HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Datenbanken: Weitere Konzepte

Dr. Matthias Uflacker, Stefan Klauck

7. Mai 2018

Vorlesungsinhalte/-aufbau

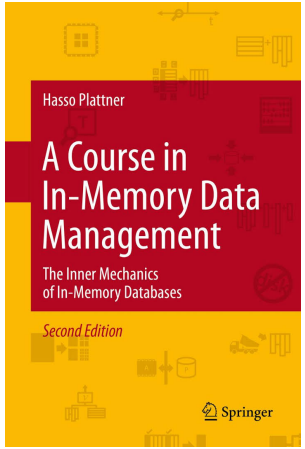
- Phase 1
  - Einführung zu Unternehmensanwendungen (2 Vorlesungen)
  - **Grundlagen von spaltenorientierten Hauptspeicherdatenbanken (4 Vorlesungen)**
  - Wöchentliche Übungsblätter
- Phase 2
  - Grundlagen des IT-gestützten Rechnungswesens und Planung (3 Vorlesungen)
  - Programmiermodelle für Unternehmensanwendungen (1 Vorlesung)
  - Zwei praktische Programmierübungen
- Klausur

Überblick

- Architektur einer spaltenbasierten HauptspeicherDB
- Datenbankkonzepte
  - Main-Delta-Architektur
  - History-Partition
  - Hot-Cold-Datenpartitionierung
  - Aggregat-Cache
  - Replikation
- Zusammenfassung

Literatur und Empfehlung

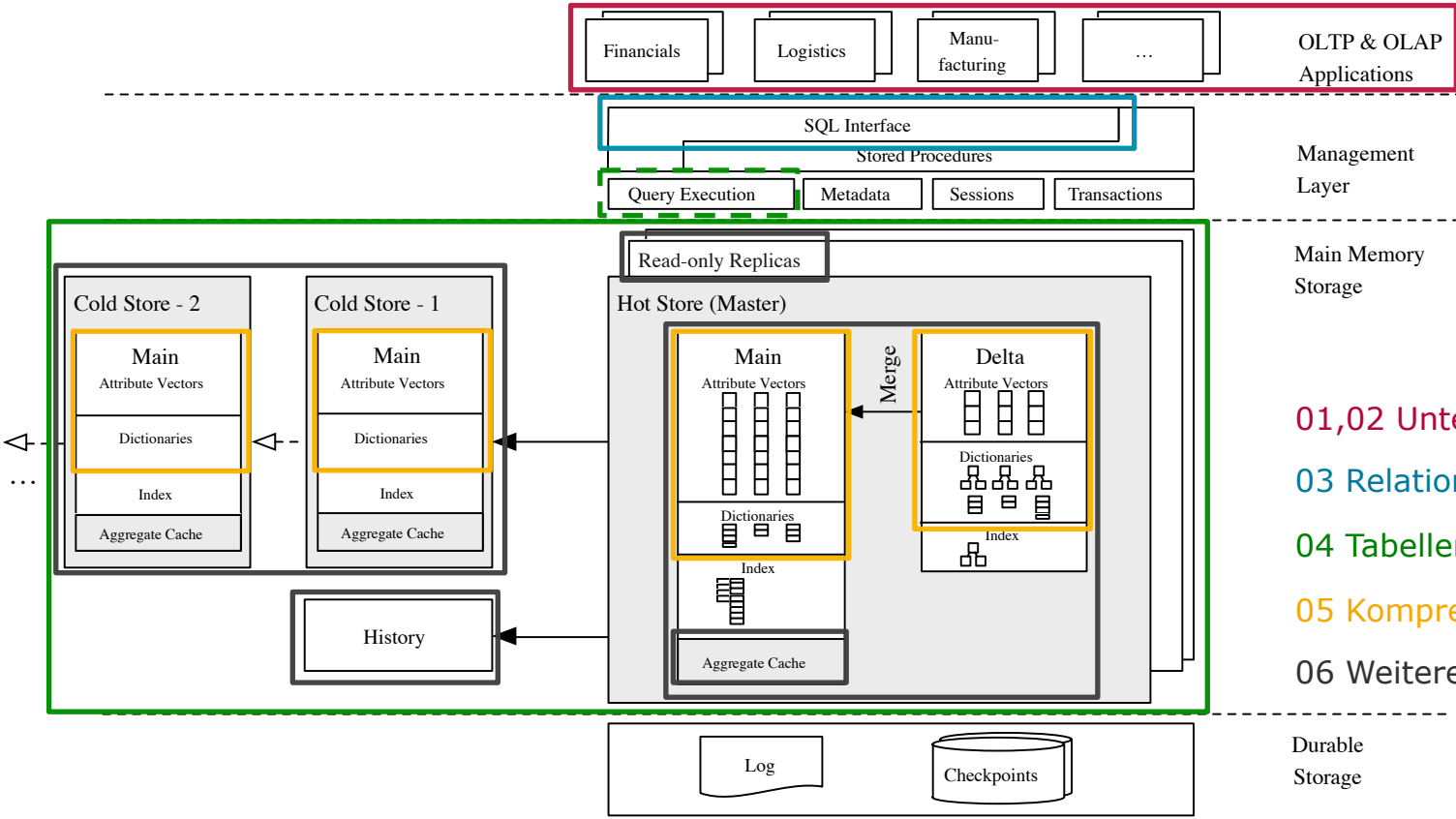
- Hasso Plattner „A Course in In-Memory Data Management”



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

Architektur einer spaltenbasierten HauptspeicherDB

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



01,02 Unternehmensanwendungen

03 Relationales Modell, SQL

04 Tabellenrepräsentation

05 Kompression

06 Weitere Konzepte

Architektur einer spaltenbasierten HauptspeicherDB
Weitere Konzepte

- Main-Delta-Architektur
- History-Partition
- Hot-Cold-Datenpartitionierung
- Aggregat-Cache
- Replikation

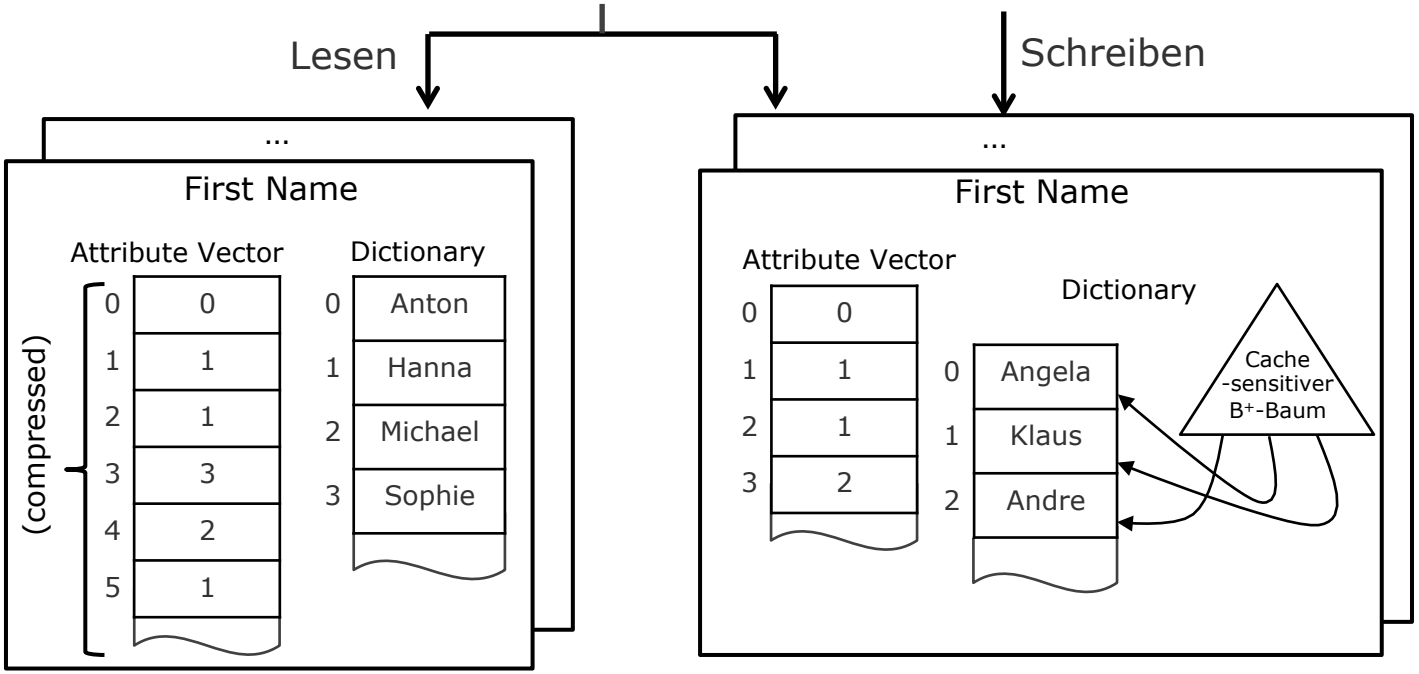
Main-Delta-Architektur
Motivation

- 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 Datenstruktur 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



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 und i.d.R. keine Attributvektorkomprimierung)

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 bekommt „Lücken“

  ->

- Datenreorganisation / Merge / Tuple-Mover notwendig

  (Anfrageoptimierung für zukünftige Queries)

Main-Delta-Architektur
Merge-Prozess

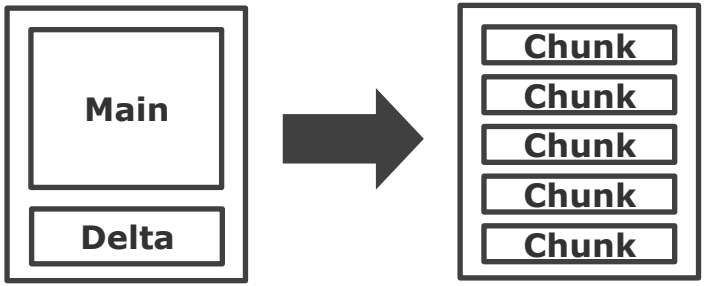
- Asynchroner Prozess in Bezug auf Anfragebearbeitung (speicherintensiv, aber optimierbar)
- Angestoßen durch Anzahl der Tupel im Delta oder durch Kostenmodell (oder manuell)

- 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

Main-Delta-Architektur
Alternative: Chunks (Hyrise)

- Motivation: Kosten für Merge-Prozess steigen mit der Zeit
- Idee:
  - Mehrere Chunks 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 (Hyrise) - Anmerkungen

- Tupel aus komprimierten Chunks können ungültig werden

  -> Merge mehrerer „löchriger" Chunks

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

History-Partition

- Insert-only Ansatz: Gelöschte Daten werden nicht überschrieben, sondern ungültig

  + Snapshot-Isolation und Zeitreise-Queries

  + Möglicherweise gesetzlich vorgeschrieben

  - Speicherverbrauch

  - Höhere Scan-Kosten

- Ungültige Daten werden in die History-Partition geschoben

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

Current-Historical Datenpartitionierung
Ein Extremfall von Hot-Cold Datenpartitionierung

Aktuelle (current) Daten

- **Durch den Anwendungsentwickler** als relevante Daten **definiert**
- Alle veränderbare 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 (und effizientere Scanoperation)

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
  - Platz-effiziente probabilistische Datenstrukturen
  - Fassen die Daten (Werte, Wertebereiche und Kombinationen) von Partitionen zusammen
  - Erkennen unnötigen Zugriff

Aggregat-Cache

- Idee: Zwischengespeicherte Anfrage(zwischen)ergebnisse der Main-Partition werden mit dem on-the-fly Aggregat der Delta-Partition kombiniert

-> Beschleunigung sich wiederholender Aggregat-Queries

- Cache-Eintrag speichert: Tabelle, Gruppierungsattribute, Filterbedingungen, Aggregate und gültige Zeilen

- Keine Aktualisierung des Cache-Eintrags (im Vergleich zu materialisierten Sichten)
  - INSERTs werden durch on-the-fly Aggregation des Deltas berücksichtigt
  - Ungültig gewordene Tupel seit der Erstellung des Cache-Eintrags in Main-Partition werden durch das Speichern der gültigen Zeilen erkannt und rausgerechnet

Aggregat-Cache
Beispiel (vereinfacht) – Cache-Eintrag erstellen

Facts							
Main				Delta			
ID	Date	Prod	Amt	ID	Date	Prod	Amt
1	1/1/2013	1	100				
2	1/1/2013	1	-50				
3	1/1/2013	2	30				
4	1/1/2013	2	60				
5	1/2/2013	1	-10				

```
SELECT Date, Prod, SUM(Amt) FROM Facts GROUP BY Date, Prod
```

↓

Aggregates		
Date	Prod	SUM(Amt)
1/1/2013	1	50
1/2/2013	1	-10
1/1/2013	2	90

Aggregat-Cache
Beispiel (vereinfacht) – Cache-Eintrag wiederverwenden

Facts							
Main				Delta			
ID	Date	Prod	Amt	ID	Date	Prod	Amt
1	1/1/2013	1	100				
2	1/1/2013	1	-50				
3	1/1/2013	2	30				
4	1/1/2013	2	60				
5	1/2/2013	1	-10				
				6	1/2/2013	1	20
				7	1/1/2013	3	50
				8	1/1/2013	3	-10

```
SELECT Date, Prod, SUM(Amt) FROM Facts GROUP BY Date, Prod
```

Aggregates		
Date	Prod	SUM(Amt)
1/1/2013	1	50
1/2/2013	1	-10
1/1/2013	2	90

Main (cached)

Delta (on-the-fly)

Aggregates		
Date	Prod	SUM(Amt)
1/2/2013	1	20
1/1/2013	3	40

```
SELECT Date, Prod, SUM(Amt)
FROM (Select * FROM CachedFactsAgg UNION ALL Select * FROM FactsDelta)
GROUP BY Date, Prod
```

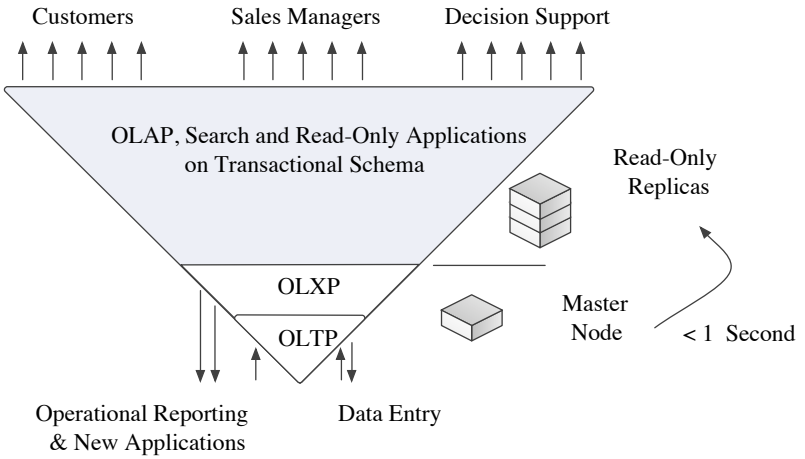
Aggregates		
Date	Prod	SUM(Amt)
1/1/2013	1	50
1/2/2013	1	10
1/1/2013	2	90
1/1/2013	3	40

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)

- Aktive Master-Replikation
  - Replikatknoten verarbeiten ausschließlich lesende Anfragen auf Snapshots des Masters ohne transaktionale Garantien zu verletzen

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

Zusammenfassung

- Main-Delta-Architektur mit leseoptimierter Main- und schreiboptimierter Delta-Partition
- History-Partition speichert gelöschte Daten
- Historische Partition speichert nicht mehr „relevante" Daten, die für die meisten Anfragen nicht mehr gelesen werden müssen
- Aggregat-Cache ist ein durch die Datenbank verwalteter Zwischenspeicher für Aggregate der Main-Partition
- Replikation ist eine Möglichkeit Leseanfragen auf zusätzlichen Computern zu skalieren