

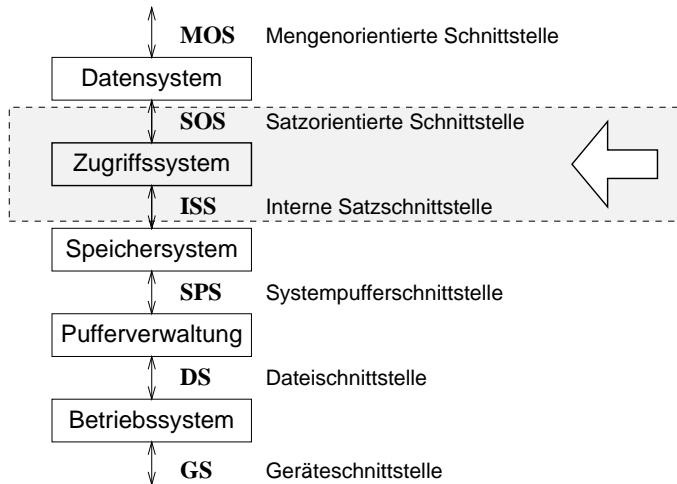
Teil VIII

Anfragebearbeitung und -optimierung

Anfragebearbeitung und -optimierung

- 1 Datenbankparameter & Grundalgorithmen
- 2 Unäre und binäre Operationen
- 3 Berechnung von Verbunden
- 4 Optimierung: Motivation und Phasen
- 5 Übersetzung und Vereinfachung
- 6 Logische Optimierung
- 7 Physische Optimierung

Einordnung



Parameter für Kostenbestimmung

- Aufwandsabschätzung basierend auf Informationen aus Data Dictionary
 - ▶ *b*size: Blockgröße; *mem*: Puffergröße in Anzahl der Blöcke
 - ▶ $|r|$: Anzahl Tupel in der Relation *r*
 - ▶ b_r : Anzahl von Blöcken, die Tupel aus *r* beinhalten
 - ▶ $size_r$: (mittlere) Größe von Tupeln aus *r*
 - ▶ f_r : Blockungsfaktor –wieviel Tupel aus *r* können in einem Block gespeichert werden:
$$f_r = \frac{bsize}{size_r}$$
bei kompakter Speicherung (nur eine Relation pro Block)

$$b_r = \left\lceil \frac{|r|}{f_r} \right\rceil$$

- ▶ $val_{A,r}$: Anzahl verschiedener Werte für Attribut *A* in *r*
- ▶ $lev_{I(R(A))}$: Anzahl der Indexebenen eines B⁺-Baums für Index $I(R(A))$

Grundannahmen

- Indexe B^+ -Bäume
- dominierender Kostenfaktor: Blockzugriff (auch für Zwischenergebnisse)
- Zugriff auf Hintergrundspeicher auch für Zwischenergebnisse
- Zwischenrelationen zunächst für jede Grundoperation
- Zwischenrelationen hoffentlich zum großen Teil im Puffer
- einige Operationen (Mengenoperationen) auf Adressmengen (TID-Listen)

Hauptspeicheralgorithmen

- wichtig für den Durchsatz des Gesamtsystems, da sie sehr oft eingesetzt werden
 - ▶ *Tupelvergleich*
(Duplikate erkennen, Sortierordnung angeben, ...) iterativ durch Vergleich der Einzelattribute, Attribute mit großer Selektivität zuerst
 - ▶ *TID-Zugriff*
TID innerhalb des Hauptspeichers: übliche Vorgehensweise bei der Auflösung indirekter Adressen

Zugriffe auf Datensätze

- *Relationen*: interner Identifikator `RelID`
- *Indexe*: interner Identifikator `IndexID`
 - ▶ *Primärindex*, etwa $I(\text{KUNDE}(\text{KN}_r))$
bei Bedingung $A = a$ wird maximal ein Tupel pro Zugriff zurück geliefert
 - ▶ *Sekundärindex*, etwa $I(\text{BESTELLUNG}(\text{KN}_r))$
Bsp.: $\text{KN}_r = 4711$ liefert i.a. mehrere Tupel

Indexzugriffe: Ergebnis TID-Listen (aus den Blättern des entsprechenden B^+ -Baums)

Zugriffe auf Datensätze /2

- **fetch-tuple** Direktzugriff auf Tupel mittels TID-Wertes holt Tupel in *Tupelpuffer*

fetch-tuple(RelID, TID) \rightarrow Tupel-Puffer

- **fetch-TID**: TID zu (Primärschlüssel-)Attributwert bestimmen

fetch-TID(IndexID, Attributwert) \rightarrow TID

- weiterhin auf Relationen und Indexen: *Scans*

Beispiel in SQL

```
select *  
from KUNDE  
where KNr = 4711
```

- Gleichheitsanfrage über einen Schlüssel

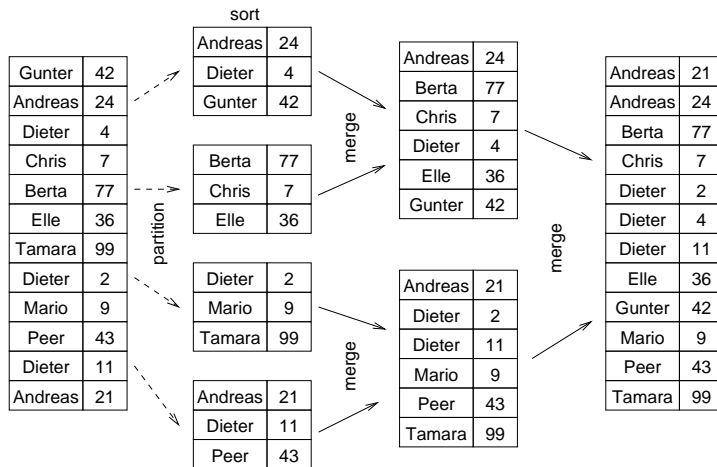
```
aktuellerTID :=  
    fetch-TID(KUNDE-KNr-Index, 4711);  
aktuellerPuffer:=  
    fetch-tuple(KUNDE-RelID, aktuellerTID);  
put(aktuellerPuffer);
```

- **put**: hier Anzeige des Ergebnisses

Externe Sortialgorithmen

- Externes Sortieren durch Mischen
- Ablauf
 - 1 Relation durch **partition** in gleich große Teile teilen, die jeweils im Hauptspeicher sortiert werden können
 - 2 n Mischläufe mit **merge**, die zwei oder mehr Teilrelationen mischen

Externe Sortieralgorithmen /2



Externes Sortieren: Kosten

- initialer Schritt: jeden Block lesen und auf Partition schreiben
- **Fall 1:** Kombination von 2 Zwischenergebnissen: n Mischläufe kombinieren 2^n Partitionen
- also:
 - ▶ initial Relation in mind. $\frac{b_r}{mem}$ Partitionen aufgeteilt
 - ▶ daher Gesamtzahl der Mischläufe:

$$\left\lceil \log_2 \left(\frac{b_r}{mem} \right) \right\rceil$$

- ▶ jeder Mischlauf liest alle Blöcke und schreibt die gleiche Anzahl (also insgesamt $2b_r$ Blöcke)

$$\text{Insgesamt also für Fall 1: } cost_{\text{SORT}} = 2b_r \left(1 + \left\lceil \log_2 \left(\frac{b_r}{mem} \right) \right\rceil \right)$$

Externes Sortieren: Kosten /2

- **Fall 2:** es werden mehr als zwei Teilrelationen gleichzeitig gemischt; genauer $mem - 1$ Teilrelationen
- daher Kosten für Fall 2:

$$cost_{\text{SORT}} = 2b_r \left(1 + \left\lceil \log_{mem-1} \left(\frac{b_r}{mem} \right) \right\rceil \right)$$

Navigationsoperationen: Scans

- Scan durchläuft Tupel einer Relation
- Arten des Scans:
 - ▶ *Relationen-Scan* (*full table scan*) durchläuft alle Tupel einer Relation in beliebiger Reihenfolge
Aufwand: b_r
 - ▶ *Index-Scan* nutzt Index zum Auslesen der Tupel in Sortierreihenfolge
Aufwand: Anzahl der Tupel plus Höhe des Indexes
- Vergleich
 - ▶ Relationen-Scan besser durch Ausnutzung der Blockung
 - ▶ Index-Scan besser, falls wenige Daten benötigt, aber schlechter beim Auslesen vieler Tupel

Operationen auf Scans

- Relationen-Scan öffnen
open-rel-scan(RelationenID) \rightarrow ScanID
liefert ScanID zurück, die bei folgenden Operationen zur Identifikation genutzt wird
- Index-Scan initialisieren
open-index-scan(IndexID, Min, Max) \rightarrow ScanID
liefert ScanID zurück; Min und Max bestimmen Bereich einer Bereichsanfrage
- **next-TID** liefert nächsten TID; Scan-Cursor weitersetzen
- **end-of-scan** liefert **true**, falls kein TID mehr im Scan abzuarbeiten
- **close-scan** schließt Scan

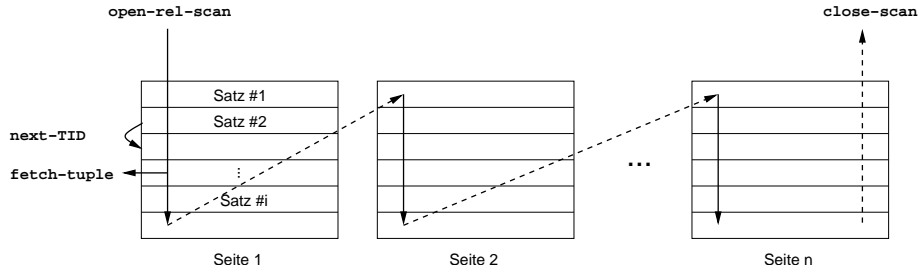
Beispiel: Scan

```
select *  
from KUNDE  
where Nachname between 'Heuer' and  
      'Jagellowsk'
```


Beispiel: Relationen-Scan

```
aktuellerScanID := open-rel-scan(KUNDE-RelationID);  
aktuellerTID := next-TID(aktuellerScanID);  
while not end-of-scan(aktuellerScanID) do  
  begin  
    aktuellerPuffer :=  
      fetch-tuple(KUNDE-RelationID, aktuellerTID);  
    if aktuellerPuffer.Nachname >= 'Heuer'  
      and aktuellerPuffer.Nachname <= 'Jagellowsk'  
    then put (aktuellerPuffer);  
    endif;  
    aktuellerTID := next-TID(aktuellerScanID);  
  end;  
close-scan (aktuellerScanID);
```

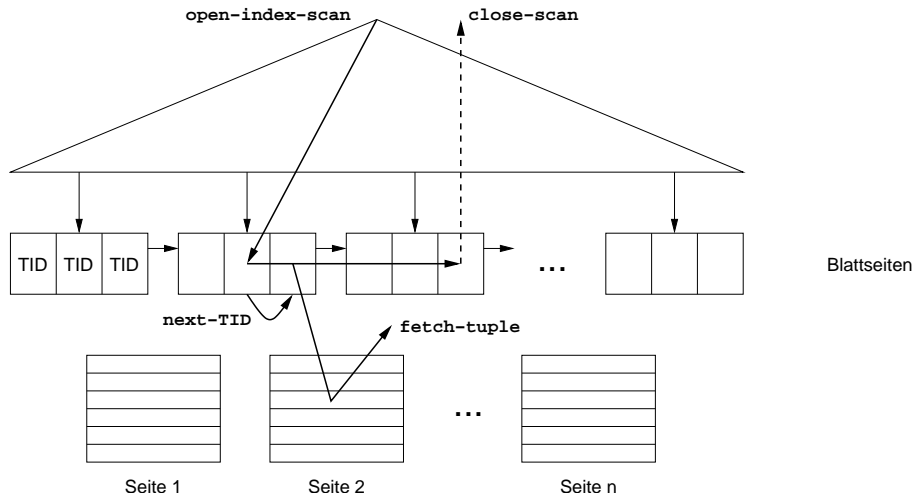
Relationen-Scan: Prinzip



Beispiel: Index-Scan

```
aktuellerScanID :=  
    open-index-scan (KUNDE-Nachname-IndexID,  
        'Heuer', 'Jagellowsk');  
aktuellerTID := next-TID (aktuellerScanID);  
while not end-of-scan (aktuellerScanID) do  
begin  
    aktuellerPuffer :=  
        fetch-tuple (KUNDE-RelationID, aktuellerTID);  
    put (aktuellerPuffer);  
    aktuellerTID := next-TID (aktuellerScanID);  
end;  
close-scan (aktuellerScanID);
```

Index-Scan: Prinzip



Scan-Operationen: Kosten

- Relationen-Scan

$$cost_{REL} = b_r$$

- Index-Scan: Bedingung $A = c$ und Primärindex

$$cost_{IND} = lev_{I(R(A))} + 1$$

- Sekundärindex: $val_{A,r}$ verschiedene Werte, ungünstigster Fall

$$cost_{IND} = lev_{I(R(A))} + \frac{|r|}{val_{A,r}}$$

Scan-Semantik

- bei Scan-basierten (positionalen) Änderungsoperationen: Festlegung einer Scan-Semantik \rightsquigarrow Wirkungsweise nachfolgender Scan-Operationen
- Beispiel: Löschen des aktuellen Satzes
- Zustände: vor dem ersten Satz, auf einem Satz, in Lücke zwischen zwei Sätzen, hinter dem letzten Satz, in leerer Menge
- weiterhin: Übergangsregeln für Zustände

Scan-Semantik /2

- Halloween-Problem (System R):

- ▶ SQL-Anweisung:

```
update PRODUKT set Preis = Preis * 1.1  
where Preis > 100
```

- ▶ satzorientierte Auswertung mittels Index-Scan über $I(\text{PRODUKT}(\text{Preis}))$ und sofortige Index-Aktualisierung
 - ▶ ohne besondere Vorkehrungen: unendliche Anzahl von Preiserhöhungen

Selektion

- *exakte Suche, Bereichsselektionen*, komplex zusammengesetzte Selektionskriterien
- zusammengesetztes Prädikat φ aus atomaren Prädikaten (exakte Suche, Bereichsanfrage) mit **and**, **or**, **not**
- Tupelweises Vorgehen
 - ▶ Gegeben $\sigma_{\varphi}(r)$
 - ▶ Relationen-Scan: für alle $t \in r$ auswerten $\varphi(t)$
 - ▶ Aufwand $O(|r|)$, genauer b_r

Selektion: Konjunktive Normalform

- Zugriffspfade bei komplexen Prädikaten einsetzen $\Rightarrow \varphi$ analysieren und geeignet umformen
- etwa φ in konjunktive Normalform (KNF) überführen; bestehend aus *Konjunkten*
- heuristisch Konjunkt auswählen, das sich besonders gut durch Indexe auswerten lässt (etwa bei $A = c$ und über A Index)
- ausgewähltes Konjunkt auswerten; für Ergebnis-TID-Liste andere Konjunkte tupelweise
- oder mehrere geeignete Konjunkte auswerten und die sich ergebenden TID-Listen schneiden

Projektion

- Relationenalgebra: mit Duplikateliminierung
- SQL: keine Duplikateliminierung, wenn nicht mit **distinct** gefordert (modifizierter Scan)
- mit Duplikateliminierung:
 - ▶ sortierte Ausgabe eines Indexes hilft bei der Duplikateliminierung
 - ▶ Projektion auf indexierte Attribute ohne Zugriff auf gespeicherte Tupel

Projektion /2

- Projektion $\pi_X(r)$:
 - 1 r nach X sortieren
 - 2 $t \in r$ werden in das Ergebnis aufgenommen, für die $t(X) \neq \mathbf{previous}(t(X))$ gilt
- Zeitaufwand: $O(|r| \log |r|)$
- Falls r schon sortiert nach X : $O(|r|)$
- Schlüssel $K \subseteq X$: $O(|r|)$

Aggregation & Gruppierung

- skalare Aggregation

```
select  agg(A) from R
```

- ▶ Implementierung durch Scan
- ▶ für **min**, **max**, **count** aus Index bzw. Data Dictionary

- Aggregatfunktionen und Gruppierung

```
select  G, agg(A)  
from    R  
group by G
```

- ▶ Zusammenfassung als Algebraoperator: $\gamma_{f_1(x_1), \dots, f_n(x_n), A}(r(t))$
- ▶ Implementierungsvarianten: Nested Loops, Sortierung, Hashing

Hashbasierte Gruppierung

- Variante der Nested-Loops-Technik
- Zuordnung zu Gruppen über Hashfunktion
- Verbesserungen
 - ▶ sukzessive Berechnung der Aggregate
 - ▶ Voraussetzung: additive Aggregatfunktionen

$$f(x_1, \dots, x_n) = g(h(x_1), \dots, h(x_n))$$

- ▶ für **min**, **max**, **sum**: $f = g = h$ (distributiv)
- ▶ für $f = \mathbf{avg}$: h muss Tupel (**sum**(x_i), **count**(x_i)) liefern und

$$g = \frac{\sum_i \mathbf{sum}(x_i)}{\sum_i \mathbf{count}(x_i)}$$

- ▶ für **median**: nicht möglich

Gruppierung: Vergleich

- Nested Loops: $O(|r|^2)$ (worst case)
- sortierbasierte Variante: $O(|r| \log |r|)$ (bei notwendiger Sortierung)

Binäre Operationen: Mengenoperationen

- Binäre Operationen meist auf Basis von tupelweisem Vergleich der einzelnen Tupelmengen
- *Nested-Loops-Technik* oder *Schleifeniteration*
 - ▶ für jedes Tupel einer äußeren Relation s wird die innere Relation r komplett durchlaufen
 - ▶ Aufwand: $O(|s| \cdot |r|)$ Vergleiche
- *Merge-Technik* oder *Mischmethode*
 - ▶ r und s (sortiert) schrittweise in der vorgegebenen Tupelreihenfolge durchlaufen
 - ▶ Aufwand: $O(|s| + |r|)$
 - ▶ Falls Sortierung noch vorzunehmen: *Sort-Merge-Technik*
 - ▶ Aufwand $|r| \log |r|$ und/oder $|s| \log |s|$

Mengenoperationen /2

- *Hash-Methoden*

- ▶ kleinere der beiden Relationen in Hash-Tabelle
- ▶ Tupel der zweiten Relation finden ihren Vergleichspartner mittels Hash-Funktion
- ▶ idealerweise Aufwand $O(|s| + |r|)$

Vereinigung mit Duplikateliminierung

- Vereinigung durch Einfügen
 - ▶ Variante der Nested-Loops-Methoden
 - ▶ Kopie einer der beiden Relationen r_2 unter dem Namen r'_2 anlegen, dann Tupel $t_1 \in r_1$ in r'_2 einfügen (Zeitaufwand abhängig von Organisationsform der Kopie)
- Spezialtechniken für die Vereinigung
 - ▶ r und s verketteten
 - ▶ Projektion auf alle Attribute der verketteten Relation
- Zeitaufwand: $O((|r| + |s|) \cdot \log(|r| + |s|))$ (wie Projektion)

Vereinigung /2

Vereinigung durch Merge-Techniken (**merge-union**)

- ① r und s sortieren, falls nicht bereits sortiert
- ② r und s mischen
 - ▶ $t_r \in r$ kleiner als $t_s \in s$: t_r in das Ergebnis, nächstes $t_r \in r$ lesen
 - ▶ $t_r \in r$ größer als $t_s \in s$: t_s in das Ergebnis, nächstes $t_s \in s$ lesen
 - ▶ $t_s = t_r$: t_r in das Ergebnis, nächste $t_r \in r$ bzw. $t_s \in s$ lesen
- Zeitaufwand: $O(|r| \cdot \log |r| + |s| \cdot \log |s|)$ mit Sortierung, $O(|r| + |s|)$ ohne Sortierung

Berechnung von Verbunden

- Varianten

- ▶ Nested-Loops-Verbund
- ▶ Block-Nested-Loops-Verbund
- ▶ Merge-Join
- ▶ Hash-Verbund
- ▶ ...

Nested-Loops-Verbund

- doppelte Schleife iteriert über alle $t_r \in r$ und alle $t_s \in s$ bei einer Operation $r \bowtie_{\varphi} s$

```
for each  $t_r \in r$  do  
begin  
  for each  $t_s \in s$  do  
    begin  
      if  $\varphi(t_r, t_s)$  then put( $t_r || t_s$ ) endif  
    end  
  end  
end
```

Block-Nested-Loops-Verbund

- statt über Tupel über Blöcke iterieren

```
for each Block  $B_r$  of  $r$  do
begin
  for each Block  $B_s$  of  $s$  do
  begin
    for each Tupel  $t_r \in B_r$  do
    begin
      for each Tupel  $t_s \in B_s$  do
      begin
        if  $\varphi(t_r, t_s)$  then put( $t_r \cdot t_s$ ) endif
      end
    end
  end
end
end
```

Nested-Loops-Verbund: Kosten

- ohne Indexunterstützung

$$cost_{\text{LOOP}} = b_r + \left\lceil \frac{b_r}{mem - 1} \right\rceil \cdot b_s$$

- mit Primärindexunterstützung in s

$$cost_{\text{LOOP}} = b_r + |r| \cdot (lev_{I(S(B))} + 1)$$

- mit Sekundärindexunterstützung

$$cost_{\text{LOOP}} = b_r + |r| \cdot (lev_{I(S(B))} + \frac{|s|}{val_{B,s}})$$

Merge-Techniken

- $X := R \cap S$; falls nicht bereits sortiert, zuerst Sortierung von r und s nach X
 - 1 $t_r(X) < t_s(X)$, nächstes $t_r \in r$ lesen
 - 2 $t_r(X) > t_s(X)$, nächstes $t_s \in s$ lesen
 - 3 $t_r(X) = t_s(X)$, t_r mit t_s und allen Nachfolgern von t_s , die auf X mit t_s gleich, verbinden
 - 4 beim ersten $t'_s \in s$ mit $t'_s(X) \neq t_s(X)$ beginnend mit ursprünglichem t_s mit den Nachfolgern t'_r von t_r wiederholen, solange $t_r(X) = t'_r(X)$ gilt

Merge-Verbund: Kosten

- Gesamtkosten

$$cost_{\text{MERGE}} = cost_r + cost_s$$

- für Relation $r_i \in \{r, s\}$

- ▶ Fall 1: r_i liegt sortiert vor

$$cost_{r_i} = b_{r_i}$$

- ▶ Fall 2: r_i muss sortiert werden

$$cost_{r_i} = b_{r_i} + b_{r_i} \log_{mem} b_{r_i}$$

Merge-Join mit Scan

- Verbund-Attribute auf beiden Relationen Schlüsseleigenschaft (**unique**, **primary key**)
- **min**(x) und **max**(x) : minimaler bzw. maximaler gespeicherter Wert für x

Verbund durch Hashing

- Idee:

- ▶ Ausnutzung des verfügbaren Hauptspeichers zur Minimierung der Externspeicherzugriffe
- ▶ Finden der Verbundpartner durch Hashing
- ▶ Anfragen der Form $r \bowtie_{r.A=s.B} s$

- Prinzip:

1. Tupel der kleineren Relation lesen und durch Anwendung einer Hashfunktion h in entsprechende Buckets einordnen
2. Tupel der zweiten Relation lesen und unter Anwendung von h Bucket mit potenziellen Verbundpartnern identifizieren
3. Verbundbedingung prüfen

Classic Hash Verbund

- Vorbereitung: kleinere Relation wird r
- Ablauf
 - 1 Tupel von r mittels Scan in Hauptspeicher lesen und mittels Hashfunktion $h(r.A)$ in Hashtabelle H einordnen
 - 2 wenn H voll (oder r vollständig gelesen):
Scan über s und mit $h(s.B)$ Verbundpartner suchen
 - 3 falls Scan über r nicht abgeschlossen:
 H neu aufbauen und erneuten Scan über S durchführen
- Aufwand: $O(b_r + p \cdot b_s)$ mit p ist Anzahl der Scans über s

Classic Hash Verbund: Algorithmus

```
R1ScanID := open-rel-scan(R1ID);  
R1TID := next-TID(R1ScanID);  
HTable := [];  
while not end-of-scan(R1ScanID) do  
    R1Puffer := fetch-tuple(R1ID, R1TID);  
    idx := hash(R1Puffer.X);  
    if overflows(HTable) then  
        join-with-hashtable(HTable, R2ID);  
    endif;  
    insert into HTable[idx] (R1Puffer);  
    R1TID := next-TID(R1ScanID);  
end;  
close-scan(R1ScanID);  
join-with-hashtable(HTable, R2ID);
```

Classic Hash Verbund: Algorithmus /2

```
procedure join-with-hashtable (HTable, R2ID)
begin
  R2ScanID := open-rel-scan(R2ID);
  R2TID := next-TID(R2ScanID);
  while not end-of-scan(R2ScanID) do
    R2Puffer := fetch-tuple(R2ID, R2TID);
    idx := hash(R2Puffer.X);
    for each t in HTable[idx] do
      begin if R2Puffer.X = t.X then
        insert into ERG(t.A1, ..., t.An, t.X,
          R2Puffer.B1, ..., R2Puffer.Bm);
      endif; end;
    R2TID := next-TID(R2ScanID);
  end;
  close-scan(R2ScanID);
  HTable := [];
end;
```

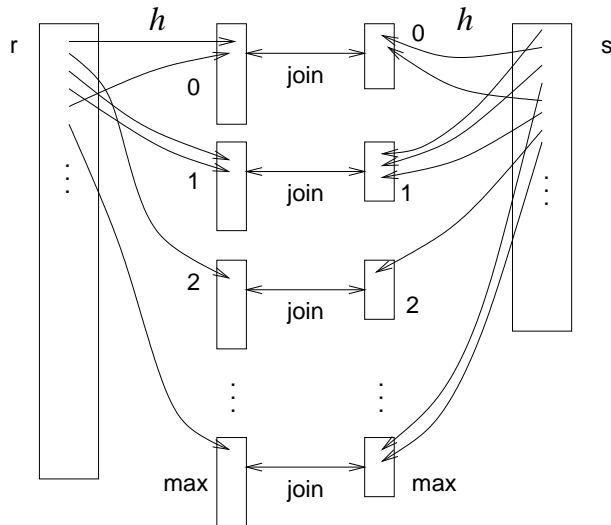
Simple Hash Verbund

- Nachteil des Classic Hash Verbunds: mehrmaliges komplettes Lesen der zweiten Relation
- Verbesserung: auch zweite Relation partitionieren
- Simple Hash Verbund
 - ▶ Bereich der Hashwerte vorab auswählen und nur betroffene Tupel in Hashtabelle einfügen
 - ▶ verbleibende Tupel in temporärer Relation (Überlaufbereich) sichern (für r und s)
 - ▶ anschließend: rekursive Behandlung der temporären Relationen
 - ▶ **Verbundkandidaten können nur in temporären Relationen sein!**

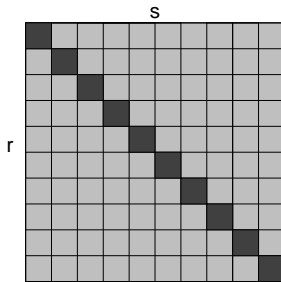
Partitionierung mittels Hashfunktion

- Trennung von Partitionierungs- und Verbundphase
- Tupel aus r und s über X in gemeinsame Datei mit k Blöcken (*Buckets*) „hashen“
- Tupel in gleichen Buckets durch Verbundalgorithmus verbinden

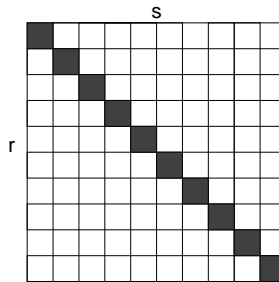
Partitionierung mittels Hashfunktion /2



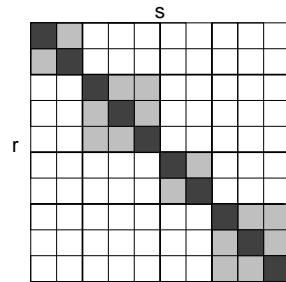
Vergleich der Techniken



Nested-Loops-Join



Merge-Join



Hash-Join

Zusammenfassung (1)

- Kostenparameter und -formeln
- Varianten von Scans
- unäre und binäre Operatoren
- Verbundimplementierungen

Grundprinzipien der Optimierung

- Basissprachen
 - ▶ SQL
 - ▶ Relationenkalküle
 - ▶ hier: Relationenalgebra
- Ziel der Optimierung
 - ▶ möglichst schnelle Anfragebearbeitung
 - ⇒ möglichst wenig Seitenzugriffe bei der Anfragebearbeitung
 - ⇒ möglichst in allen Operationen so wenig wie möglich Seiten (Tupel) berücksichtigen

Beispiel

```
select KUNDE.KNr, Nachname  
from KUNDE, BESTELLUNG  
where KUNDE.KNr = BESTELLUNG.KNr  
       and Datum = date '22-NOV-04'
```

- Relation KUNDE: 100 Tupel; eine Seite: 5 Tupel
- Relation BESTELLUNG: 10.000 Tupel; eine Seite: 10 Tupel
- 50 Bestellungen pro Tag
- Tupel der Form (KNr, Nachname): 50 auf eine Seite
- 3 Zeilen von $r(KUNDE) \times r(BESTELLUNG)$ auf eine Seite
- Puffer für jede Relation Größe 1, keine Spannsätze

Direkte Auswertung

$$1 \quad r_1 := r(\text{KUNDE}) \times r(\text{BESTELLUNG})$$

Seitenzugriffe:

- ▶ $l : (100/5) + (100/5 \cdot 10.000/10) = 20.020$
- ▶ $s : (100 \cdot 10.000)/3 = 333.000 \text{ (ca.)}$

$$2 \quad r_2 := \sigma_{\text{SEL}}(r_1)$$

- ▶ $l : 333.000 \text{ (ca.)}$
- ▶ $s : 50/3 = 17 \text{ (ca.)}$

$$3 \quad r_{\text{erg}} := \pi_{\text{PROJ}}(r_2)$$

- ▶ $l : 17$
- ▶ $s : 1$

Insgesamt ca. 687.000 Seitenzugriffe und ca. 333.000 Seiten zur Zwischenspeicherung

Optimierte Auswertung

- 1 $r_1 := \sigma_{\text{Datum}='22.11.04'}(r(\text{BESTELLUNG}))$
 - ▶ $l: 10.000/10 = 1.000$
 - ▶ $s: 50/10 = 5$
- 2 $r_2 := r(\text{KUNDE}) \bowtie_{\text{KNr}=\text{KNr}} r_1$
 - ▶ $l: 5 + 100/5 \cdot 5 = 105$
 - ▶ $s: 50/3 = 17$
- 3 $r_{\text{erg}} := \pi_{\text{PROJ}}(r_2)$
 - ▶ $l: 17$
 - ▶ $s: 1$

ca. 1.145 Seitenzugriffe (Faktor 500 verbessert)

Auswertung mit Indexausnutzung

Indexe $I(\text{BESTELLUNG}(\text{Datum}))$ und $I(\text{KUNDE}(\text{KNr}))$

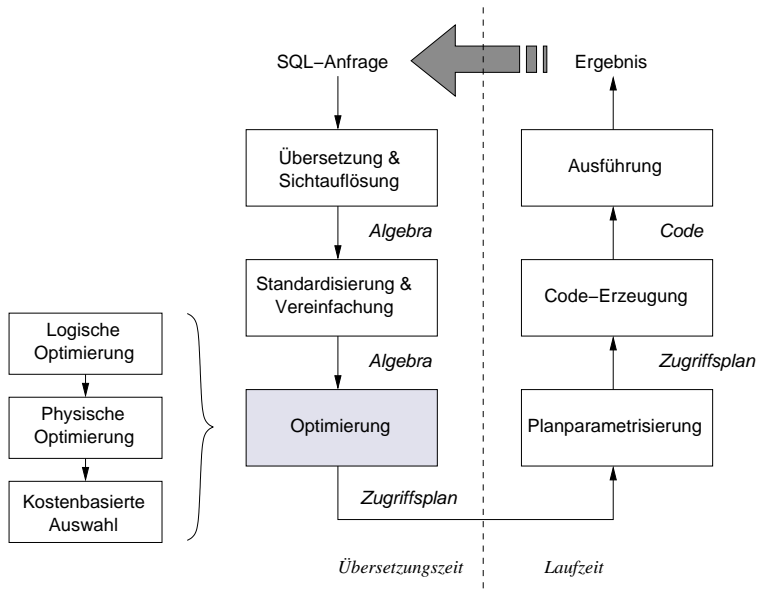
- ❶ $r_1 := \sigma_{\text{Datum}=22.11.04}(r(\text{BESTELLUNG}))$ über $I(\text{BESTELLUNG}(\text{Datum}))$
 - ▶ l : minimal 5, maximal 50; s : $50/10 = 5$
- ❷ $r_2 := \text{sortiere } r_1 \text{ nach KNr}$
 - ▶ $l + s : 5 \cdot \log 5 = 15$ (ca.)
- ❸ $r_3 := \text{KUNDE} \bowtie_{\text{KNr}=\text{KNr}} r_2$
 - ▶ l : $100/5 + 5 = 25$; s : $50/3 = 17$ (Merge-Join)
- ❹ $r_{\text{erg}} := \pi_{\text{PROJ}}(r_3)$
 - ▶ l : 17; s : 1

maximal ca. 130 und minimal ca. 85 Seitenzugriffe

Gegenüberstellung der Varianten

Variante der Ausführung	Lese- und Schreibzugriffe	Seiten für Zwischenergebnisse
direkte Auswertung	ca. 687.000	ca. 333.000
optimierte Auswertung	ca. 1.140	17
Auswertung mit Index	min. 85	17
mit Pipelining	max. 130 51 bis 96	17 5 (plus sortieren)

Phasen der Anfragebearbeitung



Phasen der Anfrageverarbeitung /2

① Übersetzung und Sichtexpansion

- ▶ in Anfrageplan arithmetische Ausdrücke vereinfachen
- ▶ Unteranfragen auflösen
- ▶ Einsetzen der Sichtdefinition

② Logische oder auch algebraische Optimierung

- ▶ Anfrageplan unabhängig von der konkreten Speicherungsform umformen; etwa Hineinziehen von Selektionen in andere Operationen

Phasen der Anfrageverarbeitung /3

3 Physische oder Interne Optimierung

- ▶ konkrete Speicherungstechniken (Indexe, Cluster) berücksichtigen
- ▶ Algorithmen auswählen
- ▶ mehrere alternative interne Pläne

4 Kostenbasierte Auswahl

- ▶ Statistikinformationen (Größe von Tabellen, Selektivität von Attributen) für die Auswahl eines konkreten internen Planes nutzen

5 Planparametrisierung

- ▶ bei vorkompilierten Anfragen (etwa Embedded-SQL): Ersetzen der Platzhalter durch Werte

6 Code-Erzeugung

- ▶ Umwandlung des Zugriffsplans in ausführbaren Code

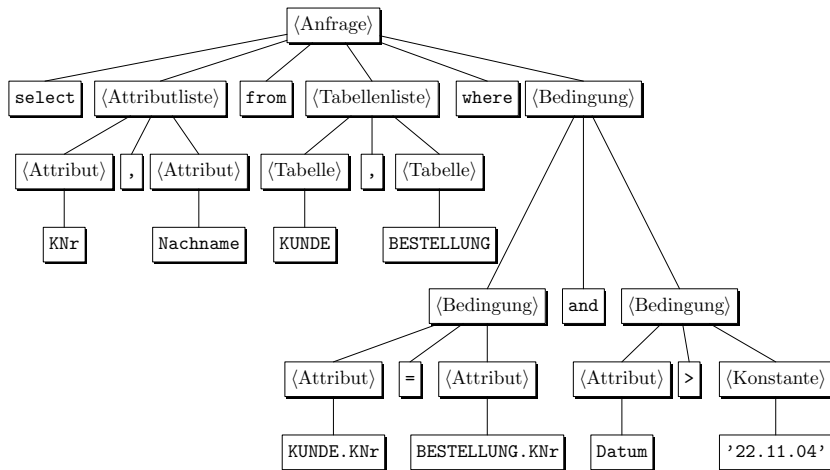
Phasen der Anfrageverarbeitung /4

- Repräsentation von Anfragen während der Verarbeitung
 - ▶ Algebraausdrücke → **Operatorbaum**
 - ★ Operatoren als Knoten
 - ★ Relationen als Blätter
 - ★ Kanten repräsentieren Datenfluss
 - ▶ spätere Phasen → **Zugriffs- oder Anfrageplan** (query execution plan – QEP)
 - ★ konkrete Algorithmen als Operatorknoten
 - ★ Verwendung von Zugriffspfaden (Indexe)

Parsen und Analysieren

- Zeichenfolge der Anfrage in Datenstruktur mit SQL-Syntaxkonstrukten überführen (Parse-Baum)
- Analyse des Eingabetextes
 - ▶ lexikalische Korrektheit: korrekte Angabe der Symbole (Schlüsselwörter etc.)
 - ▶ syntaktische Korrektheit: korrekte Reihenfolge, Einhaltung der Syntaxregeln

Parse-Baum



Übersetzung in Relationenalgebra

- Transformationsregeln

- ▶ Relationen der Tabellenliste hinter **from** untereinander durch Kreuzprodukt verknüpfen
- ▶ Bedingung im **where**-Teil als Selektion übernehmen
- ▶ Spaltenliste hinter **select** als abschließende Projektion

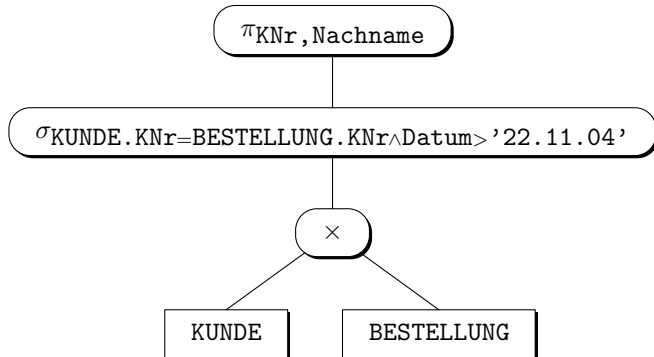
- zusätzlich noch

- ▶ Berücksichtigung von SQL-Konstrukten wie **order by**, **group by**
- ▶ Auflösen von Unteranfragen
- ▶ ...

Übersetzung in Relationenalgebra /2

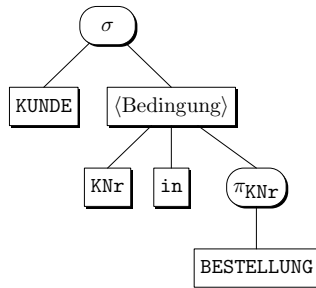
$$\pi_{\text{KNr}, \text{Nachname}}(\sigma_{\text{KUNDE.KNr}=\text{BESTELLUNG.KNr} \wedge \text{Datum} > '22.11.04'}(r(\text{KUNDE}) \times r(\text{BESTELLUNG})))$$

Kanonische Übersetzung:



Zwischenrepräsentation

```
select *  
from KUNDE  
where KNr in (select KNr  
             from BESTELLUNG)
```



Auflösung von Sichten

- Sichtexpansion: Einsetzen der Sichtdefinition in Anfrage
 - ▶ im Parsebaum
 - ▶ im Operatorbaum
- rekursiver Prozess: Sichten über Sichten möglich

Standardisierung und Vereinfachung

- Vereinfachung der folgenden Optimierungsschritte durch ein einheitliches (kanonisches) Anfrageformat
- auf Ausdrucksebene (Bedingungen):
 - ▶ Normalformen, Entfernen redundanter Ausdrücke
- auf Anfrageebene:
 - ▶ Entschachtelung

Standardisierung von Ausdrücken

- speziell für Selektions- und Verbundbedingungen
 - ▶ *konjunktive Normalform* vs. *disjunktive Normalform*
 - ▶ konjunktive Normalform (KNF) für einfache Prädikate p_{ij} ($A_i = A_j$ oder $A_i = \text{Konstante}$):

$$(p_{11} \vee p_{12} \vee \cdots \vee p_{1n}) \wedge \cdots \wedge (p_{m1} \vee p_{m2} \vee \cdots \vee p_{mn})$$

- ▶ disjunktive Normalform (DNF):

$$(p_{11} \wedge p_{12} \wedge \cdots \wedge p_{1n}) \vee \cdots \vee (p_{m1} \wedge p_{m2} \wedge \cdots \wedge p_{mn})$$

- ▶ Überführung in KNF/DNF durch Anwendung von Äquivalenzbeziehungen für logische Operationen

Normalisierung

• Äquivalenzbeziehungen

- ▶ $p_1 \wedge p_2 \iff p_2 \wedge p_1$ und $p_1 \vee p_2 \iff p_2 \vee p_1$
- ▶ $p_1 \wedge (p_2 \wedge p_3) \iff (p_1 \wedge p_2) \wedge p_3$ und $p_1 \vee (p_2 \vee p_3) \iff (p_1 \vee p_2) \vee p_3$
- ▶ $p_1 \wedge (p_2 \vee p_3) \iff (p_1 \wedge p_2) \vee (p_1 \wedge p_3)$ und $p_1 \vee (p_2 \wedge p_3) \iff (p_1 \vee p_2) \wedge (p_1 \vee p_3)$
- ▶ $\neg(p_1 \wedge p_2) \iff \neg p_1 \vee \neg p_2$ und $\neg(p_1 \vee p_2) \iff \neg p_1 \wedge \neg p_2$
- ▶ $\neg(\neg p_1) \iff p_1$

Normalisierung: Beispiel

```
select *  
from KUNDE K, BESTELLUNG B  
where K.KNr = B.KNr and Menge > 10  
      and (LName = 'Coffeeshop' or  
           LName = 'Kaffeebude')
```

- Selektionsbedingung in KNF:

$$(K.Nr = B.KNr) \wedge (Menge > 10) \wedge \\ (LName = \text{'Coffeeshop'} \vee LName = \text{'Kaffeebude'})$$

- Selektionsbedingung in DNF:

$$(K.KNr = B.KNr \wedge Menge > 10 \wedge LName = \text{'Coffeeshop'}) \vee \\ (K.KNr = B.KNr \wedge Menge > 10 \wedge LName = \text{'Kaffeebude'})$$

Vereinfachung von Ausdrücken

- Idempotenzen

- ▶ $A \vee A \iff A$
- ▶ $A \wedge A \iff A$
- ▶ $A \vee \neg A \iff \mathbf{true}$
- ▶ $A \wedge \neg A \iff \mathbf{false}$

- Konstantenpropagierung

- ▶ Ausnutzung von Transitivität
- ▶ Bsp.: $A \theta B \wedge B = c \Rightarrow A \theta c$

- unerfüllbare Ausdrücke

- ▶ $A > B \wedge B \geq C \wedge C > A \Rightarrow A > A \Rightarrow \mathbf{false}$

Entschachteln von Anfragen

- Entschachteln von Anfragen (Unteranfragen im **where**-Teil) zur Vereinfachung:
Erkennen von gemeinsamen Teilen, Nested Iteration \rightsquigarrow Join,
- Formen von Prädikaten
 - ▶ geschachtelt: $R_i.C_k \theta Q$ mit SQL-Block Q und $\theta \in \{<, >, =, \text{etc.}\}$
 - ▶ Verbundprädikat: $R_i.C_k \theta R_j.C_n$
- Ziel ist eine kanonische n -Relationen-Anfrage
 - ▶ Verbund zwischen n Relationen mit $n - 1$ Verbundprädikaten

Typ-A-Schachtelung

- innerer Block Q enthält kein Verbundprädikat, das äußere Relation referenziert
- Q berechnet Aggregat

```
select BestNr
from BESTELLUNG
where ProdNr = (select max(ProdNr)
                from PRODUKT
                where Preis < 100)
```

- Ausführung
 - 1 innere Anfrage + Aggregat berechnen
 - 2 Ergebnis in äußere Anfrage einsetzen und diese berechnen

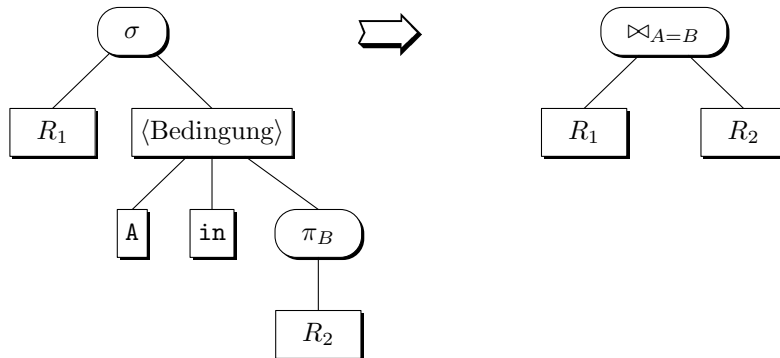
Typ-N-Schachtelung

- innerer Block Q enthält kein Verbundprädikat, das äußere Relation referenziert
- Q berechnet *kein* Aggregat

```
select BestNr
from BESTELLUNG
where ProdNr in (select ProdNr
                  from PRODUKT
                  where Preis < 100)
```

- Ausführung
 - Variante A: innere Anfrage berechnen und in äußere einsetzen
 - Variante B: Entschachtelung

Entschachteln von Typ-N-Anfragen



Typ-N-Anfragen mit **in**-Prädikaten der Tiefe $n - 1$ können in semantisch äquivalente n -Relationenanfragen transformiert werden

Logische Optimierung

- heuristische Methoden
 - ▶ etwa algebraische Optimierung (Rewriting, auch *regelbasierte Optimierung* genannt)
 - ▶ für Relationenalgebra + Gruppierung, ...
- exakte Methoden
 - ▶ Tableauroptimierung
 - ▶ Anzahl Verbunde minimieren
 - ▶ für spezielle Relationenalgebra-Anfragen

Algebraische Optimierung

- Termersetzung von Termen der Relationenalgebra anhand von Algebraäquivalenzen
- Äquivalenzen gerichtet als Ersetzungsregeln
- heuristische Methode: Operationen verschieben, um kleinere Zwischenergebnisse zu erhalten; Redundanzen erkennen

Entfernen redundanter Operationen

- bei Anfragen mit Sichten nötig

$$r(\text{AKT_PRODUKTE}) = r(\text{PRODUKT}) \bowtie \pi_{\text{ProdNr}, \text{Bezeichnung}}(\dots \sigma_{\text{Datum=current_date}}(r(\text{BESTELLUNG})))$$

- Anfrage an Sicht:

$$\pi_{\text{Bezeichnung}, \text{Preis}}(r(\text{PRODUKT}) \bowtie r(\text{AKT_PRODUKTE}))$$

- Sichtexpansion:

$$\pi_{\text{Bezeichnung}, \text{Preis}}(r(\text{PRODUKT}) \bowtie r(\text{PRODUKT}) \bowtie \pi_{\text{ProdNr}, \text{Bezeichnung}}(\dots \sigma_{\text{Datum=current_date}}(r(\text{BESTELLUNG}))))$$

- Regel: Idempotenz

$$r = r \bowtie r, \text{ d.h. } \bowtie \text{ ist idempotent}$$

Verschieben von Selektionen

$$\sigma_{\text{Preis} > 100}(r(\text{BESTELLUNG}) \bowtie r(\text{PRODUKT}))$$

günstiger:

$$r(\text{BESTELLUNG}) \bowtie (\sigma_{\text{Preis} > 100}(r(\text{PRODUKT})))$$

Regel:

Selektion und Verbund kommutieren

nur, wenn die Attribute der Selektionsprädikate dies zulassen

Reihenfolge von Verbunden

- Kenntnis der Statistikinformationen (und des Schemas) aus Katalogs nötig

$$(r(\text{KUNDE}) \bowtie r(\text{PRODUKT})) \bowtie r(\text{BESTELLUNG})$$

- erster Verbund: kartesisches Produkt, daher:

$$r(\text{KUNDE}) \bowtie (r(\text{PRODUKT}) \bowtie r(\text{BESTELLUNG}))$$

- Regel:

\bowtie ist assoziativ und kommutativ

- keine eindeutige Vorzugsrichtung bei der Anwendung dieser Regel (daher interne Optimierung, Kostenbasierung)

Algebraische Regeln

- **KommJoin**: Operator \bowtie ist kommutativ:

$$r_1 \bowtie r_2 \iff r_2 \bowtie r_1$$

- **AssozJoin**: Operator \bowtie ist assoziativ:

$$(r_1 \bowtie r_2) \bowtie r_3 \iff r_1 \bowtie (r_2 \bowtie r_3)$$

- **ProjProj**: bei Operator π dominiert in der Kombination der äußere Parameter den inneren:

$$\pi_X(\pi_Y(r_1)) \iff \pi_X(r_1), \text{ mit } X \subseteq Y$$

Algebraische Regeln /2

- **se1se1**: Kombination von Prädikaten bei σ entspricht dem logischen Und \Rightarrow Formeln können in der Reihenfolge vertauscht werden

$$\sigma_{F_1}(\sigma_{F_2}(r_1)) \iff \sigma_{F_1 \wedge F_2}(r_1) \iff \sigma_{F_2}(\sigma_{F_1}(r_1))$$

(Ausnutzung der Kommutativität des logischen Und)

Algebraische Regeln /3

- **SelfProj**: Operatoren π und σ kommutieren, sofern das Prädikat F auf den Projektionsattributen definiert ist:

$$\sigma_F(\pi_X(r_1)) \iff \pi_X(\sigma_F(r_1))$$

$$\text{falls } \text{attr}(F) \subseteq X$$

anderenfalls Vertauschung möglich, wenn Projektion um die notwendigen Attribute erweitert wird:

$$\pi_{X_1}(\sigma_F(\pi_{X_1 X_2}(r_1))) \iff \pi_{X_1}(\sigma_F(r_1))$$

$$\text{falls } \text{attr}(F) \supseteq X_2$$

Algebraische Regeln /4

- **selJoin**: Operatoren σ und \bowtie kommutieren, falls Selektionsattribute alle aus einer der beiden Relationen stammen:

$$\sigma_F(r_1 \bowtie r_2) \iff \sigma_F(r_1) \bowtie r_2$$

$$\text{falls } \text{attr}(F) \subseteq R_1$$

falls Selektionsprädikat derart aufgesplittet werden kann, dass in $F = F_1 \wedge F_2$ beide Teile der Konjunktion passende Attribute haben, gilt:

$$\sigma_F(r_1 \bowtie r_2) \iff \sigma_{F_1}(r_1) \bowtie \sigma_{F_2}(r_2)$$

$$\text{falls } \text{attr}(F_1) \subseteq R_1 \text{ und } \text{attr}(F_2) \subseteq R_2$$

Algebraische Regeln /5

- **SelJoin** (fortg.): in jeden Fall: Abspalten eines F_1 mit Attributen der Relation R_1 , wenn F_2 Attribute von R_1 und R_2 betrifft:

$$\sigma_F(r_1 \bowtie r_2) \iff \sigma_{F_2}(\sigma_{F_1}(r_1) \bowtie r_2)$$

falls $\text{attr}(F_1) \subseteq R_1$

- **SelCross**: Kreuzprodukt in Theta-Join umwandeln

$$\sigma_{A\theta B}(R \times S) \iff R \bowtie_{A\theta B} S$$

Algebraische Regeln /6

- **selUnion**: Kommutieren von σ und \cup :

$$\sigma_F(r_1 \cup r_2) \iff \sigma_F(r_1) \cup \sigma_F(r_2)$$

- **selDiff**: Kommutieren von σ und $-$:

$$\sigma_F(r_1 - r_2) \iff \sigma_F(r_1) - \sigma_F(r_2)$$

- oder (da Tupel nur aus der ersten Relation herausgestrichen werden):

$$\sigma_F(r_1 - r_2) \iff \sigma_F(r_1) - r_2$$

Algebraische Regeln /7

- **ProjJoin:** Kommutieren von π und \bowtie :

$$\pi_X(r_1 \bowtie r_2) \iff \pi_X(\pi_{Y_1}(r_1) \bowtie \pi_{Y_2}(r_2))$$

mit

$$Y_1 = (X \cap R_1) \cup (R_1 \cap R_2)$$

und

$$Y_2 = (X \cap R_2) \cup (R_1 \cap R_2)$$

- Hineinziehen der Projektion in einen Verbund, wenn durch Berechnung von Y_i dafür gesorgt wird, dass die für den natürlichen Verbund benötigten Verbundattribute erhalten bleiben (Herausprojizieren erst nach dem Verbund)

Algebraische Regeln /8

- **ProjUnion:** Kommutieren von π und \cup :

$$\pi_X(r_1 \cup r_2) \iff \pi_X(r_1) \cup \pi_X(r_2)$$

- Distributivgesetz für \bowtie und \cup , Distributivgesetz für \bowtie und $-$, Kommutieren der Umbenennung β mit anderen Operationen, etc.

Weitere Regeln

• Idempotenzen

IdemUnion:	$r_1 \cup r_1 \iff r_1$
IdemSchnitt:	$r_1 \cap r_1 \iff r_1$
IdemJoin:	$r_1 \bowtie r_1 \iff r_1$
IdemDiff:	$r_1 - r_1 \iff \{\}$

• Verknüpfung mit einer leeren Relation:

LeerUnion:	$r_1 \cup \{\} \iff r_1$
LeerSchnitt:	$r_1 \cap \{\} \iff \{\}$
LeerJoin:	$r_1 \bowtie \{\} \iff \{\}$
LeerDiffRechts:	$r_1 - \{\} \iff r_1$
LeerDiffLinks:	$\{\} - r_1 \iff \{\}$

• für \bowtie , \cup und \cap : *Kommutativ- und Assoziativgesetz*

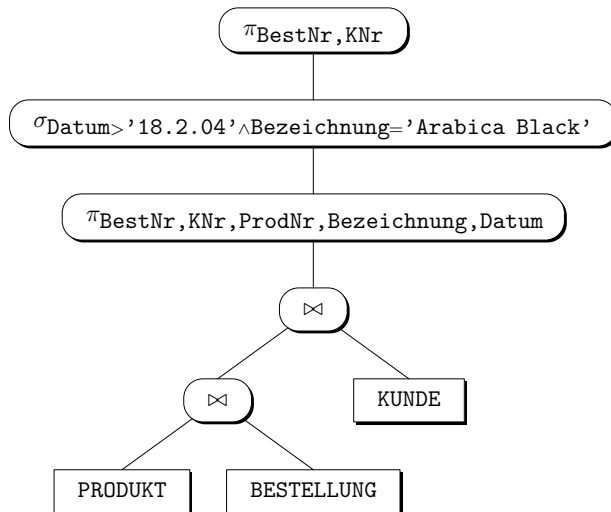
Algebraische Optimierung: Algorithmus

- einfacher Optimierungsalgorithmus
 - ① komplexe Selektionsprädikate auflösen (Regel **selSel**, ggf. Regeln der Auflösung für \neg und \vee)
 - ② mittels der Regeln **selJoin**, **selProj**, **selUnion** und **selDiff** Selektionen möglichst weit in Richtung der Blätter verschieben, ggf. Selektionen gemäß Regel **selSel** vertauschen
 - ③ Verschieben der Projektionen in Richtung Blätter mittels der Regeln **projProj**, **projJoin** und **projUnion**
- Einzelschritte werden in der genannten Reihenfolge solange ausgeführt, bis keine Ersetzungen mehr möglich sind

Algebraische Optimierung: Beispiel

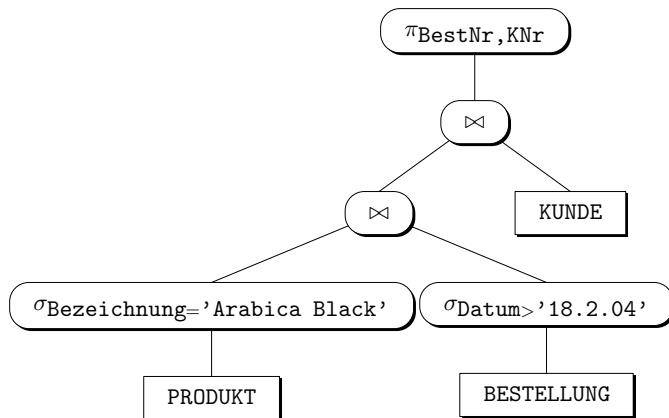
$$\pi_{\text{BestNr}, \text{KNr}}(\sigma_{\text{Datum} > '18.2.04' \wedge \text{Bezeichnung} = 'Arabica \text{ Black}'}$$
$$(\pi_{\text{BestNr}, \text{KNr}, \text{Datum}, \text{Bezeichnung}}($$
$$r(\text{PRODUKT}) \bowtie r(\text{BESTELLUNG}) \bowtie r(\text{KUNDE})))$$

Unoptimierter Anfrageplan



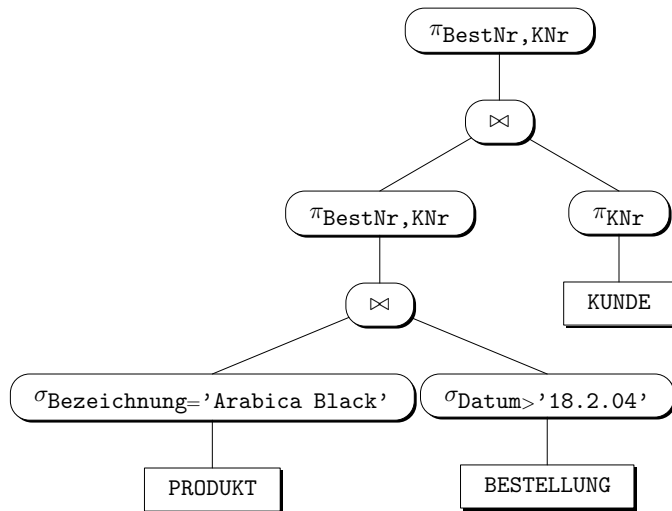
Anfrageplan /2

- Verschieben der Selektionen



Anfrageplan /3

- mit zusätzlichen Projektionen



Auswahl von Berechnungsalgorithmen

• Selektion

- ▶ $\sigma_{\varphi}^{\text{REL}}$: Selektion durch Relationen-Scan
- ▶ $\sigma_{\varphi}^{\text{IND}}$: Selektion durch Index-Scan

• Projektion

- ▶ $\pi_{\text{AttList}}^{\text{REL/mit}}$: Projektion durch Relationen-Scan; der Zusatz **mit** bezeichnet eine Projektion mit Duplikateliminierung
- ▶ $\pi_{\text{AttList}}^{\text{REL/ohne}}$: Projektion durch Relationen-Scan ohne Duplikateliminierung
- ▶ $\pi_{\text{AttList}}^{\text{SORT/mit}}$: Projektion durch Scan über einer nach `AttList` sortierten Relation mit Duplikateliminierung
- ▶ $\pi_{\text{AttList}}^{\text{SORT/ohne}}$: Projektion durch Scan über einer nach `AttList` sortierten Relation ohne Duplikateliminierung

Auswahl von Berechnungsalgorithmen /2

● Verbund

- ▶ ⌘^{LOOP}: Verbund durch Block-Nested-Loops
- ▶ ⌘^{MERGE}: Verbund durch Mischen (Voraussetzung: Eingaberelationen sind nach Verbundattribut(en) sortiert)
- ▶ ⌘^{HASH}: Verbund durch Hash-Join

Auswahl von Berechnungsalgorithmen /3

- Gruppierung

- ▶ $\gamma_{F;AttList}^{SORT}$ Gruppierung nach `AttList` und Anwendung der Aggregation F durch Sortierung
- ▶ $\gamma_{F;AttList}^{HASH}$ Gruppierung nach `AttList` und Anwendung der Aggregation F durch Hashing

Auswahl von Berechnungsalgorithmen /4

- Indexzugriff

$$\sigma_{A\Theta a}^{\text{IND}}(I(R(A))) \rightarrow \mathbf{list(tid)}$$

- Spezialfall

$$\sigma_{\mathbf{true}}^{\text{IND}}(I(R(A)))$$

- Kombination mit Projektion

$$\pi_{\text{AttList}}^{\text{IND/mit}} \text{ bzw. } \pi_{\text{AttList}}^{\text{IND/ohne}}$$

- ohne Zugriff auf Basisrelation

$$\pi_A^{\text{IND}}(I(R(A)))$$

Neue Operatoren

- für TID-Listen: „*Realisierung*“-Operator ρ

$$\rho(\langle \text{TID-Liste für } R\text{-Tupel} \rangle, r(R))$$

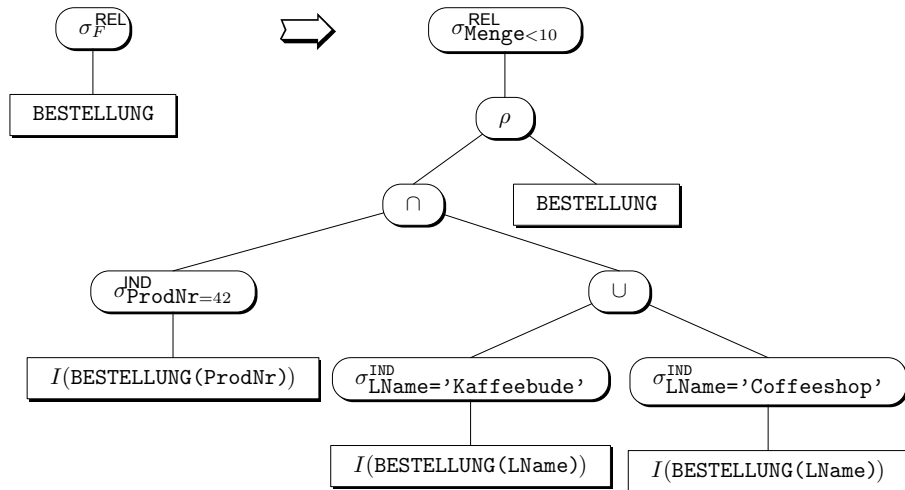
- Mengenoperatoren \cup , \cap und $-$ auf TID-Listen
- *Sortierung* von Tupelmengen ω

$$\omega_{\text{AttList}}(\langle \text{Tupel-Folge} \rangle)$$

Beispiele für Ausführungspläne

```
select *  
from BESTELLUNG  
where ProdNr = 42 and  
      (LName = 'Kaffeebude' or  
       LName = 'CoffeeShop') and  
      Menge < 10
```

Beispiele für Ausführungspläne /2




Schnittstelle von Planoperatoren

- Anforderung: Pipelining
- Iterator-Prinzip
 - ▶ **open**: Vorbereitung des Operators auf Verarbeitung, z.B. Initialisierungen vornehmen, Eingaberelationen öffnen
 - ▶ **next**: liefert bei jedem Aufruf das nächste Ergebnistupel
 - ▶ **close**: Durchführung von „Aufräumarbeiten“, z.B. Schließen von Eingaberelationen oder Freigabe temporärer Strukturen

Schnittstelle von Planoperatoren /2

Operator	open	next	close
$\pi^{\text{REL/ohne}}$	Eingabestrom öffnen	next auf Eingabestrom ausführen; Eingabetupel projizieren	Eingabestrom schließen
σ_F^{REL}	Eingabestrom öffnen	next solange auf Eingabestrom ausführen, bis ein Eingabetupel die Bedingung F erfüllt	Eingabestrom schließen
\bowtie^{MERGE}	beide Eingabeströme öffnen	next auf Eingabestrom mit kleinerem Schlüssel aufrufen, bis Verbundbedingung erfüllt	Eingabeströme schließen

Schnittstelle von Planoperatoren /3

Operator	open	next	close
 HASH	Hashtabelle anlegen; Eingabeströme öffnen; linken Eingabestrom mit next komplett lesen und in Hashtabelle einfügen; linken Eingabestrom schließen	next solange auf rechtem Eingabestrom aufrufen, bis Verbundpartner gefunden	Eingabestrom schließen
ω	Eingabestrom öffnen; partitionieren und sortieren; Partitionen mischen	nächstes Element aus gemischter Relation lesen	Partionen löschen

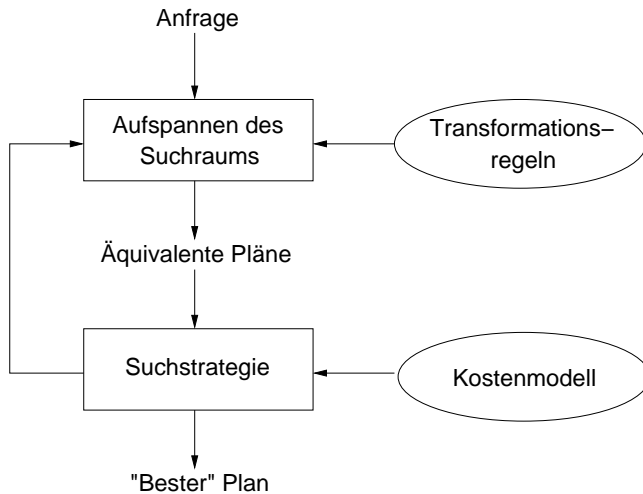
Blockierende Operatoren

- Operatoren, die zuerst die Eingaberelation(en) vollständig lesen müssen, bevor ein Ergebnistupel produziert werden kann
- Hauptteil der Arbeit in **open**-Methode
- Beispiele: sortierbasierte Operatoren

Zusammenfassung von Operatoren

- Kombination von Selektion und Projektion
- Kombination einer Selektion mit Verbundberechnung
- die Integration einer Selektion in die äußere Schleife eines Nested-Loops-Verbundes
- Integration von Selektionen in Merge-Join
- Kopplung der Selektion mit der Realisierung

Optimierung: Überblick



Aufspannen des Suchraums

- Suchraum: Menge aller äquivalenten Anfragepläne
- Aufspannen durch *Transformationsregeln* (algebraische Regeln)
- Schwerpunkt: *Verbundbäume*
- für n Relationen:
 - ▶ $n!$ verschiedene links- bzw. rechtsorientierte Bäume (Permutation der Blätter)
 - ▶ $n = 10$: 3.628.800 links-orientierte und 17.643.225.600 Bäume insgesamt!
- daher: Beschränkung des Suchraums durch
 - ▶ Heuristiken (algebraische Optimierungen)
 - ▶ vorgegebene „Form“ des Baumes

Aufspannen des Suchraums /2

- Anzahl der buschigen Bäume für Verbund zwischen n Relationen

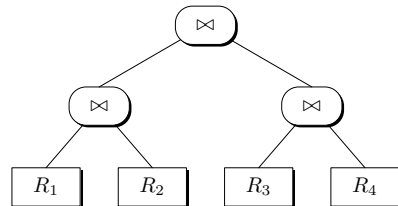
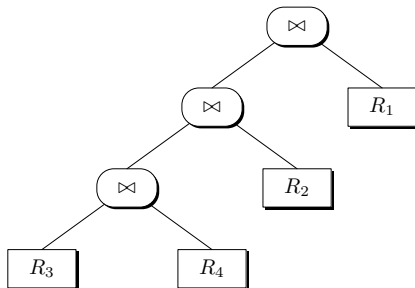
$$\mathcal{S}(1) = 1$$

$$\mathcal{S}(n) = \sum_{i=1}^{n-1} \mathcal{S}(i) \mathcal{S}(n-i)$$

- ▶ für Anzahl i aus $1 \dots n-1$ Blättern aus einem Teilbaum: $\mathcal{S}(i)$ verschiedene Baumformen
- ▶ für restliche $n-i$ Blätter: ebenfalls $\mathcal{S}(n-i)$ Formen
- ▶ zu jeder Form: n Relationen in $n!$ Varianten als Blätter zuweisen
- ▶ insgesamt: für n Relationen $\mathcal{S}(n) \cdot n!$ Varianten

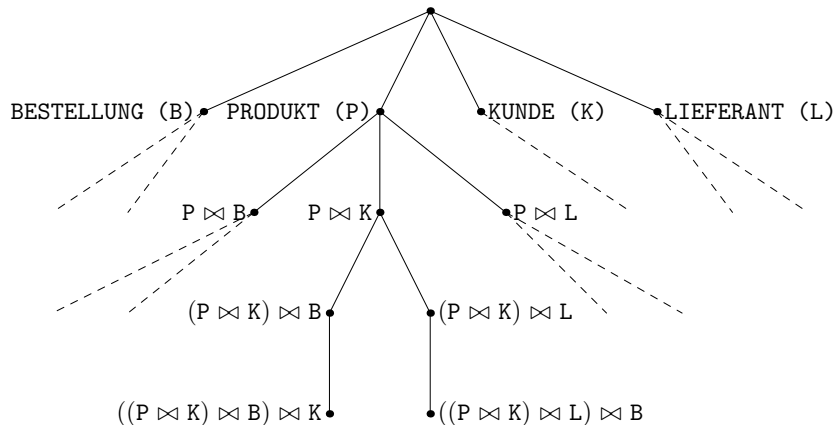
Verbundbäume

- lineare Folgen von Verbunden: links-orientiert bzw. rechts-orientiert
- „buschige“ Bäume



Aufspannen des Suchraums /2

$$r(\text{BESTELLUNG}) \bowtie r(\text{PRODUKT}) \bowtie r(\text{KUNDE}) \bowtie r(\text{LIEFERANT})$$



Suchstrategien

- „Durchlaufen“ des Suchraums
- Auswahl des kostengünstigsten Plans
- Basis: Kostenmodell
- bestimmt
 - ▶ *welche Pläne* werden betrachtet (vollständiges / partielles Durchsuchen)
 - ▶ in welcher *Reihenfolge* werden Alternativen untersucht
- Varianten: deterministisch, zufallsbasiert

Kostenmodell: Komponenten

- *Kostenfunktion*: zur Abschätzung der Kosten für Ausführung von Operationen bzw. Anfragen
- *Statistiken*: über Größe der Relationen (Kardinalität, Tupelgröße), Wertebereiche und -verteilungen
- *Formeln*: zur Berechnung der Größen von (Zwischen-)Ergebnissen auf der Basis der Statistiken

Kostenfunktion

- Kostenarten:

- ▶ I/O-Kosten: verursacht durch das Lesen und Schreiben von Blöcken vom bzw. auf den Externspeicher
- ▶ CPU-Kosten: für interne Berechnungen, Vergleiche etc.,
- ▶ Kommunikationskosten: im Fall verteilter Datenbanksysteme

- üblicherweise:

$$cost = cost_{IO} + W \cdot cost_{CPU}$$

- ▶ Faktor W zur Kalibrierung bzgl. Hardware

Kostenformeln

- Idee:

- ▶ Bestimmung des Gesamtaufwands durch Abschätzung der Kardinalitäten der Zwischenergebnisse
- ▶ Kardinalität über **Selektivität** der Operatoren

- Selektivität *sel*:

$$sel = \frac{\text{Erwartete Größe des Ergebnisses}}{\text{Kardinalität der Eingangsrelation}}$$

- Annahmen: Gleichverteilung, Unabhängigkeit der Attribute

Kostenformeln: Selektion

$$|\sigma_F(r(R))| = \text{sel}(F, R) \cdot |r|$$

- Abschätzungen (für interpolierbare, arithmetische Werte):

$$\text{sel}(A = v, R) = \frac{1}{\text{val}_{A,r}}$$

$$\text{sel}(A < v, R) = \frac{v - A_{\min}}{A_{\max} - A_{\min}}$$

$$\text{sel}(A > v, R) = \frac{A_{\max} - v}{A_{\max} - A_{\min}}$$

$$\text{sel}(A \text{ **between** } v_1 \text{ **and** } v_2, R) = \frac{v_2 - v_1}{A_{\max} - A_{\min}}$$

Kostenformeln: Selektion /2

$$sel(p(A_i) \wedge p(A_j), R) = sel(p(A_i), R) \cdot sel(p(A_j), R)$$

$$sel(p(A_i) \vee p(A_j), R) = sel(p(A_i), R) + sel(p(A_j), R) - sel(p(A_i), R) \cdot sel(p(A_j), R)$$

$$sel(A \in \{v_1, \dots, v_n\}, R) = sel(A = v, R) \cdot |\{v_1, \dots, v_n\}|$$

$$sel(\neg p(A), R) = 1 - sel(p(A), R)$$

Kostenformeln: Projektion

- ohne Duplikateliminierung

$$|\pi(r)| = |r|$$

- mit Duplikateliminierung (allgemein)

$$|\pi(r)| = 1 \dots |r|$$

- über Schlüsselattribut

$$|\pi_K(r)| = \text{val}_{K,r} \text{ falls } K \text{ Schlüssel von } R \text{ ist}$$

- über mehrere Attribute

$$|\pi_{A_1, \dots, A_n}(r)| = \min\left\{\frac{1}{2}|r|, \Pi_{i=1 \dots n} \text{val}_{A_i, r}\right\}$$

Kostenformeln: Verbund

- Verbund: $r \bowtie s$ mit $r = r(R)$ und $s = r(S)$ sowie $R(A, B)$ und $S(B, C)$
 - ▶ keine gemeinsamen B -Werte: $|r \bowtie s| = 0$.
 - ▶ Fremdschlüsselbeziehung $R.B \rightarrow S.B$, d.h. zu jedem Tupel von r gibt es genau ein Tupel in s : $|r \bowtie s| = |r|$
 - ▶ Extremfall: alle Tupel in $R.B$ und $S.B$ haben den gleichen Wert: $|r \bowtie s| = |r| \cdot |s|$

Kostenformeln: Verbund /2

- allgemeiner Fall:

$$|r \bowtie s| = \frac{|r| \cdot |s|}{\max\{val_{B,r}, val_{B,s}\}}$$

- über mehrere Verbundattribute (hier: B und C)

$$|r \bowtie s| = \frac{|r| \cdot |s|}{\max\{val_{B,r}, val_{B,s}\} \cdot \max\{val_{C,r}, val_{C,s}\}}$$

Kostenformeln: Mengenoperationen

- Vereinigung

- ▶ ohne Duplikateliminierung (**union all**): $|r \cup s| = |r| + |s|$
- ▶ allgemein: untere Schranke $\max\{|r|, |s|\}$, obere Schranke $|r| + |s|$
- ▶ Mittelwert: $|r \cup s| = \frac{\max\{|r|, |s|\} + |r| + |s|}{2}$

- Differenz

- ▶ untere Schranke $|r| - |s|$ (Extremfall: 0), obere Schranke: $|r|$
- ▶ Mittelwert: $|r - s| = \frac{|r| + |r| - |s|}{2} = |r| - \frac{1}{2}|s|$

- Schnittmenge

- ▶ untere Schranke 0, obere Schranke: $\min\{|r|, |s|\}$
- ▶ aber: eigentlich Verbundberechnung (siehe dort)

Kostenformeln: Gruppierung

- nur Gruppierung; Aggregation ändert Kardinalität nicht
- untere Schranke: 1, obere Schranke: $|r|$
- bei mehreren Gruppierungsattributen:

$$|\gamma_{F;A_1,\dots,A_n}(r)| = \min\{\frac{1}{2}|r|, \prod_{i=1\dots n} val_{A_i,r}\}$$

Kostenschätzung am Beispiel

- Datenbankparameter:

$b_{size} = 1000$, $mem = 5$

- Relation PRODUKT (P):

$|r(P)| = 5.000$, $val_{ProdNr, P} = 5.000$, $val_{LName, P} = 70$, $size_P = 100$, $b_P = 500$,
 $lev_{I(P(LName))} = 4$

- Relation BESTELLUNG (B):

$|r(B)| = 20.000$, $val_{BestNr, B} = 20.000$, $val_{ProdNr, B} = 3.000$, $val_{KNr, B} = 800$,
 $val_{Datum, B} = 730$, $size_B = 100$, $b_B = 2.000$, $lev_{I(B(Datum))} = 4$

- Relation LIEFERANT (L):

$|r(L)| = 100$, $val_{LName, L} = 100$, $size_L = 200$, $b_L = 20$

- Relation KUNDE (K):

$|r(K)| = 1.000$, $val_{KNr, K} = 1.000$, $size_K = 200$, $b_K = 200$

Kostenschätzung am Beispiel /2

- Anfrage: $\sigma_{\text{LName}='Coffeeshop'}(r(\text{PRODUKT}))$

- Variante 1:

$$\sigma_{\text{LName}='Coffeeshop'}^{\text{REL}}(r(\text{PRODUKT}))$$

- ▶ Kosten:

$$\text{cost} = b_p = 500$$

- Variante 2:

$$\rho(\sigma_{\text{LName}='Coffeeshop'}^{\text{IND}}(I(\text{PRODUKT}(\text{LName}))), r(\text{PRODUKT}))$$

- ▶ Kosten:

$$\text{cost} = \text{lev}_{I(P(\text{LName}))} + \frac{|r(\text{PRODUKT})|}{\text{val}_{\text{LName}, P}} = 4 + 72 = 76$$

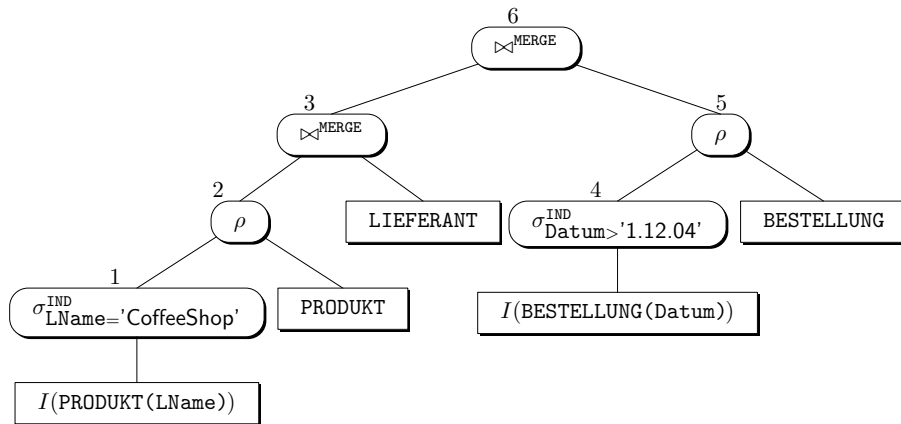
Kostenschätzung am Beispiel /3

- Anfrage: $\sigma_{\text{LName} > 'Coffeeshop'}(r(\text{PRODUKT}))$
- Selektivitätsabschätzung 0.3 (Heuristik)
- Kosten:

$$\text{cost} = \text{lev}_{I(P(\text{LName}))} + |r(\text{PRODUKT})| \cdot 0,3 \approx 1504$$

- Indexnutzung nicht immer zweckmäßig!

Kostenschätzung am Beispiel /4



Kostenschätzung am Beispiel /5

1 Knoten 1 und 2

$$|\sigma_{\text{LName}='Coffeeshop'}(r(\text{PRODUKT}))| = 5.000 \cdot \frac{1}{70} \approx 72 \text{ Tupel}$$

2 Knoten 3: Verbund über Fremdschlüssel \rightsquigarrow keine Änderung der Kardinalität

3 Knoten 4 und 5: Annahme eines bekannten Wertebereichs (hier: 1.1.2003 – 31.12.2004)

$$sel = \frac{730 - (730 - 31)}{730 - 0} = 0,042$$

$$|\sigma_{\text{Datum} > '1.12.04'}(r(\text{BESTELLUNG}))| = 20.000 \cdot 0,042 = 840$$

4 Knoten 6:

$$\frac{72 \cdot 840}{\max\{5.000, 3.000\}} \approx 13 \text{ Tupel}$$

Kostenschätzung am Beispiel /6

- 1 Selektion in Knoten 1 und 2 erfordert Traversierung des B^+ -Baums + (im ungünstigsten Fall) für jedes gefundene Tupel einen Blockzugriff

$$cost_2 = 4 + 72 = 76 \text{ Blöcke}$$

Ergebnis: $100 \cdot 72 / 1000 = 8$ Blöcke

- 2 Verbund erfordert nur noch Sortieren und Einlesen der Relation `LIEFERANT`

$$cost_3 = 20 \log_5 20 + 20 = 57,2$$

Ergebnis: $(100 + 200) \cdot 72 / 1000 = 22$ Blöcke

Kostenschätzung am Beispiel /7

- ③ Kosten der Selektion auf der Relation `BESTELLUNG`

$$cost_4 = 4 + 840 = 844 \text{ Blöcke}$$

Ergebnis: 84 Blöcke

- ④ abschließender Verbund:

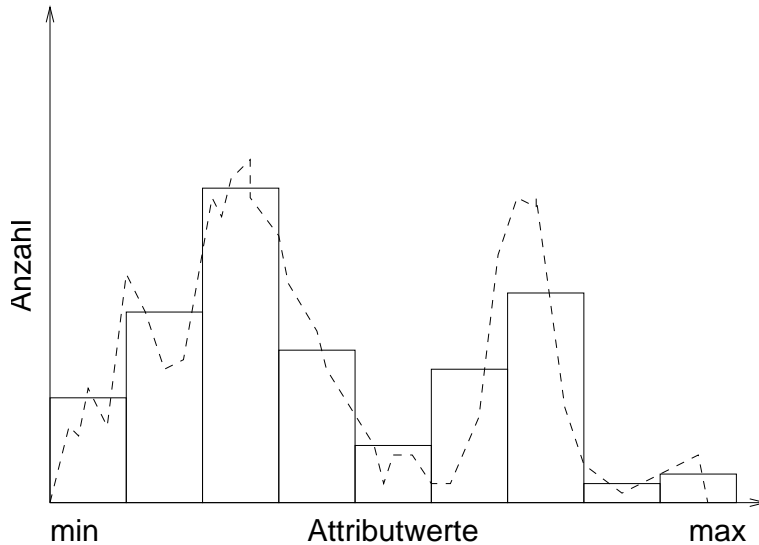
$$cost_5 = 22 \log_5 22 + 22 + 84 \log_5 84 + 84 = 379,05$$

$$cost = 76 + 57,2 + 844 + 379,05 \approx 1356 \text{ Blöcke}$$

Verbesserung der Abschätzungen

- *Parametrisierte Funktionen*: Parameter einer Funktion zur Annäherung der Datenverteilung (z.B. Normal- oder Zipf-Verteilung)
- *Stichprobe*: Selektivität anhand einer zufälligen Stichprobe bestimmen
- *Histogramme*: Approximationen der tatsächlichen Verteilung

Histogramme: Prinzip



Histogramme: Begriffe

- Menge der Werte eines Attributes A : $\mathcal{V} \subseteq \text{dom}(A)$

$$\mathcal{V} = \{v_i \mid 1 \leq i \leq |\mathcal{V}|\}$$

- für Werte von A besteht Ordnung: $i < j \rightarrow v_i < v_j$
- Häufigkeit f_i eines Wertes v_i : Anzahl der Tupel $t \in r(R)$ mit $t(A) = v_i$
- kumulative Häufigkeit: c_i der Anzahl der Tupel t mit $t(A) \leq v_i$
- Spanne: $s_i = v_{i+1} - v_i$
- Datenverteilung:

$$\mathcal{T} = \{(v_1, f_1), (v_2, f_2), \dots, (v_{|\mathcal{V}|}, f_{|\mathcal{V}|})\}$$

- kumulative Datenverteilung:

$$\mathcal{T}^c = \{(v_1, c_1), (v_2, c_2), \dots, (v_{|\mathcal{V}|}, c_{|\mathcal{V}|})\}$$

Histogramme: Begriffe /2

- Konstruktion eines Histogramms
 - ▶ Datenverteilung \mathcal{T} in $\beta \geq 1$ disjunkte Teilmengen partitionieren
- Teilmenge = **Bucket**: enthält Gruppe von \mathcal{T} -Elementen, die bezüglich eines Sortierparameters benachbart sind
- Bucket enthält Approximation bezüglich Wert und Häufigkeit

Histogramme: Freiheitsgrade

- **Anzahl der Elemente** aus \mathcal{T} , die einem Bucket zugewiesen werden können; z.B. bei End-Biased-Histogrammen: alle Buckets bis auf eines einelementig
- **Sortier- und Quellparameter**: auf welcher Basis werden Elemente in einem Bucket zusammengefasst (Sortierparameter) bzw. welcher Parameter wird im Bucket repräsentiert (Quellparameter) ; z.B. Attributwert, Häufigkeit, kumulative Häufigkeit
- **Approximation der Werte** aus einem Bucket
- **Approximation der Häufigkeit**, üblicherweise Annahme gleichverteilter Häufigkeiten (Mittelwert)
- **Constraints**: auf den Quellwerten zur Bestimmung der Partitionierung

Histogramme: Wertapproximation

- continuous value assumption: alle m Werte aus $\text{dom}(A)$, die im Wertebereich des Buckets liegen, werden berücksichtigt
- uniform spread assumption: nur $k \leq m$ äquidistante Werte im Bucket repräsentieren (gleiche Spannen)
- Beispiel:
 - ▶ Attribut A mit Werten $0, 1, 2, \dots$; Anfrage: $10 \leq A \leq 25$
 - ▶ Bucket mit Wertebereich $[1, 100]$, Anzahl verschiedener Werte: 10, Summe der Häufigkeiten: 200
 - ▶ Annahme stetiger Werte: Bucket enthält $1, 2, \dots, 100$, jeweils Häufigkeit 2 \rightsquigarrow abgeschätzte Ergebnisgröße $16 \cdot 2 = 32$
 - ▶ Annahme gleicher Spannen: Bucket enthält $1, 12, 23, \dots, 89, 100$, jeweils Häufigkeit 20 \rightsquigarrow abgeschätzte Ergebnisgröße $2 \cdot 20 = 40$

Histogramme: Partitionierung

- **Equi-sum**: Summe der Quellwerte der Buckets ist gleich; entspricht dem β -ten Teil der Summe aller Quellwerte
- **V-optimal**: Minimierung der gewichteten Varianz der Quellwerte, d.h. für k_i als Anzahl der Elemente im Bucket i und V_i als Varianz der Quellwerte Buckets so wählen, dass $\sum_{i=1}^n k_i V_i$ minimal
- **Spline-basiert**: Minimierung der maximalen absoluten Differenz zwischen einem Quellwert und dem Mittelwert aller Quellwerte des Buckets

Equi-width-Histogramm

- genauer: *Equi-sum*(V, S)
 - ▶ Sortierparameter: Attributwert
 - ▶ Quellparameter: Spanne
- Zusammenfassung zusammenhängender Bereiche von Attributwerten in einem Bucket gleicher Breite („equi-width“, d.h. für Bucket mit $[v_{\min}, v_{\max}]$):

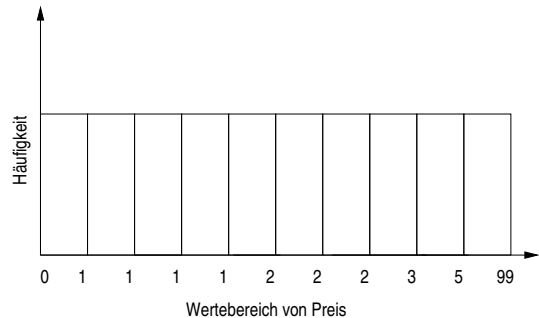
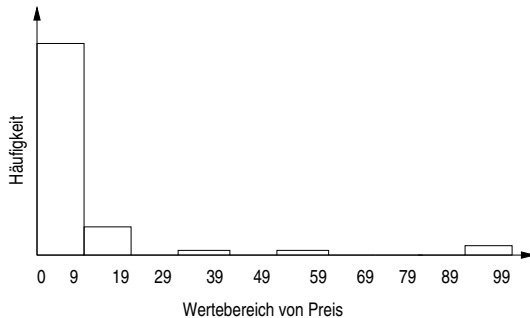
$$|v_{\max} - v_{\min}| \approx \frac{1}{\beta} (\max(\mathcal{V}) - \min(\mathcal{V}))$$

Equi-depth-Histogramm

- genauer: *Equi-sum*(V, F)
 - ▶ Sortierparameter: Attributwert
 - ▶ Quellparameter: Häufigkeit
- Häufigkeit („Höhe“) in allen Buckets gleich durch Anpassung der Breite bzw. Anzahl der belegten Buckets (Wertebereich)
- falls f_i von v_i größer als maximale Häufigkeit: über mehrere Buckets verteilen

Equi-width- vs. Equi-depth-Histogramm

Preis	1	2	3	5	10	30	50	99
Anzahl	100	75	30	20	15	3	2	5



Weitere Histogrammformen

- **V-Optimal(F, F)**: Zusammenfassung von benachbarten Häufigkeitswerten bei Minimierung der Varianz; **End-biased-Histogramme** als spezielle Form: einige der höchsten und niedrigsten Häufigkeitswerte in eigenen Buckets
- **Maxdiff**: Bucketgrenze zwischen zwei Quellwerten, wenn Abstand einer der $\beta - 1$ größten ist; speziell mit Fläche $a_i = f_i \cdot s_i$ als Constraint
- **Compressed**: die k höchsten Quellwerte getrennt in k einelementigen Buckets; Rest in Equi-sum

Nutzung von Histogrammen

- Punktanfrage `Preis=1` (100 Tupel bei angenommener Verteilung)
 - ▶ Abschätzung $\rightarrow 250/8 \approx 31$ Tupel
 - ▶ Equi-width-Histogramm: Bucket $[0, 9] \rightarrow$ Häufigkeit (250) auf 10 Werte verteilen $\rightarrow 225/10 \approx 23$ Tupel
 - ▶ Equi-depth-Histogramm: 4 Buckets mit Höhe 25 \rightarrow auf 2 Werte aufteilen $\rightarrow 50$ Tupel
 - ▶ Compressed-Histogramm: eigenes Bucket für Wert 1

Nutzung von Histogrammen /2

- Verbundanfragen: Kardinalitäten über korrespondierende Buckets berechnen

```
select *  
from BESTELLUNG, REKLAMATION  
where BESTELLUNG.ProdNr = REKLAMATION.ProdNr
```

- Equi-width-Histogramm jeweils über `ProdNr`

Nutzung von Histogrammen /3

Bucket	BESTELLUNG (B)	REKLAMATION (R)
[1 – 499]	2000	100
[500 – 999]	100	70
[1000 – 1499]	700	0
[1500 – 1999]	500	0
[2000 – 2499]	700	10
[2500 – 2999]	5000	0
[3000 – 3499]	500	40
[3500 – 3999]	4000	0
[4000 – 4499]	4500	90
[4500 – 4999]	2000	70

Nutzung von Histogrammen /4

- weitere Annahmen:

- ▶ $|r(R)| = 380$, $val_{ProdNr,R} = 250$
- ▶ $|r(B)| = 20.000$, $val_{ProdNr,B} = 3.000$

- Abschätzung:

$$|r(B) \bowtie r(R)| = \frac{380 \cdot 20.000}{\max\{250, 3000\}} \approx 2533 \text{ Tupel}$$

- mit Histogramm

$$|r(B) \bowtie r(R)| = \frac{2000 \cdot 100}{500} + \frac{100 \cdot 70}{500} + \frac{700 \cdot 10}{500} + \frac{500 \cdot 40}{500} + \frac{4500 \cdot 90}{500} + \frac{2000 \cdot 70}{500} = 1558 \text{ Tupel}$$

Histogramme: Konstruktion und Pflege

- Forderung: Histogramm muss tatsächliche Verteilung widerspiegeln – auch nach Updates
- statischer Ansatz: expliziter Aufbau, keine Berücksichtigung von Änderungen
- dynamischer Ansatz: Anpassung der Histogramme
 - ▶ Verwaltung einer Stichprobe der Relation und Abbildung der Änderungen auf Stichprobe
 - ▶ Query Feedback: Nutzung von Anfrageergebnissen zur Anpassung

Kostenbasierte Planauswahl

- Strategie zur Suche nach kostenoptimalen Plan
- Kosten: Gesamtkosten auf Basis des Kostenmodells
- Ziel: Vermeiden einer erschöpfenden Suche
- deterministische vs. zufallsbasierte Verfahren

Greedy

- Idee: vollständigen Plan schrittweise aus partiellen Lösungen (Teilplänen) konstruieren
- hier: Hinzunahme von Verbund mit minimalen Kosten zu optimalen Teilplan

Greedy /2

Input: Verbundanfrage Q mit n Relationen r_1, \dots, r_n

Output: Anfrageplan $optPlan$

foreach $r_i, r_j \in Q$ mit $i \neq j$ **do**

begin

 bestimme $|r_i \bowtie r_j|$

end

$optPlan := (r_i \bowtie r_j)$ mit $|r_i \bowtie r_j|$ ist minimal

for $k := 3$ **to** n **do begin**

foreach $r_k \in \{Q - optPlan\}$ **do**

begin

 bestimme $|optPlan \bowtie r_k|$

end

$optPlan := (optPlan \bowtie r_k)$ mit $|optPlan \bowtie r_k|$ ist minimal

end

return ($optPlan$)

Greedy: Beispiel

- Anfrage

$$r(\text{BESTELLUNG}) \bowtie r(\text{PRODUKT}) \bowtie r(\text{KUNDE}) \bowtie r(\text{LIEFERANT})$$

- 1. Schritt

Plan	Ergebnisgröße
$r(\text{KUNDE}) \bowtie r(\text{PRODUKT})$	5.000.000
$r(\text{KUNDE}) \bowtie r(\text{LIEFERANT})$	1.000.000
$r(\text{KUNDE}) \bowtie r(\text{BESTELLUNG})$	20.000
$r(\text{PRODUKT}) \bowtie r(\text{LIEFERANT})$	5.000
$r(\text{PRODUKT}) \bowtie r(\text{BESTELLUNG})$	20.000
$r(\text{LIEFERANT}) \bowtie r(\text{BESTELLUNG})$	2.000.000

Greedy: Beispiel /2

- 2. Schritt

Plan	Ergebnisgröße
$(r(P) \bowtie r(L)) \bowtie r(B)$	20.000
$(r(P) \bowtie r(L)) \bowtie r(K)$	5.000.000

- 3. Schritt: Verbund $r(\text{KUNDE})$

$$((r(L) \bowtie r(P)) \bowtie r(B)) \bowtie r(K)$$

Greedy: Anmerkungen

- mit obiger Strategie nur links-orientierte Bäume
- Vorteil von Greedy-Strategie:
 - ▶ nur wenige Pläne zu berücksichtigen (hier 8 anstelle von $5! = 120$)
- Nachteil:
 - ▶ keine Garantie, dass optimaler Plan gefunden wird
- Varianten: Berücksichtigung des Selektivitätsfaktors der Relation anstelle der Größe der Zwischenergebnisse

PostgreSQL: Statistiken

- zu Tabellen und Indexen in `pg_class`: u.a. Anzahl der Tupel (`reltuples`); Anzahl der belegten Blöcke (`relpages`)
- zu einzelnen Spalten in Tabelle `pg_statistic` bzw. Sicht `pg_stats`: der Anteil der Nullwerte (`null_frac`), durchschnittliche Größe in Bytes (`avg_width`), Anzahl der verschiedenen Attributwerte (`n_distinct`)
- Histogramme:
 - ▶ Compressed-Histogramme: Feld der häufigsten Werte (`most_common_vals`), dazu korrespondierend Feld mit den Häufigkeiten dieser Werte (`most_common_freqs`)
 - ▶ Rest der Daten in Equi-depth-Histogramm (`histogram_bounds`)

PostgreSQL: Statistiken /2

- Anlegen von Statistiken

```
analyze PRODUKT (Preis);
```

- Histogrammgröße (Anzahl der Buckets)

```
alter table PRODUKT set statistics 20;
```

- Relation > 100 Tupel: Stichprobe

PostgreSQL: Optimierer

- kostenbasierter Optimierer auf Basis dynamischer Programmierung (sowie genetischer Algorithmus)
- Beeinflussung
 - ▶ Suchraum für Verbundreihenfolge (`set join_collapse_limit = 1;`)
 - ▶ Ausschalten einzelner Planoperatoren (`set enable_nestloop = off;`)

Zusammenfassung

- Ablauf der Anfrageverarbeitung
- Vereinfachung und Normalisierung von Anfragen
- algebraische Optimierung (Rewriting)
- Kostenmodell
- physische Optimierung
- Suchstrategien zur kostenbasierten Planauswahl