Datenbanken: Weitere Datenbankoptimierungen

Stefan Halfpap, Ralf Teusner, Werner Sinzig, Michael Perscheid

12. Juni 2020

Vorlesungsinhalte/-aufbau

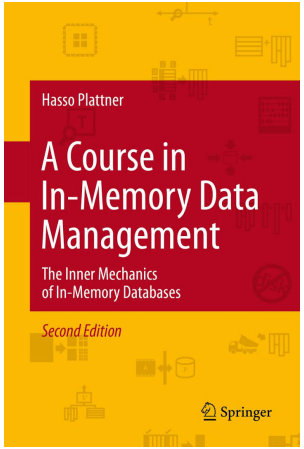
- Einführung zu Unternehmensanwendungen
- Grundlagen des IT-gestützten Rechnungswesens und der Planung
- Einführung zu relationalen Datenbanken und Anfrageverarbeitung
- **Grundlagen von (spaltenorientierten) Hauptspeicherdatenbanken**
- Trends in Hauptspeicherdatenbanken
- Klausur

Überblick

- Architektur einer spaltenbasierten HauptspeicherDB
- Datenbankoptimierungen
  - Insert-Only-Ansatz
  - Main-Delta- und Chunk-Architektur
  - History-Partition
  - Hot-Cold Datenpartitionierung
  - Replikation
- Zusammenfassung

Literatur und Empfehlung

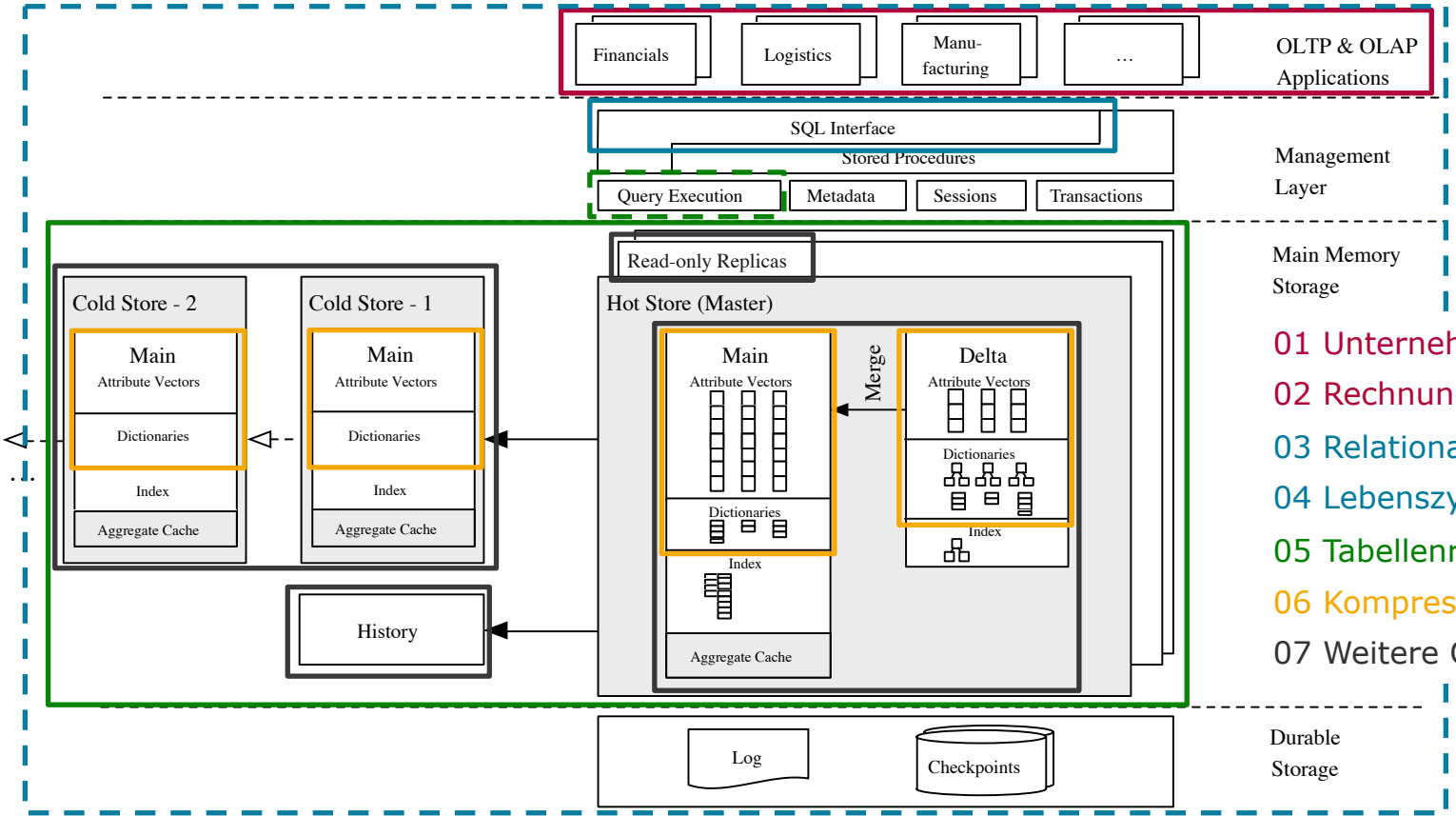
- Hasso Plattner „A Course in In-Memory Data Management”



- Hasso Plattner „The Impact of Columnar In-Memory Databases on Enterprise Systems“ (2014)
- Dreseler et al. „Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management “ (2019)

Architektur einer spaltenbasierten HauptspeicherDB

Hasso Plattner „The Impact of Columnar In-Memory Databases on Enterprise Systems“ (2014)



01 Unternehmensanwendungen

02 Rechnungswesen, Planung

03 Relationales Modell, SQL

04 Lebenszyklus einer Query

05 Tabellenrepräsentation

06 Kompression

07 Weitere Optimierungen

Architektur einer spaltenbasierten HauptspeicherDB
Datenbankoptimierungen

- Insert-Only-Ansatz
- Main-Delta- und Chunk-Architektur
- History-Partition
- Hot-Cold Datenpartitionierung
- Replikation

Insert-Only-Ansatz

- Gelöschte Daten erzeugen Lücken in den Datenstrukturen
- Reorganisation/Lückenschluss ist in spaltenorientierten Datenbanken teuer, da alle Spalten umorganisiert werden müssen und Datenkompression die Umorganisation zusätzlich erschweren kann

Insert-Only-Ansatz

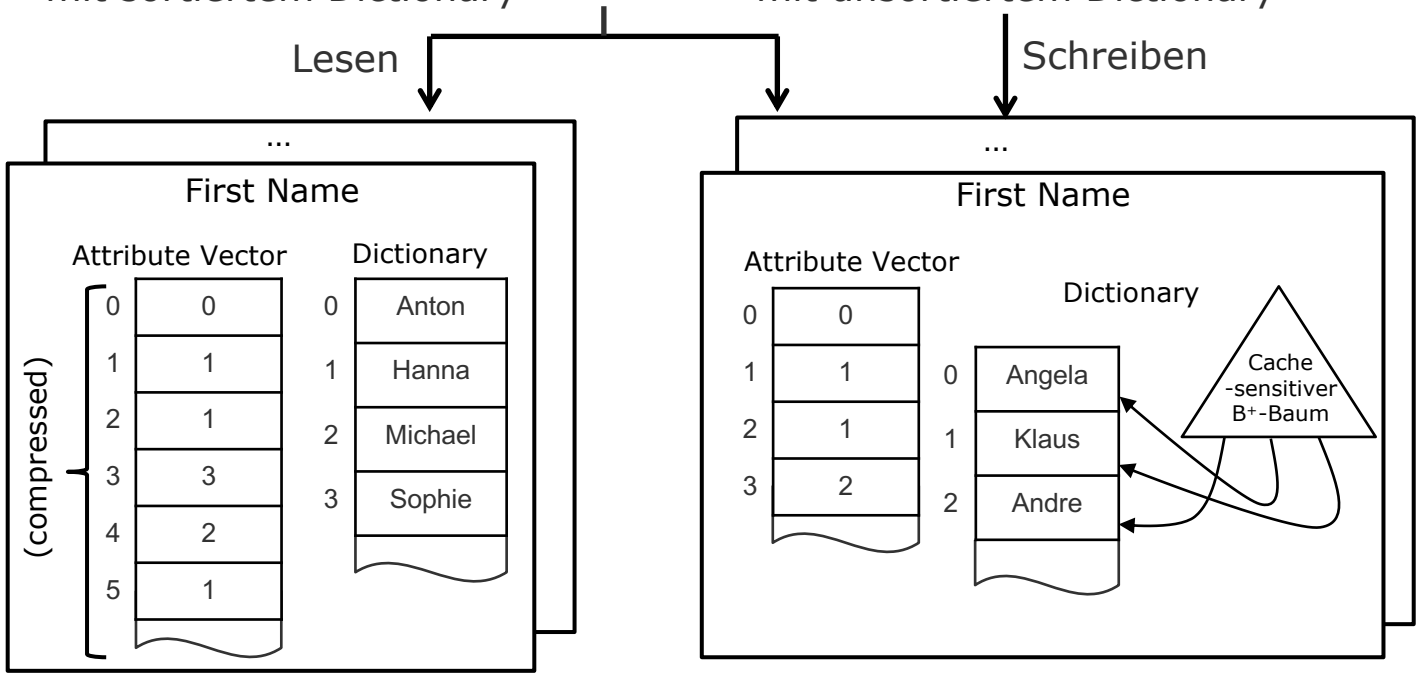
- Gelöschte Daten werden nicht überschrieben, sondern als ungültig markiert
- UPDATEs sind als INSERT und DELETE umgesetzt
- Verschiedene Implementierungen:
  - Bitmap pro Tabelle, die Gültigkeit einzelner Tupel angibt
  - **Gültigkeitsbereich pro Tupel** → ermöglicht Multi Version Concurrency Contol (MVCC) (Details in 09_DB_Nebenläufigkeitskontolle)

Main-Delta-Architektur
Motivation

- Dictionary-Kompression: Sortiertes vs. unsortiertes Dictionary (Wiederholung)
  - Pro Sortierung:
    - Suche der Wert-ID im Dictionary hat Komplexität O(log(n)) statt O(n)
    - Bereichsabfragen (range queries) beschleunigen
    - Dictionary kann besser komprimiert werden
  - Contra Sortierung:
    - Einfügen neuer Werte kann eine Reorganisation der Datenstrukturen benötigen (um Sortierung zu bewahren oder wenn sich die Anzahl der benötigten Bits pro Wert-ID verändert)
- Löschen von Tupeln führt zu „Lücken“

Main-Delta-Architektur
Idee: Zwei verschiedene horizontale Partitionen

- Leseoptimierte **Main-Partition** mit sortiertem Dictionary
- Schreiboptimierte **Delta-Partition** mit unsortiertem Dictionary



...	First Name	...
	Anton	
	Hanna	
	Hanna	
	Sophie	
	Michael	
	Hanna	
	...	
	Angela	
	Klaus	
	Klaus	
	Andre	
	...	

Main

Delta

Datenrepräsentation im Speicher

Logische Sicht

Main-Delta-Architektur
Delta-Partition

- Gute Leseperformanz und günstigeres Einfügen
  - Gute Leseperformanz, da Daten komprimiert sind
  - Günstigeres Einfügen, da Dictionary und Attributvektor nicht aufwendig umorganisiert werden müssen
- Benötigt mehr Speicher
  - zusätzlicher Baum fürs Dictionary
  - i.d.R. keine Attributvektorkomprimierung
- Keine effizienten Bereichsabfragen

Main-Delta-Architektur
Schreiboperationen

- INSERTs werden im Delta gespeichert
- DELETEs erzeugen ungültige Tupel (im Main und Delta)
- UPDATEs sind als INSERT und DELETE umgesetzt (Insert-Only-Ansatz)

  →

- Delta wächst kontinuierlich
- Main und Delta bekommen „Lücken“

  →

- Datenreorganisation / Merge / Tuple-Mover notwendig

  (Anfrageoptimierung für zukünftige Queries)

Main-Delta-Architektur
Merge-Prozess

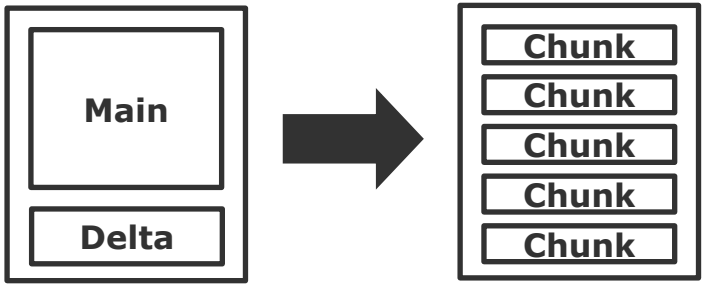
- Asynchroner Prozess in Bezug auf Anfragebearbeitung

  (speicherintensiv, aber optimierbar)
- Angestoßen durch Anzahl der Tupel im Delta oder durch Kostenmodell (oder manuell)

- Schritte:
  - „Merge“ Main- und Delta-Dictionary (Optional: Löschen nicht benötigter Werte)
  - Erstellen der Abbildung: alte Wert-ID → neue Wert-ID (für Main und Delta)
  - (Main-)Attributvektor neu schreiben
  - (Delta-Partition ist nach dem Merge-Prozess leer)

Main-Delta-Architektur
Alternative: Chunks

- Motivation: Kosten für Merge-Prozess steigen mit der Zeit
- Idee:
  - Mehrere Chunks/Blöcke fester Größe (horizontale Partitionen)



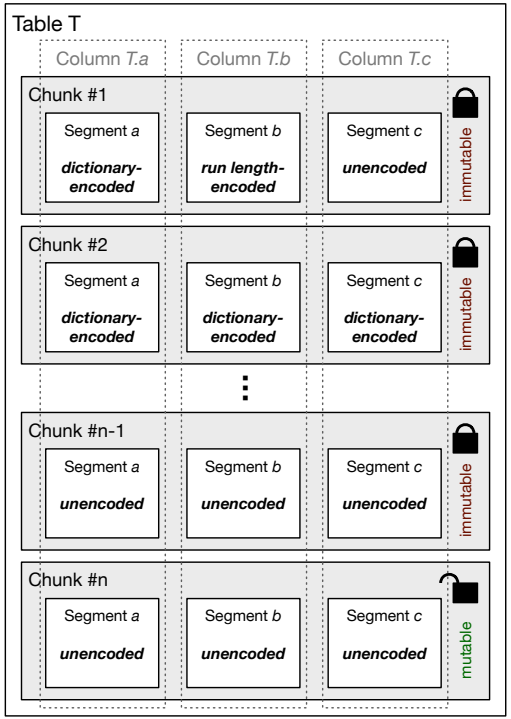
  - Drei Chunktypen (Phasen werden sequentiell durchlaufen)

    → komprimierte Chunks sind (im Vergleich zum Main) „stabil" (kein Merge)

Main-Delta-Architektur
Alternative: Chunks – Example

Beispiel eines Speicherlayouts in Hyrise mit

- *n* Chunks
- 3 Attribute



Dreseler et al. „Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management " (2019)

Main-Delta-Architektur
Alternative: Chunks - Anmerkungen

- Tupel aus komprimierten Chunks können gelöscht/ungültig werden

  → Merge mehrerer „löchriger" Chunks

- Potenziell höherer Speicherverbrauch (mehrere gleiche Dictionary-Einträge für unterschiedliche Chunks vs. weniger Bits pro Wert-ID)

- Implementiert in Hyrise

https://github.com/hyrise/hyrise

History-Partition

- Insert-Only-Ansatz: Gelöschte Daten werden nicht überschrieben, sondern ungültig
  + MVCC/Snapshot-Isolation (und Zeitreise-Queries)
  + Möglicherweise gesetzlich vorgeschrieben
  - Speicherverbrauch
  - Höhere Scan-Kosten

- Ungültige Daten werden (z.B. beim Merge) in die History-Partition geschoben und bei der (normalen) Anfragebearbeitung nicht mehr gelesen

Hot-Cold-Datenpartitionierung

- Fast jede Anwendung folgt dem „working set model“
  - “there is a subset of (pages) that are accessed distinctively more frequently”

    Peter Denning “The working set model for program behaviour” (1968)
- In Unternehmensanwendungen: aktuelles vs. erstes Geschäftsjahr

- Hot-Cold-Datenpartitionierung ist die Ausnutzung dieses Wissens bei der Datenspeicherung
- Idee:
  - Partitioniere Daten in Blöcke und speichere Metainformationen zu den Daten pro Block
  - Prüfe bei der Anfragebearbeitung anhand der Metainformationen, ob die Partition relevante Daten enthält und gelesen werden muss

Current-Historical Datenpartitionierung
Ein Extremfall von Hot-Cold Datenpartitionierung

Aktuelle (current) Daten

- **Durch den Anwendungsentwickler** als relevante Daten **definiert**
- Alle veränderbaren Daten
- Häufig zugegriffene Daten (hot)
- Auf schnellen Speicher

Historische/ältere (historical) Daten

(≠ gelöschte Daten)

- Die restlichen Daten sind „historisch”
- Daten werden nur noch gelesen
- Selten zugegriffene Daten (cold)
- Teilweise auf günstigeren (langsameren) Speicher ausgelagert

- Vorteile:
  - Performanzsteigerung durch das Weglassen des Scans der “historischen” Datenpartition
  - Geringere HW-Kosten durch günstigeren Speicher

Hot-Cold Datenpartitionierung
Pruning-Filter

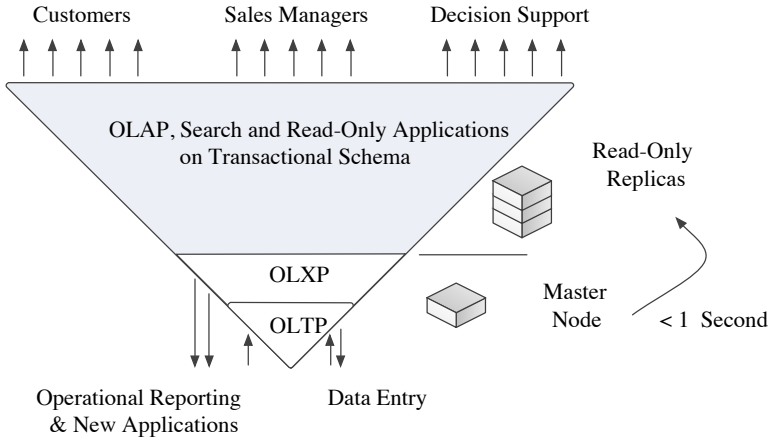
- Wie oft wird auf historische Partitionen zugegriffen?
  - Für tägliche Geschäftsprozesse fast nie
  - Die meisten analytischen Anfragen greifen auch nur auf aktuelle Daten zu

- Um unnötige Zugriff auf nicht relevante (historische) Partitionen zu vermeiden (z.B. bei defensiver Programmierung oder Anwendungen, die die Partitionierung nicht kennen), können spezielle Datenstrukturen (**Pruning-Filter**) verwendet werden, die Metainformationen zu den Daten der Partition speichern
  - Platz-effiziente probabilistische Datenstrukturen
  - Fassen die Daten (Werte, Wertebereiche und Kombinationen) von Partitionen zusammen
  - Erkennen unnötigen Zugriff

Replikation

- Neue Unternehmensanwendungen ..
  - .. locken zunehmend Nutzer an
  - .. stellen zunehmend komplexe Anfragen
  - .. unterstützen interaktive Datenexploration

  .. erfordern Skalierbarkeit



Hasso Plattner „The Impact of Columnar In-Memory Databases on Enterprise Systems" (2014)

- Verwende Datenbankreplikate (zusätzlicher Computer als Kopien des Datenbanksystems) für Anfragebearbeitung (und zur höheren Verfügbarkeit)
- Replikationsverfahren können klassifiziert werden
- Aktive Master-Replikation: Replikatknoten verarbeiten ausschließlich lesende Anfragen auf Snapshots (konsistenten Kopien) des Masters

Aktive Master-Replikation

- **Aktive** vs. passive Replikation
  - Replikate werden für Anfrageverarbeitung (aktiv) und nicht nur als Backup (passiv) verwendet
- **Master**- vs. Multimaster-/Group-Replikation
  - Ein einzelner Computer (Master) ist für die transaktionale Verarbeitung zuständig (keine teuren verteilten Transaktionen)

- Weitere Spezialisierung/Klassifikation von Replikationsverfahren:
  - Sofortige vs. „träge" Synchronisation der Replikate
  - Logische vs. physische Synchronisationsinformationen
  - Homogene vs. heterogene Replikate

Replikation
Weitere Klassifizierung von Replikationsverfahren

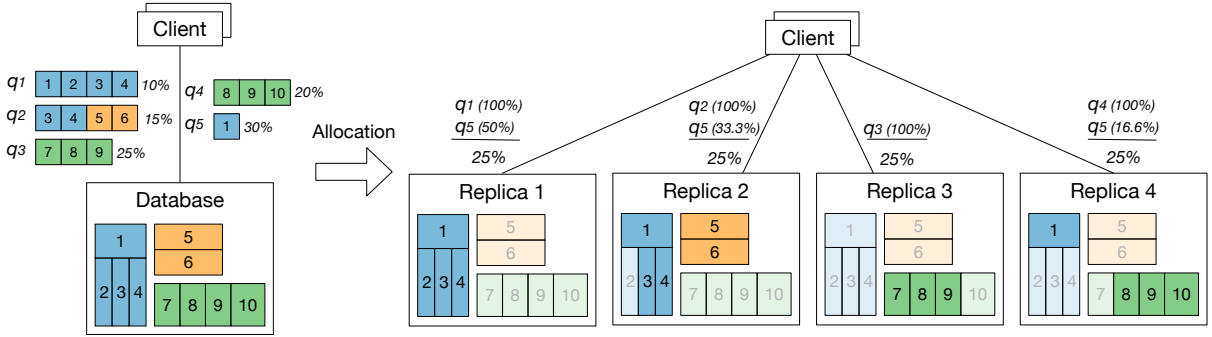
- Sofortige vs. „träge“ Synchronisation der Replikate
  - Sofortig („eager“): Aktualisiere Replikate als Teil der Transaktion
  - „Träge“ („lazy“): Aktualisiere Replikate asynchron zur Transaktion
- Logische vs. physische Synchronisationsinformationen
  - Logisch: Operationen, z.B. SQL-Anfragen
  - Physisch: Datenänderungen
- Homogene vs. heterogene Replikate
  - Homogen: Replikate als genaues „Spiegelbild“ des Masters
  - Heterogen: Replikate speichern Teilmengen der Daten oder optimierte Datenstrukturen

Replikation
Heterogene Replikation – Partielle Replikation

Replikation der Daten basierend auf Anfragen, die Teilmengen von Daten benötigen und für einen bestimmten Workload-Anteil verantwortlich sind

- Partielle Replikatknoten speichern nur Teilmengen der Daten und verbrauchen daher weniger Speicherplatz
- Partielle Replikatknoten können nur Teilmenge der Anfragen (für die alle relevanten Daten gespeichert sind) bearbeiten

→ Effiziente Skalierung



Halfpap and Schlosser „A Comparison of Allocation Algorithms for Partially Replicated Databases“ (2019)

Zusammenfassung

- Main-Delta-Architektur mit leseoptimierter Main- und schreiboptimierter Delta-Partition
- Chunk-Architektur als Alternative um steigende Merge-Kosten zu meiden
- History-Partition speichert gelöschte Daten
- Historische Partition speichert nicht mehr „relevante“ Daten, die für die meisten Anfragen nicht mehr gelesen werden müssen
- Replikation ist eine Möglichkeit Leseanfragen auf zusätzlichen Computern zu skalieren