

Datenbanksysteme II
Optimierung
(Kapitel 16)

11.6.2007
Felix Naumann

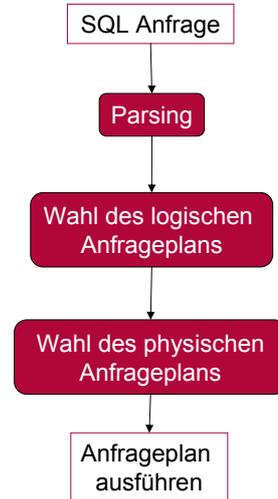
Wdh: Anfragebearbeitung –
Grundproblem

2

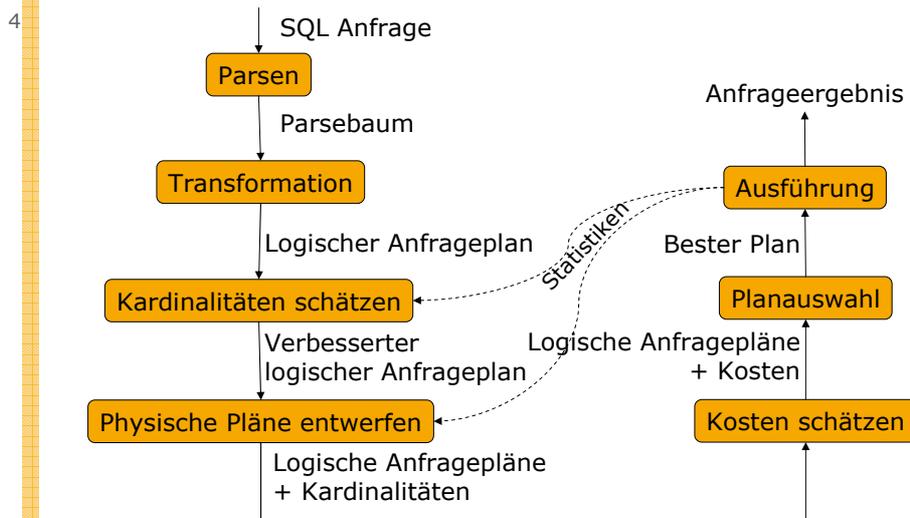
- Anfragen sind deklarativ.
 - SQL, Relationale Algebra
- Anfragen müssen in ausführbare (prozedurale) Form transformiert werden.
- Ziele
 - „QEP“ – prozeduraler Query Execution Plan
 - Effizienz
 - Schnell
 - Wenig Ressourcenverbrauch (CPU, I/O, RAM, Bandbreite)

Wdh: Ablauf der Anfragebearbeitung

- 3
1. Parsing
 - Parsen der Anfrage (Syntax)
 - Überprüfen der Elemente („Semantik“)
 - Parsebaum
 2. Wahl des logischen Anfrageplans
 - Baum mit logischen Operatoren
 - Potentiell exponentiell viele
 - Wahl des optimalen Plans
 - Logische Optimierung
 - Regelbasierter Optimierer
 - Kostenbasierter Optimierer
 3. Wahl des physischen Anfrageplans
 - Ausführbar
 - Programm mit physischen Operatoren
 - Algorithmen
 - Scan Operatoren
 - Wahl des optimalen Plans
 - physische Optimierung



Ablauf der Anfragebearbeitung



Überblick

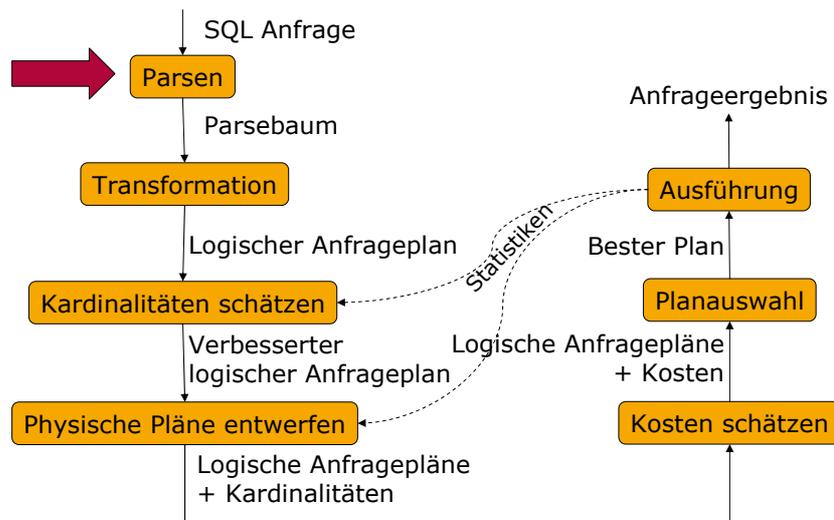
5

- ➔ (Parsing)
- (Algebraische Transformationsregeln)
- (Logische Anfragepläne)
- Kostenmodell
- Kostenbasierte Optimierung
- Joinreihenfolge
 - Dynamische Programmierung
- Physische Anfragepläne



Ablauf der Anfragebearbeitung

6



Eine Grammatik für einen Teil von SQL

7

- Anfragen
 - $\langle \text{Anfrage} \rangle ::= \langle \text{SFW} \rangle$
 - $\langle \text{Anfrage} \rangle ::= (\langle \text{SFW} \rangle)$
 - Mengenoperatoren fehlen
- SFWs
 - $\langle \text{SFW} \rangle ::= \text{SELECT } \langle \text{SelListe} \rangle \text{ FROM } \langle \text{FromListe} \rangle \text{ WHERE } \langle \text{Bedingung} \rangle$
 - Gruppierung, Sortierung etc. fehlen
- Listen
 - $\langle \text{SelListe} \rangle ::= \langle \text{Attribut} \rangle, \langle \text{SelListe} \rangle$
 - $\langle \text{SelListe} \rangle ::= \langle \text{Attribut} \rangle$
 - $\langle \text{FromListe} \rangle ::= \langle \text{Relation} \rangle, \langle \text{FromListe} \rangle$
 - $\langle \text{FromListe} \rangle ::= \langle \text{Relation} \rangle$
- Bedingungen (Beispiele)
 - $\langle \text{Bedingung} \rangle ::= \langle \text{Bedingung} \rangle \text{ AND } \langle \text{Bedingung} \rangle$
 - $\langle \text{Bedingung} \rangle ::= \langle \text{Tupel} \rangle \text{ IN } \langle \text{Anfrage} \rangle$
 - $\langle \text{Bedingung} \rangle ::= \langle \text{Attribut} \rangle = \langle \text{Attribut} \rangle$
 - $\langle \text{Bedingung} \rangle ::= \langle \text{Attribut} \rangle \text{ LIKE } \langle \text{Muster} \rangle$
- $\langle \text{Tupel} \rangle, \langle \text{Attribut} \rangle, \langle \text{Relation} \rangle, \langle \text{Muster} \rangle$ nicht durch grammatische Regel definiert
- Vollständig z.B. hier: <http://docs.openlinksw.com/virtuoso/GRAMMAR.html>

Felix Naumann | VL Datenbanksysteme II | SS 07

Prüfung der Semantik

8

Während der Übersetzung semantische Korrektheit prüfen

- Existieren die Relationen und Sichten der FROM Klausel?
- Existieren die Attribute in den genannten Relationen?
 - Sind sie eindeutig?
- Korrekte Typen für Vergleiche?
- Aggregation korrekt?
- ...

Felix Naumann | VL Datenbanksysteme II | SS 07

Überblick

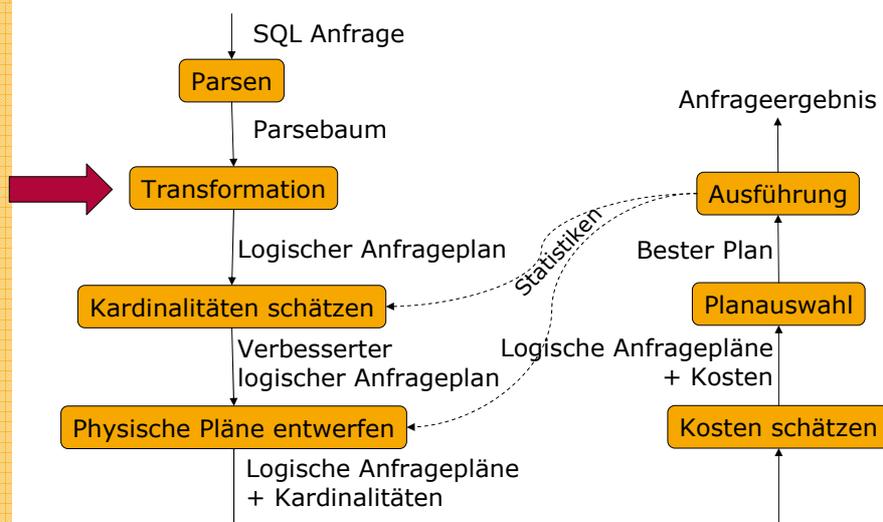
9

- (Parsing)
- (Algebraische Transformationsregeln)
- (Logische Anfragepläne)
- Kostenmodell
- Kostenbasierte Optimierung
- Joinreihenfolge
 - Dynamische Programmierung
- Physische Anfragepläne



Ablauf der Anfragebearbeitung

10



11

Transformation der internen Darstellung

- Ohne Semantik zu verändern
- Zur effizienteren Ausführung
 - Insbesondere: Kleine Zwischenergebnisse

Äquivalente Ausdrücke

- Zwei Ausdrücke der relationalen Algebra heißen äquivalent, falls
 - Gleiche Operanden (= Relationen)
 - Stets gleiche Antwortrelation
 - Stets?

Stets = Für jede mögliche Instanz der Datenbank

Felix Naumann | VL Datenbanksysteme II | SS 07

12

- \times ist kommutativ und assoziativ
 - $R \times S = S \times R$
 - $(R \times S) \times T = R \times (S \times T)$
- \cup ist kommutativ und assoziativ
 - $R \cup S = S \cup R$
 - $(R \cup S) \cup T = R \cup (S \cup T)$
- \cap ist kommutativ und assoziativ
 - $R \cap S = S \cap R$
 - $(R \cap S) \cap T = R \cap (S \cap T)$
- \bowtie ist kommutativ und assoziativ
 - $R \bowtie S = S \bowtie R$
 - $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

Gilt jeweils für Mengen und Multimengen

Ausdrücke können in beide Richtungen verwendet werden.

Felix Naumann | VL Datenbanksysteme II | SS 07

Weitere Regeln

13

- Selektion
 - $\sigma_{c1 \text{ AND } c2}(R) = \sigma_{c1}(\sigma_{c2}(R))$
 - $\sigma_{c1 \text{ OR } c2}(R) = \sigma_{c1}(R) \cup \sigma_{c2}(R)$
 - Nicht bei Multimengen
 - $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
 - $\sigma_c(R \Phi S) \equiv (\sigma_c(R)) \Phi (\sigma_c(S))$
 - $\Phi \in \{\cup, \cap, -, \bowtie\}$
 - $\sigma_c(R \Phi S) \equiv (\sigma_c(R)) \Phi S$
- Projektion
 - $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))$
 - $\pi_L(R \bowtie_C S) = \pi_L(\pi_M(R) \bowtie_C \pi_N(S))$
 - $\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))$
 - $\pi_L \sigma_C(R) = \pi_L(\sigma_C(\pi_M(R)))$

Überblick

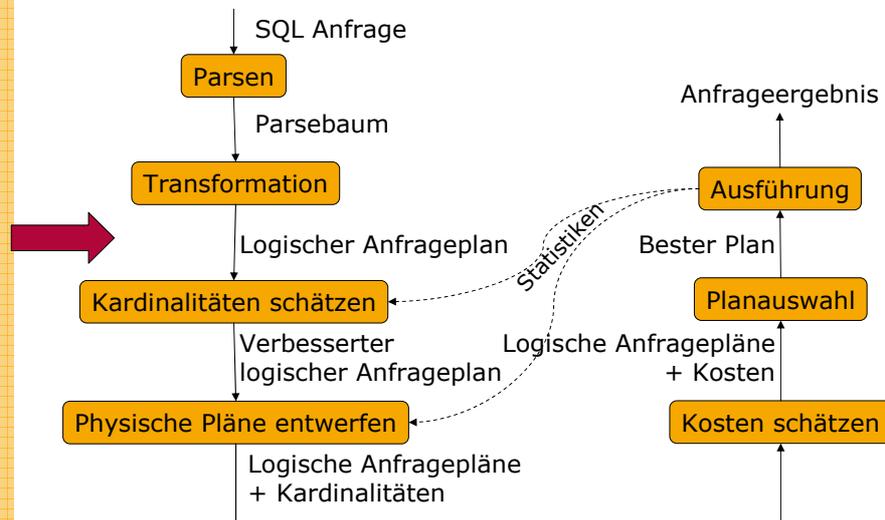
14

- (Parsing)
- (Algebraische Transformationsregeln)
- (Logische Anfragepläne)
- Kostenmodell
- Kostenbasierte Optimierung
- Joinreihenfolge
 - Dynamische Programmierung
- Physische Anfragepläne



Ablauf der Anfragebearbeitung

15



Felix Naumann | VL Datenbanksysteme II | SS 07

Zwei Schritte

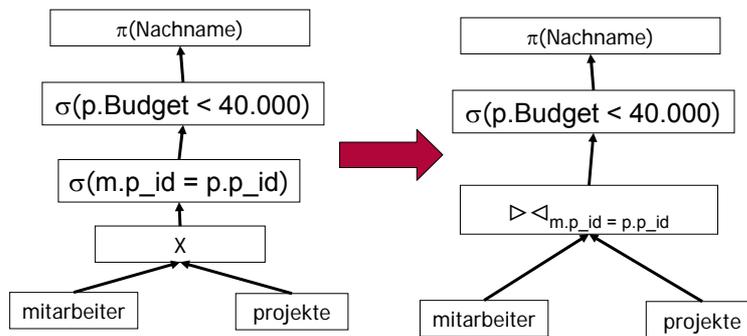
16

- Schritt 1
 - Parsebaum in Ausdruck der relationalen Algebra übersetzen
 - Darstellung als Baum
 - U.a.: Subanfrage auflösen (nicht hier)
- Schritt 2
 - Umformung des Baums gemäß Transformationsregeln
 - „Voroptimierung“ (Heuristiken)
 - Selektionen pushen
 - Projektionen pushen (und einbauen)
 - Duplikateliminierung verschieben
 - Selektion + Kreuzprodukt = Join
 - Gruppierung von Vereinigung und Joinoperatoren

Felix Naumann | VL Datenbanksysteme II | SS 07

Anfragebearbeitung – Beispiel

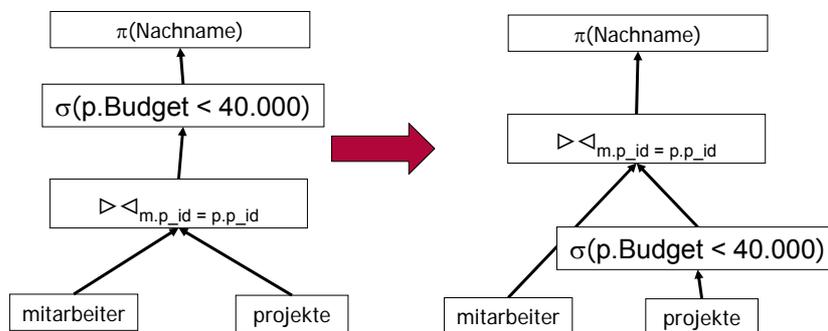
17



Felix Naumann | VL Datenbanksysteme II | SS 07

Anfragebearbeitung – Beispiel

18



Felix Naumann | VL Datenbanksysteme II | SS 07

Übergang zum Physischen Anfrageplan

19

Diverse Freiheitsgrade

1. Reihenfolge und Gruppierung von assoziativen und kommutativen Operatoren
 - Joins, Vereinigung, Schnittmenge
 2. Wahl eines Algorithmus für jeden Operator
 - Siehe voriger Foliensatz
 3. Zusätzliche Operatoren, die im logischen Plan nicht auftauchen
 - Scan, Sort
 4. Modus des Datentransports zwischen Operatoren
 - Temporäre Tabelle, Pipeline mit Iterator
- Dafür zunächst: Kostenabschätzung

Felix Naumann | VL Datenbanksysteme II | SS 07

Überblick

20

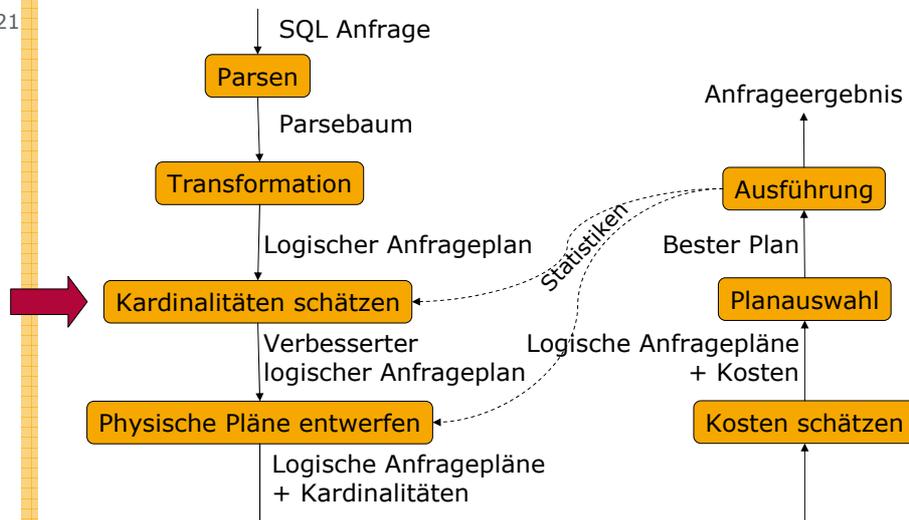
- (Parsing)
- (Algebraische Transformationsregeln)
- (Logische Anfragepläne)
- ➔ ■ Kostenmodell
- Kostenbasierte Optimierung
- Joinreihenfolge
 - Dynamische Programmierung
- Physische Anfragepläne



Felix Naumann | VL Datenbanksysteme II | SS 07

Ablauf der Anfragebearbeitung

21



Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenbasierte Optimierung

22

- Konzeptionell: Generiere alle denkbaren Anfrageausführungspläne
 - Bewerte deren Kosten anhand eines Kostenmodells
 - Statistiken und Histogramme
 - Kalibrierung gemäß verwendeter Rechner
 - Abhängig vom verfügbaren Speicher
 - Aufwands-Kostenmodell
 - Durchsatz-maximierend
 - Nicht Antwortzeit-minimierend
 - Führe billigsten Plan aus
- } Achtung: Nicht zu lange optimieren!

Felix Naumann | VL Datenbanksysteme II | SS 07

Problemgröße (Suchraum)

23

- Konzeptionell: Generiere alle denkbaren Anfrageausführungspläne
- Anzahl Bushy-Pläne mit n Tabellen

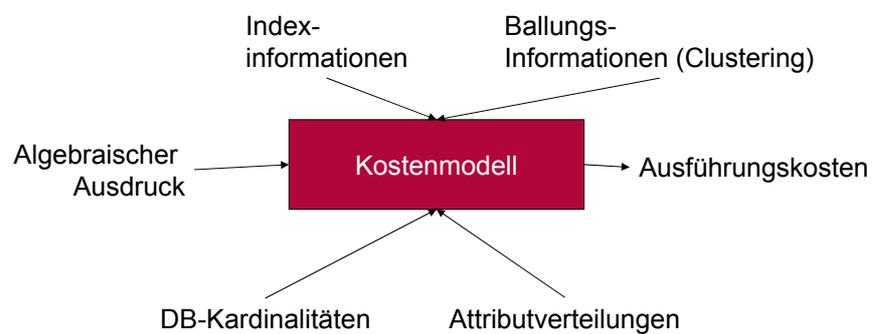
$$\frac{(2(n-1))!}{(n-1)!}$$

n	2 ⁿ	(2(n-1))!/(n-1)!
2	4	2
5	32	1.680
10	1.024	1,76*10 ¹⁰
20	1.048.576	4,3*10 ²⁷

- Plankosten unterscheiden sich um viele Größenordnungen.
- Optimierungsproblem ist NP-hart

Kostenmodell

24



Statistiken

25

- Zu jeder Basisrelation
 - Anzahl der Tupel
 - Tupelgröße
- Zu (jedem) Attribut
 - Min / Max
 - Werteverteilung (Histogramm)
 - Anzahl der distinct Werte
- Zum System
 - Speichergröße
 - Bandbreite
 - I/O Zeiten
 - CPU Zeiten

Problem: Erstellung und Update der Statistiken

- Deshalb meist nur explizit/manuell zu initiieren
 - `runstats()`

Felix Naumann | VL Datenbanksysteme II | SS 07

Kosten von Operationen - Zwischenergebnisse

26

Wesentliches Kostenmerkmal: Anzahl der Tupel im Input

- Insbesondere: Passt die Relation in den Hauptspeicher?
- Selektion, Projektion, Sortierung, Join

Output ist Input des nächsten Operators.

Deshalb: Ein Kostenmodell schätzt u.a. für jede Operation die Anzahl der Ausgabebetupel.

- „Selektivität“ in Bezug auf Inputgröße
- $\# \text{Ausgabebetupel} = \# \text{Eingabetupel} \times \text{Selektivität}$
- Auch „Selektivitätsfaktor“ (*selectivity factor, sf*)

Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenschätzung – Projektion

27

- Größe des Zwischenergebnisses kann exakt ausgerechnet werden
- Beispiele
 - Relation $R(A \text{ integer}(4), B \text{ integer}(4), C \text{ varchar}(100))$
 - Tupelheader: 12 Byte
 - => 120 Byte pro Tupel
 - Block: 1024 Byte mit 24 Byte header
 - => 8 Tupel pro Block
 - $T(R) = 10,000 \Rightarrow B(R) = 1250$
 - $S = \pi_{A+B,C}(R) \Rightarrow 116 \text{ Byte pro Tupel}$
 - => 8 Tupel pro Block: $B(S) = 1250$
 - $T = \pi_{A,B}(R) \Rightarrow 20 \text{ Byte pro Tupel}$
 - => 50 Tupel pro Block und $B(T) = 200$

Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenschätzung – Selektion

28

- Anzahl Tupel sinkt, Tupelgröße bleibt
- $S = \sigma_{A=c}(R)$
 - $T(S) = T(R)/V(R,A)$
 - $V(R,A)$ ist Anzahl der distinct Werte von Attribut A.
 - Annahme: Werte sind gleichverteilt
 - Annahme: c ist einer dieser Werte
 - Bessere Abschätzung mittels Histogramme
- $S = \sigma_{A<c}(R)$
 - Erste Abschätzung: $T(S) = T(R) / 2$
 - Typischer: $T(S) = T(R) / 3$
- $S = \sigma_{A \neq c}(R)$
 - Erste Abschätzung: $T(S) = T(R)$
 - Etwas genauer: $T(S) = T(R) \cdot (V(R,A) - 1) / V(R,A)$
- Bei Konjunktionen mehrerer Selektionsbedingungen: Multiplikation der Selektivitätsfaktoren
 - Annahme: Unabhängigkeit der Bedingungen

Felix Naumann | VL Datenbanksysteme II | SS 07

TABLE 1 SELECTIVITY FACTORS

```

column = value
  F = 1 / ICARD(column index) if there is an index on column
  This assumes an even distribution of tuples among the index key
  values.
  F = 1/10 otherwise

column1 = column2
  F = 1/MAX(ICARD(column1 index), ICARD(column2 index))
  if there are indexes on both column1 and column2
  This assumes that each key value in the index with the smaller
  cardinality has a matching value in the other index.
  F = 1/ICARD(column-i index) if there is only an index on column-i
  F = 1/10 otherwise

column > value (or any other open-ended comparison)
  F = (high key value - value) / (high key value - low key value)
  Linear interpolation of the value within the range of key values
  yields F if the column is an arithmetic type and value is known at
  access path selection time.
  F = 1/3 otherwise (i.e. column not arithmetic)
  There is no significance to this number, other than the fact that
  it is less selective than the guesses for equal predicates for
  which there are no indexes, and that it is less than 1/2. We
  hypothesize that few queries use predicates that are satisfied by
  more than half the tuples.

column BETWEEN value1 AND value2
  F = (value2 - value1) / (high key value - low key value)

  A ratio of the BETWEEN value range to the entire key value range is
  used as the selectivity factor if column is arithmetic and both
  value1 and value2 are known at access path selection.
  F = 1/4 otherwise
  Again there is no significance to this choice except that it is
  between the default selectivity factors for an equal predicate and
  a range predicate.

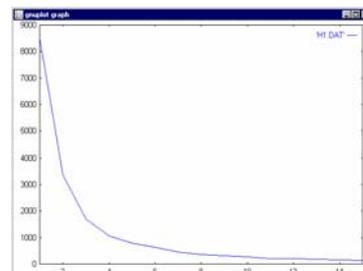
```

Felix Naumann | VL Datenbanksysteme II | SS 07

Zipf-Verteilung

30

- Modelliert Worthäufigkeiten in Texten einer Sprache
 - Linguistik
- Sortierung nach Häufigkeit
 - h sei Häufigkeit des häufigsten Wertes
- i -ter Wert taucht $h/i^{1/2}$ mal auf
 - Frage: Wie sieht Kurve aus?
- Frage: Wie sollte Kardinalitätsschätzung angepasst werden?
- Antwort: Gar nicht! Warum?
 - Durchschnittliche Antwortkardinalität ist weiter $T(R)/V(R,A)$
 - Annahme: Konstanten werden gleichverteilt gewählt
 - Bei Zipf-Verteilung der Konstanten: Unterschätzung!



Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenschätzung – Selektion

31

Beispiel

- $S = \sigma_{A=10 \text{ AND } B < 20}(R)$
- $T(R) = 10.000$ und $V(R,A) = 50$
- $\Rightarrow T(S) = 10.000 \cdot 1/50 \cdot 1/3 = 67$

- $S = \sigma_{A=10 \text{ AND } A > 20}(R)$
 - $\Rightarrow T(S) = 10.000 \cdot 1/50 \cdot 1/3 = 67 ?$
 - Besser: $T(S) = 0$
 - Optimierer sollte solche Fälle erkennen

Kostenschätzung – Selektion mit Disjunktion

32

- $S = \sigma_{C1 \text{ OR } C2}(R)$
- Idee 1: Summe der Ergebniskardinalitäten
 - Annahme: Kein Tupel erfüllt beide Bedingungen
 - Problem: $T(S) > T(R)$
- Idee 2: $\text{MIN}[R, \text{Summe der Ergebniskardinalitäten}]$
- Idee 3: Wahrscheinlichkeitstheorie
 - Annahme: C1 und C2 sind unabhängig
 - $T(R) = n$ und $T(\sigma_{C1}(R)) = m_1$ und $T(\sigma_{C2}(R)) = m_2$
 - $\Rightarrow T(S) = n \left(1 - \left(1 - \frac{m_1}{n}\right) \left(1 - \frac{m_2}{n}\right)\right)$

Anteil Tupel, die nicht C1 entsprechen

Anteil Tupel, die nicht C2 entsprechen

Kostenschätzung – Selektion mit Disjunktion

33

Beispiel

- $S = \sigma_{A=10 \text{ OR } B < 20}(R)$
- $T(\sigma_{A=10}(R)) = T(R) / V(R,A) = 10.000 / 50 = 200$
- $T(\sigma_{B < 20}(R)) = T(R) / 3 = 3.333$
- Einfache Schätzung (Idee 1): $T(S) = 200 + 3.333 = 3.533$
- Bessere Schätzung (Idee 3):
$$n(1 - (1 - \frac{m_1}{n})(1 - \frac{m_2}{n})) = 10000(1 - (\frac{200}{10000})(1 - \frac{3333}{10000})) = 3466$$

Kostenschätzung – Join

34

- Hier nur natural join
 - Equijoin analog
 - Thetajoin: „<“, „>“, usw: Wie zuvor: 1/3
- $R(X,Y) \bowtie S(Y,Z)$
 - Vereinfachende Annahme: Y ist nur ein Attribut.
- Problem: Beziehung zwischen R.Y und S.Y
 - Disjunkte Werte: $T(R \bowtie S) = 0$
 - Fremdschlüsselbeziehung (Schlüssel in S): $T(R \bowtie S) = T(R)$
 - Fast alle gleiche Werte: $T(R \bowtie S) = T(R) \cdot T(S)$
- Vereinfachende Annahmen
 - *Containment of Value Sets*: Werte eines Attributs, das in mehreren Relationen auftaucht, werden vom Beginn einer festen Liste gewählt.
 - Falls $V(R,Y) \leq V(S,Y) \Rightarrow$ Jeder Y-Wert in R taucht in S auf
 - Realistisch? Wann?
 - *Preservation of Value Sets*: Anzahl der distinct-Werte eines nicht-Joinattributs bleiben erhalten.
 - $V(R \bowtie S, A) = V(R, A)$
 - Realistisch? Wann?

Kostenschätzung – Join

35

- Sei $V(R,Y) \leq V(S,Y)$
 - \Rightarrow Jedes Tupel aus R hat eine $1/V(S,Y)$ Chance, mit einem gegebenen S-Tupel zu joinen
 - \Rightarrow (da $T(S)$ S-Tupel): Ein Tupel aus R joint mit $T(S) \cdot 1/V(S,Y)$ Tupeln aus S
 - \Rightarrow (da $T(R)$ R-Tupel): $T(R \bowtie S) = T(R) \cdot T(S) / V(S,Y)$
- Falls $V(R,Y) \geq V(S,Y)$: $T(R \bowtie S) = T(R) \cdot T(S) / V(R,Y)$
- Allgemein: $T(R \bowtie S) = T(R) \cdot T(S) / \max[V(R,Y), V(S,Y)]$

Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenschätzung – Join

36

Beispiel

- $R(A,B) \bowtie S(B,C) \bowtie U(C,D)$
- $T(R) = 1.000$; $T(S) = 2.000$; $T(U) = 5.000$
- $V(R,B) = 20$; $V(S,B) = 50$; $V(S,C) = 100$; $V(U,C) = 500$
- Betrachtung: $(R \bowtie S) \bowtie U$
- $T(R \bowtie S) = T(R) \cdot T(S) / \max[V(R,B), V(S,B)] = 1.000 \cdot 2.000 / 50 = 40.000$
- $T(R \bowtie S \bowtie U) = T(R \bowtie S) \cdot T(U) / \max[V(R \bowtie S, C), V(U, C)] = 40.000 \cdot 5.000 / \max[100, 500] = 400.000$
 - Warum $V(R \bowtie S, C) = 100$?
 - Preservation of Value Sets: $V(R \bowtie S, C) = V(S, C)$
 - C ist nicht-Joinattribut
- Probe: $T(R \bowtie (S \bowtie U)) = 40.000$
 - Zu Hause

Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenschätzung – Join

37

- $R(X,Y) \bowtie S(Y,Z)$
 - Y enthält mehr als ein Attribut
 - Schreibweise hier: $R(X,Y_1, Y_2) \bowtie S(Y_1, Y_2, Z)$
- $T(R \bowtie S) =$
 $T(R) \cdot T(S) / \max[V(R,Y_1), V(S,Y_1)] \max[V(R,Y_2), V(S,Y_2)]$
- Allgemein
 - Ergebniskardinalität von $R \bowtie S$ entspricht dem Produkt der Kardinalitäten von R und S, dividiert durch das größere von $V(R,Y)$ und $V(S,Y)$ für jedes Attribut.

Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenschätzung – Mehrfacher Join

38

- Allgemeiner Fall: $S = R_1 \bowtie R_2 \bowtie \dots \bowtie R_n$
- Notation: $V(R_i, A) = v_i$
- Attribut A erscheine in k Relationen.
- Es gelte $v_1 \leq v_2 \leq \dots \leq v_k$.
- Gegeben: 1 Tupel aus jeder Relation
- Gesucht: Wahrscheinlichkeit, dass alle im A-Wert übereinstimmen
 - Containment of Value Sets: Jeder A-Wert von Tupeln aus R_1 taucht in den anderen Relationen auf.
 - Ein Tupel aus R_i hat Wahrscheinlichkeit $1/v_i$ mit gegebenen Tupel aus R_1 übereinzustimmen.
 - Zusammen: $1/v_2 v_3 \dots v_k$
- Gesamtverfahren: Ausgangspunkt ist Produkt aller Kardinalitäten
 - Selektivität: Für jedes Attribut, das mehr als einmal auftaucht: Dividiere durch Produkt aller v_i bis auf das kleinste.

Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenschätzung – Mehrfacher Join

39

Beispielanfrage

- $R(A,B,C) \bowtie S(B,C,D) \bowtie U(B,E)$
- Kardinalitäten: $T(R) = 1000$; $T(S) = 2000$; $T(U) = 5000$
- DISTINCT Werte:
 - $V(R,B): 20$; $V(R,C): 200$; $V(S,B): 50$; $V(S,C): 100$; $V(U,B): 200$

Vorgehen

- Zunächst alle Kombinationen aufzählen
 - $1000 \times 2000 \times 5000 = 10.000.000.000$
- Nun mit Wahrscheinlichkeiten für gemeinsame Attribute multiplizieren
 - Gemeinsame Attribute: B dreimal, C zweimal
 - Für B: $1/50 \times 1/200$
 - Für C: $1/200$
 - Also: $10.000.000.000 / (50 \times 200 \times 200) = 10.000.000.000 / 2.000.000 = 5.000$

Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenschätzung – Weitere Operationen

40

- Vereinigung ($R \cup S$)
 - Multimenge: Summe der Inputs
 - Menge: $\max[T(R), T(S)] \leq T(R \cup S) \leq T(R) + T(S)$
 - Z.B: $\text{AVG}[T(R) + T(S), \max[T(R), T(S)]]$
 - $= \max[T(R), T(S)] + \min[T(R), T(S)]/2$
- Schnittmenge ($R \cap S$)
 - $0 \leq T(R \cap S) \leq \min[T(R), T(S)]$
 - Idee 1: $\min[T(R), T(S)]/2$
 - Idee 2: Als extremen Join auffassen
 - Anomalien bei Multimengen
- Differenz
 - $T(R) - T(S) \leq T(R - S) \leq T(R)$
 - Z.B.: $T(R) - T(S)/2$

Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenschätzung – Weitere Operationen

41

- Duplikateliminierung
 - $1 \leq \delta(R) \leq T(R)$
 - $\delta(R) \leq \prod_i V(R, A_i)$
 - Z.B.: $\delta(R) = \min[T(R)/2, \prod_i V(R, A_i)]$
- Gruppierung und Aggregation
 - $1 \leq \gamma_L(R) \leq T(R)$
 - Falls nur ein Gruppierungsattribut: $T(\gamma_L(R)) = V(R, L)$
 - $\gamma_L(R) \leq \prod_i V(R, L_i)$
 - Z.B.: $\gamma_L(R) = \min[T(R)/2, \prod_i V(R, L_i)]$

Überblick

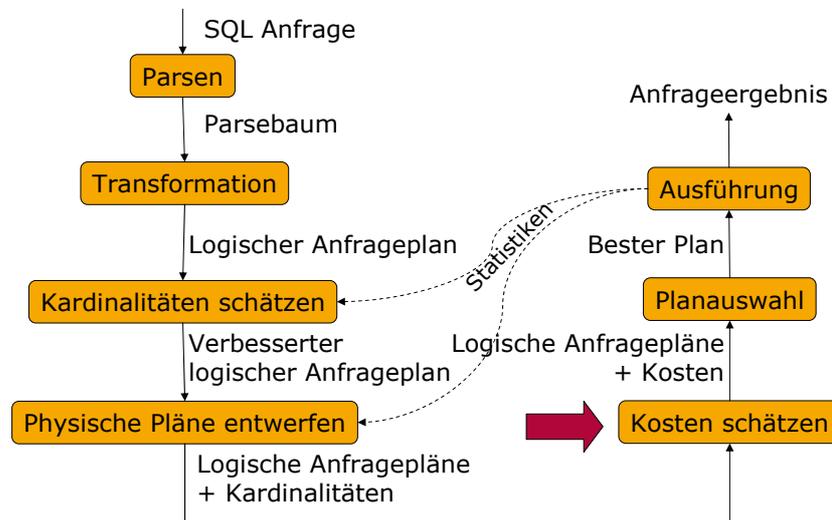
42

- (Parsing)
- (Algebraische Transformationsregeln)
- (Logische Anfragepläne)
- Kostenmodell
- ➔ ■ Kostenbasierte Optimierung
- Joinreihenfolge
 - Dynamische Programmierung
- Physische Anfragepläne



Ablauf der Anfragebearbeitung

43



Felix Naumann | VL Datenbanksysteme II | SS 07

Kosten eines Plans

44

Wie immer: I/O-Kosten

- Beeinflusst durch
 - Wahl der logischen Operatoren um Anfrage umzustezen
 - Größe der Zwischenergebnisse
 - Wahl der physischen Operatoren
 - Reihenfolge der Operatoren
 - Insbesondere Joinreihenfolge
 - Datentransport
 - Pipelining vs. blocking

Felix Naumann | VL Datenbanksysteme II | SS 07

Schätzung der Statistiken

45

Statistiken sind notwendig um die Größe von Zwischenergebnissen zu berechnen

- Insbesondere $T(R)$ und $V(R,A)$
- Einholen der Statistiken auf Befehl des Administrators
- $T(R)$ mittels Scan von R
- $V(R,A)$ mittels einer der vorigen Algorithmen
 - Ähnlich Gruppierung, separat für jedes Attribut
- $B(R)$
 - Zählen, falls R clustered
 - Berechnen ($T(R)$ / Tupel pro Block) falls nicht clustered

Felix Naumann | VL Datenbanksysteme II | SS 07

Schätzung der Statistiken – Histogramme

46

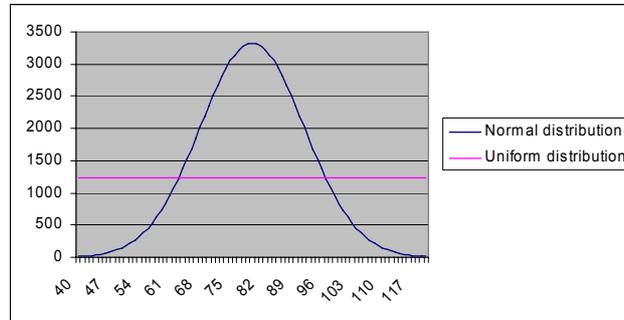
Histogramme stellen speichereffizient Werteverteilungen dar.

- Idee:
 - Fasse Gruppen von Werten (zusammenhängende Wertebereiche) in Buckets zusammen
 - Equal-width, Equal-height,
 - Idee: Speichere zusätzlich Häufigkeiten für häufigste Werte
 - Speichere pro Bucket die durchschnittliche Anzahl der Tupel pro Einzel-Wert.
 - D.h. man nimmt innerhalb des Buckets Gleichverteilung an.
- Vorteile
 - Geringere Schätzfehler, da Verteilungsannahmen nur in kleineren Bereichen
 - Geringer Speicherverbrauch durch Zusammenfassen in Gruppen
- Design und Wartung
 - Wie werden Bucketgrenzen bestimmt?
 - Was speichern wir pro Bucket?
 - Wie werden Histogramme aktuell gehalten?

Felix Naumann | VL Datenbanksysteme II | SS 07

Verteilungen – Körpergewicht

47

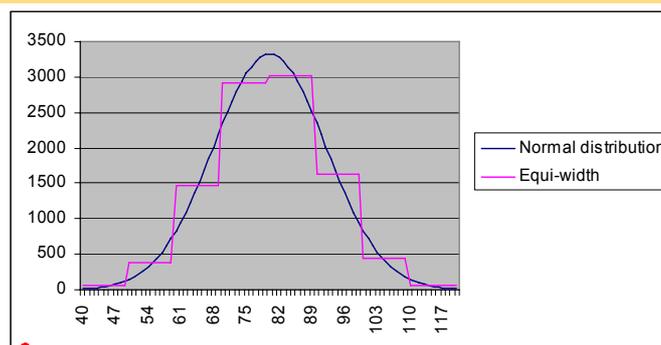


- Normalverteilung
 - Wertebereich: $120-40=80$, Mittelwert: 80, stddev: 12; 100000 people
- Gleichverteilung
 - $100000/80=1250$ for each possible weight
 - In fast jedem Bereich fehleranfällig

Felix Naumann | VL Datenbanksysteme II | SS 07

Equi-Width Histogramme

48

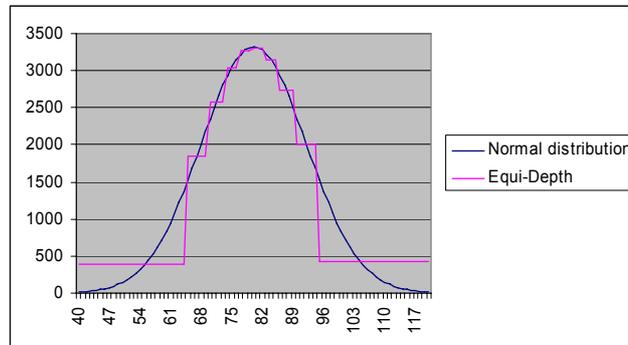


- Fest Anzahl Buckets, jeweils gleicher Breite
 - Vorteil: Grenzen müssen nicht mitgespeichert werden. Startpunkt und Breite genügen.
- Annahme: Gleichverteilung in jedem Bucket
- Berechnung mittels eines Scans von R
- Quellen für weitere Schätzfehler
 - Anzahl der Buckets
 - Werteverteilung innerhalb der Buckets

Felix Naumann | VL Datenbanksysteme II | SS 07

Equi-height-Histogramme

49



- Auch: Equal-depth-Histogramme bzw. Perzentile
- Feste Anzahl Buckets (z.B. 10 Stück für 10%, 20%, ...)
- Wähle Bucketgrenzen so, dass jeder Bucket ungefähr gleiche Anzahl an Tupeln enthält
 - Hier: Ca. 10.000 Personen pro Bucket
 - Bucketgrenzen müssen gespeichert werden
 - Berechnung durch Sortierung und gleichgroßen Sprüngen.

Felix Naumann | VL Datenbanksysteme II | SS 07

Histogramme zur Schätzung für Joins

50

- `SELECT Januar.Tag, Juli.Tag`
`FROM Januar, Juli`
`WHERE Januar.temp = Juli.temp`
- Schätzung pro Wertebereich
 - $\text{Histo1} \cdot \text{Histo2} / \text{Breite}$
- $10 \cdot 5 / 10 + 5 \cdot 20 / 10 = 5 + 10 = 15$
- Herkömmliche Schätzung
 - Je 245 Tupel mit gleichverteilten Temperaturen
 - $245 \cdot 245 / 100 = 600$ Tupel

Wertebereich Temperatur °F	Januar	Juli
0-9	40	0
10-19	60	0
20-29	80	0
30-39	50	0
40-49	10	5
50-59	5	20
60-69	0	50
70-79	0	100
80-89	0	60
90-99	0	10

Felix Naumann | VL Datenbanksysteme II | SS 07

Histogramme zur Schätzung für Joins

51

- Beispiel: $R(A,B) \bowtie S(B,C)$
- Histogramme listen jeweils die 3 häufigsten Werte und gruppieren den Rest.
 - Histogramm für R.B: 1: 200; 0:150; 5: 100; Rest: 550
 - Histogramm für S.B: 0: 100; 1: 80; 2: 70; Rest: 250
- $V(R,B) = 14$ und $V(S,B) = 13$
 - => Rest in R (550 Tupel) haben 11 verschiedene Werte
 - Annahme: je 50 Tupel
 - => Rest in S (250 Tupel) hat 10 verschiedene Werte
 - Annahme: je 25 Tupel
- Genau Schätzung für B-Werte 0 und 1: $150 \cdot 100 + 200 \cdot 80 = 31000$
- Schätzung: „2“ kommt 50 mal in R vor => $50 \cdot 70 = 3500$
- Vermutung: „5“ ist in R sehr häufig, kommt also vermutlich auch in S 25 mal vor: $100 \cdot 25 = 2500$
- 9 weitere Werte je in R und S: $9 \cdot (50 \cdot 25) = 11250$

Felix Naumann | VL Datenbanksysteme II | SS 07

Erhebung von Statistiken

52

Statistiken werden nur periodisch erhoben

- Statistiken ändern sich nicht laufend und radikal
- Auch falsche Statistiken funktionieren, wenn sie konsistent angewendet werden
- Statistiken sollen nicht selbst hot-spot werden.
 - Werden oft gelesen => Sollten nicht dauern geändert werden
- Auslösung
 - Regelmäßig periodisch
 - Nach einer festen Menge an Updates
 - Durch den Administrator
- Berechnung der Statistiken ist aufwändig. Lösung: Sampling

Felix Naumann | VL Datenbanksysteme II | SS 07

- Idee 1: Vollständige Enumeration
 - Entlang aller Freiheitsgrade
 - Reihenfolge und Gruppierung von assoziativen und kommutativen Operatoren
 - Wahl eines Algorithmus für jeden Operator
 - Zusätzliche Operatoren, die im logischen Plan nicht auftauchen
 - Modus des Datentransports zwischen Operatoren
 - Jeweils Kostenberechnung
 - Wahl des Plan mit geringsten Kosten
 - Problem: Zu viele Pläne
- Es gibt diverse bessere Methoden
 - Heuristische Auswahl, Branch-and-Bound, Hill-Climbing, Dynamische Programmierung, Selinger-Style

- Idee: Wende eine Sequenz bekannter Heuristiken an
- Z.B. für Joinreihenfolge (Greedy)
 - Wähle zuerst Joinpaar mit kleinstem Zwischenergebnis.
 - Join hinzu Relation, die wiederum kleinstes Zwischenergebnis erzeugt.
 - Usw.
- Weitere Heuristiken
 - Falls Selektion und Index auf Selektionsattribut: Wähle Index-scan
 - Führe mehrere Selektionen auf gleicher Relation zugleich aus.
 - Falls Index auf Joinattribut, wähle Index-Join
 - Falls ein Joininput sortiert ist, wähle sort-merge-join (falls kein Index vorhanden)

Branch and Bound & Hill-Climbing

55

- Branch and Bound
 - Idee: Verwende Heuristiken zum Finden eines guten Plans
 - Dessen Kosten bilden obere Schranke, auch für Teilpläne
 - Enumeriere Pläne für diverse Teile der Anfrage
 - » Verwerfe Teilpläne, die mehr als Schranke kosten
 - Senke Schranke falls besserer Gesamtplan gefunden wird.
 - Vorteil: Optimierung kann jederzeit abgebrochen werden.
- Hill-Climbing
 - Idee: Verwende Heuristiken zum Finden eines guten Plans
 - Suche schrittweise ähnliche Pläne mit niedrigeren Kosten
 - Ähnlich: Nur eine Änderung
 - Falls kein ähnlicher Plan besser ist: Fertig
 - Nachteil: Lokales Optimum vs. Globales Optimum
 - Varianten zur Verbesserung:
 - Iterative Improvement: Mit 10 verschiedenen Startplänen loslegen
 - Simulated Annealing: Lasse auch Verschlechterungen zu

Felix Naumann | VL Datenbanksysteme II | SS 07

Dynamische Programmierung & Selinger-style

56

- Dynamische Programmierung
 - Idee: Bottom-up Vorgehen im Baum
 - Suche jeweils besten Teilplan und verwende diese um höherem Teilplan zu bauen.
- Selinger-style
 - Idee: Erweiterung der Dynamischen Programmierung
 - Merke nicht nur besten Plan sondern verschiedene Sortiervarianten (interesting order)
 - Dürfen mehr kosten, haben aber später Vorteile
 - Beeinflusst nicht Kardinalitäten von Zwischenergebnisse, aber beeinflusst I/O-Kosten

Felix Naumann | VL Datenbanksysteme II | SS 07

Überblick

57

- (Parsing)
- (Algebraische Transformationsregeln)
- (Logische Anfragepläne)
- Kostenmodell
- Kostenbasierte Optimierung
- Joinreihenfolge
 - Dynamische Programmierung
 - Anwendung für Left-Deep Bäume
- Physische Anfragepläne



Anfragebearbeitung – Optimierung

58

Freiheitsgrade:

- Algebraische Anfrageumformung
- **Joinreihenfolge**
- Joinmethode
 - Nested Loop, Sort-Merge, Hash ...
- Indexzugriff oder Full-Table-Scan
- Operatorreihenfolge
- ...

59

Joinreihenfolge

- Wichtig hier: nur Joinreihenfolge, nicht Parallelität!
- Join ist i.d.R. teuerster Operator.
- Optimierung konzentriert sich auf beste Reihenfolge.
 - Weitere Optimierungsschritte (Selektionen nach unten, etc.) später
- Bei n Relationen $n!$ Alternativen
 - Aber: Die meisten Alternativen haben kartesisches Produkt.
- Viele Algorithmen
 - Dynamische Programmierung
- Bei mehr-Prozessor Systemen nicht nur Reihenfolge sondern auch Parallelisierung.

Felix Naumann | VL Datenbanksysteme II | SS 07

Richard Bellman

60

- 1920 – 1984
- PhD Princeton (3 Monate)
- Los Alamos (1944-1946)
- 1953 Rand Corporation: Erfindung der Dynamischen Programmierung
- Viele andere Beiträge zur Mathematik
 - Bellman-Ford Algorithmus

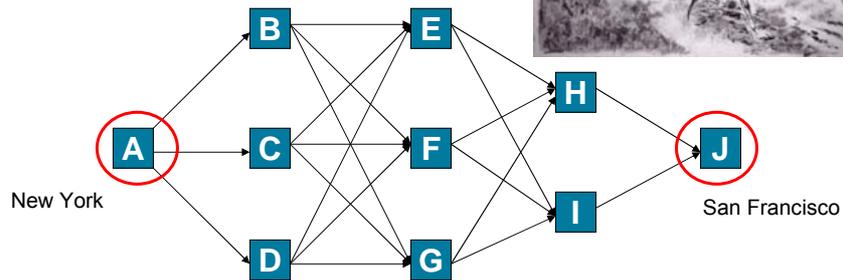


Felix Naumann | VL Datenbanksysteme II | SS 07

The Stagecoach story

61

some 150 years ago there was a salesman travelling west by stagecoach ..

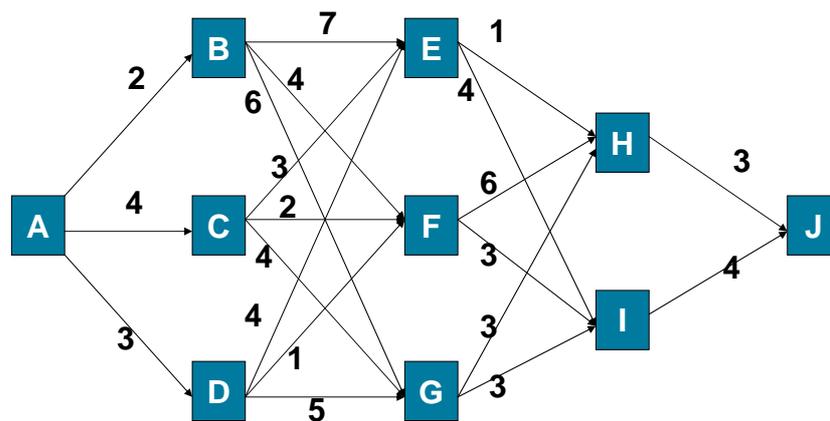


Quelle: Folien Ioana Popescu
<http://faculty.insead.edu/popescu/ioana/>

Felix Naumann | VL Datenbanksysteme II | SS 07

Versicherungskosten

62



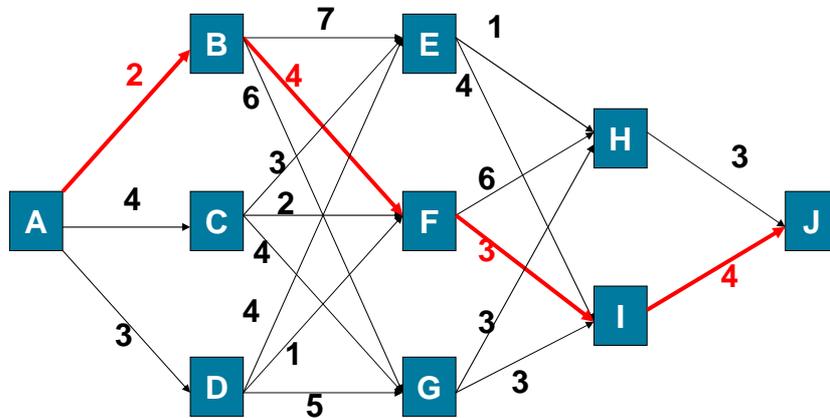
Frage: Welches ist der beste (billigste) Weg?

Quelle: Folien Ioana Popescu
<http://faculty.insead.edu/popescu/ioana/>

Felix Naumann | VL Datenbanksysteme II | SS 07

Billigster Weg – Greedy Suche

63



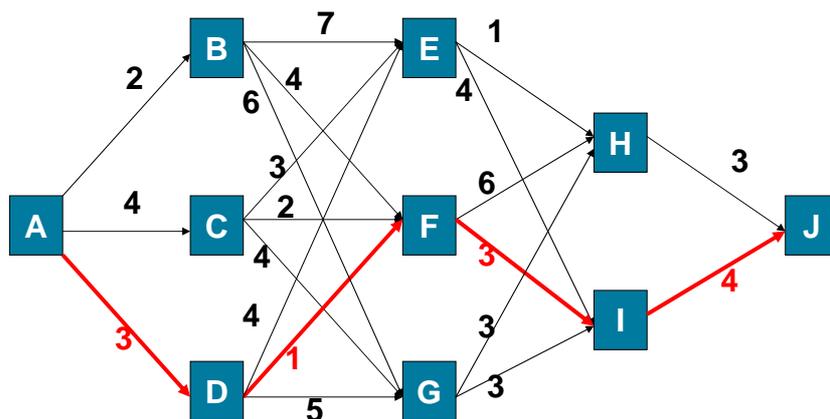
Greedy: A-B-F-I-J = 13

Frage: Gibt es einen besseren Weg?

Quelle: Folien Ioana Popescu
<http://faculty.insead.edu/popescu/ioana/>

Billigster Weg – Vollständige Suche

64



Besser: A-D-F-I-J = 11

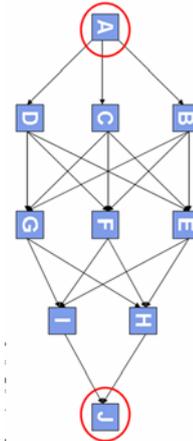
Anzahl der möglichen Wege: $3 \times 3 \times 2 = 18$

Quelle: Folien Ioana Popescu
<http://faculty.insead.edu/popescu/ioana/>

The Stagecoach Solution

65

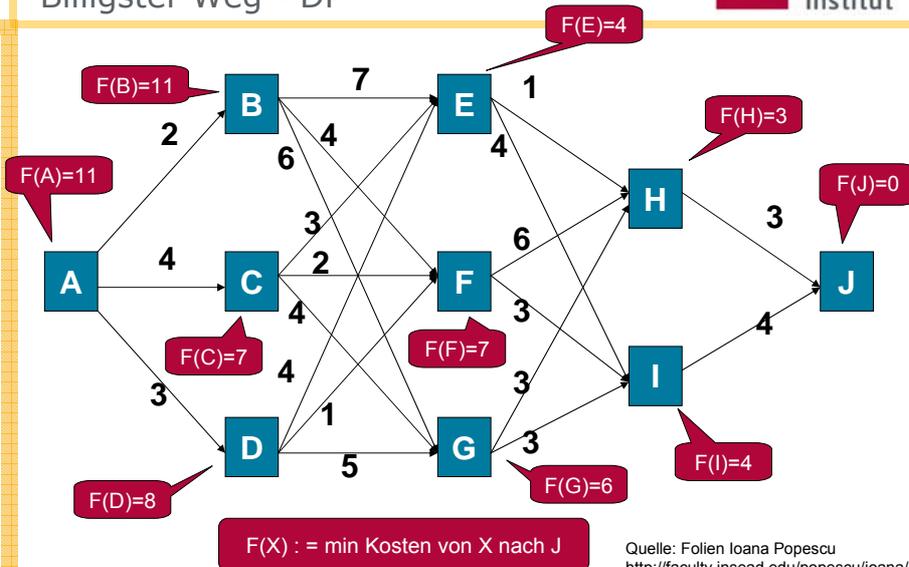
- Idee der Dynamischen Programmierung
 - Hier: "Rückwärtsberechnung"
- Voraussetzung: Prinzip der Optimalität
 - Teilplan eines optimalen Plans ist ebenfalls optimal
- Idee
 - Ausgehend vom Zielknoten stufenweise rückwärts beste Teilpfade berechnen
 - $F(X) :=$ minimale Kosten von X nach J



Quelle: Folien Ioana Popescu
<http://faculty.insead.edu/popescu/ioana/>

Billigster Weg - DP

66



Quelle: Folien Ioana Popescu
<http://faculty.insead.edu/popescu/ioana/>

Dynamische Programmierung

67

- Optimaler Algorithmus
- „Schwierigkeiten“
 - Prinzip der Optimalität muss gelten.
 - Aufteilung des Problems in Teilprobleme
- Aufwand kann exponentiell sein
- Klassische Anwendungen
 - Knapsack Problem
 - Traveling Salesman Problem
 - Maschinenbelegung
 - Transportproblem

Nun: Anwendung auf die Anfrageplanung

Felix Naumann | VL Datenbanksysteme II | SS 07

Überblick

68

- (Parsing)
- (Algebraische Transformationsregeln)
- (Logische Anfragepläne)
- Kostenmodell
- Kostenbasierte Optimierung
- Joinreihenfolge
 - Dynamische Programmierung
- ➔ □ Anwendung für Left-Deep Bäume
- Physische Anfragepläne



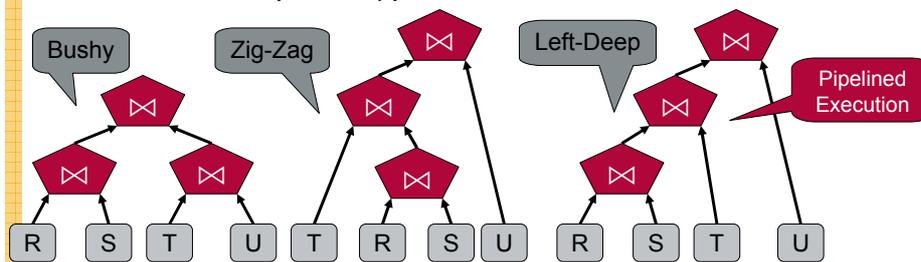
Felix Naumann | VL Datenbanksysteme II | SS 07

Anfrageplanung

69

Heuristische Einschränkung des Suchraums

- Keine Kreuzprodukte
 - Außer explizite Kreuzprodukte in der Anfrage
- Prädikate so früh wie möglich
- Nur links-tiefe (left-deep) Bäume



Felix Naumann | VL Datenbanksysteme II | SS 07

Dynamische Programmierung: Optimierung im System-R

70

- A.k.a. "Selinger-style query optimization"
 - Der klassische Artikel zur Anfrageoptimierung: [SAC+79]
 - Ursprünglich im IBM System-R
 - Heutzutage weit verbreitet
- Grundidee:
 - Nur "Left-deep" Anfragebäume
 - D.h. nur Joinreihenfolge interessant
 - Innere und äußere Relation unberücksichtigt
 - Bottom-up Generierung von Anfrageplänen
 - Dynamische Programmierung (DP)
 - Zusätzlich: interesting orders (interessante Sortierungen)
 - (Zusätzlich: interesting sites (interessante Ausführungsorte))



Felix Naumann | VL Datenbanksysteme II | SS 07

Quelle: u.a. Folien Prof. Chen Li

Access Path Selection
in a Relational Database Management System

P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
K. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and complex queries (such as joins), given a user specification of desired data as a boolean expression of predicates. System R is an experimental database management system developed to carry out research on the relational model of data. System R was designed and built by members of the IBM San Jose Research Laboratory.

1. Introduction

System R is an experimental database management system based on the relational model of data which has been under development at the IBM San Jose Research Laboratory since 1975 <1>. The software was developed as a research vehicle in relational database, and is not generally available outside the IBM Research Divi-

retrieval. Nor does a user specify in what order joins are to be performed. The System R optimizer chooses both join order and an access path for each table in the SQL statement. Of the many possible choices, the optimizer chooses the one which minimizes "total access cost" for performing the entire statement.

This paper will address the issues of access path selection for queries. Retrieval for data manipulation (UPDATE, DELETE) is treated similarly. Section 2 will describe the place of the optimizer in the processing of a SQL statement, and section 3 will describe the storage component access paths that are available on a single physically stored table. In section 4 the optimizer cost formulas are introduced for single table queries, and section 5 discusses the joining of two or more tables, and their corresponding costs. Nested queries (queries in predicates) are covered in section 6.

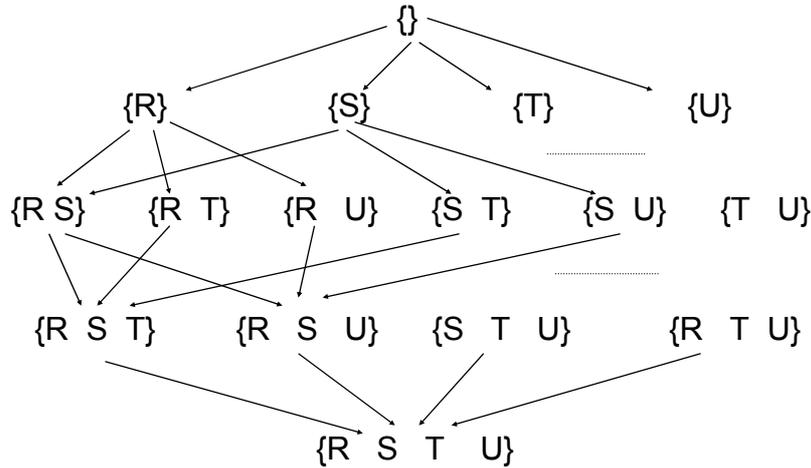
Bottom-up Anfrageplangenerierung



72

- Grundannahme 1:
Nach dem Join über k Relationen ist die Join-Methode die $k+1$ te Relation um hinzuzujoinen unabhängig von den vorigen Join-Methoden.
 - Joinmethoden: Nested Loops, Hashjoin, Sort-Merge Join usw.
- Grundannahme 2:
Jeder Teilplan eines optimalen Plans ist ebenfalls optimal.
 - Entspricht dem Prinzip der Optimalität: Wenn sich zwei Pläne nur in einem Teilplan unterscheiden, so ist der Plan mit dem besseren Teilplan auch der bessere Gesamtplan
- Bottom-up Anfrageplangenerierung:
 - Berechne die optimalen Pläne für den Join über (jede Kombination von) k Relationen
 - Suboptimale Pläne werden verworfen
 - Erweitere diese Pläne zu optimalen Plänen für $k+1$ Relationen.
 - usw. bis $k = n$

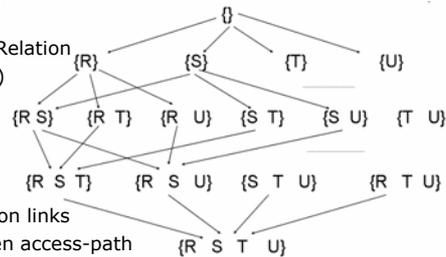
73



DP – Grundidee für Anfrageoptimierung

74

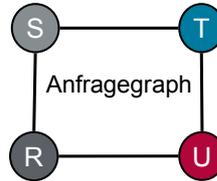
- Für jede Kombination merke (in einer Hilfstabelle):
 - Geschätzte Größe des Ergebnisses (Kardinalität)
 - Geschätzte minimale Kosten
 - Hier zur Vereinfachung: Größe des Zwischenergebnisses
 - Joinreihenfolge, die diese Kosten verursacht (= optimaler Teilplan)
- Induktion über Anzahl der Relationen im Plan
 - N=1: Für jede Relation
 - Kardinalität = Kardinalität der Relation
 - Kosten = 0 (zur Vereinfachung)
 - Joinreihenfolge: n/a
 - N=2: Für jedes Relationenpaar R, S
 - Kardinalität = |R| x |S| x sf
 - Kosten = 0
 - Joinreihenfolge: kleinere Relation links
 - Clou: R und S jeweils mit besten access-path
 - N=3: Für jedes Tripel R, S, T
 - Clou: Nur bestes Relationenpaar aus dem Tripel wird um dritte Relation ergänzt
 - ...



DP – Beispiel

75

- Anfrage über Relationen R, S, T, U.
- Vier Join-Bedingungen



{R}	{S}	{T}	{U}
1000	1000	1000	1000
0	0	0	0
scan(R)	scan(S)	scan(T)	scan(U)

u.U. auch IndexScan(...)

DP – Beispiel

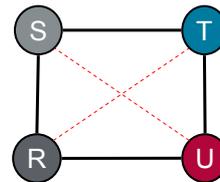
76

	{R,S}	{R,T}	{R,U}	{S,T}	{S,U}	{T,U}
Kardinalität	5000	1M	10000	2000	1M	1000
Kosten	0	0	0	0	0	0
opt. Plan	R ⋈ S	R ⋈ T	R ⋈ U	S ⋈ T	S ⋈ U	T ⋈ U

Kreuzprodukte nicht berücksichtigen!

	{R,S,T}	{R,S,U}	{R,T,U}	{S,T,U}
Kardinalität	10000	50000	10000	2000
Kosten	2000	5000	1000	1000
opt. Plan	(S ⋈ T) ⋈ R	(R ⋈ S) ⋈ U	(T ⋈ U) ⋈ R	(T ⋈ U) ⋈ S

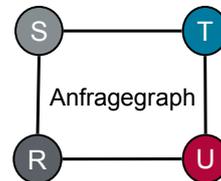
Besser als z.B. S ⋈ (T ⋈ R) oder (R ⋈ S) ⋈ T



DP – Beispiel

77

	{R,S,T}	{R,S,U}	{R,T,U}	{S,T,U}
Kardinalität	10000	50000	10000	2000
Kosten	2000	5000	1000	1000
opt. Plan	$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(T \bowtie U) \bowtie S$



Plan	Kosten
$((S \bowtie T) \bowtie R) \bowtie U$	12k
$((R \bowtie S) \bowtie U) \bowtie T$	55k
$((T \bowtie U) \bowtie R) \bowtie S$	11k
$((T \bowtie U) \bowtie S) \bowtie R$	3k

Bisher unberücksichtigt:
Wahl des Join-Algorithmus

Optimaler (left-deep) Plan

DP - interesting orders (Interessante Sortierung)

78

- WdH.: Prinzip der Optimalität: Wenn sich zwei Pläne nur in einem Teilplan unterscheiden, so ist der Plan mit dem besseren Teilplan auch der bessere Gesamtplan.
- Gegenbeispiel?
 - $R(A,B) \bowtie S(A,C) \bowtie T(A,D)$
 - Bester (lokaler) Plan für $R \bowtie S$: Hash-Join
 - Best (globaler) Gesamtplan:
 - 1. Sort-merge Join über R und S
 - 2. Sort-merge Join mit T
- Warum könnte dies so sein?
 - Das Zwischenergebnis von $R \bowtie_{\text{sort-merge}} S$ ist nach Join-Attribut A sortiert.
 - Dies ist eine interesting order, die später ausgenutzt werden kann:
 - Spätere sort-merge Joins
 - Gruppierung (GROUP BY)
 - Sortierung (ORDER BY)
 - Eindeutige Tupel (DISTINCT)

DP - interesting orders (Interessante Sortierung)

79

- Bei Auswahl des besten Teilplans:
 - Kostenvergleich genügt nicht.
 - Es gibt keine vollständige Ordnung der Teilpläne nach Kosten.
 - Auch Sortierungen müssen berücksichtigt werden.
- Lösung: Für jede Kombination von Relationen, speichere mehrere Sortiervarianten:
 - Nach jeder Variante der beteiligten Teilpläne
 - Die "leere" Sortierung
 - DP Tabellen werden „breiter“.
- Merke außerdem Join- und Sortieroperationen, die diese Sortierung erzeugen.

Felix Naumann | VL Datenbanksysteme II | SS 07

DP – Algorithmus

80

Input: SPJ query q on relations R_1, \dots, R_n

Output: A query plan for q

```
1: for  $i = 1$  to  $n$  do {
2:    $optPlan(\{R_i\}) = accessPlans(R_i)$ 
3:    $prunePlans(optPlan(\{R_i\}))$ 
4: }
5: for  $i = 2$  to  $n$  do {
6:   for all  $S \subseteq \{R_1, \dots, R_n\}$  such that  $|S| = i$  do {
7:      $optPlan(S) = \emptyset$ 
8:     for all  $O \subset S$  do {
9:        $optPlan(S) = optPlan(S) \cup joinPlans(optPlan(O), optPlan(S - O))$ 
10:       $prunePlans(optPlan(S))$ 
11:    }
12:  }
13: }
14: return  $optPlan(\{R_1, \dots, R_n\})$ 
```

Alle Zugriffspläne für jede Relation

Schlechtere Zugriffspläne verwerfen

Achtung: Nicht left-deep!

Quelle: [Ko00]

Felix Naumann | VL Datenbanksysteme II | SS 07

Überblick

81

- (Parsing)
- (Algebraische Transformationsregeln)
- (Logische Anfragepläne)
- Kostenmodell
- Kostenbasierte Optimierung
- Joinreihenfolge
 - Dynamische Programmierung
- Physische Anfragepläne



Letzte Schritte

82

- Wahl des jeweiligen Algorithmus
 - Wenn nicht schon zuvor (z.B. bei DP) geschehen
 - Hier nur beispielhaft: Selektion und Join
- Pipelining vs. Blocking
- Zugriffsmethoden für Relationen

Wahl der Selektionsmethode

83

- Schon kennengelernt
 - Variante 1: Ganz R lesen und Selektionsbedingung auf jedes Tupel anwenden
 - Variante 2: Falls Index auf Selektionsattribut vorhanden: Zugriff über Index
 - Voraussetzung: Index und Gleichheitsbedingung
- Jetzt: Verallgemeinerung auf mehrere Selektionen auf verschiedenen Attributen
 - Mit oder ohne Index
 - Gleichheit oder Ungleichheit ($<$, $>$, \leq , \geq , \neq)

Wahl der Selektionsmethode

84

- Annahme: Mindestens eine Selektionsbedingung kann einen Index verwenden.
- Vorgehen mit Indizes (jeweils viele Alternativen):
 - Verwende Indizes um Schnittmenge der Pointermengen zu ermitteln
 - Lese diese Tupel ein (Index-scan)
 - Wende darauf „Filter“-Operator an: Prüft alle übrigen Bedingungen
- Vorgehen ohne Indizes
 - Table-scan für ganz R
 - Wende Filter-Operator für alle Bedingungen an
- Filter-Operator findet nur im Hauptspeicher statt: Keine Kosten
- Jetzt: Kostenvergleich der Alternativen

Kostenvergleich der Selektionsmethoden

85

- Bisher: Kostenschätzung durch Schätzung der Ergebnisgröße
 - Kardinalität des Zwischenergebnisses
- Jetzt: Nur Implementierungsvarianten mit jeweils gleichem Ergebnis
 - Deshalb wieder: Disk I/Os
 - Annahme: Indizes kosten nichts (da sehr kleine Mengen)
- Beispiel: $\sigma_{A=10, B<20}(R)$
 - Variante 1: Tablescan
 - B(R) falls R clustered
 - T(R) falls R nicht clustered
 - Variante 2: Index auf A verwenden
 - B(R)/V(R,A) falls Index clustering
 - T(R)/V(R,A) falls Index nicht clustering
 - Variante 3: Index auf B verwenden
 - B(R)/3 falls Index clustering
 - T(R)/3 falls Index nicht clustering

Felix Naumann | VL Datenbanksysteme II | SS 07

Kostenvergleich der Selektionsmethoden – Beispiel

86

- $\sigma_{X=1, Y=2, Z<5}(R)$
 - T(R) = 5000, B(R) = 200, V(R,X)=100, V(R,Y)=500
 - R sei clustered
 - Indizes auf X und Y nicht clustering
 - Index auf Z clustering
- Variante 1: Table-scan und Filter
 - Kosten: B(R) = 200 I/Os
- Variante 2: Index-scan mit X-Index; Filter für den Rest
 - Kosten: T(R)/V(R,X) = 5000/100 = 50 I/Os
- Variante 3: Index-scan mit Y-Index; Filter für den Rest
 - Kosten: T(R)/V(R,Y) = 5000/500 = 10 I/Os
- Variante 4: Index-scan mit (clustering) Z-Index; Filter für den Rest
 - Kosten: B(R)/3 = 200/3 = 67 I/Os

Felix Naumann | VL Datenbanksysteme II | SS 07

Wahl der Join-Methode

87

- Kosten je nach Joinmethode (siehe voriger Foliensatz)
 - Annahme: Man kennt M (verfügbarer Hauptspeicher)
 - Und M ändert sich nicht während der Ausführung
 - Annahme: Man kennt B(R), T(R), V(R, ...)
- Ideen falls Annahmen nicht stimmen
 - One-pass oder Nested-loop Algorithmus als default
 - Prinzip „Hoffnung“
 - Wähle Sort-merge-join falls mindestens ein Input bereits nach Joinattribut sortiert ist.
 - Wähle Sort-merge-join bei mehr als ein Join auf gleichem Attribut
 - $(R(A,B) \bowtie S(B,C)) \bowtie R(A,D)$
 - $R(A,B) \bowtie S(B,C)$: Falls R klein und Index auf S.B: Wähle Index-Join
 - Falls weder Sortierung noch Indizes vorhanden sind: Wähle Hash-Join
 - Kosten hängen nur von kleinerem Input ab, nicht von beiden Inputs
- Analoge Überlegungen für Mengenoperationen

Felix Naumann | VL Datenbanksysteme II | SS 07

Pipelining vs. Blocking

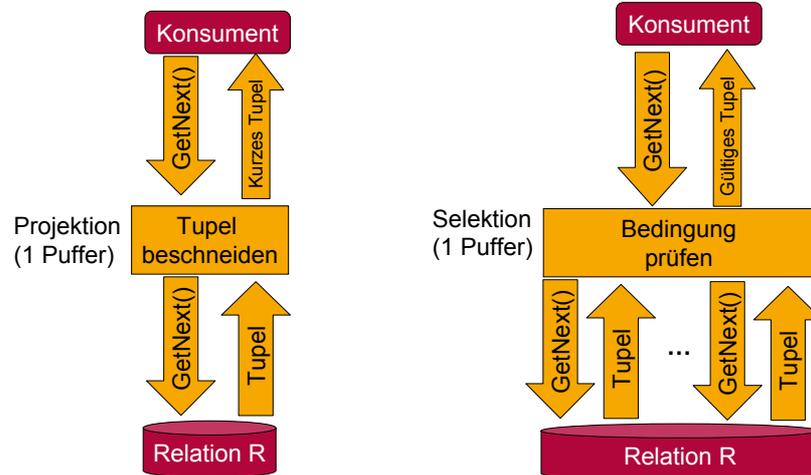
88

- Naive: Blocking (auch „Materialisierend“)
 - Jeder Operator speichert sein Zwischenergebnis auf Disk
- Besser: Vermischung der Ausführung verschiedener Operatoren
 - Pipelining
 - Kette von Iteratoren
- Vorteile von Pipelining
 - Weniger I/O
 - Frühe Ergebnisse bei der Anwendung
- Nachteile des Pipelining?
 - Nicht jeder Operator funktioniert
 - Jeder Operator hat weniger Hauptspeicher
 - => Thrashing
- Pipelining also nicht immer besser!

Felix Naumann | VL Datenbanksysteme II | SS 07

Pipelines mit unären Operatoren

89

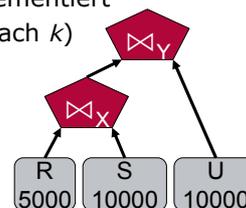


Felix Naumann | VL Datenbanksysteme II | SS 07

Pipelines mit binären Operatoren

90

- 1 Puffer für Output aber mehrere Puffer für Input
- Nun: Betrachtung des shared memory am Beispiel
- $(R(W,X) \bowtie S(X,Y)) \bowtie U(Y,Z)$
 - $B(R) = 5000, B(S) = B(U) = 10000$
 - Zwischenergebnis $R \bowtie S$ nehme k Blöcke ein
 - Wird variieren
 - Beide Joins seien als Hash-join implementiert
 - One-pass oder two-pass (je nach k)
 - $M = 101$



Felix Naumann | VL Datenbanksysteme II | SS 07

Pipelines mit binären Operatoren – Beispiel

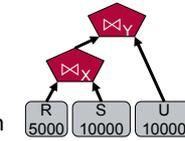
91

■ Zunächst $R \bowtie S$

- Keines passt in Hauptspeicher => Two-pass Hash-join
- Pass 1: $M = 101 \Rightarrow 100$ Buckets => Jeder Bucket von R hat 50 Blocks
- Pass 2 benötigt 51 Puffer => 50 verbleiben für Join mit U

■ Variation von k (Blöcke des Zwischenergebnisses $R \bowtie S$)

- $k \leq 49$
 - Pipeline der Ergebnis Tupel in freien Speicher unter Aufbau einer Hashtabelle
 - Mindestens ein freier Puffer verbleibt zum Einlesen von U (also one-pass)
 - I/O: 45000 für two-pass hashjoin + 10000 für U lesen
- $49 < k \leq 5000$
- $k > 5000$

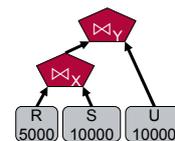


Pipelines mit binären Operatoren – Beispiel

92

■ $49 < k \leq 5000$

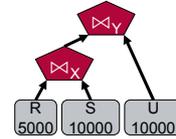
- Wie vorher: Pipeline des Ergebnisses von $R \bowtie S$
- Aber: Join mit U als 50-bucket two-pass Hash-join
- 1. Hashe U in 50 Buckets á 200 Blocks
- 2. Two-pass Hashjoin von $R \bowtie S$ auf 51 buckets
- 3. Schreibe Joinergebnisse in einen der 50 U-buckets (wie immer auf Disk schreiben, falls voll)
- 4. Bucket-weise Join von $R \bowtie S$ mit U
 - $k \leq 5000 \Rightarrow (R \bowtie S)$ -buckets maximal 100 groß
- Kosten:
 - 20000 für Lesen und bucket-Schreiben von U
 - 45000 für two-pass hash-join $R \bowtie S$
 - k zum Schreiben der buckets von $R \bowtie S$
 - $K + 10000$ zum Lesen der $(R \bowtie S)$ -buckets und probing mit U
- Zusammen: $75000 + 2k$



Pipelines mit binären Operatoren – Beispiel

93

- $k > 5000$
 - Hashing von $R \bowtie S$ klappt nicht mehr: Einer der 50 Buckets passt nicht mehr in Hauptspeicher
- Variante 1: three-pass Join; Kosten $20000 + 2k$ zusätzlich
 - 2 I/O für jede Seite (für U und für $R \bowtie S$)
 - Zusammen: $95000 + 4k$
- Variante 2: Auf pipeline verzichten
 - $R \bowtie S$ mit two-pass Hash-join
 - Ergebnis auf Disk schreiben
 - Join mit U mittels two-pass Hash-join
 - Da $B(U) = 10000$: Build mit 100 buckets ist möglich
 - Also eigentlich $U \bowtie (R \bowtie S)$
 - Kosten:
 - $45000 + k$ für $R \bowtie S$ und anschließendes Schreiben
 - $30000 + 3k$ für two-pass Hash-join mit U
 - Zusammen: $75000 + 4k$



Felix Naumann | VL Datenbanksysteme II | SS 07

Überblick

94

- (Parsing)
- (Algebraische Transformationsregeln)
- (Logische Anfragepläne)
- Kostenmodell
- Kostenbasierte Optimierung
- Joinreihenfolge
 - Dynamische Programmierung
- Physische Anfragepläne
- Picasso: Erratische Optimierer
<http://dsl.serc.iisc.ernet.in/projects/PICASSO/picasso.html>



Felix Naumann | VL Datenbanksysteme II | SS 07

Example Query [Q7 of TPC-H]

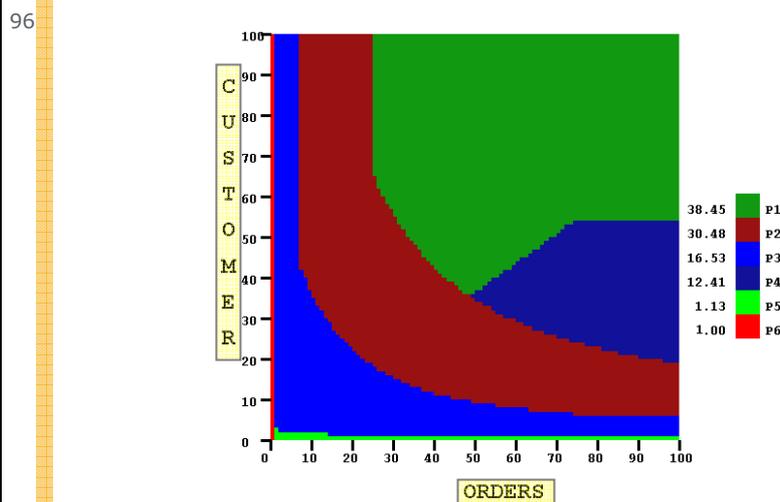
```

95 select
    supp_nation, cust_nation, l_year, sum(volume) as revenue
  from
    (select n1.n_name as supp_nation, n2.n_name as cust_nation,
      extract(year from l_shipdate) as l_year,
      l_extendedprice * (1 - l_discount) as volume
    from supplier, lineitem, orders, customer, nation n1, nation n2
   where s_suppkey = l_suppkey and o_orderkey = l_orderkey and
      c_custkey = o_custkey and s_nationkey = n1.n_nationkey
      and c_nationkey = n2.n_nationkey and
      ((n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY') or
      (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE')) and
      l_shipdate between date '1995-01-01' and date '1996-12-31'
      and o_totalprice < C1 and c_acctbal < C2 ) as shipping
  group by supp_nation, cust_nation, l_year
  order by supp_nation, cust_nation, l_year

```

Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
 Felix Naumann | VL Datenbanksysteme II | SS 07

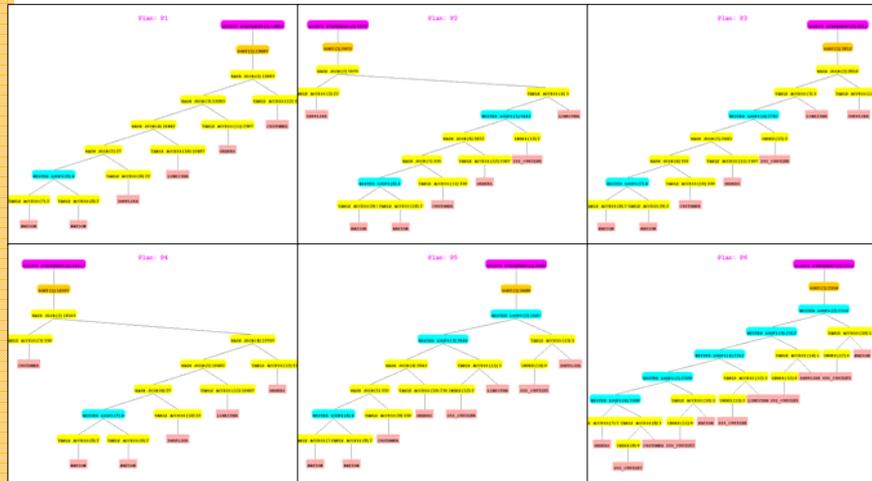
Example Plan Diagram



Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
 Felix Naumann | VL Datenbanksysteme II | SS 07

Specific Plan Choices

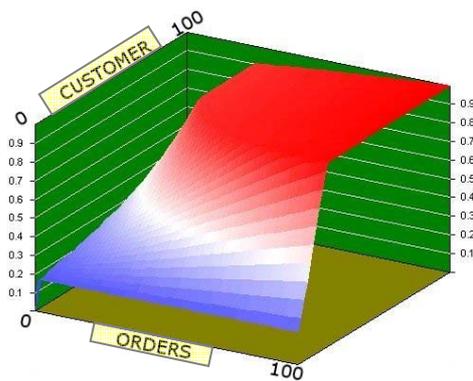
97



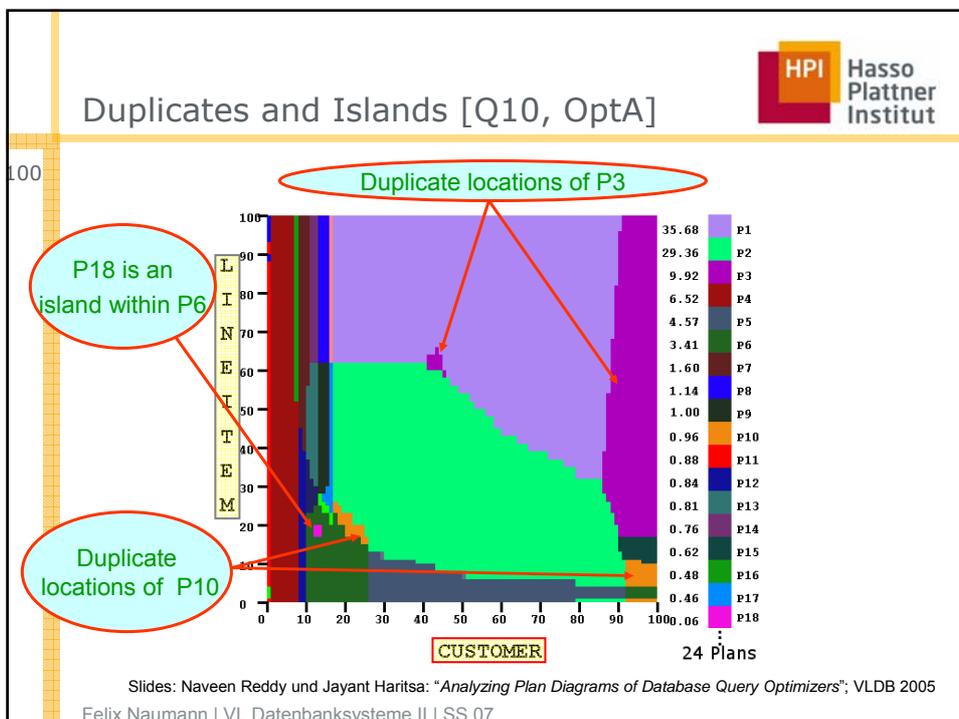
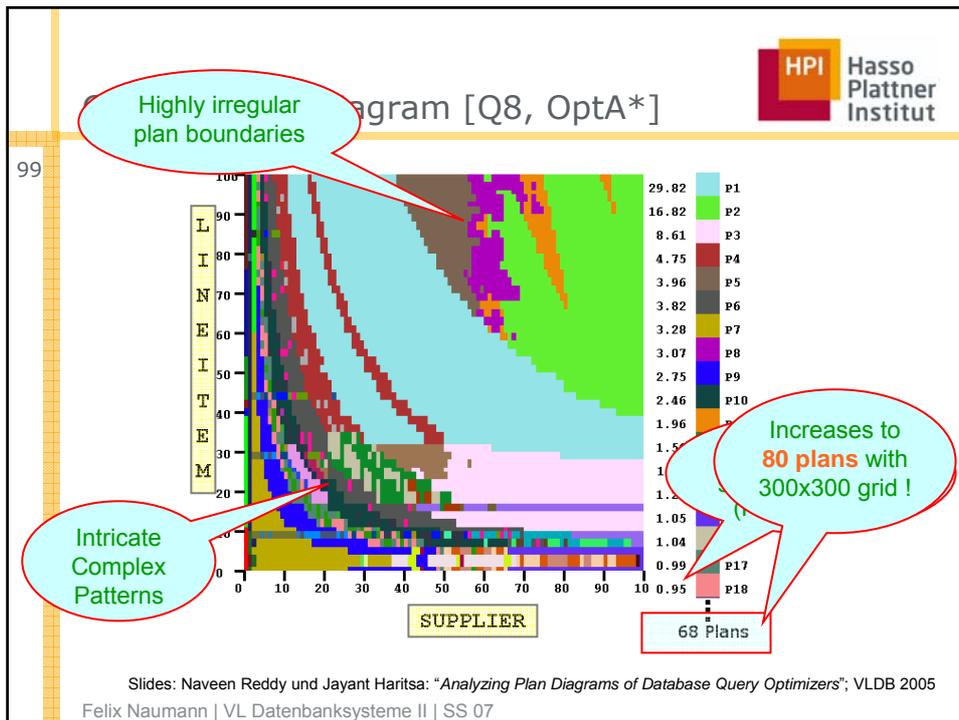
Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
 Felix Naumann | VL Datenbanksysteme II | SS 07

Example Cost Diagram

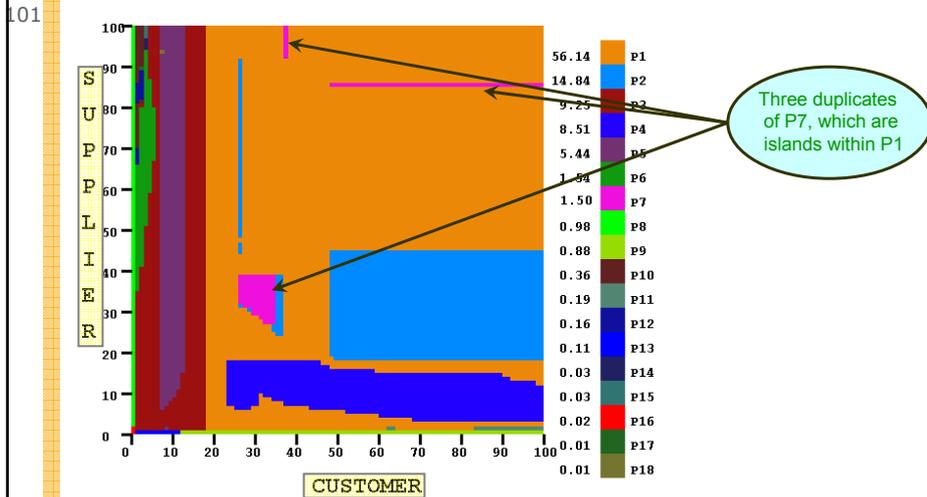
98



Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
 Felix Naumann | VL Datenbanksysteme II | SS 07

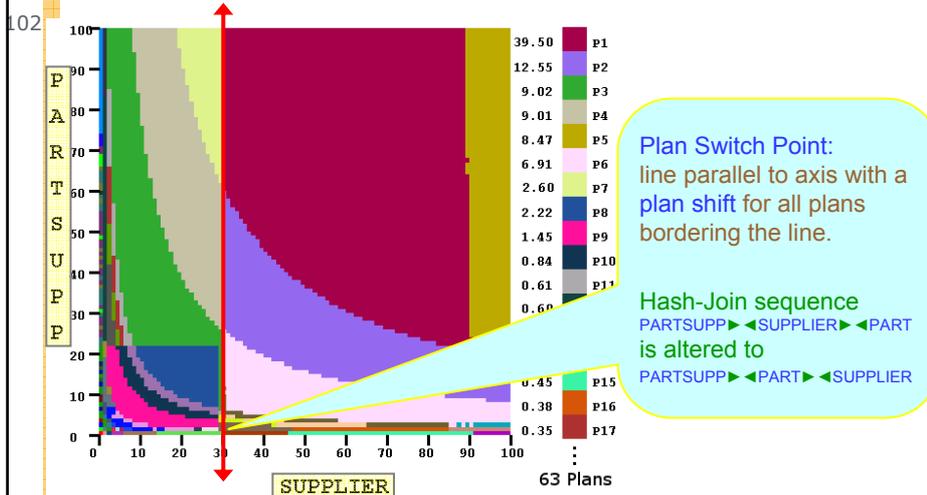


Duplicates and Islands [Q5, OptC]



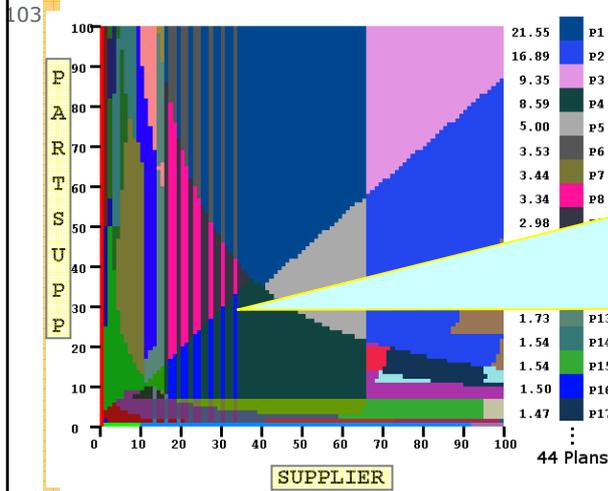
Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
 Felix Naumann | VL Datenbanksysteme II | SS 07

Plan Switch Points [Q9, OptA]



Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
 Felix Naumann | VL Datenbanksysteme II | SS 07

Venetian Blinds [Q9, Opt B]

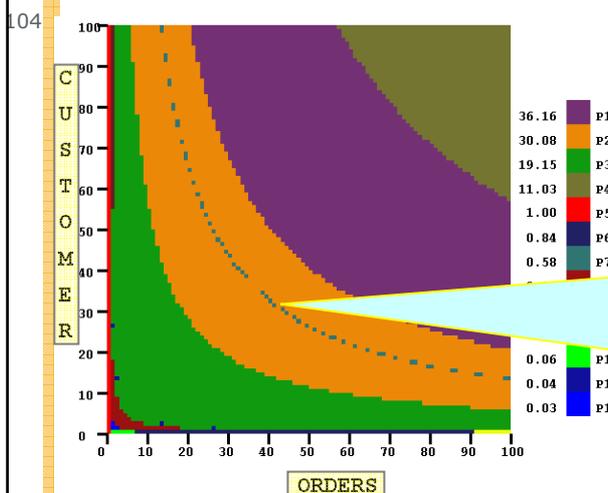


Six plans simultaneously change with rapid alternations to produce a "Venetian blinds" effect.

Left-deep hash join across NATION, SUPPLIER and LINEITEM relations gets replaced by a right-deep hash join.

Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
Felix Naumann | VL Datenbanksysteme II | SS 07

Footprint Pattern [Q7, Opt A]

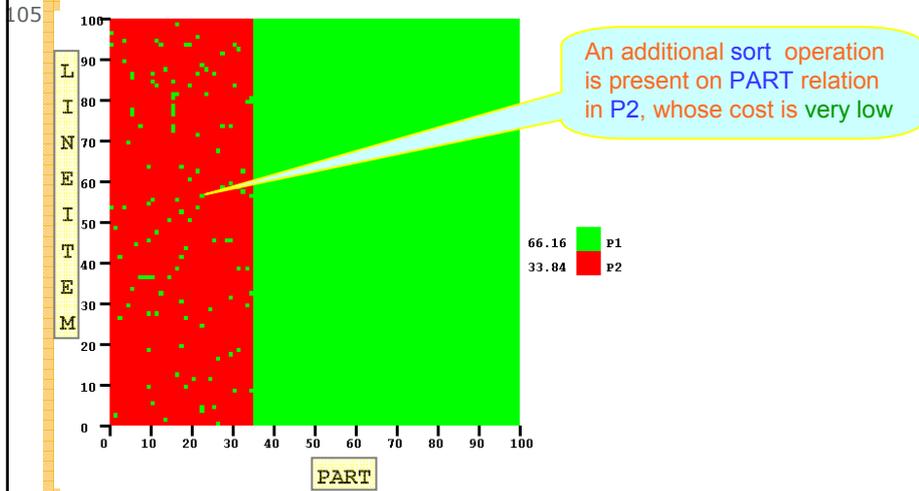


P7 exhibits a thin broken curved pattern in the middle of P2's region.

P2 has sort-merge-join at the top of the plan tree, while P7 has hash-join

Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
Felix Naumann | VL Datenbanksysteme II | SS 07

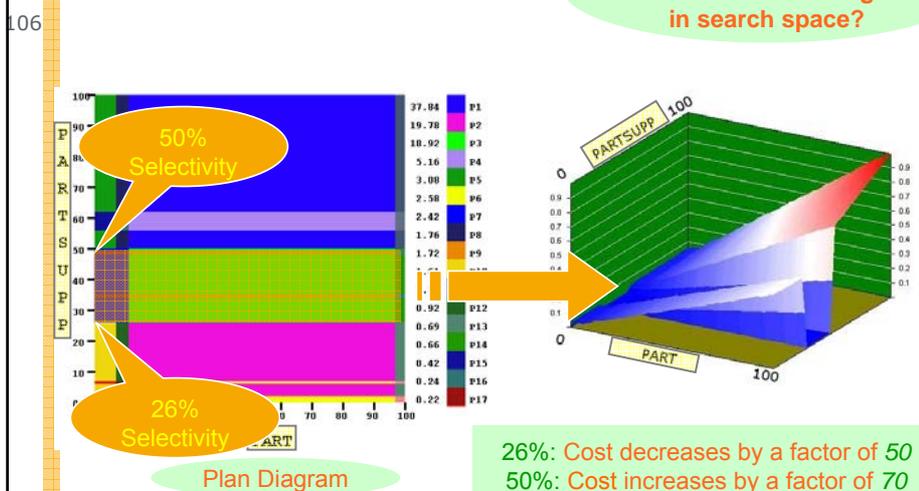
Speckle Pattern [Q17, Opt A*]



Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
 Felix Naumann | VL Datenbanksysteme II | SS 07

Plan-Switch Non-Monotonic Costs [Q2, Opt A]

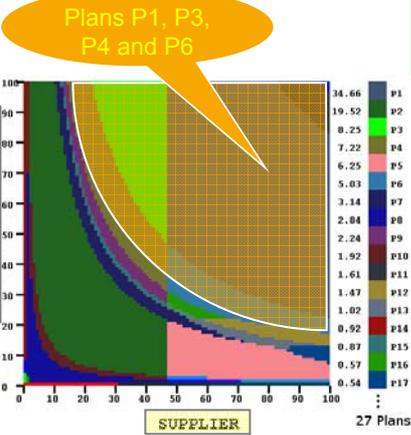
Presence of Rules?
 Parameterized changes
 in search space?



Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
 Felix Naumann | VL Datenbanksysteme II | SS 07

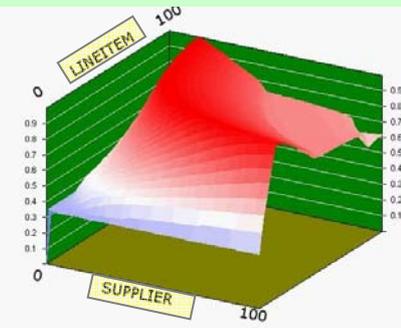
Intra-Plan Non-Monotonic Costs [Q21, Opt A]

107



Plan Diagram

Nested loops join whose cost decreases with increasing input cardinalities



Cost Diagram

Slides: Naveen Reddy und Jayant Haritsa: "Analyzing Plan Diagrams of Database Query Optimizers"; VLDB 2005
 Felix Naumann | VL Datenbanksysteme II | SS 07