
Datenbanken

Anfrageoptimierung

Dr. Özgür Özçep

Prof. Dr. Ralf Möller

Universität zu Lübeck

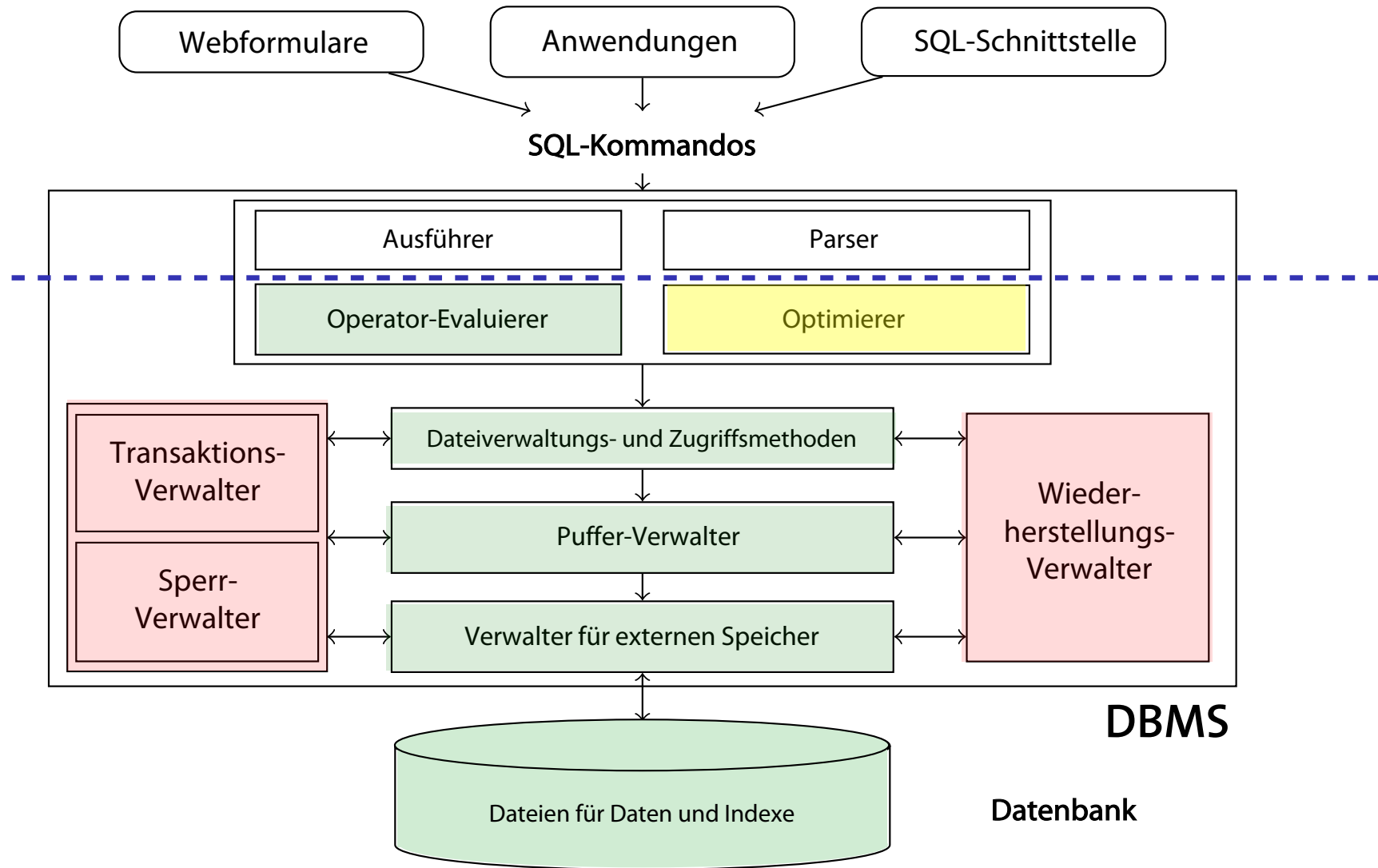
Institut für Informationssysteme

Felix Kuhr (Übungen)

und studentische Tutoren



Anfrageoptimierung

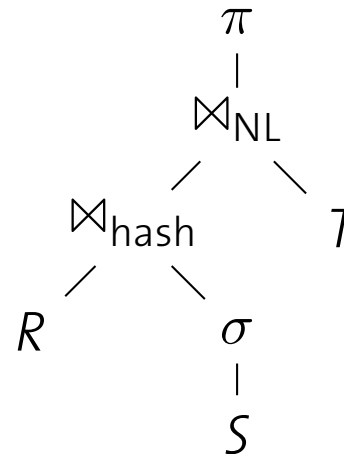
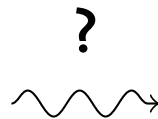


Danksagung

- Diese Vorlesung ist inspiriert von den Präsentationen zu dem Kurs:
„Architecture and Implementation of Database Systems“ von Jens Teubner an der ETH Zürich
- Graphiken und Code-Bestandteile wurden mit Zustimmung des Autors (und ggf. kleinen Änderungen) aus diesem Kurs übernommen
- Einige Inhalte sind angelehnt den Kurs „Architecture and Implementation of Database Systems, Summer 14“ von Torsten Grust, Uni Tübingen

Anfrageoptimierung

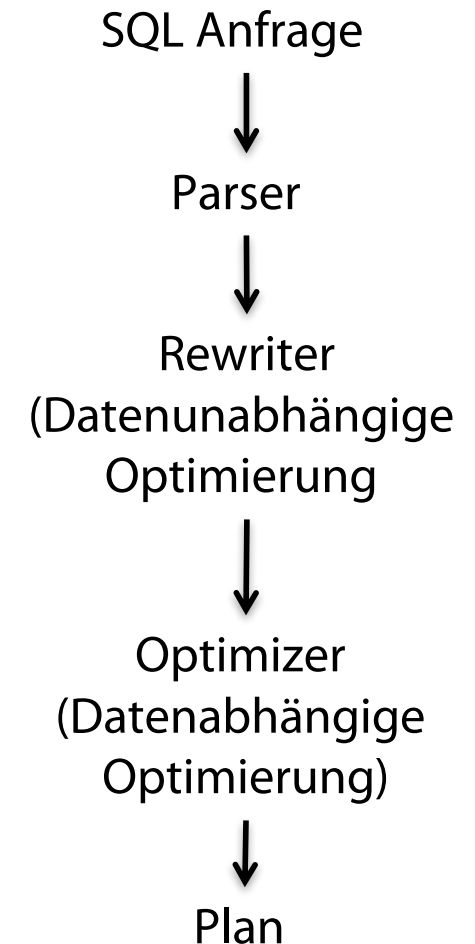
SELECT ...
FROM ...
WHERE ...



- Es gibt mehr als eine Art, eine Anfrage zu beantworten
 - Welche Implementation eines Verbundoperators?
 - Welche Parameter für Blockgrößen, Pufferallokation, ...
 - Automatisch einen Index aufsetzen?
- Die Aufgabe, den besten Ausführungsplan zu finden, ist der **heilige Gral** der Datenbankimplementierung

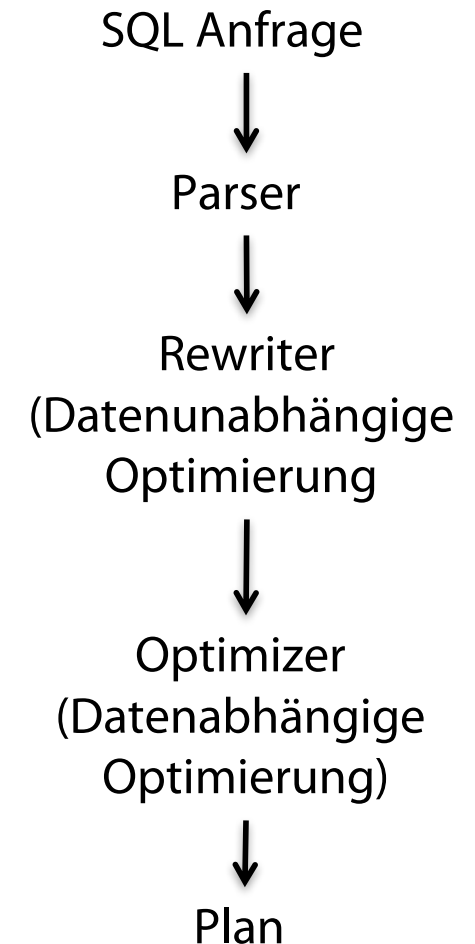
Optimierung

- Optimierungen können unabhängig von den Daten erfolgen; man spricht dann auch vom Umschreiben (**Rewriting**)
 - Selektionsprädikate früh anwenden
 - Vermeide Duplikatenelimination, wenn möglich
 - ...
- Datenabhängige Optimierung (Optimizer)
 - Kostenbasiert auf Basis der Daten in der DB bzw. statistisch relevanter Größen der DB



Optimierung

- Rewriting könnte z.B. Repräsentation von SQL-Anfragen in Datalog verwenden, um die Ideen aus dem Bereich des automatisierten Schließens nutzen (Datalog! Bzw. Logik i.A. ist wichtig)
- Hier nicht näher besprochen
 - Minimierung einer Anfrage durch Elimination einer Unteranfrage
 - Elimination eines teuren Operators
 - Bestimmung relevanter Tabellen



M. Benedikt: How Can Reasoners Simplify Database Querying (And Why Haven't They Done It Yet)? (Invited Talk at PODS 2018.)

Auswirkungen auf die Performanz

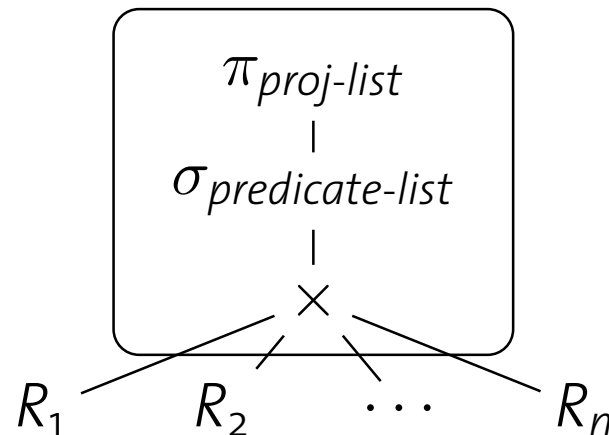
```
SELECT L.L_PARTKEY, L.L_QUANTITY, L.L_EXTENDEDPRICE
FROM LINEITEM L, ORDERS O, CUSTOMER C
WHERE L.L_ORDERKEY = O.O_ORDERKEY
        AND O.O_CUSTKEY = C.C_CUSTKEY
        AND C.C_NAME = 'IBM Corp.'
```



- Bezogen auf die Ausführungszeit können die Unterschiede „Sekunden vs. Tage“ bedeuten

Abschätzung der Ergebnisgröße

Betrachte Anfrageblock für Select-From-Where-Anfrage Q



Abschätzung der Ergebnisgröße von Q durch

- die Größe der Eingabetabellen $|R_1|, |R_2|, \dots, |R_n|$ und
- die **Selektivität** $sel(predicate-list)$

$$|Q| = |R_1| \cdot |R_2| \cdot \dots \cdot |R_n| \cdot sel(predicate-list)$$

Tabellenkardinalitäten

- Die Größe einer Tabelle ist über den Systemkatalog verfügbar (hier IBM DB2)
- Vor Ausführung der Anfrage verfügbar: offline (bei DB-Änderungen wird Tabelle upgedatet)

```
db2 => SELECT TABNAME, CARD, NPAGES
db2 (cont.) => FROM SYSCAT.TABLES
db2 (cont.) => WHERE TABSCHEMA = 'TPCH';
```

| TABNAME | CARD | NPAGES |
|----------|---------|--------|
| ORDERS | 1500000 | 44331 |
| CUSTOMER | 150000 | 6747 |
| NATION | 25 | 2 |
| REGION | 5 | 1 |
| PART | 200000 | 7578 |
| SUPPLIER | 10000 | 406 |
| PARTSUPP | 800000 | 31679 |
| LINEITEM | 6001215 | 207888 |

8 record(s) selected.

Grobe Abschätzung der Selektivität

... durch Induktion über die Struktur des Anfrageblocks

column = value

$$sel(\cdot) = \begin{cases} 1/|I| & \text{falls es einen Index } I \text{ auf Spalte } column \text{ gibt} \\ 1/10 & \text{sonst} \end{cases}$$

|I| gibt Aufschluss über die Anzahl der verschiedenen Werte

column₁ = column₂

$$sel(\cdot) = \begin{cases} \frac{1}{\max\{|I_1|, |I_2|\}} & \text{falls es Indexe auf beiden Spalten gibt} \\ \frac{1}{|I_k|} & \text{falls es einen Index nur auf eine Spalte gibt} \\ 1/10 & \text{sonst} \end{cases}$$

p₁ AND p₂

$$sel(\cdot) = sel(p_1) \cdot sel(p_2)$$

p₁ OR p₂

$$sel(\cdot) = sel(p_1) + sel(p_2) - sel(p_1) \cdot sel(p_2)$$

Verbesserung der Selektivitätsabschätzung

- Annahmen
 - Gleichverteilung der Datenwerte in einer Spalte
 - Unabhängigkeit zwischen einzelnen Prädikaten
- Annahmen nicht immer gerechtfertigt
- Sammlung von Datenstatistiken (offline)
 - Speicherung im Systemkatalog
 - IBM DB2: RUNSTATS ON TABLE
 - Meistverwendet: Histogramme

Aufgabe:

Wo wird in den groben Abschätzungen die Gleichverteilung (uniformity) genutzt? Wo die Unabhängigkeit (independence)?

Aufgabe:

Wo wird in den groben Abschätzungen die Gleichverteilung (uniformity) genutzt? Wo die Unabhängigkeit (independence)?

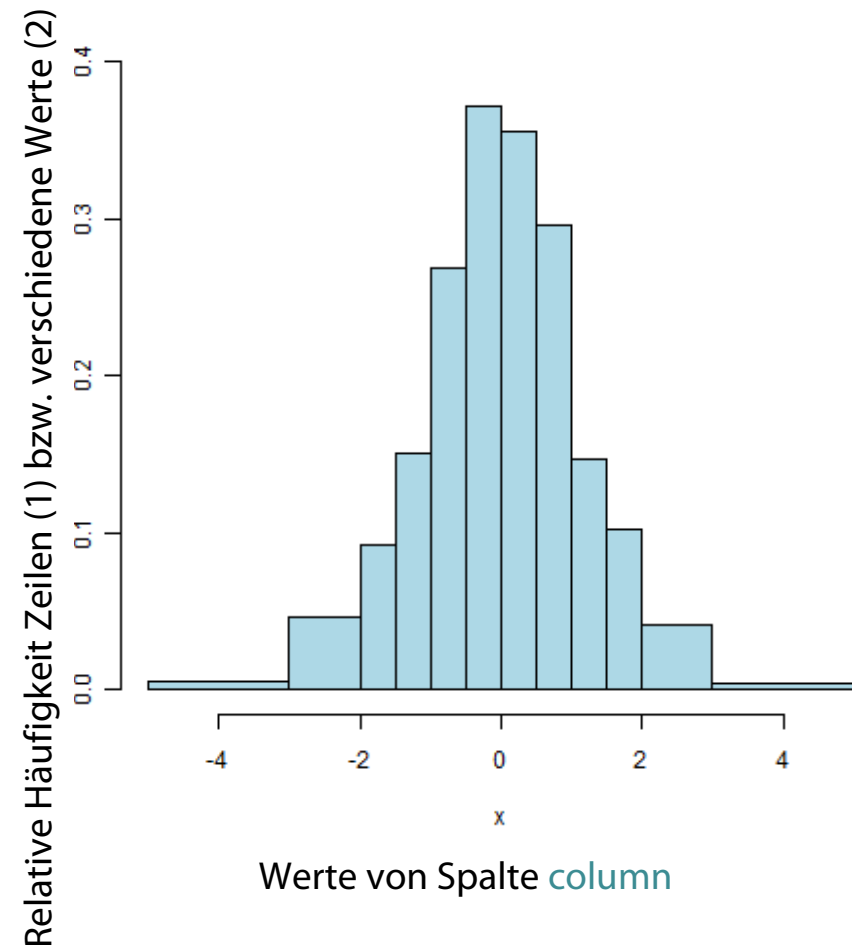
Lösung:

Die Annahme der Gleichverteilung ist beispielsweise für die Selektivitätsabschätzung für $\text{column} = \text{value}$ nötig, falls value nicht gelistet ist unter den häufigsten Elementen: unabhängig von value wird sie mit $1/|I|$ bzw. $1/10$ angenommen.

Unabhängigkeit: Unabhängigkeit wird für die Konjunktion von Prädikaten angenommen (keine "bedingten Häufigkeiten"). Eine Konjunktion steckt auch implizit in $\text{column1} = \text{column2}$ (= Es gibt c mit $\text{column1} = c$ und $c = \text{column2}$; hier wird außerdem angenommen, dass es zu jedem Wert der Kolumne mit der geringeren Anzahl an verschiedenen Werten einen korrespondierenden Wert der anderen Kolumne gibt.)

Histogramme

- Mit Histogrammen können echte Verteilungen von Werten einer Kolumne `column` approximiert werden
- Alle Werte von `column` werden in angrenzende Intervalle geteilt mit Grenzwerten x_i
- Sammle statistische Parameter für jedes Intervall, z.B.
 1. Anzahl Zeilen z mit $x_{i-1} < z.column \leq x_i$
 2. Oder: Anzahl verschiedener Werte von `column` im Intervall $(x_{i-1}, x_i]$, absolut oder relativ)



Von MM-Stat - Eigenes Werk (Originaltext: selbst erstellt), CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=10459764>

Histogramme

```
SELECT SEQNO, COLVALUE, VALCOUNT
FROM SYSCAT.COLDIST
WHERE TABNAME = 'LINEITEM'
AND COLNAME = 'L_EXTENDEDPRICE'
AND TYPE = 'Q';
```

| SEQNO | COLVALUE | VALCOUNT |
|-------|-------------------|----------|
| 1 | +0000000000996.01 | 3001 |
| 2 | +0000000004513.26 | 315064 |
| 3 | +0000000007367.60 | 633128 |
| 4 | +0000000011861.82 | 948192 |
| 5 | +0000000015921.28 | 1263256 |
| 6 | +0000000019922.76 | 1578320 |
| 7 | +0000000024103.20 | 1896384 |
| 8 | +0000000027733.58 | 2211448 |
| 9 | +0000000031961.80 | 2526512 |
| 10 | +0000000035584.72 | 2841576 |
| 11 | +0000000039772.92 | 3159640 |
| 12 | +0000000043395.75 | 3474704 |
| 13 | +0000000047013.98 | 3789768 |

SYSCAT.COLDIST enthält Informationen wie

- n-häufigste Werte (und deren Anzahl)
- Auch Anzahl der verschiedenen Werte pro Histogramm-Rasterplatz anfragbar

Tatsächlich können Histogramme auch absichtlich gesetzt werden, um den Optimierer zu beeinflussen

Bessere Abschätzung der Selektivität

- $MCV(\text{column}, R) = n$ -häufigsten (top- n) Werte in der Kolumne column einer Tabelle R
- $MCF(\text{column}, R) =$ Häufigkeiten dieser Werte
- Verbesserte Abschätzung für $\text{column} = \text{value}$
 - $\text{column} = \text{value}$

$$\text{– sel() = } \begin{cases} 1 / MCF(R, \text{column})[\text{value}] & \text{falls } \text{value} \in MCV(R, \text{column}) \\ 1/|I| & \text{falls nicht } \text{value} \in MCV(R, \text{column}), \text{ es aber einen Index} \\ & \text{auf Attribut } \text{column} \text{ gibt} \\ 1/10 & \text{sonst} \end{cases}$$

Kardinalitätsabschätzung für Projektion

- Anfrage $Q : \pi_L(R)$ mit $L = (A_1, \dots, A_N)$ Liste von Spalten
- $V(A,R)$ = Anzahl verschiedener Werte von A in R

$$\bullet \quad |Q| = \begin{cases} V(A,R) \\ (= ||) & \text{falls } L = (A) \\ & \text{falls Index auf Spalte } A \text{)} \\ |R| & \text{falls Schlüsselkolumnen von } R \text{ in } L \\ |R| & \text{ohne Duplikateneliminierung} \\ \text{Min} (|R|, \prod_{A_i \in L} V(A_i,R)) & \text{sonst} \end{cases}$$

Aufgabe:

Schätzen Sie die Kardinalitäten für die Vereinigung (\cup), die Differenz (\setminus) und das kartesische Produkt (\times) ab.

Aufgabe:

Schätzen Sie die Kardinalitäten für die Vereinigung (\cup), die Differenz (\setminus) und das kartesische Produkt (\times) ab.

Lösung:

- $|R \cup S| \leq |R| + |S|$
- $\max(0, |R| - |S|) \leq |R \setminus S| \leq |R|$
- $|R \times S| = |R| * |S|$

Kardinalitätsabschätzung für Join

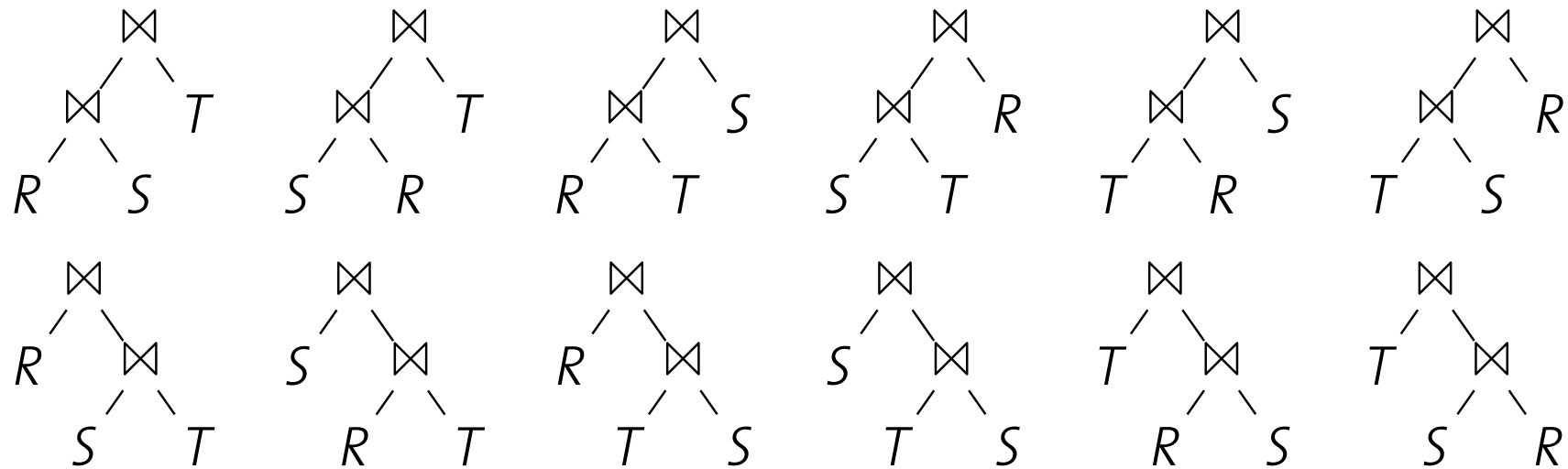
- Im Allgemeinen nicht-trivial
- Wenn Fremdschlüsselbeziehungen vorliegen (kommt häufig vor), wie folgt abschätzbar:

$$- |R \bowtie_{R.A=S.A} S| = |S| \text{ falls } S.A \text{ Fremdschlüssel auf } R.A$$

$$- |R \bowtie_{R.A=S.B} S| = \begin{cases} |R| * |S| / V(A,R) \text{ falls } S.B \text{ Fremdschlüssel auf } R.A \\ |R| * |S| / V(B,S) \text{ falls } R.A \text{ Fremdschlüssel auf } S.B \end{cases}$$

Verbund-Optimierung

Auflistung der möglichen Ausführungspläne, d.h. alle 3-Wege-Verbundkombinationen für jeden Block



Suchraum

- Der sich ergebende Suchraum ist enorm groß:
Schon bei 4 Relationen ergeben sich 120 Möglichkeiten

| <u>number of relations n</u> | <u>join trees</u> |
|---|-------------------|
| 2 | 2 |
| 3 | 12 |
| 4 | 120 |
| 5 | 1,680 |
| 6 | 30,240 |
| 7 | 665,280 |
| 8 | 17,297,280 |
| 10 | 17,643,225,600 |

Suchraum

- Der sich ergebende Suchraum ist enorm groß:
Schon bei 4 Relationen ergeben sich 120 Möglichkeiten

| number of relations n | join trees |
|-------------------------|----------------|
| 2 | 2 |
| 3 | 12 |
| 4 | 120 |
| 5 | 1,680 |
| 6 | 30,240 |
| 7 | 665,280 |
| 8 | 17,297,280 |
| 10 | 17,643,225,600 |

Anzahl der Bäume für $n + 1$ Inputrelationen:

$$C_n * (n+1)! = (2n)!/n!$$

$$C_n = \text{n-te Catalanzahl} = (2n)!/(n+1)!n!$$

Enzyklopädie von Sequenzen (oeis.org)

[login](#)

This site is supported by donations to [The OEIS Foundation](#).

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES®

founded in 1964 by N. J. A. Sloane

[The On-Line Encyclopedia of Integer Sequences® \(OEIS®\)](#)

Enter a sequence, word, or sequence number:

[Hints](#) [Welcome](#) [Video](#)

For more information about the Encyclopedia, see the [Welcome](#) page.

Languages: [English](#) [Shqip](#) [العربية](#) [Bangla](#) [Български](#) [Català](#) [中文 \(正體字, 简化字 \(1\), 简化字 \(2\)\)](#)
[Hrvatski](#) [Čeština](#) [Dansk](#) [Nederlands](#) [Esperanto](#) [Eesti](#) [فارسی](#) [Suomi](#) [Français](#) [Deutsch](#) [Ελληνικά](#) [ગુજરાતી](#) [עברית](#)
[हिंदी](#) [Magyar](#) [Ibo](#) [Bahasa Indonesia](#) [Italiano](#) [日本語](#) [ಕನ್ನಡ](#) [한국어](#) [Lietuvių](#) [मराठी](#) [Bokmål](#) [Nynorsk](#) [Polski](#) [Português](#)
[Română](#) [Русский](#) [Српски](#) [Slovenščina](#) [Español](#) [Svenska](#) [Tagalog](#) [தமிழ்](#) [Türkçe](#) [Українська](#) [العبرית](#) [Tiếng Việt](#) [Cymraeg](#)

[Lookup](#) | [Welcome](#) | [Wiki](#) | [Register](#) | [Music](#) | [Plot 2](#) | [Demos](#) | [Index](#) | [Browse](#) | [More](#) | [WebCam](#)
[Contribute new seq. or comment](#) | [Format](#) | [Style Sheet](#) | [Transforms](#) | [Superseeker](#) | [Recent](#) | [More pages](#)
[The OEIS Community](#) | Maintained by [The OEIS Foundation Inc.](#)

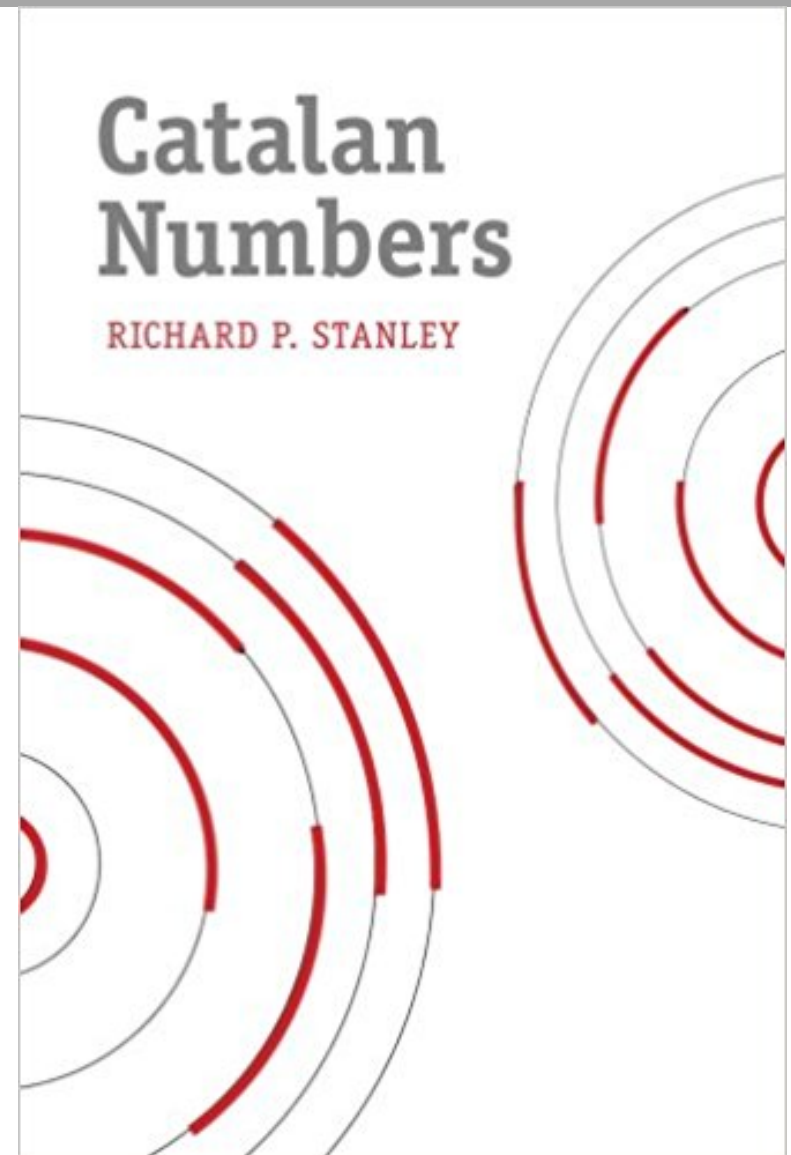
[License Agreements](#), [Terms of Use](#), [Privacy Policy](#).

Last modified June 22 08:29 EDT 2018. Contains 305669 sequences. (Running on oeis4.)



Ein ganzes Buch wert...

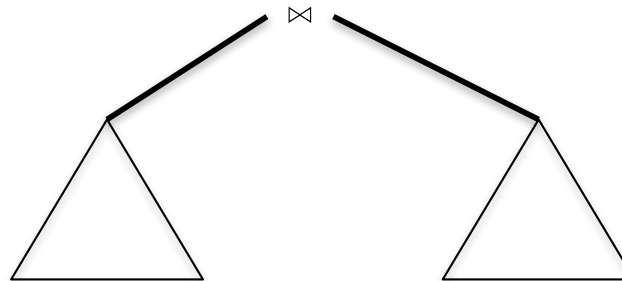
Catalanzahlen und ihre 214
kombinatorischen
Interpretationen



Herleitung

- Verbund über $n+1$ Relation benötigt n binäre Joins
- Die Wurzel eines jeden Plans bildet den Verbund von Sub-Plänen von k und $n-k-1$ Verbundoperatoren
($0 \leq k \leq n-1$)

k binäre Verbünde
 R_1, \dots, R_{k+1}



$n-k-1$ binäre Verbünde
 R_{k+2}, \dots, R_{n+1}

- $C_i =$ Anzahl der Möglichkeiten, einen binären Baum mit i inneren Knoten (binäre Verbünde) zu erstellen
- $C_n = \sum_{k=0}^{n-1} C_k * C_{n-k-1}$
- Rekurrenzgleichung mit Mittel der Analysis zu lösen

Suchraum

- Der sich ergebende Suchraum ist enorm groß:
Schon bei 4 Relationen ergeben sich 120 Möglichkeiten

| number of relations n | join trees |
|-------------------------|----------------|
| 2 | 2 |
| 3 | 12 |
| 4 | 120 |
| 5 | 1,680 |
| 6 | 30,240 |
| 7 | 665,280 |
| 8 | 17,297,280 |
| 10 | 17,643,225,600 |

- Noch nicht berücksichtigt: Anzahl v der verschiedenen Verbundalgorithmen (liefert zusätzlichen Faktor v^n)

Dynamische Programmierung

- Beispiel 4-Wege-Verbund
- Sammle gute Zugriffspläne für Einzelrelation (z.B. auch mit Indexscan und mit Ausnutzung von Ordnungen)
- Grundannahme: Optimalitätsprinzip
Um den global optimalen Plan zu finden, reicht es aus, die optimalen Pläne bzgl. der Unteranfragen zu betrachten.

P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie and T.G. Price, Access path selection in a relational database management system, In Proc. ACM SIGMOD Conf. on the Management of Data, pages 23-34, 1979

Beispiel: 4-Wege-Verbund

Pass 1 (best 1-relation plans)

Find the best **access path** to each of the R_i individually
(considers index scans, full table scans, and interesting orders (those required by GROUP BY or ORDER BY))

Pass 2 (best 2-relation plans)

For each **pair** of tables R_i and R_j , determine the best order to join R_i and R_j ($R_i \bowtie R_j$ or $R_j \bowtie R_i$):

$$\text{optPlan}(\{R_i, R_j\}) \leftarrow \text{best of } R_i \bowtie R_j \text{ and } R_j \bowtie R_i .$$

→ 12 plans to consider.

Pass 3 (best 3-relation plans)

For each **triple** of tables R_i , R_j , and R_k , determine the best three-table join plan, using sub-plans obtained so far:

$$\begin{aligned} \text{optPlan}(\{R_i, R_j, R_k\}) \leftarrow & \text{best of } R_i \bowtie \text{optPlan}(\{R_j, R_k\}), \\ & \text{optPlan}(\{R_j, R_k\}) \bowtie R_i, R_j \bowtie \text{optPlan}(\{R_i, R_k\}), \dots . \end{aligned}$$

→ 24 plans to consider.

Beispiel (Fortsetzung)

Pass 4 (best 4-relation plan)

For each set of **four** tables $R_i, R_j, R_k,$ and R_l , determine the best four-table join plan, using sub-plans obtained so far:

$$\begin{aligned} \text{optPlan}(\{R_i, R_j, R_k, R_l\}) \leftarrow & \text{best of } R_i \bowtie \text{optPlan}(\{R_j, R_k, R_l\}), \\ & \text{optPlan}(\{R_j, R_k, R_l\}) \bowtie R_i, R_j \bowtie \text{optPlan}(\{R_i, R_k, R_l\}), \dots, \\ & \text{optPlan}(\{R_i, R_j\}) \bowtie \text{optPlan}(\{R_k, R_l\}), \dots \end{aligned}$$

→ 14 plans to consider.

- Insgesamt: 50 (Unter-)Pläne betrachtet (anstelle von 120 möglichen 4-Wege-Verbundplänen, siehe Tabelle)
- Alle Entscheidungen basieren auf vorher bestimmten Unterplänen (keine Neuevaluierung von Subplänen: Nutzung einer Lookup Tabelle)

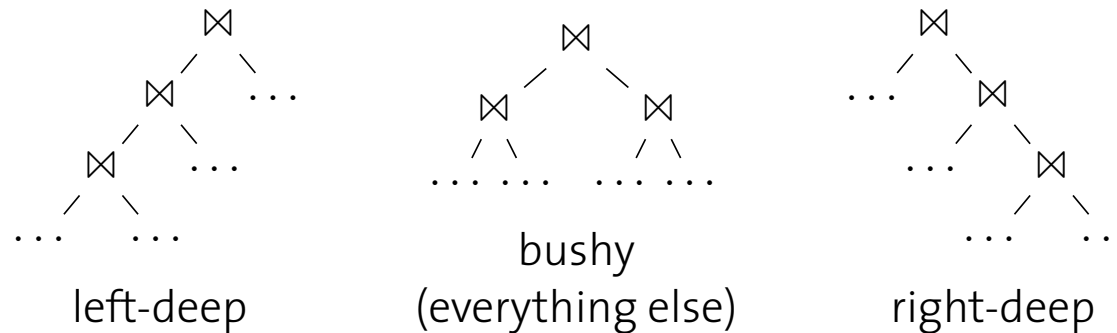
Optimaler n -Wege Verbundplan mit DP

```
Function: find_join_tree_dp (q(R1, ..., Rn))
for i=1 to n do
  optPlan({Ri}) ← access_plans (Ri);
  prune_plans( optPlan({Ri}) );
for i=2 to n do
  foreach S ⊆ {R1, ..., Rn} such that |S| = i do
    optPlan(S) ← ∅;
    foreach O ⊂ S with O ≠ ∅ do
      optPlan(S) ← optPlan(S) ∪
                    possible_joins( optPlan(O) ⋈ optPlan(S \ O) )
    prune_plans( optPlan(S) );
return optPlan({R1, ..., Rn});
```

- possible_joins zählt mögliche Joins zwischen Argumenten auf, z.B. nested loop join, merge join etc.

prune_plans behält nur den besten Plan

Links-tiefe vs. buschige Verbundplan-Bäume



Implementierte Systeme generieren meist links-tiefe Pläne¹

- Die innere Relation ist immer eine Basisrelation
- Index-Verbund lässt sich anwenden
- Gut geeignet für die Verwendung von Pipeline-Verarbeitungsprinzipien (s. folgende Aufgabe)

Aufgabe:

Warum wäre z.B. ein rechts-tiefer Verbundplan (Nested-Loop Join verwendend) nicht für eine Pipelinestrategie geeignet?

Erinnerung: Nested-loop join

Einfache Implementierung des Verbundes:

```
1 Function: nljoin (R, S, p)
2 foreach record r ∈ R do
3     foreach record s ∈ S do
4         if ⟨r, s⟩ satisfies p then
5             append ⟨r, s⟩ to result
```

Sei N_R und N_S die Seitenzahl in R und S , sei p_R und p_S die Anzahl der Datensätze pro Seite in R und S

Anzahl der Plattenzugriffe:

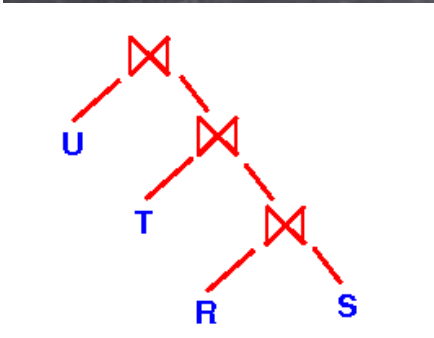
$$\frac{N_R + p_r \cdot N_R \cdot N_S}{\text{\#Tupel in } R}$$

Aufgabe:

Warum wäre z.B. ein rechts-tiefer Verbundplan (Nested-Loop Join verwendend) nicht für eine Pipelinestrategie geeignet?

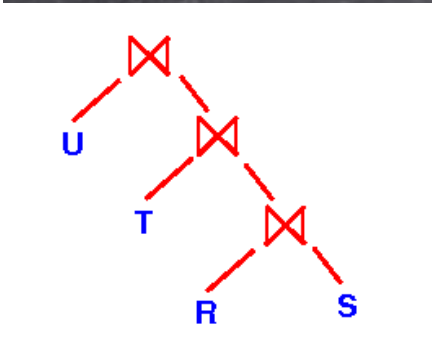
Lösung: Nested Loop Join liest das linke Argument nur einmal ein, das rechte Argument mehrmals. Beim rechts-tiefen Baum ist die rechte Seite eine Anfrage: Die Inputs dieser Anfrage würden also mehrmals gelesen. Das passt nicht zum Pipelining, da man die Tupel einmal liest und dann "vergisst".

Lösung (Forts.):

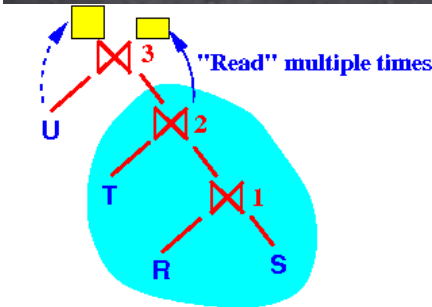


1. Rechts-tiefer Plan für
 $R \times S \times T \times U$

Lösung (Forts.):

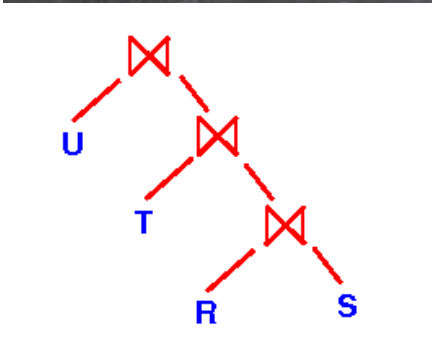


1. Rechts-tiefer Plan für $R \bowtie S \bowtie T \bowtie U$

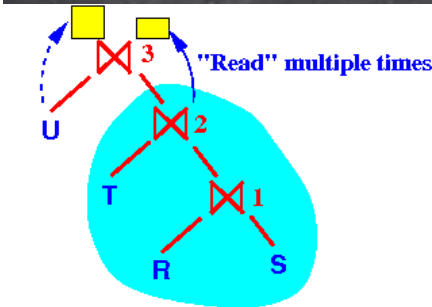


2. \bowtie_3 liest ein Tupel vom linken Argument U

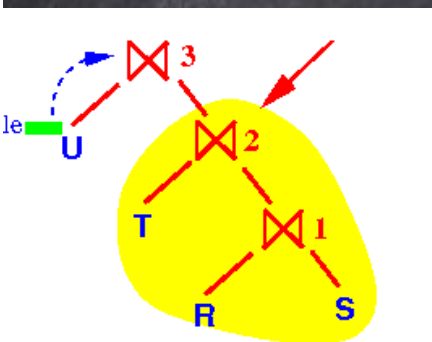
Lösung (Forts.):



1. Rechts-tiefer Plan für
 $R \bowtie S \bowtie T \bowtie U$

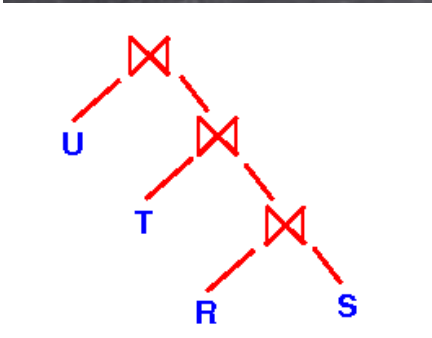


2. \bowtie_3 liest ein Tupel vom
linken Argument U

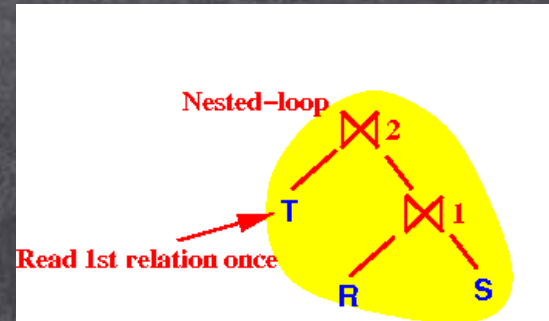


3. \bowtie_3 liest rechtes Argument
(alle Tupel) für das eine
linke Tupel

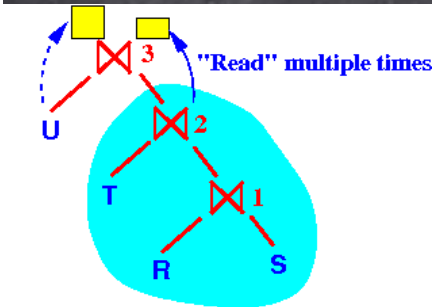
Lösung (Forts.):



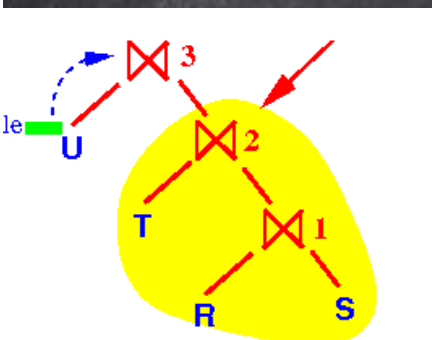
1. Rechts-tiefer Plan für $R \bowtie S \bowtie T \bowtie U$



4. Zur Auswertung dieser Unterfrage muss T (und R) Einmal vollständig gelesen werden

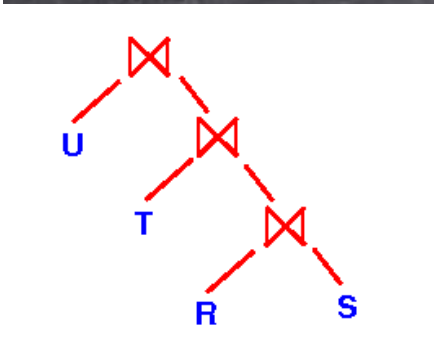


2. \bowtie_3 liest ein Tupel vom linken Argument U

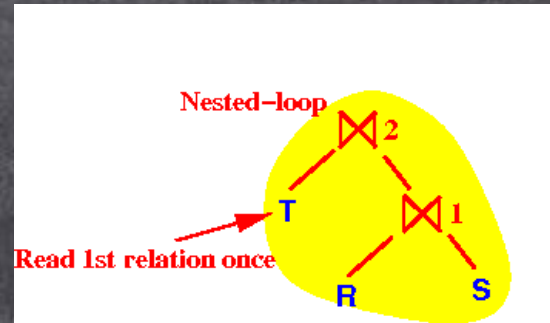


3. \bowtie_3 liest rechtes Argument (alle Tupel) für das eine linke Tupel

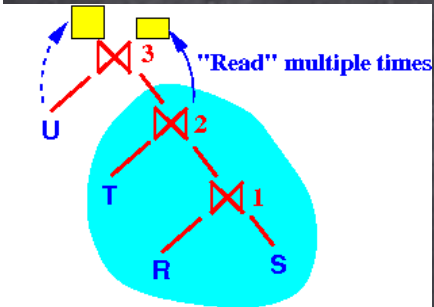
Lösung (Forts.):



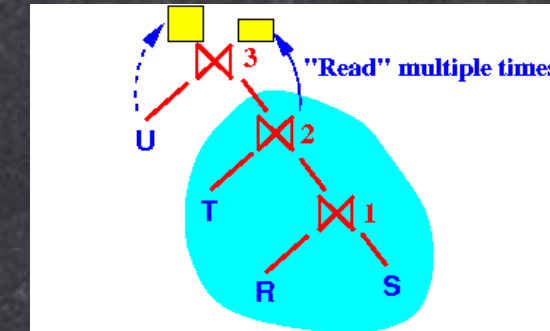
1. Rechts-tiefer Plan für $R \bowtie S \bowtie T \bowtie U$



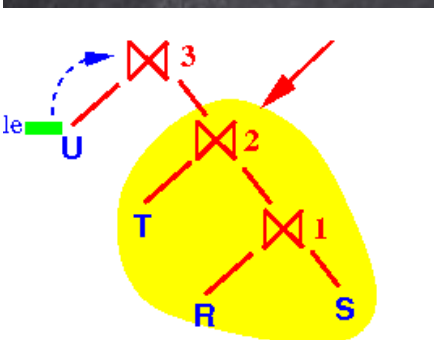
4. Zur Auswertung dieser Unterfrage muss T (und R) Einmal vollständig gelesen werden



2. \bowtie_3 liest ein Tupel vom linken Argument U

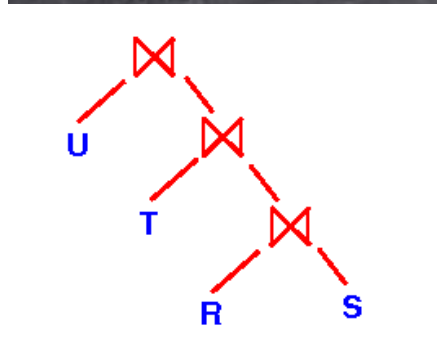


5. \bowtie_3 liest ein weiteres Tupel vom linken Argument U

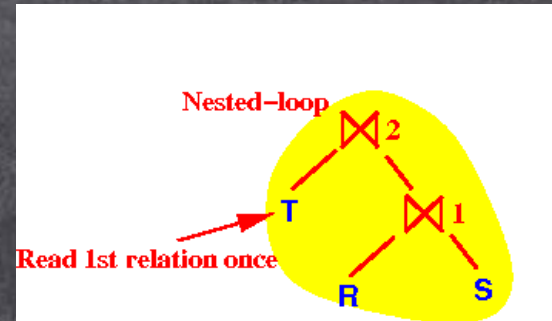


3. \bowtie_3 liest rechtes Argument (alle Tupel) für das eine linke Tupel

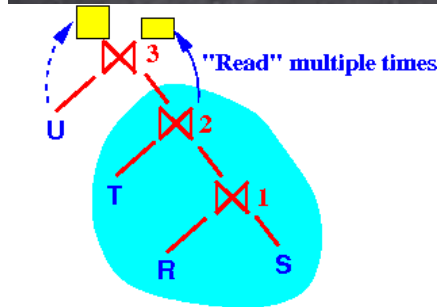
Lösung (Forts.):



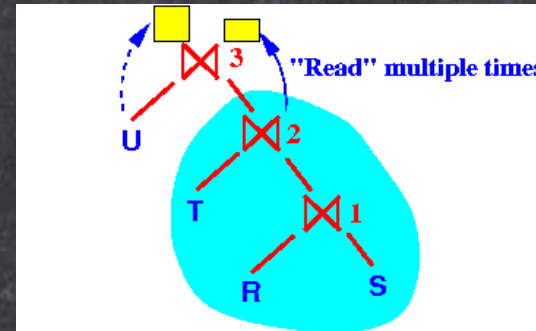
1. Rechts-tiefer Plan für $R \bowtie S \bowtie T \bowtie U$



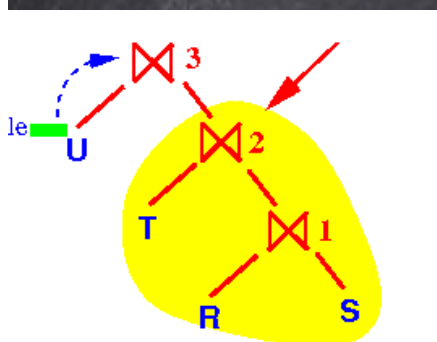
4. Zur Auswertung dieser Unterfrage muss T (und R) Einmal vollständig gelesen werden



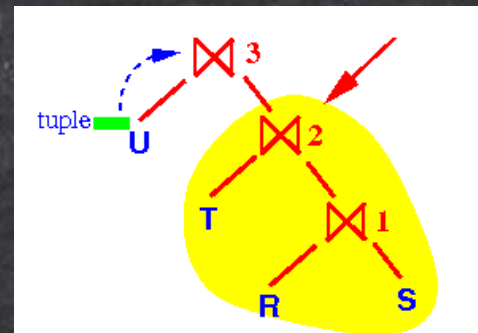
2. \bowtie_3 liest ein Tupel vom linken Argument U



5. \bowtie_3 liest ein weiteres Tupel Vom linken Argument U



3. \bowtie_3 liest rechtes Argument (alle Tupel) für das eine linke Tupel



6. Für das neue Tupel wieder vollständig Unteranfrage zu lesen: Damit aber muss T (und R) Vollständig gelesen werden. Aber T wurde schon „durchgelesen“ in Schritt 4.

Verbund über mehrere Relationen

- Dynamische Programmierung erzeugt in diesem Kontext exponentiellen Aufwand [Ono & Lohmann, 90]
 - Zeit: $O(3^n)$
 - Platz: $O(2^n)$
- Das kann zu teuer sein...
 - für Verbunde mit mehreren Relationen (10-20 und mehr)
 - für einfache Anfragen über gut-indizierte Daten, für die ein sehr guter Plan einfach zu finden wäre
- Neue Plangenerierungsstrategie:
Greedy-Join-Enumeration

Greedy-Join-Enumeration

```
1 Function: find_join_tree_greedy ( $q(R_1, \dots, R_n)$ )
2 worklist  $\leftarrow \emptyset$  ;
3 for  $i = 1$  to  $n$  do
4    $\lfloor$  worklist  $\leftarrow$  worklist  $\cup$  best_access_plan ( $R_i$ ) ;
5 for  $i = n$  downto 2 do
6    $\lfloor$  // worklist =  $\{P_1, \dots, P_i\}$ 
7    $\lfloor$  find  $P_j, P_k \in$  worklist and  $\bowtie \dots$  such that  $cost(P_j \bowtie \dots P_k)$  is minimal ;
8    $\lfloor$  worklist  $\leftarrow$  worklist  $\setminus \{P_j, P_k\} \cup \{(P_j \bowtie \dots P_k)\}$  ;
   // worklist =  $\{P_1\}$ 
8 return single plan left in worklist ;
```

- Wähle in jeder Iteration den kostengünstigsten Plan, der über den verbliebenen Unterplänen zu realisieren ist

Prädikatsvereinfachung (Rewriting)

Beispiel: Schreibe

Non-Sargable¹⁾

```
SELECT *  
FROM LINEITEM L  
WHERE L.L_TAX * 100 < 5
```

um in

Sargable

```
SELECT *  
FROM LINEITEM L  
WHERE L.L_TAX < 0.05
```

- Prädikatsvereinfachung ermöglicht Verwendung von Indexen und vereinfacht die Erkennung von effizienten Verbundimplementierungen

Zusätzliche Verbundprädikate

Implizite Verbundprädikate wie in

```
SELECT *  
FROM A, B, C  
WHERE A.a = B.b AND B.b = C.c
```

können explizit gemacht werden

```
SELECT *  
FROM A, B, C  
WHERE A.a = B.b AND B.b = C.c AND A.a = C.c
```

Hierdurch werden Pläne möglich wie $(A \bowtie C) \bowtie B$

Geschachtelte Anfragen

SQL bietet viele Wege, geschachtelte Anfrage zu schreiben

- Unkorrelierte Unteranfragen

```
SELECT *  
FROM ORDERS O  
WHERE O_CUSTKEY IN (SELECT C_CUSTKEY  
                     FROM CUSTOMER  
                     WHERE C_NAME = 'IBM Corp.')
```

- Korrelierte Unteranfragen

```
SELECT *  
FROM ORDERS O  
WHERE O_CUSTKEY IN (SELECT C.C_CUSTKEY  
                     FROM CUSTOMER  
                     WHERE C.C_ACCTBAL = O.O_TOTALPRICE)
```

Optimierung von geschachtelten Anfragen

- Bei unkorrelierten Anfragen muss die Unteranfrage nur einmal ausgewertet werden
- Meist sind Unteranfragen nur syntaktische Varianten von Joins
 - beim Rewriting werden die Joins explizit gemacht, so dass die Anfrage in der Join-Order-Optimierung genutzt werden kann

Schnitt von Indexstrukturen

- Gegeben: Tabelle R(A,B,C,D,E)
 - B-Baum-Index für A
 - B-Baum-Index für B
 - Kein zusammengesetzter Index

- Anfrage

```
SELECT Count(*)  
FROM R  
WHERE A=5 AND B<10
```

- Verwende **Indexschnitt** als Ausführungsplanoperator
 - Durchsuche Index für A nach A=5
 - Indexeinträge <A, RID>
 - 2-spaltige Tabelle I1 mit Schema I1(A, RID)
 - Durchsuche Index für B nach B<10
 - Indexeinträge <B, RID>
 - 2-spaltige Tabelle I2 mit Schema I2(B, RID)
 - Verbund I1 \bowtie I2

Schnitt von Indexstrukturen

- Funktioniert gut für Konjunktion mit **mittelselektiven** Filtern

```
SELECT Count (*)  
FROM R  
WHERE A=5 AND B<10
```

- Annahmen
 - 5% der Datensätze erfüllen $A=5$
 - 5% der Datensätze erfüllen $B<10$
 - $5\% * 5\% = 0,25\%$ der Datensätze erfüllen $A=5 \wedge B<10$
- Holen der vollen Datensätze bzgl. einzelner Indexe sehr langsam
- Mehraufwand für Index-Verbund hingegen gering

Bitmap-Index

- Vorher diskutiert: RID-Listen in B-Bäumen
 - Spart Platz bei mehrfach vorkommenden Werten
- Bitmap-Indexe führen die Idee weiter
 - Verwende Bitmap statt RID-Liste
 - Jeder mögliche Wert (RID) in der ganzen Relation wird durch ein Bit repräsentiert
 - 1 = RID kommt in RID-Liste vor
 - 0 = RID kommt nicht in RID-Liste vor
 - Bitmaps werden komprimiert (z.B. Lauflängenkodierung)
- Oracle:

```
CREATE BITMAP INDEX index_name  
ON tbl_name (index_col_name,...)
```

Bitmap-Index – Beispiel

<M,10100011>

<F,01011100>

| ID | Name | Sex |
|----|-------|-----|
| 1 | Fred | M |
| 2 | Jill | F |
| 3 | Joe | M |
| 4 | Fran | F |
| 5 | Ellen | F |
| 6 | Kate | F |
| 7 | Matt | M |
| 8 | Bob | M |

Anwendung von Bitmap-Indexstrukturen

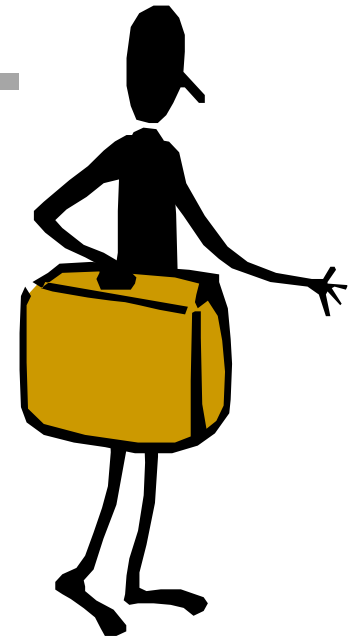
- Index-Schnitt-Ausführungspläne
 - Bitweises AND
 - Kein Verbund wie bei Index-Schnitt mit B-Bäumen
- Beispiel

```
SELECT Count(*)  
FROM R  
WHERE A=5 AND B<10
```

 - Bitmap-Indexe für A und B
 - OR für B-Bitmaps und Werte < 10
 - AND von Ergebnis mit Bitmap für A=5
- Vorteile im Platzverbrauch
- Geeignet für Attribute mit wenigen Werten
(nicht geeignet z.B. für Zeichenketten)

Zusammenfassung

- **Anfrageparser**
 - Übersetzung der Anfrage in Anfrageblock
- **Umschreiber (Rewriter)**
 - Logische Optimierung (unabhängig vom DB-Inhalt)
 - Prädikatsvereinfachung
 - Anfrageentschachtelung
- **Verbundoptimierung**
 - Bestimmung des „günstigsten“ Plan auf Basis
 - eines Kostenmodells (I/O-Kosten, CPU-Kosten) und
 - Statistiken (Histogramme) sowie
 - Physikalischen Planeigenschaften (interessante Ordnungen)
 - Dynamische Programmierung, Greedy-Join
- **Indexschnitt, Bitmap-Indexe**



Noch zu diskutieren...

