Datenbanken

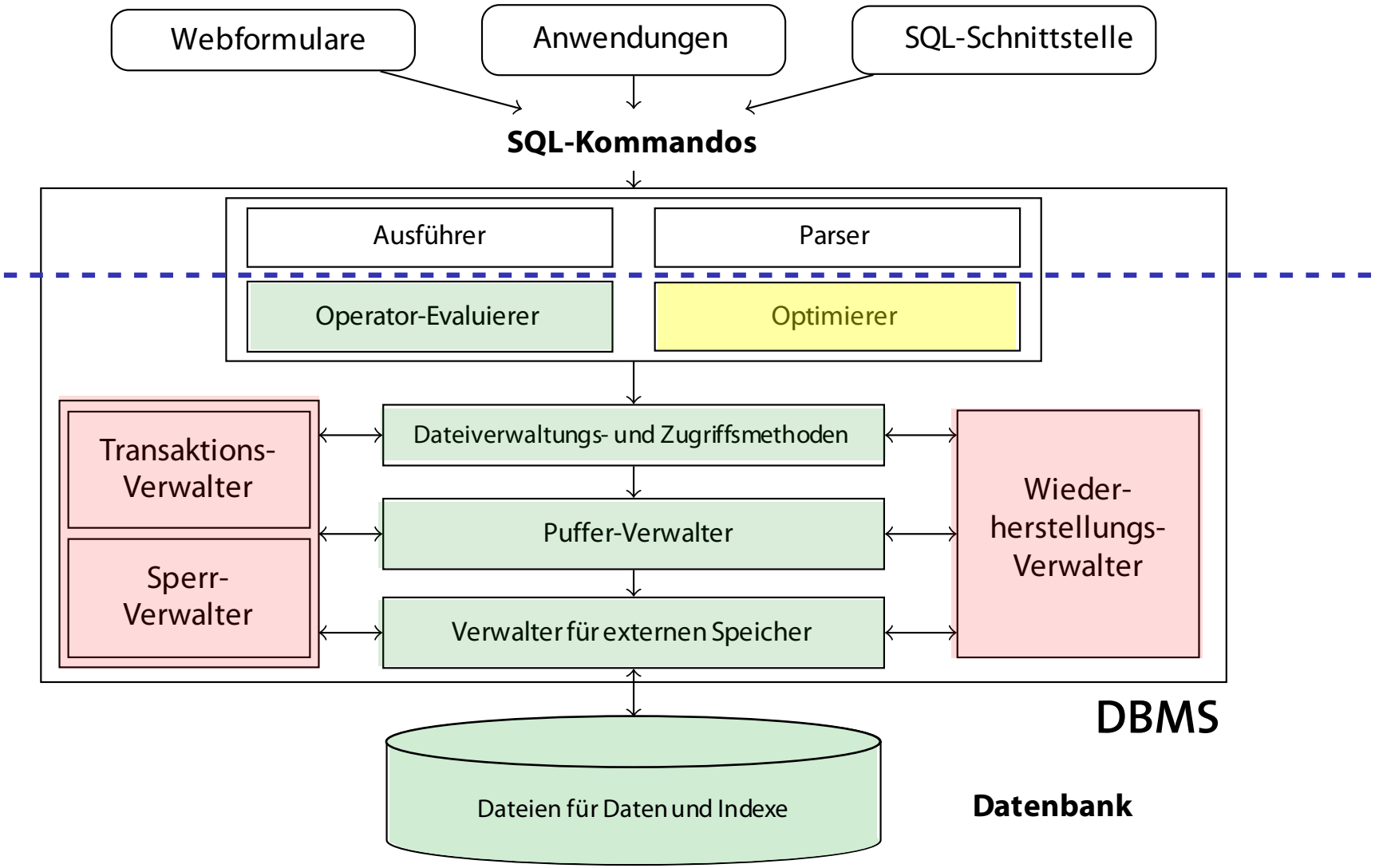
Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Karsten Martiny (Übungen)

Anfrageoptimierung



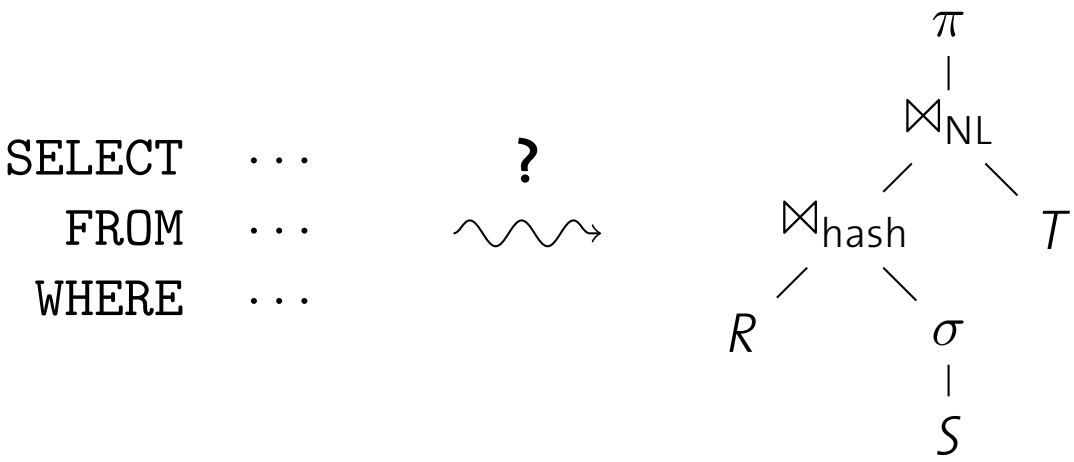
Danksagung

- 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

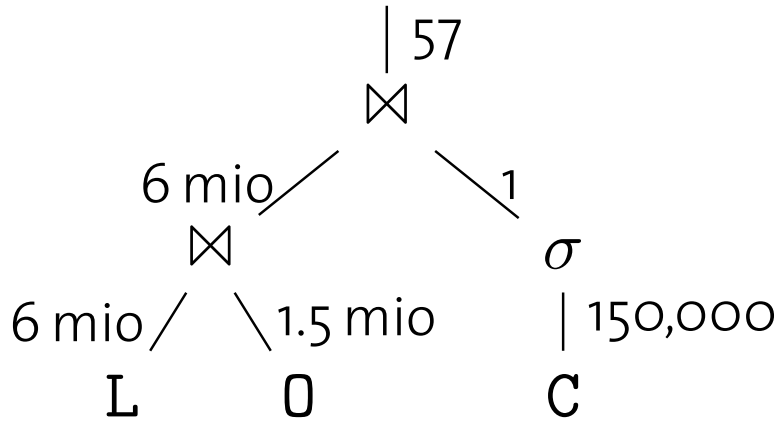
Anfrageoptimierung

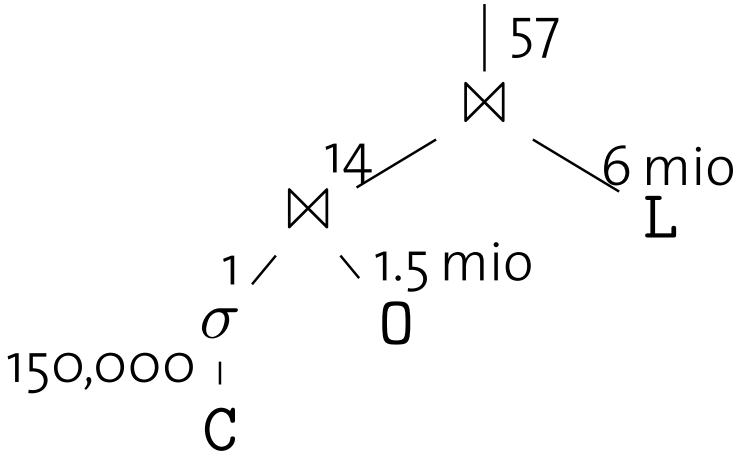


- Es gibt mehr als eine Art, eine Anfrage zu beantworten
  - Welche Implementation eines Verbundoperators?
  - Welche Parameter für Blockgrößen, Pufferallokation, ...
  - Automatisch einen Index aufsetzen?
- Die Aufgabe, den besten Ausführungsplan zu finden, ist der heilige Gral der Datenbankimplementierung

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.'
```

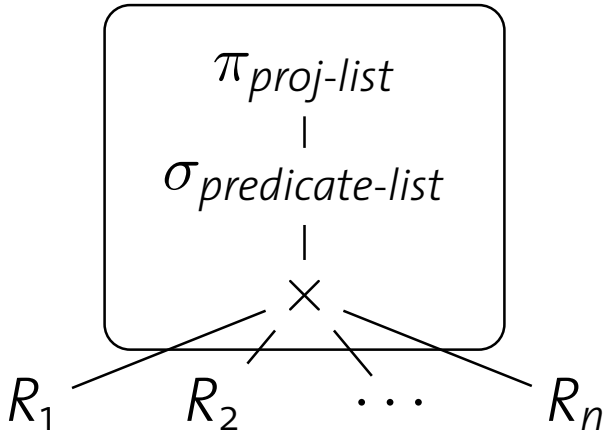




- Bezogen auf die Ausführungszeit können die Unterschiede „Sekunden vs. Tage“ bedeuten

Abschätzung der Ergebnisgröße

Betrachte Anfrageblock für Select-From-Where-Anfrage *Q*



Abschätzung der Ergebnisgröße von Q durch

- die Größe der Eingabetabellen $|R_1|, |R_2|, \ldots, |R_n|$ 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)

```
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.
```

Abschätzung der Selektivität

... durch Induktion über die Struktur des Anfrageblocks

column = *value*

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

$column_1 = column_2$

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

p_1 `AND` p_2

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

p_1 `OR` p_2

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

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

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
```

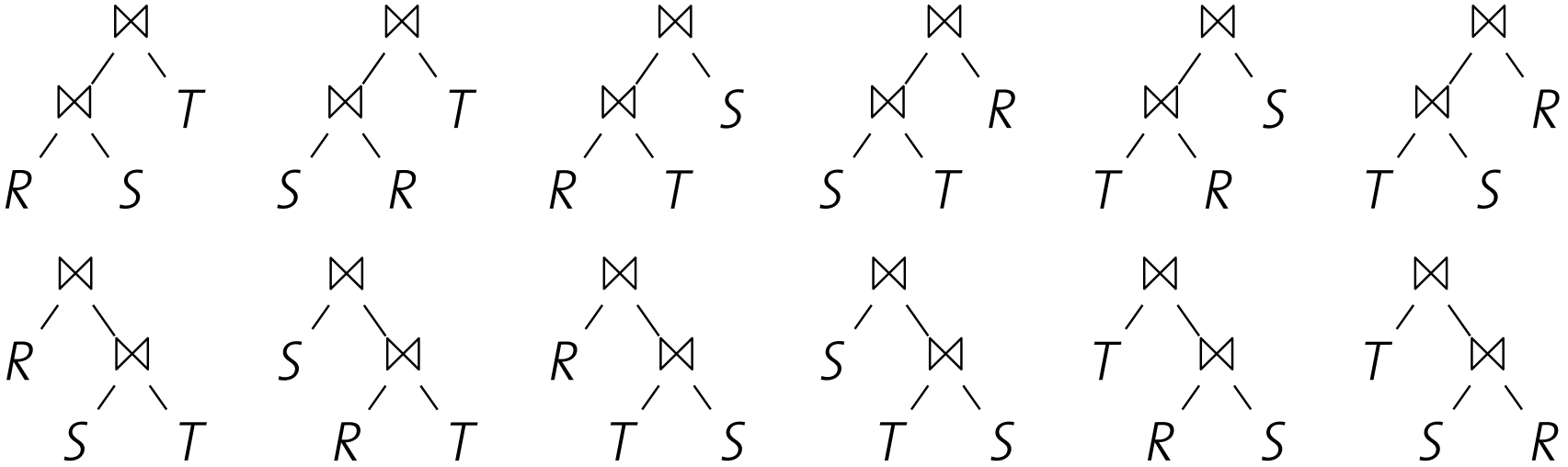
SYSCAT.COLDIST enthält Informationen wie

- 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

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

- Noch nicht berücksichtigt: Anzahl v der verschiedenen Verbundalgorithmen

Dynamische Programmierung

- Beispiel 4-Wege-Verbund

- Sammle gute Zugriffspläne für Einzelrelation (z.B. auch mit Indexscan und mit Ausnutzung von Ordnungen)

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 a set of good access paths to each of the R_i individually
(considers index scans, full table scans, interesting orders)

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 $R_j \bowtie R_i$?):

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

$\longrightarrow$ 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:

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

$\longrightarrow$ 24 plans to consider.

Beispiel (Fortsetzung)

Pass 4 (best 4-relation plan)

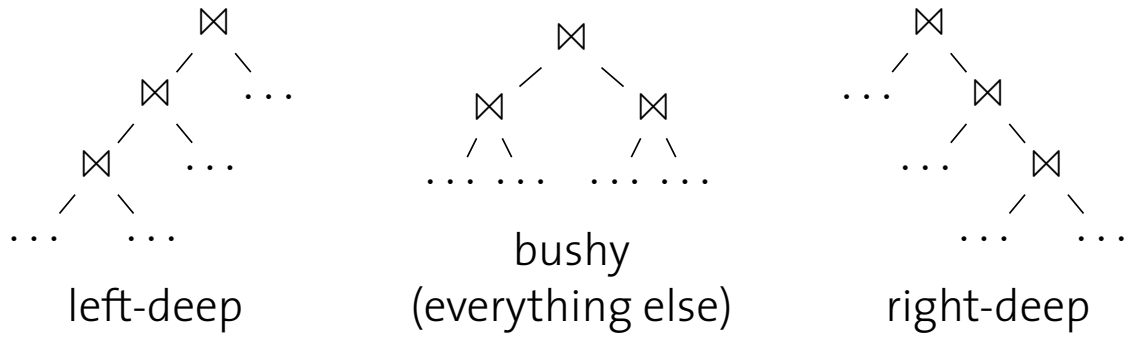
For each set of **four** tables R_i, R_j, R_k, and R_l, determine the best four-table join plan, using sub-plans obtained so far:

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

$\longrightarrow$ 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 Plänen)

Links-tiefe vs. buschige Verbundplan-Bäume



Implementierte Systeme generieren meist links-tiefe Pläne[1]

- Die innere Relation ist immer eine Basisrelation
- Bevorzugung der Verwendung von Index-Verbunden
- Gut geeignet für die Verwendung von Pipeline-Verarbeitungsprinzipien

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

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, 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 $\quad worklist \leftarrow worklist \setminus \{P_j, P_k\} \cup \{(P_j \bowtie_\ldots P_k)\}$;
// $worklist = \{P_1\}$
8 **return** single plan left in $worklist$;

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

Prädikatsvereinfachung

Beispiel: Schreibe

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

um in

```
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)
```


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

Zusammenfassung

- Anfrageparser
  - Übersetzung der Anfrage in Anfrageblock
- Umschreiber
  - 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

Noch zu diskutieren...