶ Tuple mover übermittelt Daten vom WS in RS zu geringer Last (offline)
- SybaseIQ (erster kommerzieller Column Store)
 - ▶ Ähnlich zum C-Store-Ansatz
 - ▶ System unterteilt in Read- und Write- bzw. Read/Write-Knoten
 - ▶ Abgleich im Hintergrund zu Zeiten geringer Last
- Redundanz
 - ▶ Daten column- und row-orientiert im Hauptspeicher
 - ▶ Datenbank redundant als Column und Row Store
 - ▶ Virtualisierung des Datenwürfels
- ...

Systeme

- Kommerziell
 - ▶ SybaseIQ
 - ▶ Vertica
 - ▶ Infobright ICE
 - ▶ Tenbase (webbasiert)
 - ▶ BigTable (Google, nicht relational)
 - ▶ ...
- Frei
 - ▶ Infobright ICE Community Edition
 - ▶ LucidDB
 - ▶ MonetDBX100 (Ingres/Vectorwise)
 - ▶ C-Store (benötigt alten gcc)
 - ▶ Hbase (Apache), Hypertable, Cassandra (Facebook) alles BigTable-Derivate
 - ▶ ...
- Im Gegensatz zu Row Stores weichen Column Store Implementierungen untereinander sehr stark ab

Zusammenfassung

- Row Stores nicht optimal für OLAP- und DWH-Anwendungen
- Column Stores besser geeignet für Aggregatsfunktionen
- Column Stores verringern Datenvolumen z.T. drastisch ↔ geringerer I/O, bessere Ausnutzung des Hauptspeichers
- Darstellung von Tupeln erzeugt Kosten in Column Stores (Tupelrekonstruktion)
- Column Stores zeigen Schwächen bei Inserts und Updates
- Sehr viele und stark unterschiedliche Implementierungen für Column Stores

Referenzen

- D. J. Abadi. Query execution in column-oriented database systems. PhD thesis, Cambridge, MA, USA, 2008. Adviser: Madden, Samuel.
- D. J. Abadi, S. R. Madden, and N. Hachem. Column-stores vs. row-stores: How different are they really? In SIGMOD '08, pages 967-980, New York, NY, USA, 2008. ACM.
- G. P. Copeland and S. N. Khoshafian. A decomposition storage model. In SIGMOD '85, pages 268-279, New York, NY, USA, 1985. ACM.
- H. Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In SIGMOD '09, pages 1-2, New York, NY, USA, 2009. ACM.
- J. Schaffner, A. Bog, J. Krüger, and A. Zeier. A hybrid row-column OLTP database architecture for operational reporting. In BIRTE '08, 2008.
- D. Slezak, J. Wroblewski, V. Eastwood, and P. Synak. Bighthouse: an analytic data warehouse for ad-hoc queries. PVLDB, 1(2):1337-1345, 2008.
- M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-Store: A column-oriented DBMS. In VLDB '05, pages 553-564. VLDB Endowment, 2005.
- ...