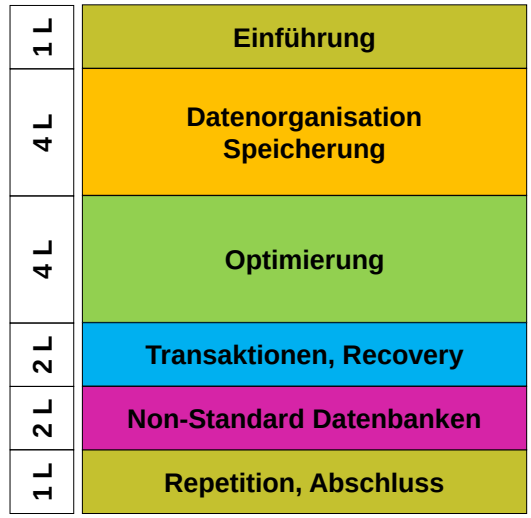
Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Optimierung

Lehrbuch Kapitel 7.5, 8.1 bis 8.3

Gliederung

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie



"You are here"

2

Rückblick

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Verständnis gewinnen für die 'Kosten' eines Verfahrens
- Einige der in einem RDBMS eingesetzten Algorithmen kennen und deren 'Kosten' beurteilen können:
  1. Hauptspeicheralgorithmen
  2. Zugriffe auf Datensätze
  3. Scans
  4. Sortierung
  5. Joins / Verbund (1. Teil)

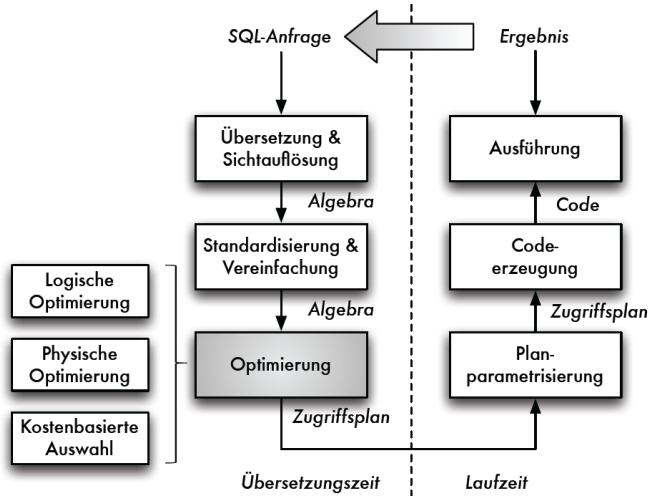
3

Lernziele heute

- Abschliessen der Verbundoperationen:
  5. Joins / Verbund (2. Teil): Hash-Join-Varianten
- Grundlagen der Optimierung verstehen

4

Heute werden wir die Betrachtung der Basisalgorithmen abschliessen, ein einführendes Beispiel für eine logische Optimierung aufzeigen und die mathematischen Grundlagen zur logischen Optimierung einführen. In der nächsten Lektion werden wir dann den gesamten Prozess von der SQL-Anfrage bis zum Ausführungsplan detailliert betrachten.



Hash-Join: Grundidee

Durch gleiche Hashfunktion auf beide Relationen sind Verbundkandidaten in den selben Blöcken (im Hauptspeicher) zu finden.

Vorgehen:

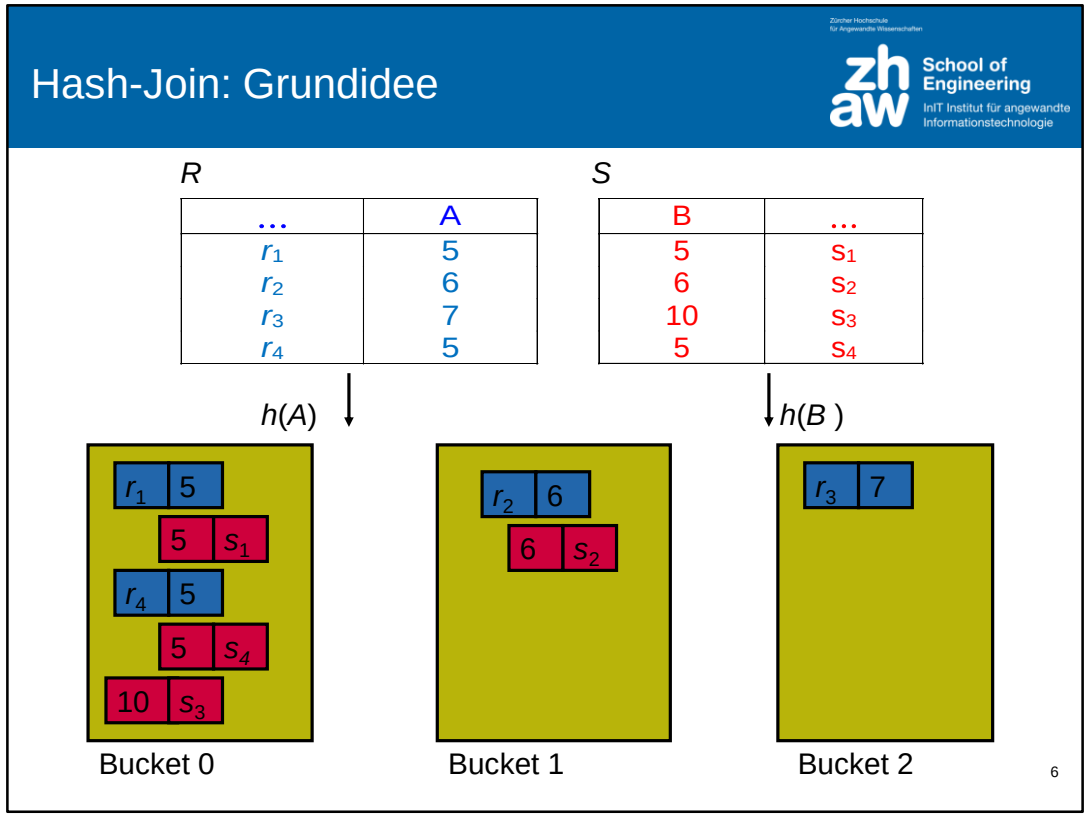
1. Build-Phase: Kleinere Relation durch Hashfunktion h in Buckets einordnen.
2. Probe-Phase: Durch Anwendung von h auf grössere Relation Buckets mit Verbundkandidaten identifizieren.
3. Verbundbedingung pro Bucket prüfen und Resultate ausgeben.

Nur passend für Anfragen der Form: $r \bowtie_{r.A = s.B} s$ (equi-join oder natural join)

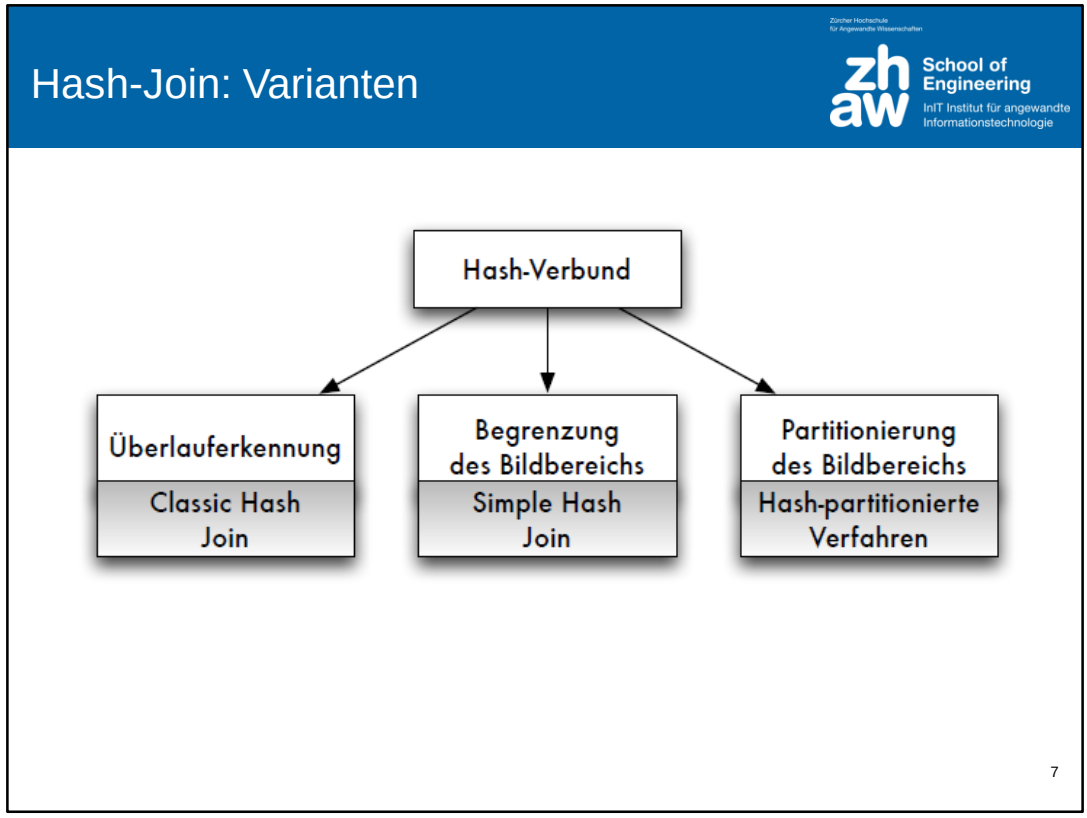
5

Grundidee: Durch Anwendung der gleichen Hashfunktion auf die Verbundsattribute der zu verbindenden Relationen sind die potenzielle Verbundkandidaten in Blöcken (engl. buckets) mit gleichen Hashwerten zu finden. Da die effektive Abbildung auf Buckets nur ein Mal erfolgend muss (im 2. Schritt wird nur noch gesucht), wird die kleinere Relation in Buckets abgebildet.

Je nach Variante des Algorithmus (siehe später) Schritt 1 bis 3 wiederholen, bis alle Tupel der grösseren Relation gelesen sind.



Im Beispiel werden die Verbundattribute A und B mit der Hashfunktion ‘mod 5’ auf die Buckets mit den Adressen 0 bis 2 abgebildet. Dadurch kommen die potentiell zu verbindenden Tupel in den selben Buckets zu liegen. Die Abbildung der 2. Relation in die Buckets wird in der Realität nicht wirklich durchgeführt (es wird direkt der Verbund ausgeführt), sondern ist hier nur zum besseren Verständnis aufgezeigt.



Das in der Grundidee aufgezeigt Verfahren des Hash-Verbunds setzt voraus, dass zumindest die Hashtabelle in den Hauptspeicher passt. Volle Buckets der Hashtabelle (d.h. beim Überlauf) müssen allenfalls auf den Hintergrundspeicher ausgelagert werden, da im Hauptspeicher hierfür kein Platz 'mehr' ist. Dabei können extrem viele Buckets betroffen sein, da der Bildbereich der Hashfunktion ja der Grösse des Hauptspeichers angepasst werden muss.

Um dies zu verhindern, werden verschiedene Partitionierungsverfahren eingesetzt. Das Bild zeigt drei Arten von derartigen Partitionierungsverfahren (gemäss unserem Lehrbuch), welche wir im Folgenden genauer betrachten werden.

Classic Hash-Join

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Hashtabelle für r wird so gross angelegt, dass diese im Hauptspeicher Platz hat.
- Dadurch muss r allenfalls partitioniert werden und s für jede Partition vollständig gelesen werden.
- Aufwand: $O(b_r + p \cdot b_s)$
  - b_r = Anzahl von Blöcke dieTupel von r beinhalten
  - p = Anzahl Partitionen von r = Anzahl der Scans über s

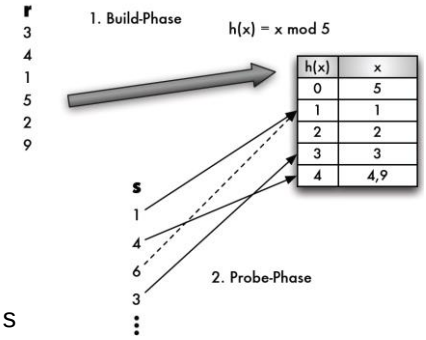
8

Beim Classic Hash-Join wird die kleinere Relation so partitioniert, dass die einzelne Partition jeweils gerade in die Hashtabelle aufgenommen werden kann (d.h. es werden so viele Tupel der Relation r gelesen, bis die Hashtabelle voll ist). Danach werden diese mit den Tupeln aus s verbunden. Anschliessend wird die Hashtabelle wieder geleert und die verbleibenden Tupel aus r in die Hashtabelle eingelesen. Dies wird so lange wiederholt, bis alle Tupel aus r verarbeitet wurden.

Die Relation r wird also genau einmal gelesen, während die Relation s so oft gelesen wird, wie r an Anzahl Partitionen unterteilt wird (allfällige Schreib-Operationen wurden vernachlässigt).

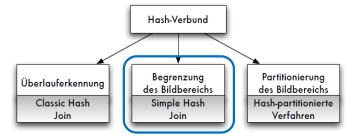
Classic Hash-Join

- Ablauf:
  1. Build-Phase (Scan über r)
     Tupel mittels Hashfunktion h(r.A)
     in Hashtabelle H einordnen.
  2. Probe-Phase (Scan über s)
     Wenn H voll oder vollständig gelesen:
     Scan über s und mit h(s.B)
     Verbundpartner suchen.
  3. Falls Scan über r nicht abgeschlossen:
     H neu aufbauen und erneuten Scan über s



Simple Hash-Join

- Verbesserung Classic Hash-Join:
  Hash-Adressbereich wird so gewählt, dass alle Tupel aus r in der Hashtabelle h Platz finden. Dafür muss h in mehrere Bildbereiche unterteilt werden.
  - Ablauf:
    1. Build-Phase:
       Scan über r: Tupel von r mittels Hashfunktion h(r.A) in aktuellen, begrenzten Bildbereich der Hashtabelle H ablegen. Tupel ausserhalb Bildbereich in Überlaufbereich ablegen.
    2. Probe-Phase:
       Scan über s: Überprüfung mit h(s.B) ob das Tupel im Bildbereich und Verbundpartner suchen. Tupel ausserhalb Bildbereich in Überlaufbereich ablegen.
    3. Hashtabelle h löschen, Bildbereich ändern und Verfahren mit Tupeln aus Überlaufbereich wiederholen.



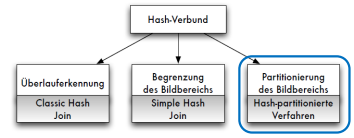
10

Beim Classic Join muss die grössere Tabelle s mehrfach gelesen werden. Um die Anzahl der Lese-Operation zu verringern, soll im Simple Hash-Join nicht nur r sondern auch s partitioniert werden. Wir wählen dazu die Hashfunktion h unabhängig vom verfügbaren Speicherplatz so, dass alle Tupel aus r in der Hashtabelle Platz finden. Da jetzt die Hashtabelle nicht mehr im Hauptspeicher Platz hat, muss diese partitioniert werden (und damit auch r und s), sie wird in mehrere Bildbereiche unterteilt (z.B. in 0-10, 11-20 und 21-26).

Die Verarbeitung erfolgt jetzt entlang den Partitionen (den unterschiedlichen Bildbereichen) der Hashtabelle. Im ersten Schritt werden die Tupel des erste Bildbereichs verarbeitet und verbunden. Tupel aus r uns s die nicht in diesen Bereich fallen, werden in einem Überlaufbereich (einen für r und einen für s) gespeichert. Diese temporären Relationen werden im nachfolgenden Schritt mit dem nächsten Bildbereich verarbeitet, usw.

Hash-partitionierender Join

- Verbesserung Simple Hash-Join:
  Mehrfaches Lesen wird vermieden, indem zu Beginn die Relationen mit einer Hashfunktion h auf dem Externspeicher partitioniert werden.
  - Ablauf:
    1. Partitionierung:
       Die Relationen r und s werden mittels h_1 auf dem Externspeicher in Partitionen r_i und s_i eingeordnet.
    2. Für jede Partition i werden folgende Schritte ausgeführt:
       1. Build-Phase:
          Scan über r_i: Tupel von r_i mittels Hashfunktion $h_2(r.A)$ in Hashtabelle H ablegen.
       2. Probe-Phase:
          Scan über s_i: Mit $h_2(s.B)$ Verbundpartner suchen.



11

Der Nachteil des Simple Hash-Join (vorherige Folie) ist offensichtlich. Ab dem zweiten Schritt werden Tupel wiederholt gelesen und geschrieben. Je mehr Partitionen für die Hashtabelle gebildet werden müssen, desto mehr Lese- und Schreiboperationen.

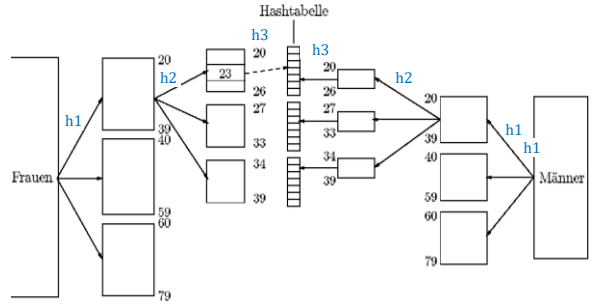
Die Hash-partitionierenden Verfahren vermeiden dies, indem die Phase der Partitionierung (von r und s) und deren Verbund getrennt durchgeführt werden, indem zwei Mal gehashed wird. Für die Partitionierung auf dem Externspeicher und für den anschliessenden Verbund im Hauptspeicher werden dann auch zwei verschiedene Hashfunktionen h_1 und h_2 eingesetzt. Während h_1 Partitionen auf dem Externspeicher bildet, wird h_2 dann auf diesen Partitionen (von h_1) im Hauptspeicher für den eigentlichen Verbund angewendet.

Beim Partitionieren mit h1 werden mehrere Partitionen r_i und s_i von r und s gebildet. Da Verbundkandidaten nur innerhalb der selben Partitionen i vorliegen, können die zusammengehörigen Partitionen aus r und s unabhängig voneinander und damit parallel verarbeitet werden.

Unser Klassifizierungsbaum ist natürlich theoretischer Natur, konkrete Verfahren dieser Klasse sind der GRACE-Join und der Hybrid-Hash-Join (siehe Literatur).

Hash-partitionierender Join

- Beispiel: Frauen $\bowtie_{\text{Frauen.Alter = Männer.Alter}}$ Männer
  - Frauen sei grössere Relation ($\rightarrow$ Probe Input)
  - Hashfunktionen ordnen nach dem Alter
  - Im Beispiel 2 Partitionierungen h1 und h2 (damit Hashtabelle in den Hauptspeicher passt)



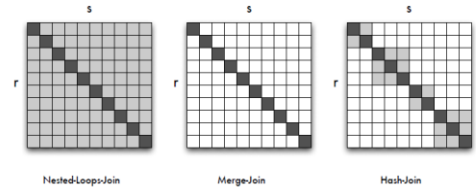
Das Bild zeigt einen Fall, bei dem nicht nur eine Hashfunktion h_1 zur Partitionierung auf dem Externspeicher angewendet wird, sondern bei dem eine zweite Hashfunktion h_2 eingesetzt wird. Dies ist dann notwendig, wenn die tatsächlich gebildeten Partitionen noch zu gross sind, als dass diese in der Hashtabelle Platz finden würden. Mittels h_3 erfolgt schliesslich die Abbildung in die Hashtabelle selbst.

Vergleich der Verbundoperationen

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Verfahren	Vorbedingung	Komplexität	Anwendbarkeit
Nested Loops	keine	$O(\|r\| \cdot \|s\|)$	alle Verbunde
Block Nested Loops	Relationen in Blöcken organisiert	$O(b_r \cdot b_s)$	
Merge	Relationen sortiert nach Verbundattributen	$O(b_r + b_s)$	Equi-Join
Sort-Merge	keine	$O(b_r \log_{mem} b_r + b_s \log_{mem} b_s)$	Equi-Join
Classic Hash	keine	$O(\|b_r\| + k \cdot \|b_s\|)$	Equi-Join

(Sort-)Merge kann auch bei Theta-Joins verwendet werden, wenn der Join Operator >, >=, <, <= ist.



13

In der Tabelle sind die wichtigsten Verbundoperationen aufgeführt. Der Nested Loop ist das einzige Verfahren, das nicht nur beim Equi-Join eingesetzt werden kann. In der Praxis treten Joins aber praktisch ausschliesslich als Equi-Joins auf (allenfalls ergänzt um Filterbedingungen).

Es ist schwierig, generelle Aussagen zu machen, welche Methode der SQL Server wann anwendet. Folgende Aussagen müssen daher vorsichtig genossen werden:

- Nested Loop wird angewendet wenn:
  1. Beide Relationen sehr klein (ca. 10 Tupel) sind
  2. Eine Relation sehr klein ist und die andere Relation über das Verbundattribut indexiert ist
- Merge: Beide Relationen über Verbundattribute indexiert
- Hash Join: In allen anderen Fällen

Bei der Datenmodellierung können mittels Denormalisierung (gezielte Einführung von Redundanzen ins Datenmodell, z.B. dem Zusammenlegen zweier Tabellen – z.B. Kunden und Adressen) Join-Operationen vermieden werden. Dies zum Preis der Redundanzen und deren Pflege. Grundsätzlich sollte versucht werden auf Denormalisierung zu verzichten.

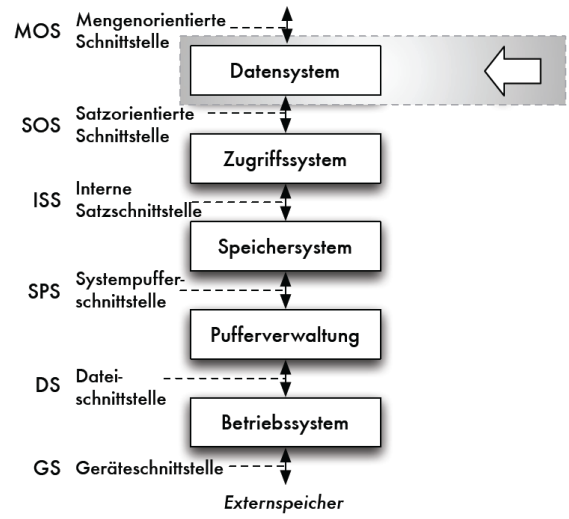
Vertikal partitionierte Tabellen (Attribute eines Tupels auf mehrere Tabellen verteilt), welche einen eindeutigen, geclusterten Primärindex haben, werden im SQL Server mittels Merge-Join zusammengeführt. Dies ist sehr performant (sehr geringer Overhead). Falls häufig nur wenige Attribute einer grossen Tabelle verwendet werden, ist die vertikale Partitionierung der Tabelle daher ein probates Mittel zur Performanceoptimierung!

Anfrageoptimierung: Motivation

School of Engineering
InIT Institut für angewandte Informationstechnologie

- Eingaben sind mengenorientierte Anfragen, die in die satzorientierte Schnittstelle des Zugriffssystems umgesetzt werden müssen
- Eine Hauptaufgabe ist die logische Optimierung der höheren Sprachkonstrukte

SQL ist deklarativ (nicht imperativ)



14

Ausgangslage der Optimierung:

1. Sehr hohes Abstraktionsniveau der mengenorientierten Schnittstelle (SQL)
2. SQL ist deklarativ, nicht-prozedural, d.h. es wird spezifiziert, was man finden möchte, aber nicht wie (z.B. welcher Index verwendet werden soll)
3. Das wie bestimmt sich aus der Abbildung der mengenorientierten Operatoren auf Schnittstellen-Operatoren der internen Ebene
4. Zu einem was kann es zahlreiche wies geben: Effiziente Anfrageauswertung durch Anfrageoptimierung erwünscht
5. Im Allgemeinen wird aber nicht die optimale Auswertungsstrategie gesucht (bzw. gefunden) sondern eine «genügend gute»

Anfrageoptimierung – Beispiel

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Ausgangslage «Verlagsdatenbank»:
  - Buchhandlung (Name, Ort, Einkaeufer) (15 Tupel)
  - Verkauf (Name, Ort, ISBN, Anzahl, JahrMonat) (45 Tupel)
  - Buch (ISBN, Titel, Typ, Auflage, Fachgebiet) (19 Tupel)
- Gesucht (Beispielanfrage):
  - Alle Einkäufer (Buchhandlung.Einkaeufer) derjenigen Aachener (Buchhandlung.Ort) Buchhandlungen, die im Dezember 2014 (Verkauf.JahrMonat) Bücher zu 'Datenbanken' (Buch.Fachgebiet) verkauft haben.

15

Das Beispiel auf den folgenden Folien soll exemplarisch zeigen, wie mittels logischer Optimierung aus einer «normalen» SQL-Anfrage eine effizientere Anfrage erstellt werden kann.

Anfrageoptimierung – Beispiel

- Lösung mit SQL:

```
SELECT DISTINCT Einkaeufer
FROM Buchhandlung bh, Verkauf v, Buch b
WHERE bh.Name = v.Name AND bh.Ort = v.Ort AND v.ISBN = b.ISBN
AND v.JahrMonat = '1412'
AND bh.Ort = 'Aachen'
AND b.Fachgebiet = 'Datenbanken'
```

Alle Einkäufer derjenigen Aachener (Ort) Buchhandlungen, die im Dezember 2014 (JahrMonat) Bücher zu 'Datenbanken' (Fachgebiet) verkauft haben.

- (Kanonische) Übersetzung in Relationenalgebra:
  - $\pi_{\text{Einkaeufer}}(\sigma_{\text{v.JahrMonat = '1412'} \wedge \text{bh.Ort = 'Aachen'} \wedge \text{b.Fachgebiet = 'Datenbanken'} \wedge \text{bh.name = v.name} \wedge \text{bh.ort = v.ort} \wedge \text{v.isbn = b.isbn}}(\text{Buchhandlung} \times \text{Verkauf} \times \text{Buch}))$

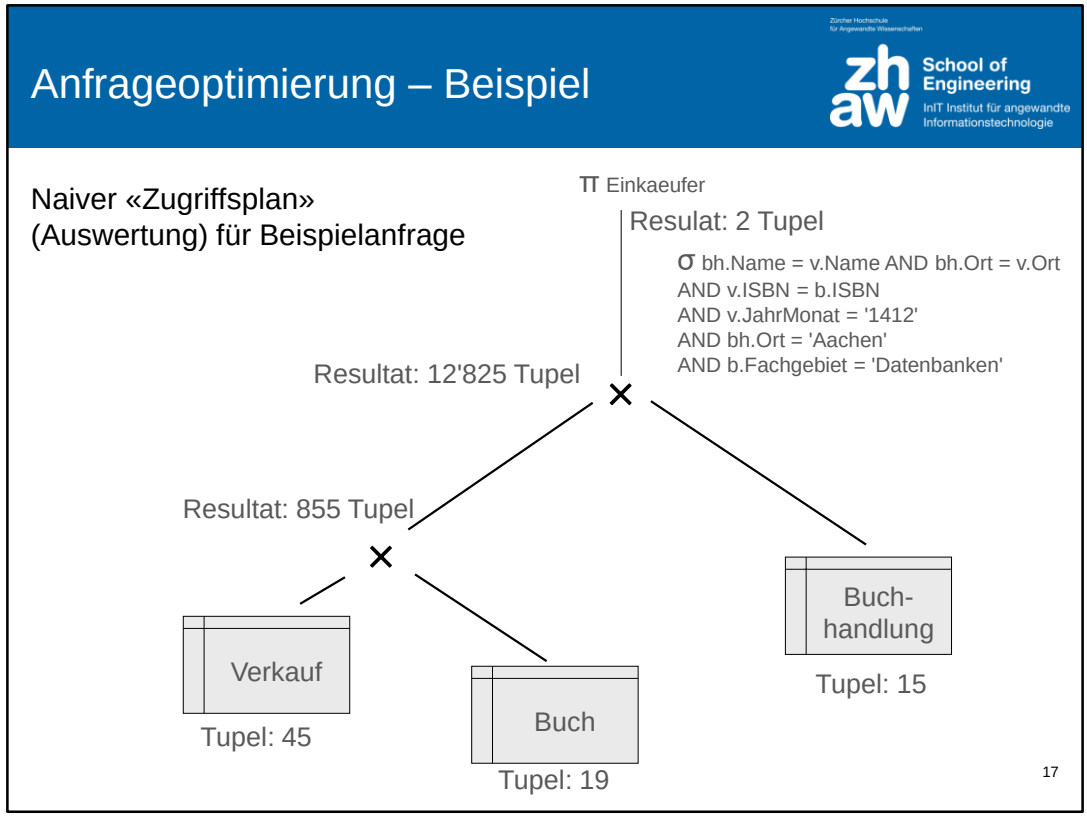
Join in obigem SQL ist nicht explizit spezifiziert (θ-Join)

Kreuzprodukt über alle Tabellen

16

Es gibt für SQL eine Standardübersetzung (kanonische Übersetzung) in Relationenalgebra. D.h., es gibt ein einfaches Verfahren, das SQL in standardisierte, relationen-algebraische Ausdrücke umwandelt. In der nächsten Lektion werden wir dieses Verfahren zeigen: wie wir ausgehend von SQL, über den Parse-Baum zum Operator-Baum kommen, der die relationale Algebra grafisch darstellt.

Wir betrachten diese Übersetzung von SQL zu rel. Algebra erst später, da die Übersetzung für unsere betrachteten Beispiele auf den folgenden Folien trivial ist. Die nächste Folie zeigt den Algebra-Ausdruck dieser Folie als ‘vereinfachten’ Operator-Baum.



Die Hochrechnung ist für das Kreuzprodukt gerechnet (nicht für einen Join). Beim Erstellen des Kreuzproduktes von Verkauf und Buch resultieren 855 Tupel. Eine Ebene höher werden 12’825 Tupel gebildet (855 x 15 = 12’825). Es werden daher in der anschliessenden Selektion 12’825 Tupel verarbeitet. Schliesslich wird nur eine kleine Teilmenge das Resultat bilden - im Beispiel wird angenommen, dass dies 2 Tupel sind.

Anfrageoptimierung – Beispiel

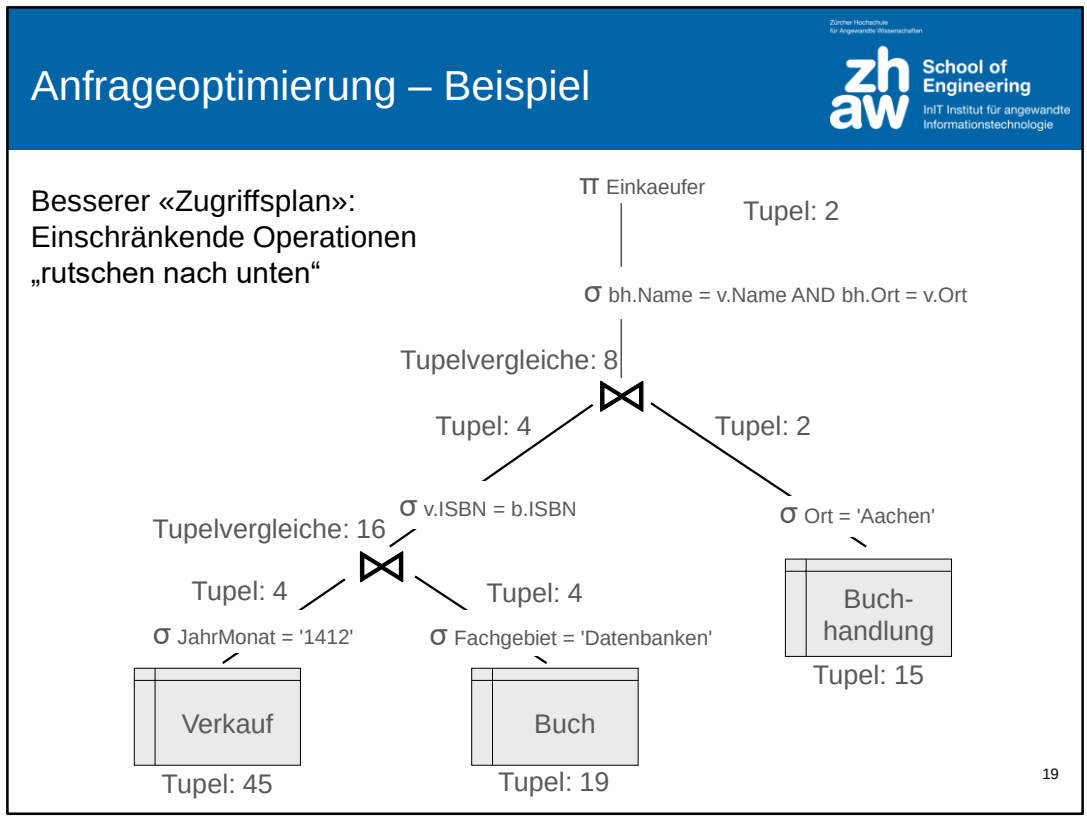
- Bessere Lösung (ohne Kreuzprodukt über alle Tabellen):

```
SELECT DISTINCT Einkaeufer
FROM Buchhandlung bh
NATURAL JOIN Verkauf v
NATURAL JOIN Buch b
WHERE v.JahrMonat = '1412'
AND bh.Ort = 'Aachen'
AND b.Fachgebiet = 'Datenbanken'
```

Alle Einkäufer derjenigen Aachener (Ort) Buchhandlungen, die im Dezember 2014 (JahrMonat) Bücher zu 'Datenbanken' (Fachgebiet) verkauft haben.

- (Kanonische) Übersetzung in Relationenalgebra:
  - $\pi_{\text{Einkaeufer}}(\sigma_{\text{v.JahrMonat = '1412'} \wedge \text{bh.Ort = 'Aachen'} \wedge \text{b.Fachgebiet = 'Datenbanken'}}$ (Buchhandlung $\bowtie$ Verkauf $\bowtie$ Buch))

Natural Join



Dieser Zugriffsplan entspricht nicht exakt dem SQL Statement auf der vorhergehenden Folie. Die Selektions-Operationen werden in diesem Zugriffplan ja vor dem jeweiligen Join ausgeführt. Das korrespondierende SQL Statement lautet daher eigentlich:

```
SELECT DISTINCT Einkaeufer
FROM (SELECT * FROM Buchhandlung WHERE Ort = 'Achen')
NATURAL JOIN Verkauf ON Verkauf.JahrMonat = '1412'
NATURAL JOIN Buch ON Buch.Fachgebiet = 'Datenbanken'
```

Die hinter der Join-Operation aufgeführten Join-Bedingungen werden vor der Join-Operation angewendet, während die Where-Bedingungen am Ende des SQL-Statements (vorhergehende Folie) *nach* allen Join-Operationen ausgeführt werden. Dies ist vor allem für Outer-Joins von Bedeutung, da hier für die beiden SQL-Statements die Resultate nicht die selben sind – und natürlich für die Performance!

Logische Optimierung – Grundprinzipien

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Ziel: Möglichst schnelle Anfragebearbeitung (d.h. möglichst wenig Seitenzugriffe bei der Anfragebearbeitung)

- Wie kann das erreicht werden? Überlegungen:
  - Möglichst kleine Zwischenergebnisse (damit sie im Hauptspeicher Platz haben) → Selektionen so früh wie möglich ausführen.
  - Basisoperationen wenn möglich zusammenfassen (Selektion/Projektion) und ohne Zwischenspeicherung von Zwischenergebnissen als ein Berechnungsschritt realisieren.
  - Nur Berechnungen ausführen, die auch einen Beitrag zum Gesamtergebnis liefern. Redundante Operationen oder nachweisbar leere Zwischenrelationen aus Berechnungen entfernen.
  - Zusammenfassen gleicher Teilausdrücke (Wiederverwendung von Zwischenergebnissen).
  - Eigenschaften der relationalen Algebra ausnutzen.

20

Optimierung: Ablauf

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Der Ablauf einer einfachen, **logischen Optimierung** ist z.B. (Details kommen in der nächsten Lektion):

1. SQL in Parse-Baum umwandeln
2. Semantische Analyse
3. Parse-Baum in Operator-Baum umwandeln (Bag-Algebra)
4. Auflösen von Sichten (Views)
5. Standardisierung von Selektions- und Verbundbedingungen (konjunktive Normalform - KNF)
6. Vereinfachung / Transformation (inkl. Entschachtelung)

Anschliessend an die logische Optimierung folgt die **physische Optimierung**: Es werden ausführbare Pläne (Execution Plan) erstellt (mit verschiedenen Basisoperationen) und nach einer Kostenabschätzung der «optimale» Plan gewählt.

21

Eigenschaften der Relationenalgebra

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Äquivalenzbeziehungen:
  - $p_1 \wedge p_2 \Leftrightarrow p_2 \wedge p_1$
  - $p_1 \vee p_2 \Leftrightarrow p_2 \vee p_1$
  - $p_1 \wedge (p_2 \wedge p_3) \Leftrightarrow (p_1 \wedge p_2) \wedge p_3$
  - $p_1 \vee (p_2 \vee p_3) \Leftrightarrow (p_1 \vee p_2) \vee p_3$
  - $p_1 \vee (p_2 \wedge p_3) \Leftrightarrow (p_1 \vee p_2) \wedge (p_1 \vee p_3)$
  - $\neg(\neg p_1) \Leftrightarrow p_1$
  - …
- Idempotenzen:
  - $A \vee A \Leftrightarrow A$
  - $A \wedge A \Leftrightarrow A$
  - $A \vee \neg A \Leftrightarrow$ true
  - $A \wedge \neg A \Leftrightarrow$ false
  - …

22

Die folgenden Folien greifen etwas vor, um die nächste Lektion vorzubereiten. Sie zeigen die Regeln mittels welchen rel. Ausdrücke standardisiert und anschliessend vereinfacht werden.

Um die logische Optimierung zu vereinfachen ist es sinnvoll, die relationalen Ausdrücke in ein standardisiertes (kanonisches) Anfrageformat zu bringen. Dazu werden die logischen Bedingungen meist in konjunktive Normalform (KNF) gebracht (diese können einfach in Folgen von Selektionen zerlegt werden). Dies sind logische Ausdrücke in der Form:

$$(p_{11} \vee p_{12} \vee \ldots p_{1n}) \wedge \ldots \wedge (p_{m1} \vee p_{m2} \vee \ldots p_{mo})$$

Die Äquivalenzbeziehungen auf der Folie (Liste ist nicht vollständig) dienen dazu, Ausdrücke in KNF zu bringen.

Nach der Umwandlung des relationalen Ausdrucks in KNF kann nun versucht werden, die Ausdrücke weiter zu vereinfachen und überflüssige Terme zu eliminieren. Hierzu können die weiteren Eigenschaften der Relationenalgebra genutzt werden. Wir werden in der nächsten Lektion auf diese Mechanismen eingehen und dabei auch die Entschachtelung (als ein Schritt der Vereinfachung) von Anfragen betrachten.

Eigenschaften der Relationenalgebra

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Verbund, Vereinigung, Durchschnitt und Produkt sind kommutativ:
  - $R1 \bowtie R2 = R2 \bowtie R1$
  - $R1 \cup R2 = R2 \cup R1$
  - $R1 \cap R2 = R2 \cap R1$
  - $R1 \times R2 = R2 \times R1$
- Selektionen sind untereinander vertauschbar:
  - $\sigma_P(\sigma_Q(R)) = \sigma_Q(\sigma_P(R))$
- Verbund, Vereinigung, Durchschnitt und Produkt sind assoziativ:
  - $R1 \bowtie (R2 \bowtie R3) = (R1 \bowtie R2) \bowtie R3$
  - $R1 \cup (R2 \cup R3) = (R1 \cup R2) \cup R3$
  - $R1 \cap (R2 \cap R3) = (R1 \cap R2) \cap R3$
  - $R1 \times (R2 \times R3) = (R1 \times R2) \times R3$

23

Eigenschaften der Relationenalgebra

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Konjunktionen in einer Selektionsbedingung können in mehrere Selektionen aufgebrochen werden:
  - $\sigma_{p1} \wedge \sigma_{p2} \wedge \ldots \wedge \sigma_{pn}\ (R)) = \sigma_{p1}(\sigma_{p2}(\ldots(\sigma_{pn}(R))\ldots))$
- Eine Projektion kann mit der Vereinigung vertauscht werden:
  - $\pi_L(R1 \cup R2) = \pi_L(R1) \cup \pi_L(R2)$
- Eine Disjunktion kann in eine Konjunktion umgewandelt werden:
  - $\neg\ (P1 \vee P2) = \neg\ P1 \wedge \neg\ P2$
  - $\neg\ (P1 \wedge P2) = \neg\ P1 \vee \neg\ P2$
- … und viele weitere.

24

Ausblick

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

- Das nächste Mal: Optimierung (Fortsetzung)
- Lesen: Kapitel 8.4

25