Datenbanken

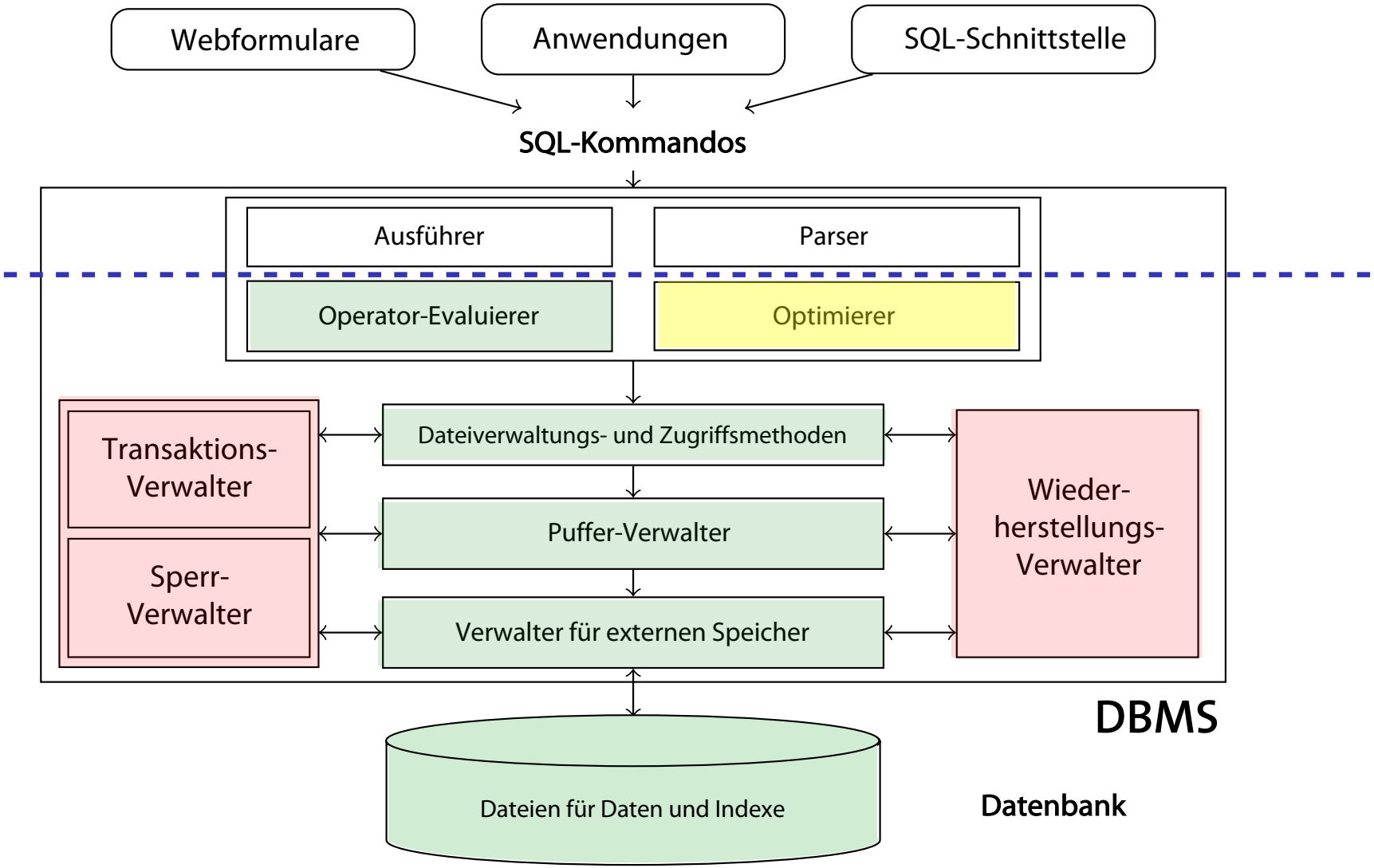
Anfrageoptimierung

Dr. Özgür Özçep

Universität zu Lübeck

Institut für Informationssysteme

Anfrageoptimierung



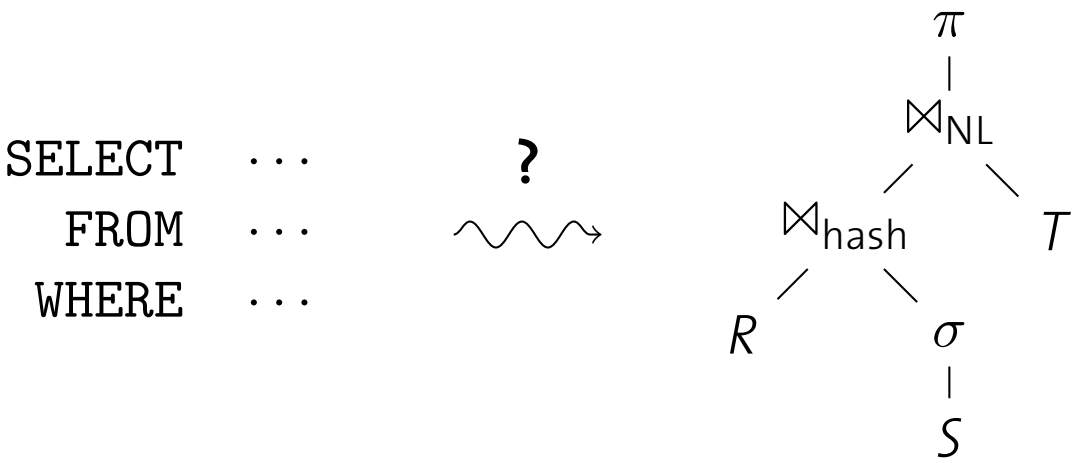
Danksagung

- 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 an den Kurs „Architecture and Implementation of Database Systems, Summer 14“ von Torsten Grust, Uni Tübingen

Optimierung: Motivation

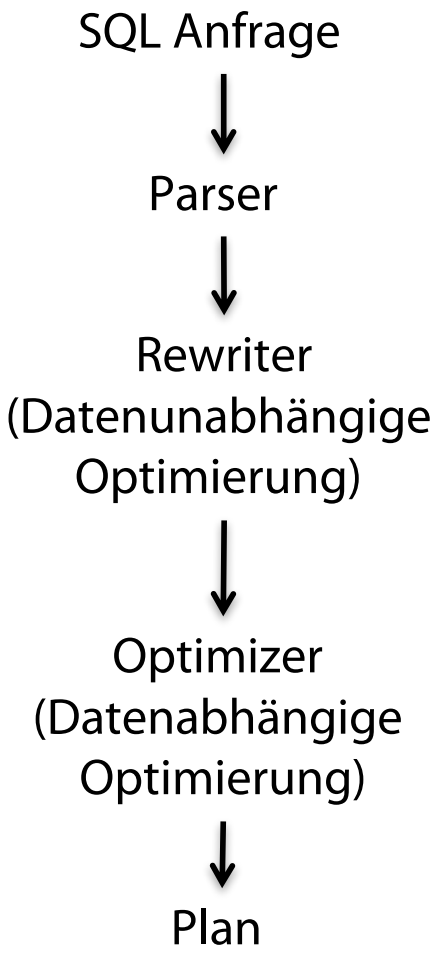
Anfrageoptimierung



- 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

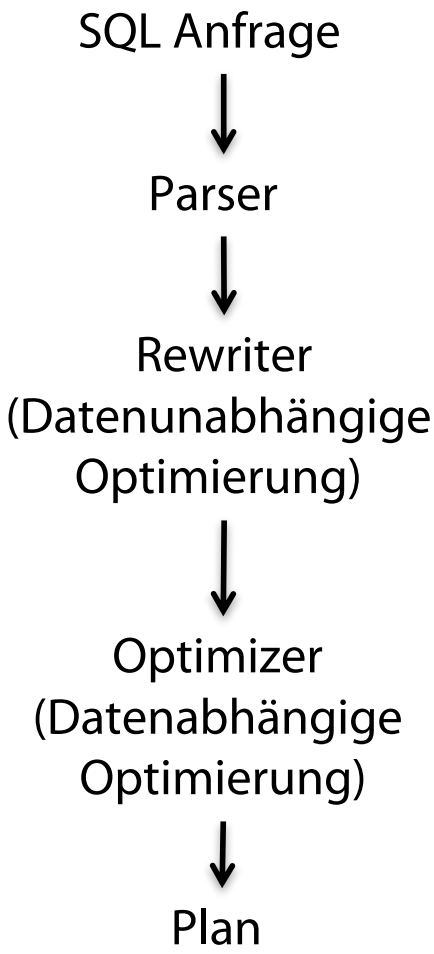
Optimierung

- 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

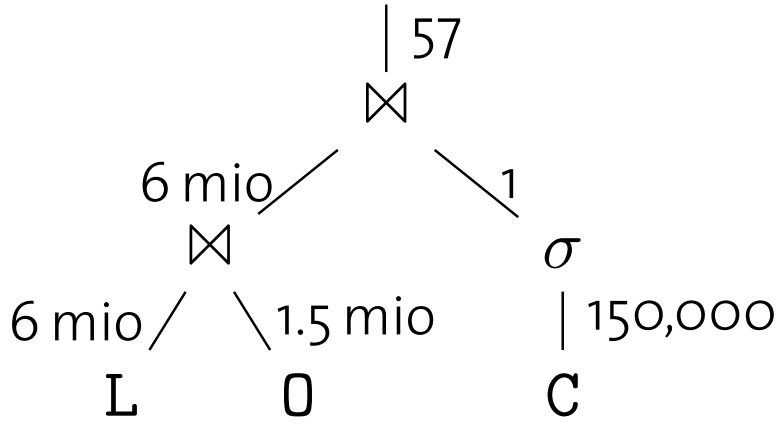
- Rewriting könnte z.B. Repräsentation von SQL-Anfragen in Datalog verwenden, um die Ideen aus dem Bereich des automatisierten Schließens zu nutzen
  Datalog bzw. **Logik** i.A. ist **wichtig für DB**
- Hier nicht näher besprochen
  - Minimierung einer Anfrage durch Elimination einer Unteranfrage
  -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

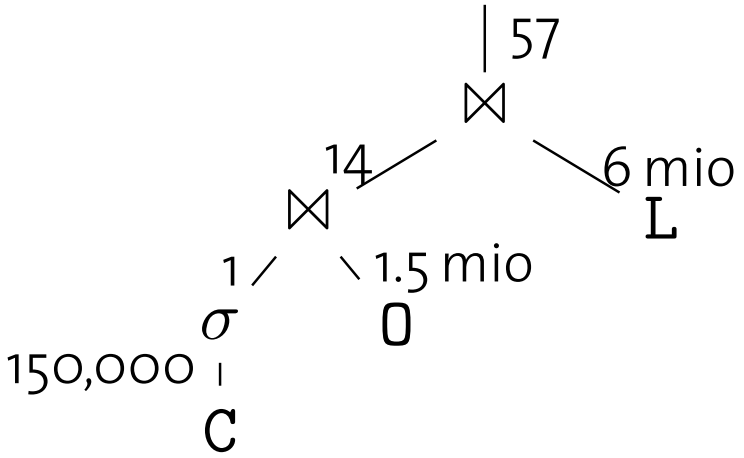


M. Benedikt: How Can Reasoners Simplify Database Querying (And Why Haven't They Done It Yet)? (Invited Talk at PODS 2018.)

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.'
```



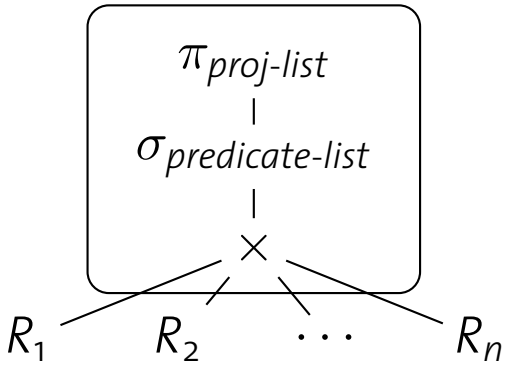


- Bezogen auf die Ausführungszeit können die Unterschiede „Sekunden vs. Tage“ bedeuten

Datenabhängige Optimierung

Abschätzung der Ergebnisgröße

Parser betrachtet Anfrageblock für Select-From-Where-Anfrage *Q*



Abschätzung der Ergebnisgröße von Q durch

- die **Kardinalitäten der Eingabetabellen bzw. Anfrageblöcke** $|R_1|, |R_2|, \ldots, |R_n|$ und
- die **Selektivität** *sel(predicate-list)*

$$|Q| = |R_1| \cdot |R_2| \cdot \ldots \cdot |R_n| \cdot sel(predicate\text{-}list)$$

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

Verbesserung der Selektivitätsabschätzung

- Annahmen
  - Gleichverteilung der Datenwerte in einer Spalte
  - Unabhängigkeit zwischen einzelnen Prädikaten

- Annahmen nicht immer gerechtfertigt, daher:

- 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)?

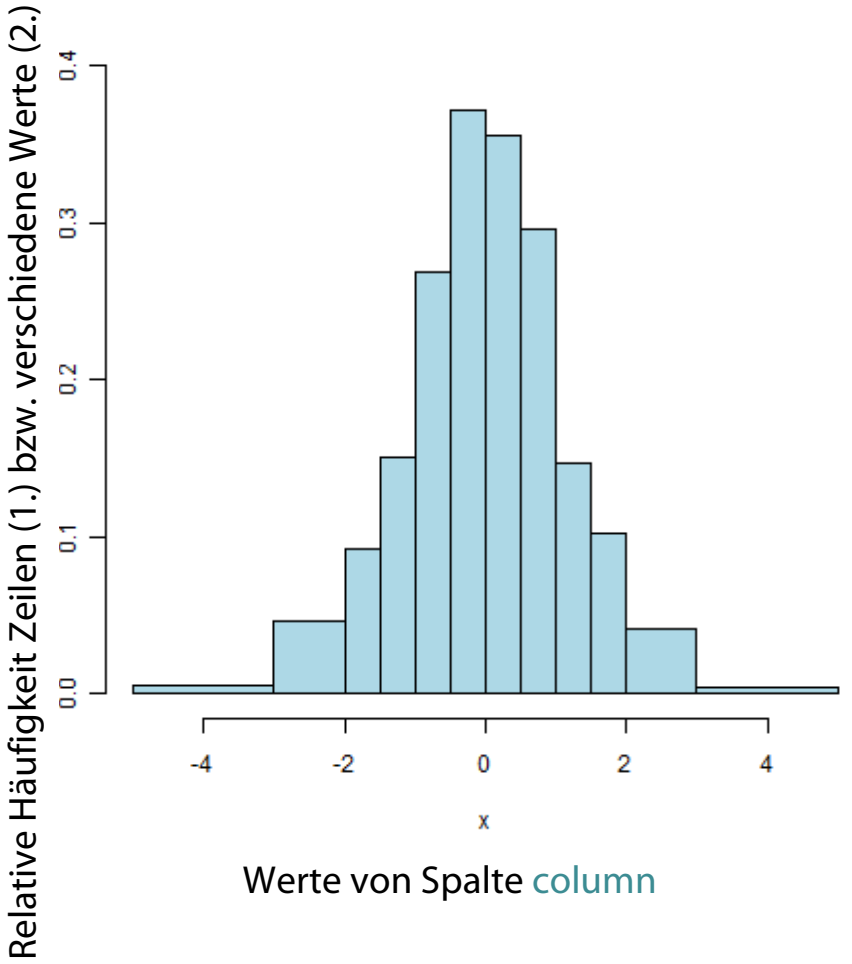
Lösung:

Die Annahme der Gleichverteilung ist beispielsweise für die Selektivitätsabschätzung für column = value nötig: unabhängig von value wird sie mit 1/|I| bzw. 1/10 angenommen.

Unabhängigkeit: Unabhängigkeit wird für die Konjunktion von Prädikaten angenommen (es wird nicht mit "bedingten Häufigkeiten" gerechnet).

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 $x_{i-1} < z.column <= x_i$
  2. Oder: Anzahl verschiedener Werte von column im Intervall $(x_{i-1},x_i]$ (absolut oder relativ)



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-tes Quantil (Type = ‚Q‘) oder n-häufigste Werte (Type = `F‘) 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(column, R) = n-häufigste (top-n) Werte in der Kolumne column einer Tabelle R
- MCF(column,R) = Häufigkeiten dieser Werte
- Verbesserte Abschätzung für column = value
  - column = value
  - $sel() = \begin{cases} MCF(R,column)[value] & \text{falls } value \in MCV(R, column) \\ 1/|I| & \text{falls nicht } value \in MCV(R, column), \text{ es aber einen Index auf Attribut column gibt} \\ 1/10 & \text{sonst} \end{cases}$

Kardinalitätsabschätzung für Projektion

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

- $|Q| = \begin{cases} V(A,R) & \text{falls } L = (A) \\ (= |I| & \text{falls Index auf Spalte } A\ ) \\ |R| & \text{falls Schlüsselkolumnen von } R \text{ in } L \text{ enthalten} \\ |R| & \text{ohne Duplikateneliminierung} \\ \text{Min}\,(|R|, \Pi_{Ai \in L}\, V(A_i,R)) & \text{sonst} \end{cases}$

Aufgabe:

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

Aufgabe:

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

Lösung:

- $|R \cup S| \leq |R| + |S|$
- $\text{Max}(0, |R|-|S|) \leq |R \setminus S| \leq |R|$
- $|R \times S| = |R| * |S|$

Kardinalitätsabschätzung für Join

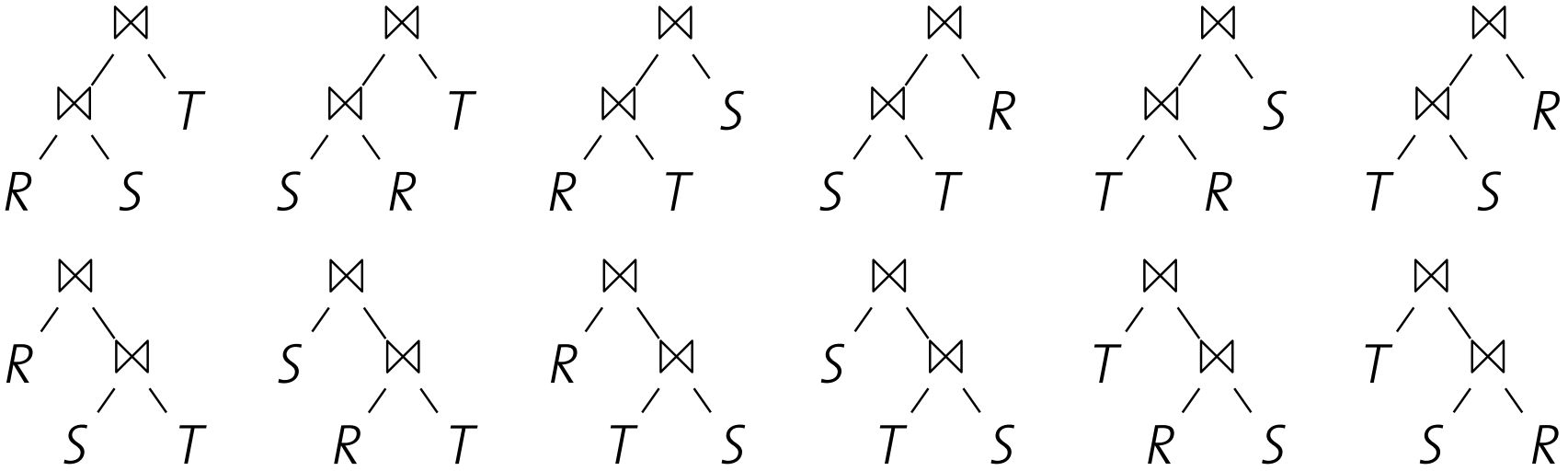
- Im Allgemeinen nicht-trivial
- Wenn Fremdschlüsselbeziehung vorliegen (kommt häufig vor), wie folgt abschätzbar:
  - $|R \bowtie_{R.A = S.A} S| = |S|$ falls S.A Fremdschlüssel auf R.A

- Bei Inklusionsdependenz wie folgt abschätzbar:
  - $|R \bowtie_{R.A = S.B} S| = \begin{cases} |S| * (|R| / V(A,R)) \text{ falls } \pi_B(S) \subseteq \pi_A(R) \\ |R| * (|S| / V(B,S)) \text{ falls } \pi_A(R) \subseteq \pi_B(S) \end{cases}$

Optimierung von mehreren Verbünden

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

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

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

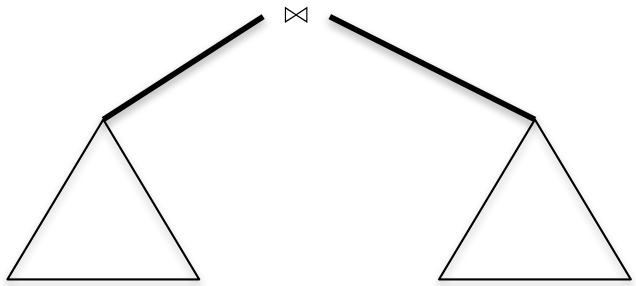
Anzahl der Bäume für n +1 Inputrelationen:
$C_n * (n+1)! = (2n)!/n!$

C_n = n-te Catalanzahl = $(2n)!/(n+1)!n!$

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
  $(0 \leqq k \leqq n\text{-}1)$

k binäre Verbünde
$R_1, \ldots, R_{k+1}$



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

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

- 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 $R_j \bowtie R_i$?):

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

$\longrightarrow$ 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:

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

$\longrightarrow$ 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:

$$optPlan(\{R_i, R_j, R_k, R_l\}) \leftarrow \text{ best of } R_i \bowtie optPlan(\{R_j, R_k, R_l\}),$$
$$optPlan(\{R_j, R_k, R_l\}) \bowtie R_i, \quad R_j \bowtie optPlan(\{R_i, R_k, R_l\}), \ldots,$$
$$optPlan(\{R_i, R_j\}) \bowtie optPlan(\{R_k, R_l\}), \ldots \quad .$$

$\longrightarrow$ 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)

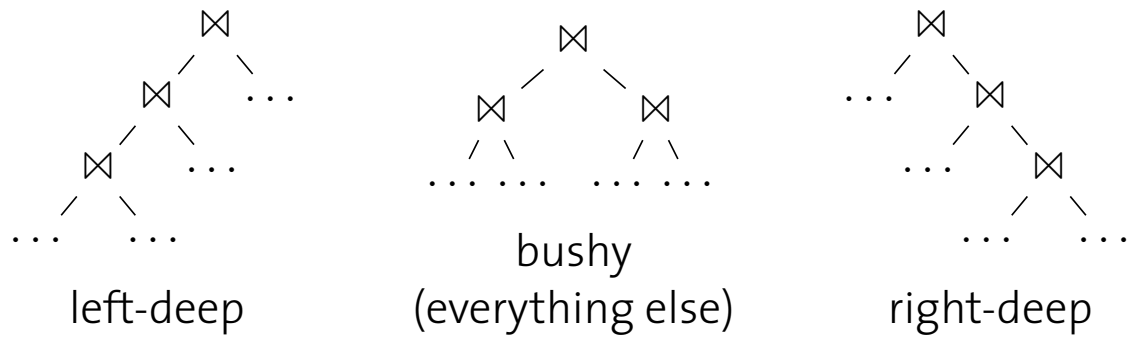
Optimaler *n*-Wege Verbundplan mit DP

Function: find_join_tree_dp (q(R_1 , . . . , R_n))

for i=1 **to** n **do**
  optPlan({R_i}) ← access_plans (R_i) ;
  prune_plans(optPlan({R_i})) ;
for i=2 **to** n **do**
  foreach S ⊆ {R_1,...,R_n} **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({R_1 , . . . , R_n}) ;

- possible_joins zählt mögliche Join-Implemenationen 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 Pläne[1]

- Die innere Relation ist immer eine Basisrelation
- Index-Verbund lässt sich anwenden
- Gut geeignet für die Verwendung von Nested-Join oder 1-pass-Algorithmen (s. folgende Aufgabe)

[1] So z.B. der Prototyp von System R (IBM)

Aufgabe:

Warum ist z.B. ein rechts-tiefer Verbundplan nicht bei Wahl eines Nested-Loop Joins verwendbar?

```
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
```

Lösung:

- Nested loop join liest das linke Argument nur einmal ein, das rechte Argument mehrmals.
- Beim rechts-tiefen Baum ist die rechte Seite eine ganze Unteranfrage, keine einzelne Tabelle.
- Die Inputs dieser Anfrage müssten für jedes Tupel des linken Arguments der übergeordneten Anfrage also mehrmals gelesen werden.

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

K. Ono, G.M. Lohman, Measuring the Complexity of Join Enumeration in Query Optimization, Proc. VLDB 1990

Greedy-Join-Enumeration

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

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

Andere Optimierungen: Rewriting

Prädikatsvereinfachung (Rewriting)

Beispiel: Schreibe

Non-Sargable[1)]

```
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

1) siehe Vorlesung Anfrageverarbeitung

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

- Korrelierte Unteranfragen

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

- Unkorrelierte Unteranfragen

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

Bei unkorrelierten Anfragen muss die Unteranfrage nur einmal ausgewertet werden

Won Kim. On Optimizing an SQL-like Nested Query.
ACM TODS, vol. 7, no. 3, 1982

Optimierung von geschachtelten Anfragen

- 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

Won Kim. On Optimizing an SQL-like Nested Query.
ACM TODS, vol. 7, no. 3, 1982

Andere Optimierungen: Nutzung von Indexen

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 ⋈ 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 ∧ B<10

- Holen der vollen Datensätze bzgl. einzelner Indexe sehr langsam

- Mehraufwand für Index-Verbund hingegen gering

Bitmap-Index

- Vorher diskutiert (Vorlesung 7, Seite 27): 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
- Oracle:

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

Bitmap-Index – Beispiel

<M,10100011>
<F , 01011100>

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

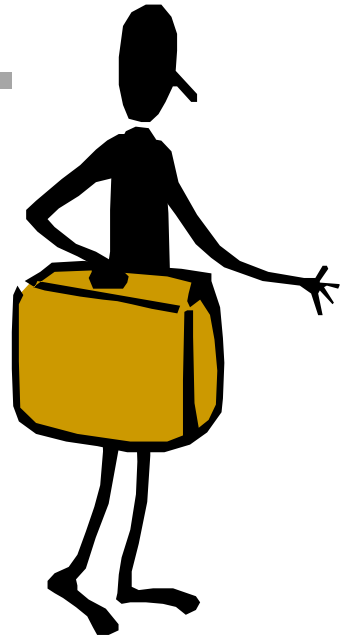
Anwendung von Bitmap-Indexstrukturen

- 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...