

Vorlesung  
**Datenbanken II**  
*OvGU Magdeburg, SomSem 2004*

Gunter Saake

saake@iti.cs.uni-magdeburg.de

# Überblick

---

1. Aufgaben und Prinzipien von Datenbanksystemen
2. Architektur von Datenbanksystemen
3. Verwaltung des Hintergrundspeichers
4. Dateiorganisation und Zugriffsstrukturen
5. Zugriffsstrukturen für spezielle Anwendungen
6. Basisalgorithmen für Datenbankoperationen
7. Optimierung von Anfragen
8. Weitere Aspekte und Ausblick

# Nötiges Vorwissen

---

Datenbanken I:

- Grundprinzipien Datenbanksysteme
- Tabellen, Attribute, Schlüssel
- Relationale Algebra und SQL

*Wird am Anfang der Vorlesung kurz wiederholt!*

# Literatur

---

- Saake, G.; Heuer, A.: *Datenbanken — Implementierungskonzepte*. mitp-Verlag, Mai 1999  
*Achtung — Sonderbestellung beim Verlag notwendig da zur Zeit ausverkauft!*
- Härder, T.; Rahm, E.: *Datenbanksysteme — Konzepte und Techniken der Implementierung*. Springer-Verlag, 2001
- Garcia-Molina, H.; Ullman, J.; Widom, J.: *Database System Implementation*. Addison-Wesley, 1999.
- Silberschatz, A.; Korth, H. F.; Sudarshan, S.: *Database System Concepts*. Wiley & Sons, 2001.

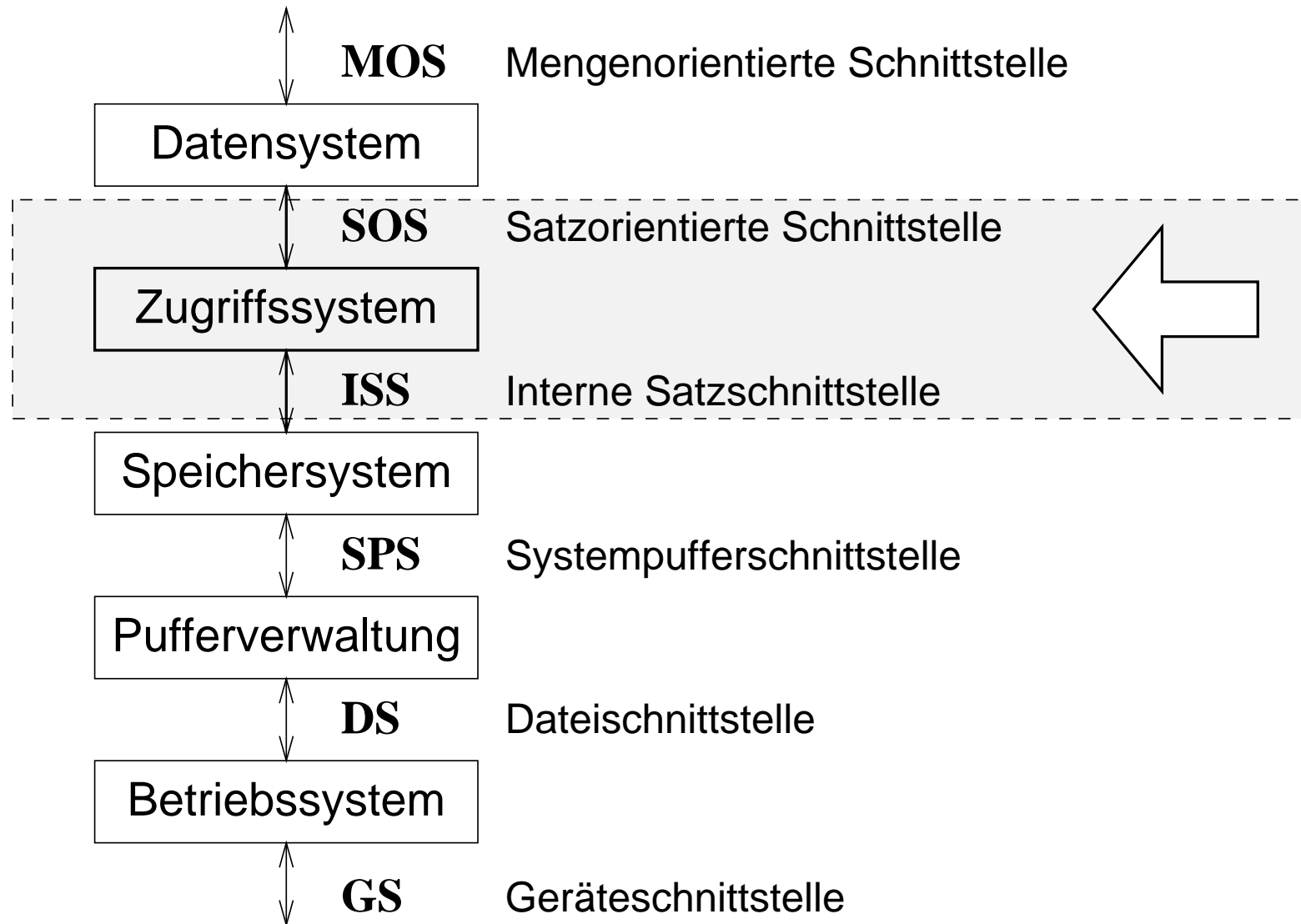
# 6. Basialgorithmen für DB-Operationen

---

- Datenbankparameter
- Komplexität von Grundalgorithmen
- Unäre Operationen (Scan, Selektion, Projektion)
- Binäre Operationen: Mengenoperationen
- Berechnung von Verbunden

# Einordnung

---



# Datenbankparameter

---

- Komplexitätsbetrachtungen ( $O(n^2)$ )
- Aufwandsabschätzungen (konkret)
- Datenbankparameter als Grundlage
- müssen im Katalog des Datenbanksystems gespeichert werden

# Datenbankparameter (II)

---

- $n_r$ : Anzahl Tupel in Relation  $r$
- $b_r$ : Anzahl von Blöcken (Seiten), die Tupel aus  $r$  beinhalten
- $s_r$ : (mittlere) Größe von Tupeln aus  $r$  ( $s$  für *size*)
- $f_r$ : *Blockungsfaktor* (Tupel aus  $r$  pro Block)

$$f_r = \frac{b_s}{s_r},$$

mit  $b_s$  Blockgröße

- Tupel einer Relation kompakt in Blöcken:

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

# Datenbankparameter (III)

---

- $V(A, r)$ : Anzahl verschiedener Werte für das Attribut  $A$  in der Relation  $r$  ( $V$  für *values*):  $V(A, r) = |\pi_A(r)|$
- $A$  Primärschlüssel:  $V(A, r) = n_r$
- $SC(A, r)$ : *Selektionskardinalität (selection cardinality)*; durchschnittliche Anzahl von Ergebnistupeln bei  $\sigma_{A=x}(r)$  für  $x \in \pi_A(r)$
- Schlüsselattribut  $A$ :  $SC(A, r) = 1$
- Allgemein:

$$SC(A, r) = \frac{n_r}{V(A, r)}$$

Weiterhin: Verzweigungsgrad bei B-Baum-Indexen, Höhe des Baums, Anzahl von Blätterknoten

# Komplexität von Grundalgorithmen

---

## Grundannahmen

- Indexe B<sup>+</sup>-Bäume
- dominierender Kostenfaktor: Blockzugriff
- Zugriff auf Hintergrundspeicher auch für Zwischenrelationen
- 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  $I(\text{Personen}(\text{PANr}))$   
bei  $A = a$  wird maximal ein Tupel pro Zugriff
    - ◆ *Sekundärindex*, etwa  $I(\text{Ausleihe}(\text{PANr}))$   
Bsp.:  $\text{PANr} = 4711$  liefert i.a. mehrere Tupel
- Indexzugriffe: Ergebnis TID-Listen

# Zugriffe auf Datensätze (II)

---

- **fetch-tupel** Direktzugriff auf Tupel mittels TID-Wertes holt Tupel in *Tupel-Puffer*

**fetch-tupel**(RelID, TID) → *Tupel-Puffer*

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

**fetch-TID**(IndexID, Attributwert) → TID

- weiterhin auf Relationen und Indexen: *Scans*

# Beispiel in SQL

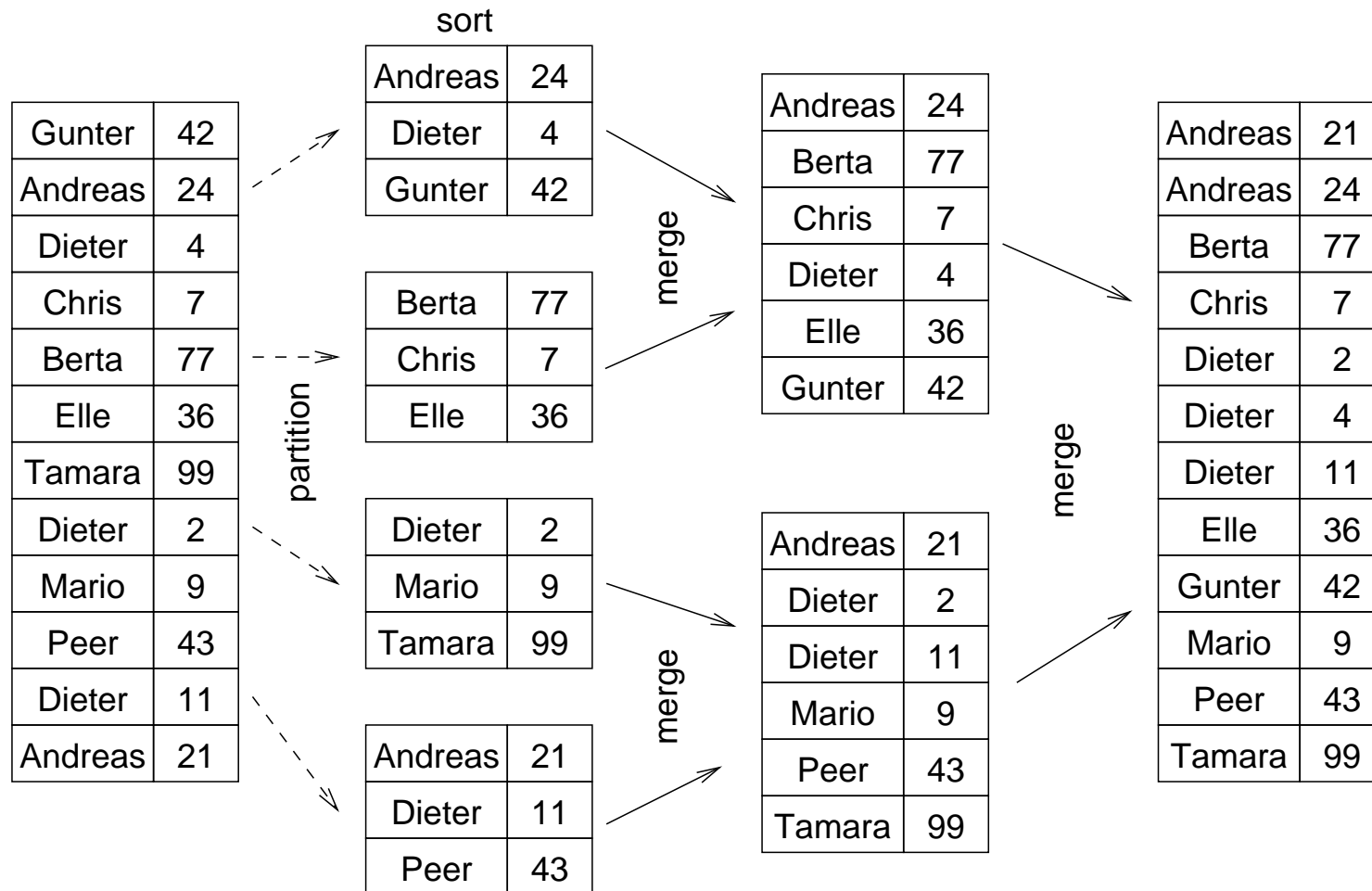
---

```
select *  
from KUNDE  
where KName = 'Meier'
```

- Gleichheitsanfrage über einen Schlüssel
- **put**: hier Anzeige des Ergebnisses

```
aktuellerTID :=  
    fetch-TID(KUNDE-KName-Index, 'Meyer');  
aktuellerPuffer :=  
    fetch-tupel(KUNDE-RelationID, aktuellerTID);  
put(aktuellerPuffer);
```

# Externe Sortieralgorithmen



Externes Sortieren durch Mischen; Komplexität  $O(n \log n)$   
Vertauschoperationen

# Unäre Operationen

---

Scan durchläuft Tupel einer Relation

- *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) → ScanID  
liefert ScanID zurück, die bei folgenden Operationen zur Identifikation genutzt wird
- Index-Scan initialisieren  
**open-index-scan**(IndexID, Min, Max) → 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 Personen  
where Nachname between 'Heuer' and  
        'Jagellowsk'
```

# Beispiel: Relationen-Scan

---

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

# Beispiel: Index-Scan

---

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

# Selektion

---

- *exakte Suche, Bereichsselektionen*, komplex zusammengesetzte Selektionskriterien
- zusammengesetztes Prädikat  $\varphi$  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  $O(n_r)$ , genauer  $b_r$

# Selektion: Konjunktive Normalform

---

- Zugriffspfade bei komplexen Prädikaten einsetzen  $\Rightarrow \varphi$  analysieren und geeignet umformen
- etwa  $\varphi$  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

# Selektion: Filtermethoden

---

- bei *Filtermethode* alle Bedingungen auf **true** setzen, die nicht durch eine Zugriffsmethode unterstützt werden
- resultierendes Prädikat:  $\varphi'$ .
- $r' = \sigma_{\varphi'}(r)$  unter Ausnutzung der Indexe auswerten
- $\sigma_{\varphi}(r')$  auf dem (hoffentlich viel kleineren) Zwischenergebnis  $r'$  mittels tupelweisem Vorgehen auswerten
- Filtermethoden nur gut, wenn  $\varphi'$  tatsächlich Datenvolumen reduziert (Vorsicht bei Disjunktionen)

# 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 (II)

---

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

# 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 (II)

---

Halloween-Problem (System R):

- SQL-Anweisung:

```
update employee e  
  set salary = salary * 1.05
```

- satzorientierte Auswertung mittels Index-Scan über  $I_{\text{employee}}(\text{salary})$  und sofortige Index-Aktualisierung
- ohne besondere Vorkehrungen: unendliche Anzahl von Gehaltserhöhungen

# 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:  $O(n_s * n_r)$
- *Merge-Technik* oder *Mischmethode*
  - ◆  $r$  und  $s$  (sortiert) schrittweise in der vorgegebenen Tupelreihenfolge durchlaufen
  - ◆ Aufwand:  $O(n_s + n_r)$
  - ◆ Falls Sortierung noch vorzunehmen:  
*Sort-Merge-Technik*
  - ◆ Aufwand  $n_r \log n_r$  und/oder  $n_s \log n_s$

# Mengenoperationen (II)

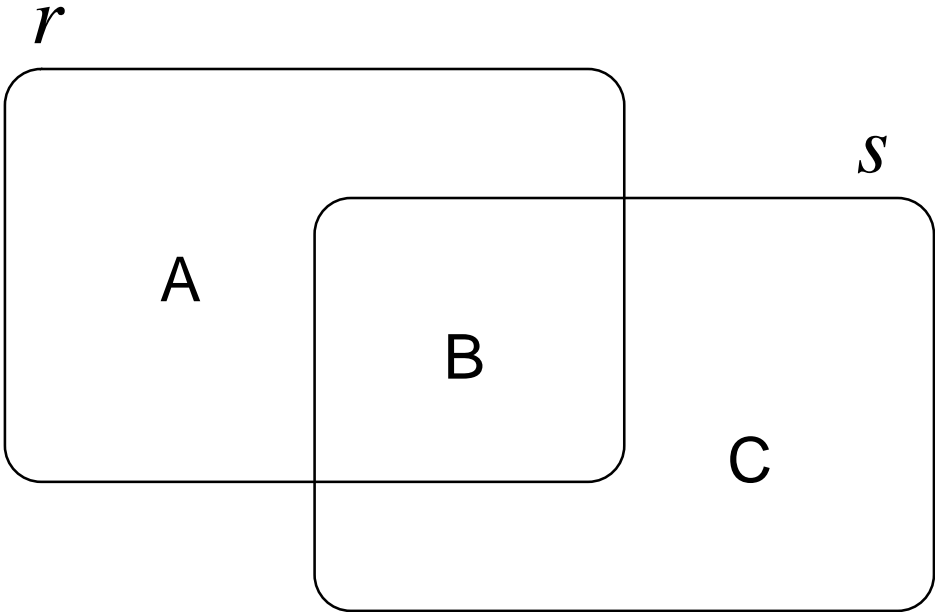
---

## ■ *Hash-Methoden*

- ◆ kleinere der beiden Relationen in Hash-Tabelle
- ◆ Tupel der zweiten Relation finden ihren Vergleichspartner mittels Hash-Funktion
- ◆ idealerweise Aufwand  $O(n_s + n_r)$

# Klassen binärer Operationen

---



# Klassen binärer Operationen (II)

Ergebnis- extensionen	Übereinstimmung auf allen Attributen	Übereinstimmung auf einigen Attributen
$A$	Differenz $r - s$	Anti-Semi- Verbund
$B$	Schnitt $r \cap s$	Verbund, Semi- Verbund
$C$	Differenz $s - r$	Anti-Semi- Verbund
$A \cup B$		Left Outer Join
$A \cup C$	symmetrische Dif- ferenz $(r - s) \cup (s - r)$	Anti-Verbund
$B \cup C$		Right Outer Join
$A \cup B \cup C$	Vereinigung $r \cup s$	Full Outer Join

# 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:  $O((n_r + n_s) \times \log(n_r + n_s))$  (wie Projektion)

# Vereinigung (II)

---

## 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:  $O(n_r \times \log n_r + n_s \times \log n_s)$  mit Sortierung,  $O(n_r + n_s)$  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  $t_1 \in r$  und alle  $t_2 \in s$  bei einer Operation  $r \bowtie s$

$r \bowtie_{\varphi} s$ :

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

# Nested-Loops-Verbund mit Scan

---

```
R1ScanID := open-rel-scan(R1ID);
R1TID := next-TID(R1ScanID);
while not end-of-scan(R1ScanID) do
begin
    R1Puffer := fetch-tupel(R1ID,R1TID);
    R2ScanID := open-rel-scan(R2ID);
    R2TID := next-TID(R2ScanID);
    while not end-of-scan(R2ScanID) do
    begin
        .../* Scan über innere Relation */
    end;
    close (R2ScanID);
    R1TID := next-TID(R1ScanID);
end;
close (R1ScanID);
```

# Nested-Loops-Verbund mit Scan II

---

```
/* Scan über innere Relation */
R2Puffer := fetch-tupel(R2ID,R2TID);
if R1Puffer.X = R2Puffer.Y
then insert into ERG
    (R1.Puffer.A1, ..., R1.Puffer.An, R1.Puffer.X,
     R2.Puffer.B1, ..., R1.Puffer.Bm);
endif;
R2TID := next-TID(R2ScanID);
```

Verbesserung: Nested-Loops-Verbund verbindet alle  $t_1 \in r$  mit Ergebnis von  $\sigma_{X=t_1(X)}(s)$  (gut bei Index auf  $X$  in  $r_2$ )

# 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
```

Aufwand:  $b_r * b_s$

# 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-Techniken: Aufwand

---

- alle Tupel haben den selben  $X$ -Wert:  $O(n_r \times n_s)$
- $X$  Schlüssel von  $R$  oder  $S$ :  $O(n_r \log n_r + n_s \log n_s)$
- bei vorsortierten Relationen sogar:  $O(n_r + n_s)$

# Merge-Join mit Scan

---

- Verbund-Attribute auf beiden Relationen  
Schlüsseleigenschaft
- $\min(x)$  und  $\max(x)$ : minimaler bzw. maximaler  
gespeicherter Wert für  $x$

# Merge-Join mit Scan (II)

---

```
R1ScanID := open-index-scan(R1XIndexID,  
    min(X), max(X));  
R1TID := next-TID(R1ScanID);  
R1Puffer := fetch-tupel(R1ID,R1TID);  
R2ScanID := open-index-scan(R2YIndexID,  
    min(Y), max(Y));  
R2TID := next-TID(R2ScanID);  
R2Puffer := fetch-tupel(R2ID,R2TID);  
while not end-of-scan(R1ScanID)  
    and not end-of-scan(R2ScanID) do  
begin  
    .../* merge */  
end;  
close (R1ScanID);  
close (R2ScanID);
```

# Merge-Join mit Scan (III)

---

```
/* merge */
if R1Puffer.X < R2Puffer.Y
then R1TID := next-TID(R1ScanID);
    R1Puffer := fetch-tupel(R1ID,R1TID);
else if R1Puffer.X > R2Puffer.y
    then R2TID := next-TID(R2ScanID);
        R2Puffer := fetch-tupel(R2ID,R2TID);
    else insert into ERG
        (R1.Puffer.A1, ..., R1.Puffer.An, R1.Puffer.X,
         R2.Puffer.B1, ..., R1.Puffer.Bm);
        R1TID := next-TID(R1ScanID);
        R1Puffer := fetch-tupel(R1ID,R1TID);
        R2TID := next-TID(R2ScanID);
        R2Puffer := fetch-tupel(R2ID,R2TID);
    endif;
endif;
```

# Verbund durch Hashing

---

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

# Classic Hashing

---

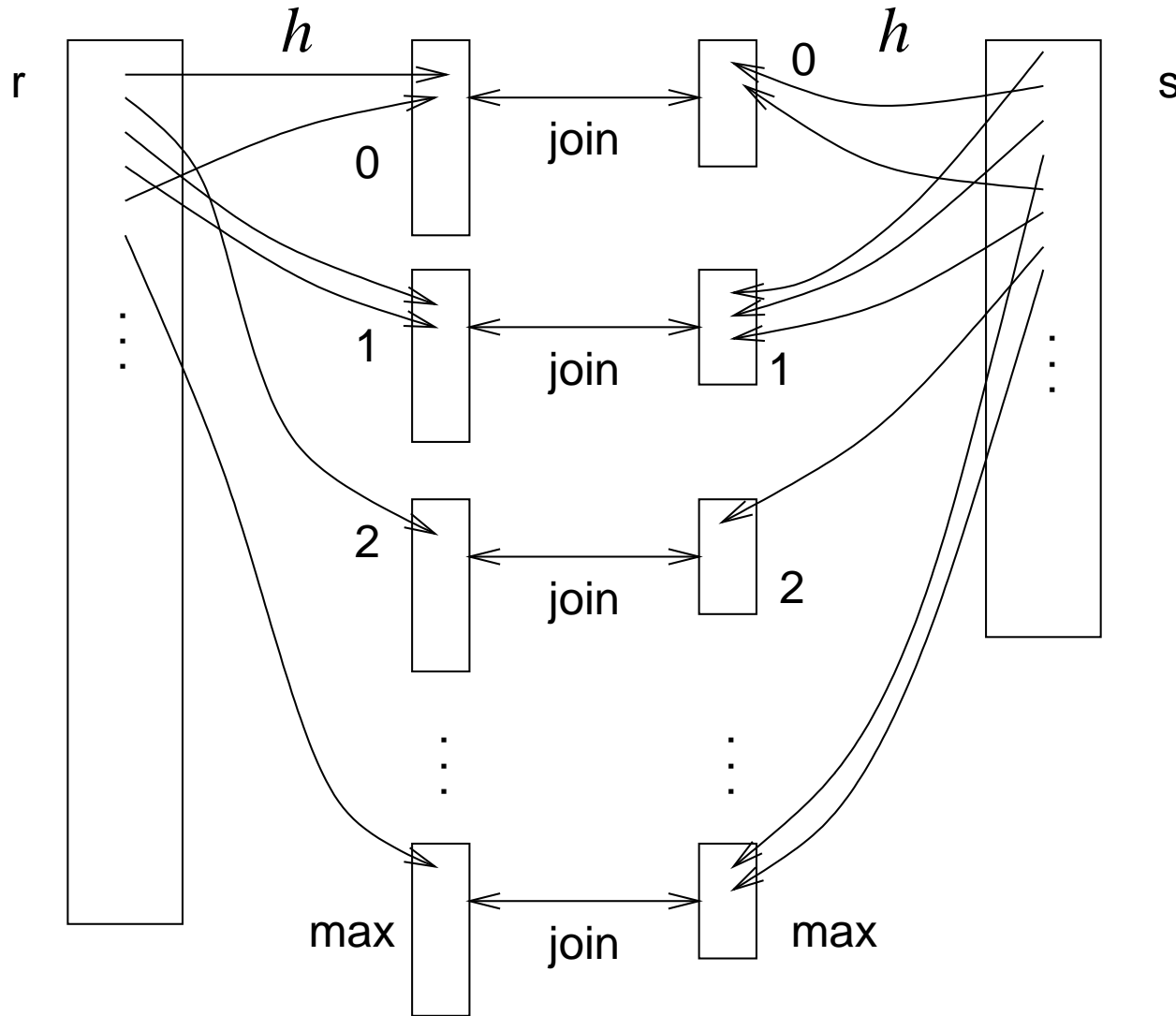
- Vorbereitung: kleinere Relation wird  $r$
- Ablauf
  1. Tupel von  $r$  mittels Scan in Hauptspeicher lesen und mittels Hashfunktion  $h(r.A)$  in Hashtabelle  $H$  einordnen
  2. wenn  $H$  voll (oder  $r$  vollständig gelesen): Scan über  $S$  und mit  $h(s.B)$  Verbundpartner suchen
  3. falls Scan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

# Partitionierung mittels Hashfunktion

---

- Tupel aus  $r$  und  $s$  über  $X$  in gemeinsame Datei mit  $k$  Blöcken (*Buckets*) „gehasht“
- Tupel in gleichen Buckets durch Verbundalgorithmus verbinden

# Partitionierung mittels Hashfunktion (II)



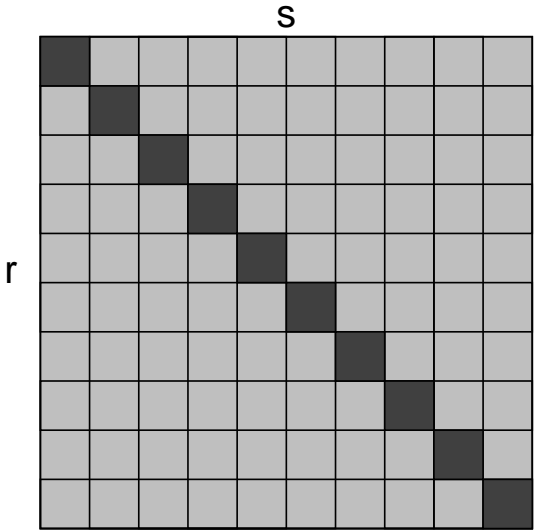
# Partitionierung mittels Hashfunktion (III)

---

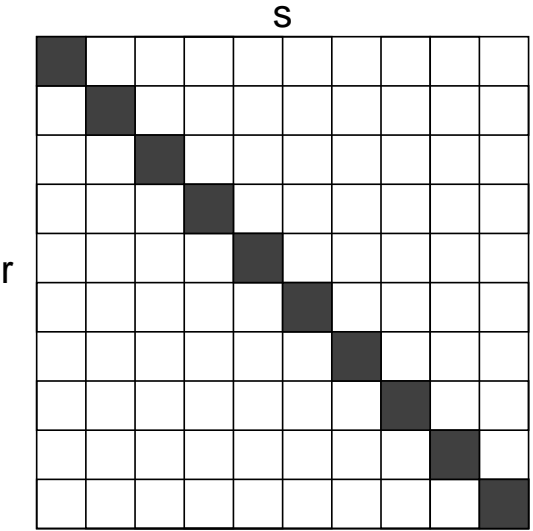
```
for each  $t_r$  in  $r$  do
  begin
     $i := h(t_r(X))$ ;
     $H_i^r := H_i^r \cup t_r(X)$ ;
  end;
for each  $t_s$  in  $s$  do
  begin
     $i := h(t_s(X))$ ;
     $H_i^s := H_i^s \cup t_s(X)$ ;
  end;
for each  $k$  in  $0 \dots \max$  do
   $H_k^r \bowtie H_k^s$ ;
```

# Vergleich der Techniken

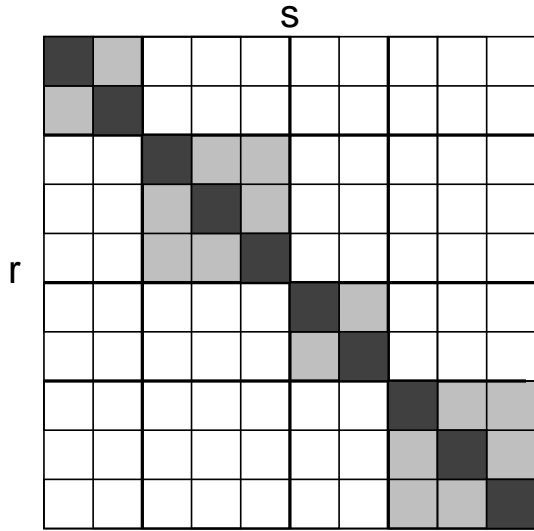
---



Nested-Loops-Join



Merge-Join



Hash-Join

# Aggregation & Gruppierung

---

- Anfragen:

```
select A, count ( * )  
from T  
group by A
```

- Algebraoperator:  $\gamma_{\text{count}(*),A}(r(t))$

- Implementierungsvarianten:

- ◆ Nested Loops
- ◆ Sortierung
- ◆ Hashing

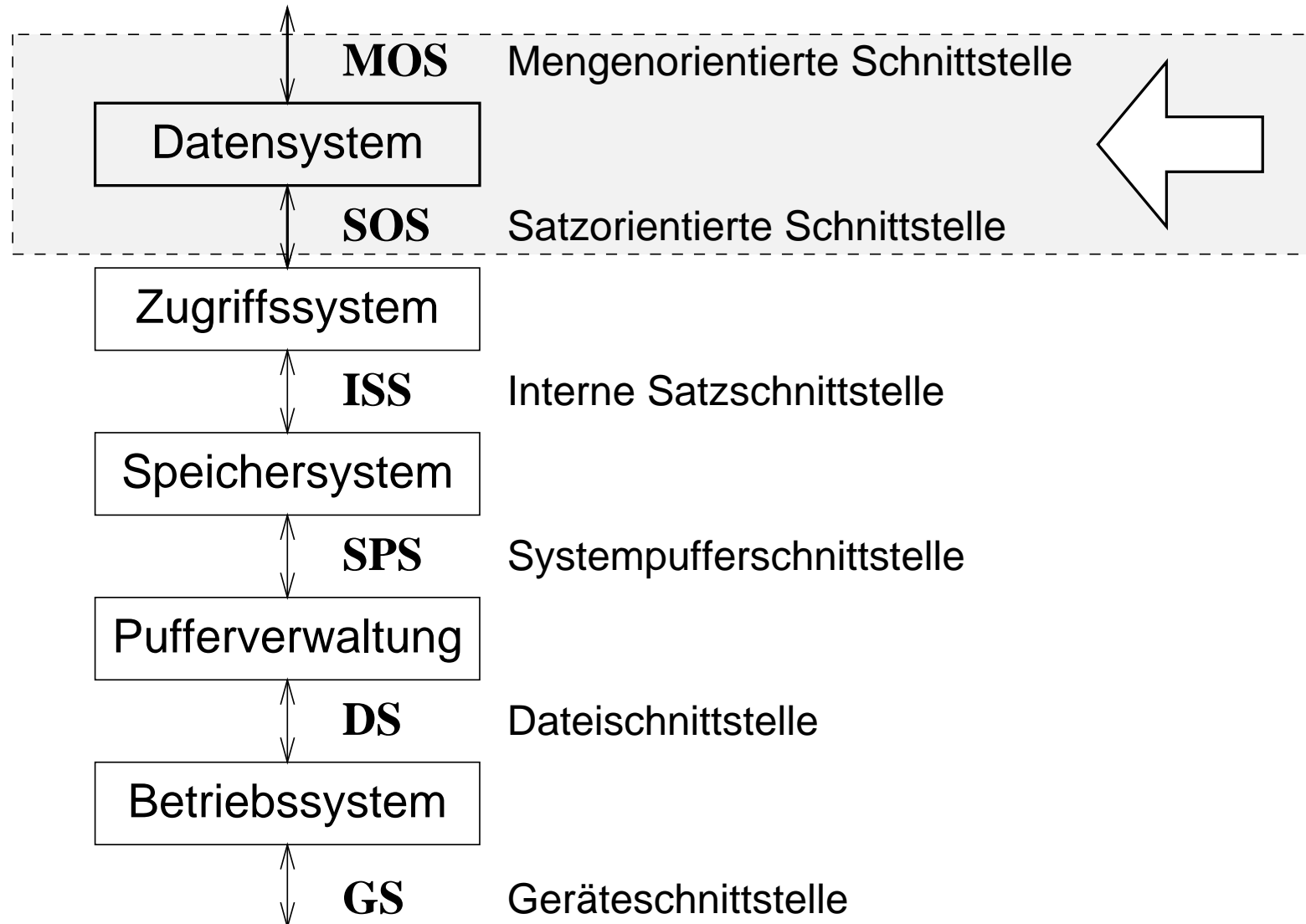
# 7. Optimierung von Anfragen

---

- Grundprinzipien, motivierende Beispiele
- Phasen der Anfrageverarbeitung
- Übersetzung von SQL in Relationenalgebra
- Logische Optimierung (algebraisch, Tableau)
- Interne Optimierung
- Kostenbasierte Auswahl

# Einordnung

---



# Grundprinzipien

---

## Basissprachen

- SQL
- Relationenkalküle
- hier: Relationenalgebra

## Ziel der Optimierung

- möglichst schnelle Anfragebearbeitung  $\Rightarrow$
- möglichst wenig Seitenzugriffe bei der Anfragebearbeitung  $\Rightarrow$
- möglichst in allen Operationen so wenig wie möglich Seiten (Tupel) berücksichtigen

# Teilziele einer Optimierung

---

1. Selektionen so früh wie möglich
2. Basisoperationen zusammengefasst und ohne Zwischenspeicherung realisieren
3. Redundante Operationen, Idempotenzen oder leere Zwischenrelationen entfernen
4. Zusammenfassen gleicher Teilausdrücke:  
Wiederverwendung von Zwischenergebnissen

# Beispiel

---

KUNDE { KName, Kadr, Kto }  
AUFTRAG { KName, Ware, Menge }

```
select KUNDE.KName, Kto  
from KUNDE, AUFTRAG  
where KUNDE.KName = AUFTRAG.KName  
       and Ware = 'Kaffee'
```

- Relation KUNDE: 100 Tupel; eine Seite: 5 Tupel
- Relation AUFTRAG: 10.000 Tupel; eine Seite: 10 Tupel
- 50 der Aufträge betreffen Kaffee
- Tupel der Form (KName, Kto): 50 auf eine Seite
- 3 Zeilen von KUNDE × AUFTRAG auf eine Seite
- Puffer für jede Relation Größe 1, keine Spannsätze

# Direkte Auswertung

---

1.  $R_1 := \text{KUNDE} \times \text{AUFTRAG}$

Seitenzugriffe:

■  $l : (100/5 * 10.000/10) = 20.000$

■  $s : (100 * 10.000)/3 = 333.000 \text{ (ca.)}$

2.  $R_2 := \sigma_{\text{SEL}}(R_1)$

■  $l : 333.000 \text{ (ca.)}$

■  $s : 50/3 = 17 \text{ (ca.)}$

3.  $ERG := \pi_{\text{PROJ}}(R_2)$

■  $l : 17$

■  $s : 1$

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

# Optimierte Auswertung

---

1.  $R_1 := \sigma_{\text{Ware}='Kaffee'}(\text{AUFTRAG})$

■  $l : 10.000/10 = 1.000$

■  $s : 50/10 = 5$

2.  $R_2 := \text{KUNDE} \bowtie_{\text{KName}=\text{KName}} R_1$

■  $l : 100/5 * 5 = 100$

■  $s : 50/3 = 17$

3.  $ERG := \pi_{\text{PROJ}}(R_2)$

■  $l : 17$

■  $s : 1$

ca. 1.140 Seitenzugriffe (Faktor 500 verbessert)

# Auswertung mit Indexausnutzung

---

Indexe  $I(\text{AUFTRAG}(\text{Ware}))$  und  $I(\text{KUNDE}(\text{KName}))$

1.  $R_1 := \sigma_{\text{Ware}='Kaffee'}(\text{AUFTRAG})$  über  $I(\text{AUFTRAG}(\text{Ware}))$ 
  - $l$  : minimal 5, maximal 50;  $s$  :  $50/10 = 5$
2.  $R_2 := \text{sortiere } R_1 \text{ nach KName}$ 
  - $l + s$  :  $5 * \log 5 = 15$  (ca.)
3.  $R_3 := \text{KUNDE} \bowtie_{\text{KName}=\text{KName}} R_2$ 
  - $l$  :  $100/5 + 5 = 25$ ;  $s$  :  $50/3 = 17$
4.  $ERG := \pi_{\text{PROJ}}(R_3)$ 
  - $l$  : 17;  $s$  : 1

maximal ca. 130 und minimal ca. 85 Seitenzugriffe

# Gegenüberstellung der Varianten

---

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

# Phasen der Anfrageverarbeitung

---

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 II

---

## 3. *Interne Optimierung*

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

## 4. *Kostenbasierte Auswahl*

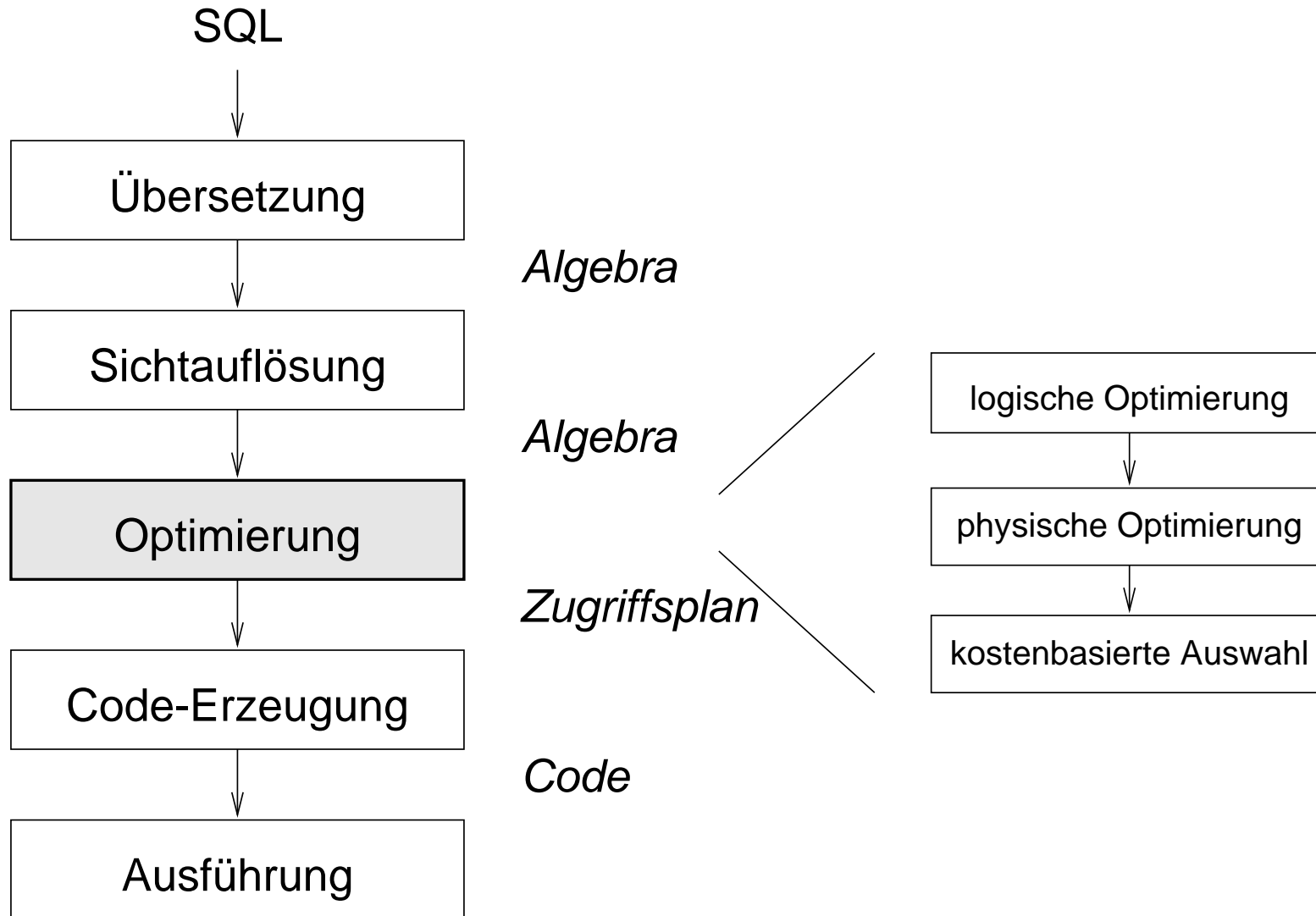
- Statistikinformationen (Größe von Tabellen, Selektivität von Attributen) für die Auswahl eines konkreten internen Planes nutzen

## 5. *Code-Erzeugung*

- Umwandlung des Zugriffsplans in ausführbaren Code

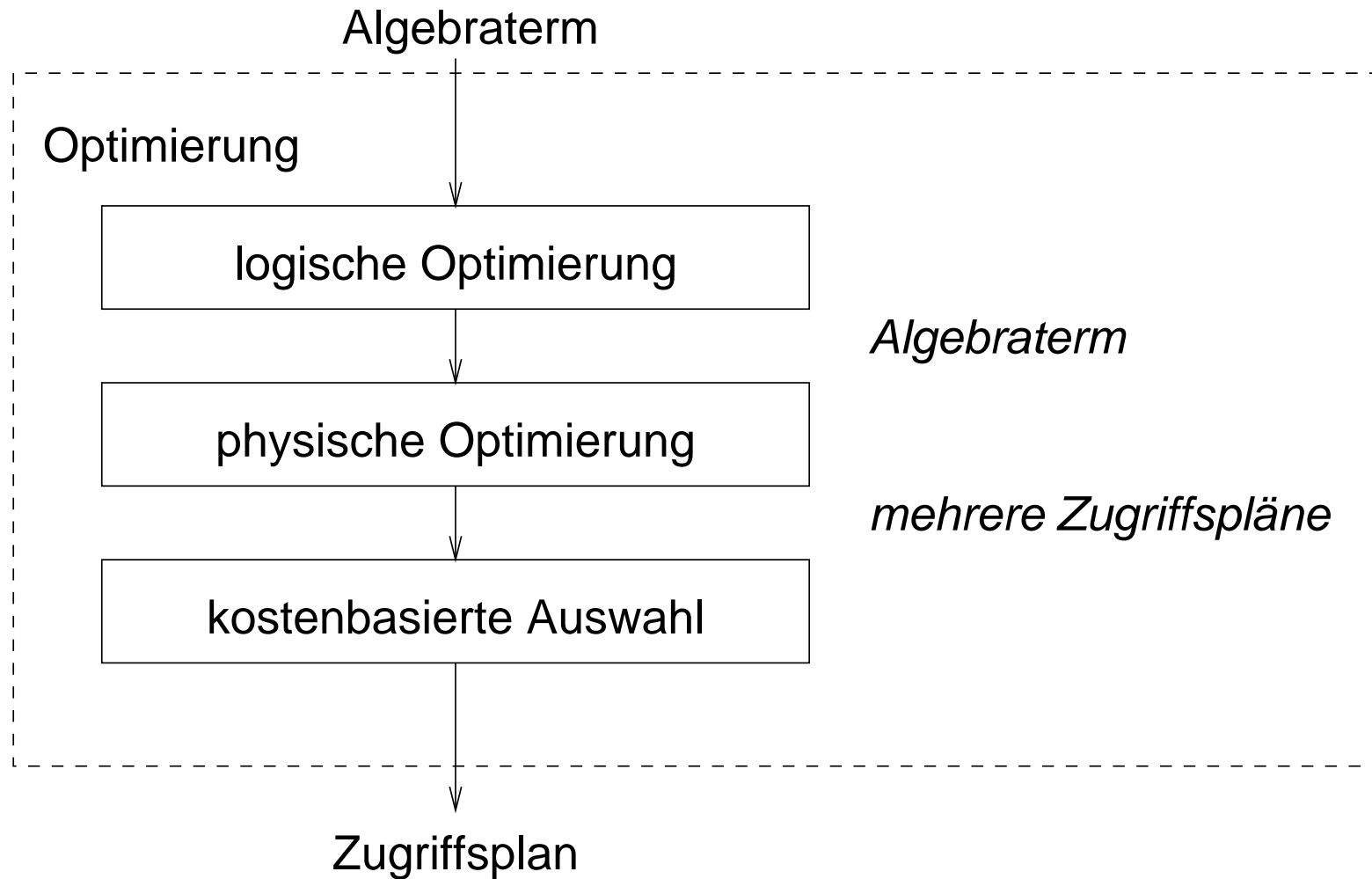
# Ablauf und Sprachen

---



# Phasen der Optimierung

---



# Übersetzung in Relationenalgebra

---

```
select A1, ..., Am
from R1, R2, ..., Rn
where F
```

Umsetzung in Relationenalgebra:

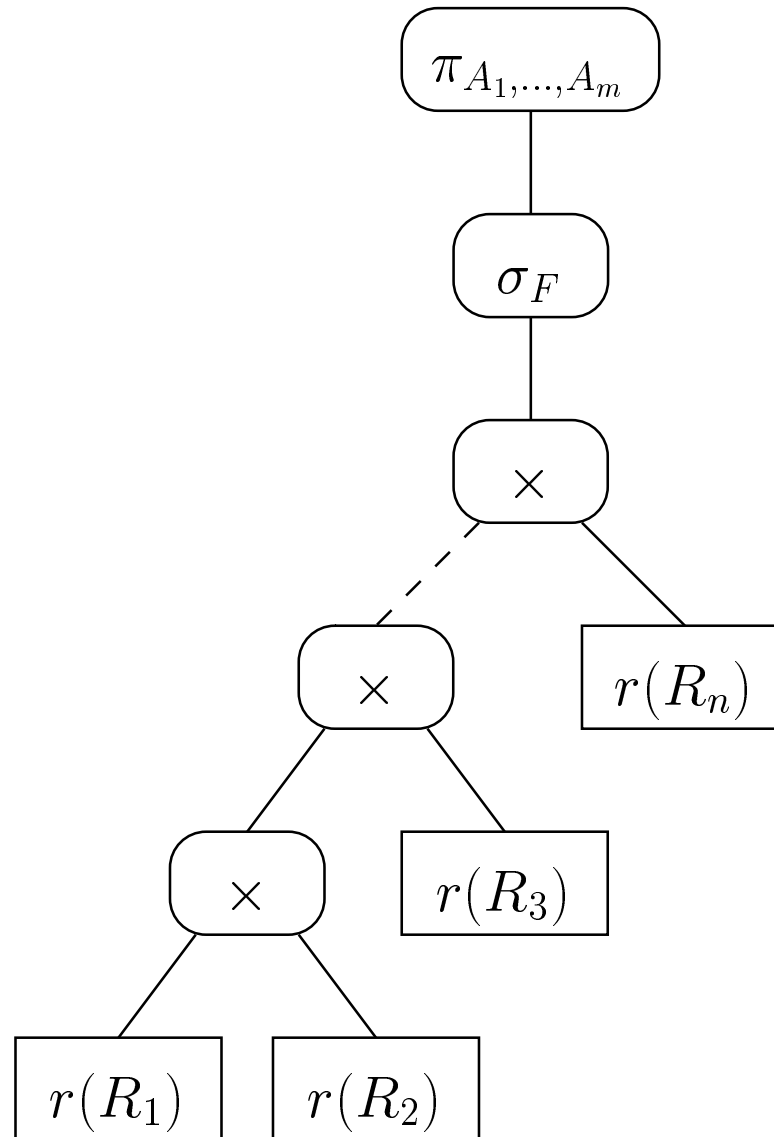
$$\pi_{A_1, \dots, A_m}(\sigma_F(r(R_1) \times r(R_2) \times r(R_3) \times \dots \times r(R_n)))$$

Anfragebaum (folgende Folie) verbessern gemäß:

- *Erkennen von Verbunden* statt Kreuzprodukten
- *Auflösung von Unteranfragen* (**not exists**-Anfragen in Differenz)
- SQL-Konstrukte, die in der Relationenalgebra kein Gegenstück haben: **group by**, **order by**, Arithmetik, Multimengensemantik

# Umsetzung des SFW-Blocks

---



# Normalisierung

---

- Vereinfachung der folgenden Optimierungsschritte durch ein einheitliches (kanonisches) Anfrageformat
- speziell für Selektions- und Verbundbedingungen
  - ◆ *konjunktive Normalform vs. disjunktive Normalform*
  - ◆ konjunktive Normalform (KNF) für einfache Prädikate  $p_{ij}$ :

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

- ◆ disjunktive Normalform (DNF):

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

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

# Normalisierung /2

---

## ■ Äquivalenzbeziehungen

◆  $p_1 \wedge p_2 \longleftrightarrow p_2 \wedge p_1$  **und**  $p_1 \vee p_2 \longleftrightarrow p_2 \vee p_1$

◆  $p_1 \wedge (p_2 \wedge p_3) \longleftrightarrow (p_1 \wedge p_2) \wedge p_3$  **und**  
 $p_1 \wedge (p_2 \vee p_3) \longleftrightarrow (p_1 \wedge p_2) \vee p_3$

◆  $p_1 \wedge (p_2 \vee p_3) \longleftrightarrow (p_1 \wedge p_2) \vee (p_1 \wedge p_3)$  **und**  
 $p_1 \vee (p_2 \wedge p_3) \longleftrightarrow (p_1 \vee p_2) \wedge (p_1 \vee p_3)$

◆  $\neg(p_1 \wedge p_2) \longleftrightarrow \neg p_1 \vee \neg p_2$  **und**  
 $\neg(p_1 \vee p_2) \longleftrightarrow \neg p_1 \wedge \neg p_2$

◆  $\neg(\neg p_1) \longleftrightarrow p_1$

# Normalisierung: Beispiel

---

- Anfrage:

```
select * from Projekt P, Zuordnung Z
where P.PNr = Z.PNr and
        Budget > 100.000 and
        (Ort = 'MD' or Ort = 'B')
```

- Selektionsbedingung in KNF:

$$P.PNr = Z.PNr \wedge \text{Budget} > 100.000 \wedge (\text{Ort} = \text{'MD'} \vee \text{Ort} = \text{'B'})$$

- Selektionsbedingung in DNF:

$$(P.PNr = Z.PNr \wedge \text{Budget} > 100.000 \wedge \text{Ort} = \text{'MD'}) \vee \\ (P.PNr = Z.PNr \wedge \text{Budget} > 100.000 \wedge \text{Ort} = \text{'B'})$$

# Logische Optimierung

---

- heuristische Methoden
  - ◆ etwa algebraische Optimierung
  - ◆ für Relationenalgebra + Gruppierung, ...
- exakte Methoden
  - ◆ Tableauoptimierung
  - ◆ 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

# Prinzipien algebraischer Optimierung

---

Beispiel:

BÜCHER = { Titel, Autor, Verlag, ISBN }

VERLAGE = { Verlagsname, VerlagsAdr }

ENTLEIHER = { EntlName, EntlAdr, EntlKarte }

AUSLEIHE = { EntlKarte, ISBN, Datum }

# Entfernen redundanter Operationen

---

- bei Anfragen mit Sichten nötig

$$r(\text{LANGEWEG}) = r(\text{BÜCHER}) \bowtie$$

$$\pi_{\text{ISBN,DATUM}}(\dots\dots\dots \sigma_{\text{DATUM} < '31.12.1995'}(r(\text{AUSLEIHE})))$$

- Anfrage an Sicht:

$$\pi_{\text{TITEL}}(r(\text{BÜCHER}) \bowtie r(\text{LANGEWEG}))$$

- Sichtexpansion:

$$\pi_{\text{TITEL}}(r(\text{BÜCHER}) \bowtie r(\text{BÜCHER}) \bowtie \pi_{\dots}(\dots))$$

- Regel: Idempotenz

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

# Verschieben von Selektionen

---

$$\sigma_{\text{AUTOR}='Heuer'}(r(\text{BÜCHER}) \bowtie \pi_{\text{ISBN,DATUM}}(\dots))$$

günstiger:

$$(\sigma_{\text{AUTOR}='Heuer'}(r(\text{BÜCHER}))) \bowtie \pi_{\text{ISBN,DATUM}}(\dots)$$

Regel:

Selektion und Verbund kommutieren

nur, wenn die Attribute der Selektionsprädikate dies zulassen

# Reihenfolge von Verbunden

---

- Kenntnis der Statistikinformationen des Katalogs nötig

$$(r(\text{VERLAGE}) \bowtie r(\text{AUSLEIHE})) \bowtie r(\text{BÜCHER})$$

- erster Verbund: kartesisches Produkt, daher:

$$r(\text{VERLAGE}) \bowtie (r(\text{AUSLEIHE}) \bowtie r(\text{BÜCHER}))$$

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:

$$r_1 \bowtie r_2 \longleftrightarrow r_2 \bowtie r_1$$

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

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

- **ProjProj:** bei Operator  $\pi$  dominiert in der Kombination der äußere Parameter den inneren:

$$\pi_X(\pi_Y(r_1)) \longleftrightarrow \pi_X(r_1)$$

# Algebraische Regeln (II)

---

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

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

(Ausnutzung der Kommutativität des logischen Und)

# Algebraische Regeln (III)

---

- **SelProj:** Operatoren  $\pi$  und  $\sigma$  kommutieren, sofern das Prädikat  $F$  auf den Projektionsattributen definiert ist:

$$\sigma_F(\pi_X(r_1)) \longleftrightarrow \pi_X(\sigma_F(r_1))$$

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

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

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

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

# Algebraische Regeln (IV)

---

- **SelJoin:** Operatoren  $\sigma$  und  $\bowtie$  kommutieren, falls Selektionsattribute alle aus einer der beiden Relationen stammen:

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

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:

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

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

# Algebraische Regeln (V)

---

- **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:

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

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

# Algebraische Regeln (VI)

---

- **SelUnion:** Kommutieren von  $\sigma$  und  $\cup$ :

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

- **SelDiff:** Kommutieren von  $\sigma$  und  $-$ :

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

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

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

# Algebraische Regeln (VII)

---

- **ProjJoin**: Kommutieren von  $\pi$  und  $\bowtie$ :

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

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 (VIII)

---

- **ProjUnion:** Kommutieren von  $\pi$  und  $\cup$ :

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

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

# Weitere Regeln

---

## ■ *Idempotenzen*

$$\begin{array}{ll} \text{IdemUnion:} & r_1 \cup r_1 \longleftrightarrow r_1 \\ \text{IdemSchnitt:} & r_1 \cap r_1 \longleftrightarrow r_1 \\ \text{IdemJoin:} & r_1 \bowtie r_1 \longleftrightarrow r_1 \\ \text{IdemDiff:} & r_1 - r_1 \longleftrightarrow \{\} \end{array}$$

## ■ Verknüpfung mit einer leeren Relation:

$$\begin{array}{ll} \text{LeerUnion:} & r_1 \cup \{\} \longleftrightarrow r_1 \\ \text{LeerSchnitt:} & r_1 \cap \{\} \longleftrightarrow \{\} \\ \text{LeerJoin:} & r_1 \bowtie \{\} \longleftrightarrow \{\} \\ \text{LeerDiffRechts:} & r_1 - \{\} \longleftrightarrow r_1 \\ \text{LeerDiffLinks:} & \{\} - r_1 \longleftrightarrow \{\} \end{array}$$

- für  $\bowtie$ ,  $\cup$  und  $\cap$  gelten *Kommutativ-* und *Assoziativgesetz* (Bezeichnung: **Komm\*** und **Assoz\***)

# Ein einfacher Optimierungsalgorithmus

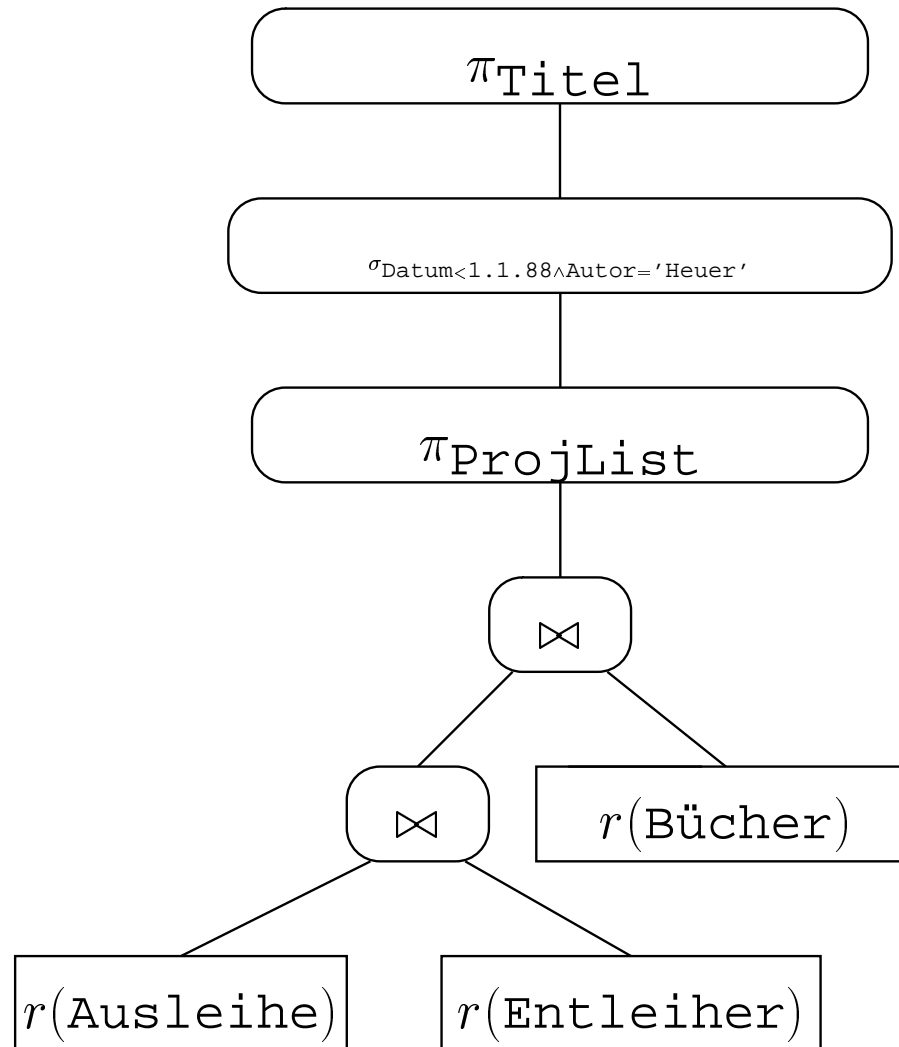
---

- komplexe Selektionsprädikate aufgelösen (Regel **SelSel**, ggf. Regeln der Auflösung für  $\neg$  und  $\vee$ )
- mittels der Regeln **SelJoin**, **SelProj**, **SelUnion** und **SelDiff** Selektionen möglichst weit in Richtung der Blätter verschieben, ggf. Selektionen gemäß Regel **SelSel** vertauschen
- 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

# Unoptimierter Anfrageplan

---

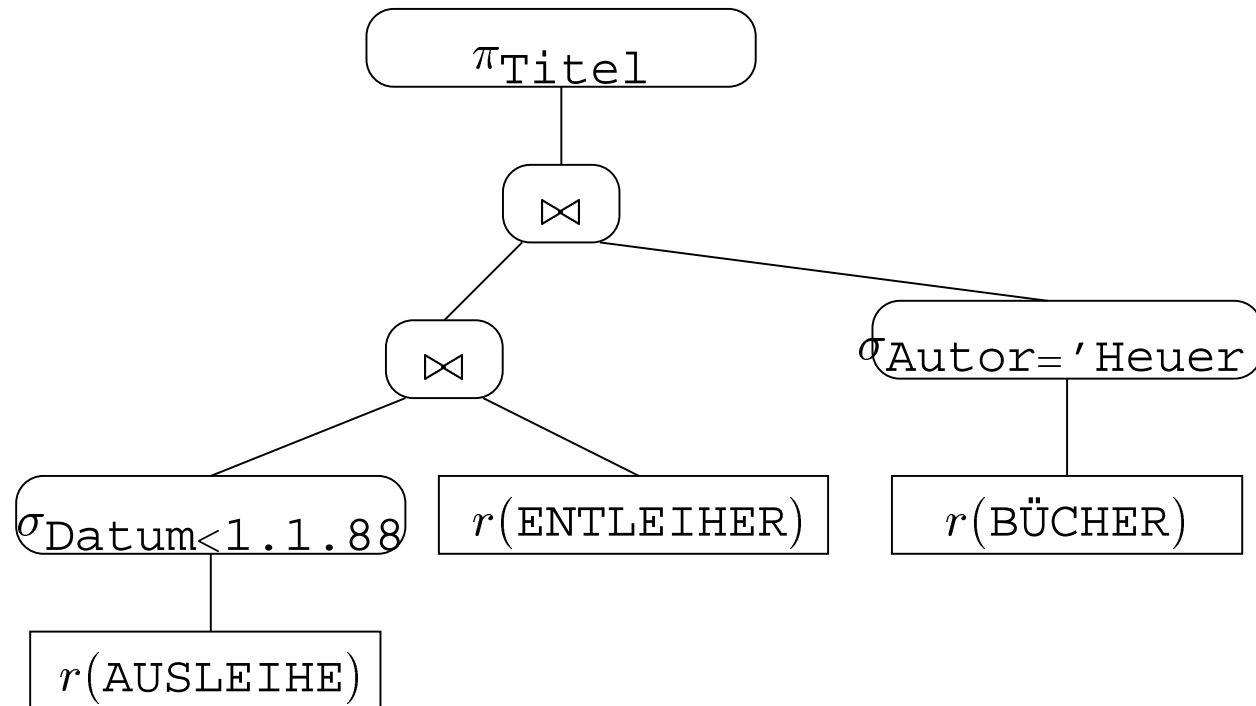


ProjList=Titel,Autor,Verlag,ISBN,EntlName,EntlAdr...

# Anfrageplan (II)

---

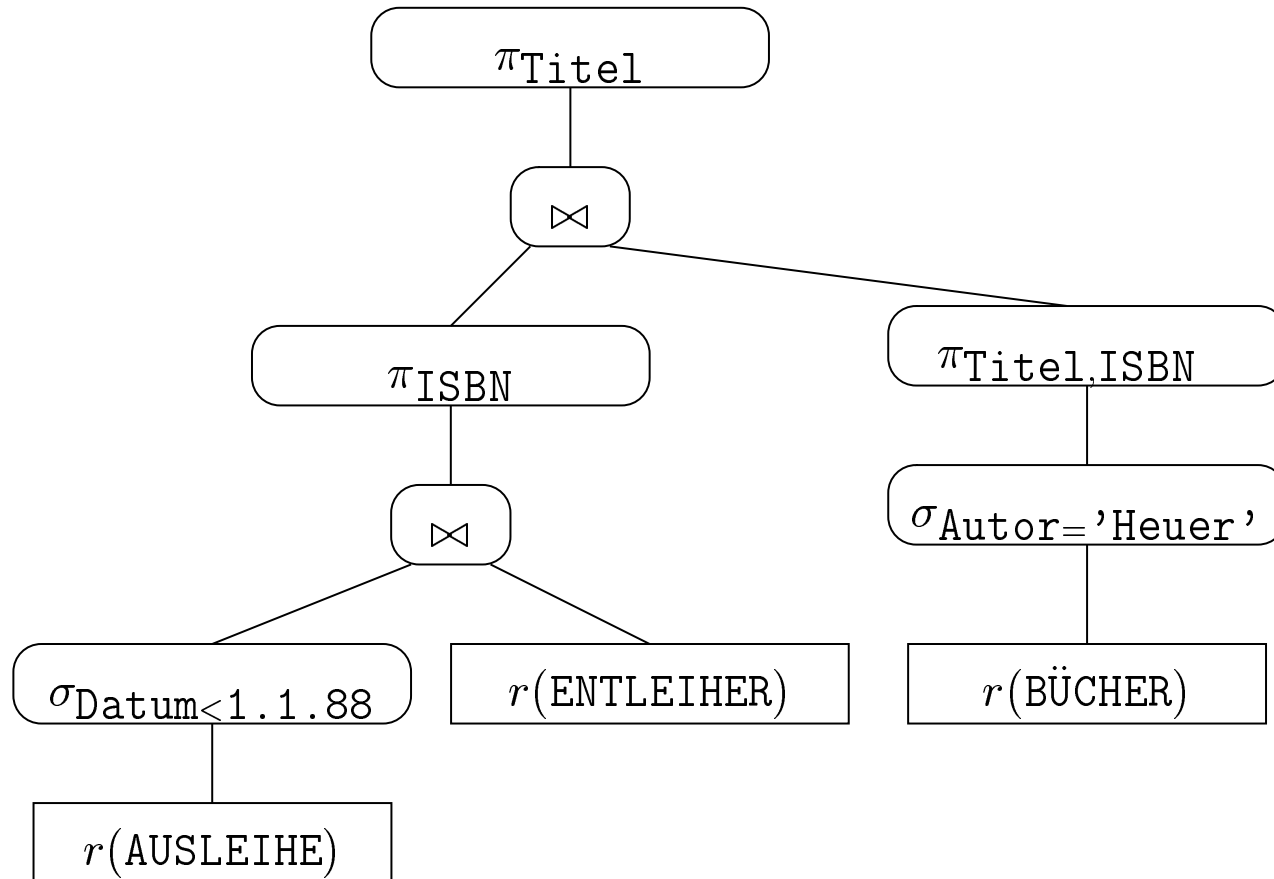
nach Verschieben der Selektionen



# Anfrageplan (III)

---

mit zusätzlichen Projektionen



# Verbundoptimierung mit Tableaus

---

- exakte Optimierung (minimale Anzahl von Verbunden)
- für eine eingeschränkte Klasse von Anfragen ( $\sigma, \pi, \bowtie$ )

sinnvoll bei

- Optimierung von Anfragen über Sichten
- Anfrageinterpretation von Universalrelationenschnittstellen

# Tableaus — Informale Einführung

---

matrixförmige Darstellung einer Relationenalgebra- oder Relationenkalkül-Anfrage

- Spalten der Matrix: Attribute des Universums
- erste Zeile des Tableaus: *summary*; *blanks* und *ausgezeichnete Variablen*  $a_i$
- weitere Zeilen: *blanks*, ausgezeichnete Variablen, Konstanten und *nichtausgezeichnete Variablen*  $b_j$ ; besitzt *tag* (Namen einer Basisrelation)

# Äquivalente Klassen von Anfragen

---

- *Tableau-Anfragen*
- *Konjunktive Anfragen* Teilmenge des Bereichskalküls

$$\{a_1 \dots a_n \mid \exists b_1 \dots \exists b_m : F_1 \wedge \dots \wedge F_k\}$$

wobei  $F_i = R(c_1 \dots c_r)$ , d.h.  $c_1 \dots c_r \in r(R)$ ;  $c_j$  ausgezeichnete oder nichtausgezeichnete Variablen oder Konstanten

- *Eingeschränkte relationenalgebraische Ausdrücke.*

$\sigma_{A=c}, \sigma_{A=B}, \sigma_F, \pi, \bowtie$  mit  $F$  bestehend aus  
 $A = c, A = B, \wedge$ .

# Beispiel für Tableau-Anfragen

$\mathcal{U} = \{A, B, C\}$  mit  $R_1 = \{A, B\}$  und  $R_2 = \{B, C\}$

- Algebra-Ausdruck:  $\pi_A(\sigma_{C=3}(r(R_1) \bowtie r(R_2)))$
- Konjunktive Anfrage:  $\{a_1 \mid \exists b_1 : R_1(a_1, b_1) \wedge R_2(b_1, 3)\}$
- Tableau-Anfrage:

					tags
		A	B	C	↓
summary	→	$w_0$	$a_1$		
rows	→	$w_1$	$a_1$	$b_1$	$R_1$
		$w_2$	$b_1$	<b>3</b>	$R_2$

# Interne Optimierung

---

- erster Schritt: Relationenalgebraoperationen in interne Operationen aus Kapitel 6
- weitere Operationen
  - ◆ Tupelmengen zum Teil durch sortierte Tupellisten ersetzen
  - ◆ statt Mengen Multimengen
  - ◆ neben Tupeln auch TID-Listen verarbeiten
  - ◆ Zugriffe auf Indexe

# Auswahl von Berechnungsalgorithmen

---

## ■ Projektion:

- ◆  $\pi_{AttList}^{REL/mit}$ : **REL** Projektion durch Relationen-Scan (**mit** bezeichnet eine Projektion mit Duplikateliminierung)
- ◆  $\pi_{AttList}^{REL/ohne}$ : Projektion durch Relationen-Scan ohne Duplikateliminierung
- ◆  $\pi_{AttList}^{SORT/mit}$ : **SORT** Projektion durch Scan über einer nach `AttList` sortierten Relation mit Duplikateliminierung
- ◆  $\pi_{AttList}^{SORT/ohne}$ : Projektion durch Scan über einer nach `AttList` sortierten Relation ohne Duplikateliminierung

# Auswahl von Berechnungsalgorithmen II

---

- Selektion:
  - ◆  $\text{REL}_{\sigma_{\varphi}^{\text{REL}}}$ : Selektion durch Relationen-Scan
  - ◆ Selektion über Index (später detailliert)
- Verbund:
  - ◆  $\bowtie^{\text{DIRECT}}$ : Verbund durch Nested-Loops **DIRECT**
  - ◆  $\bowtie^{\text{MERGE}}$ : Verbund durch Mischen **MERGE**  
(Voraussetzung: Eingaberelationen nach Verbundattribut(en) sortiert).
  - ◆  $\bowtie^{\text{HASH}}$ : Verbund durch Hash-Join **HASH**
- Operatoren für Vereinigung, Differenz und Durchschnitt analog zum Verbund; Vereinigung mit oder ohne Duplikateliminierung

# Indexzugriff

---

- Zugriff über Index

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

liefert TID-Liste

- Spezialfall  $\sigma_{\text{true}}^{\text{IND}}(\text{I}(\text{R}(A)))$ : *alle* Indexeinträge sortiert nach  $A$
- Varianten:
  - ◆ duplikatfreier Primärindex: Ergebnis bei  $A = a$  ein einzelnes Tupel
  - ◆ Tupel direkt im Index: Ergebnis des Indexzugriffs ist sortierte Liste von Tupeln
  - ◆ *Bereichsanfrage*: Prädikat  $a_1 \leq A \leq a_2$

# Indexzugriff II

---

- Index kann auch Projektion unterstützen
- Ergebnis von  $\sigma_{\text{true}}^{\text{IND}}(\text{I}(\text{R}(\text{A})))$  als Eingabe für  $\pi^{\text{SORT/-}}$  Operator nehmen
- kombinierter Zugriff:  $\pi_{\text{AttList}}^{\text{IND/mit}}$  bzw.  $\pi_{\text{AttList}}^{\text{IND/ohne}}$
- in  $\pi_A^{\text{IND/-}}(\text{I}(\text{R}(\text{A})))$  kann auf Zugriff auf Basisrelation ganz verzichtet werden

# Neue Operatoren

---

- Für TID-Listen: ‘*Realisierung*’-Operator  $\rho$ :

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

- Auf TID-Listen  $\cup$ ,  $\cap$  und  $-$
- *Sortierung* von Tupelmengen:  $\omega$

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

- Relation sortieren:

$$\rho(\sigma_{\text{true}}^{\text{IND}}(\text{I}(R(\text{AttList}))), r(R)) = \omega_{\text{AttList}}(r(R))$$

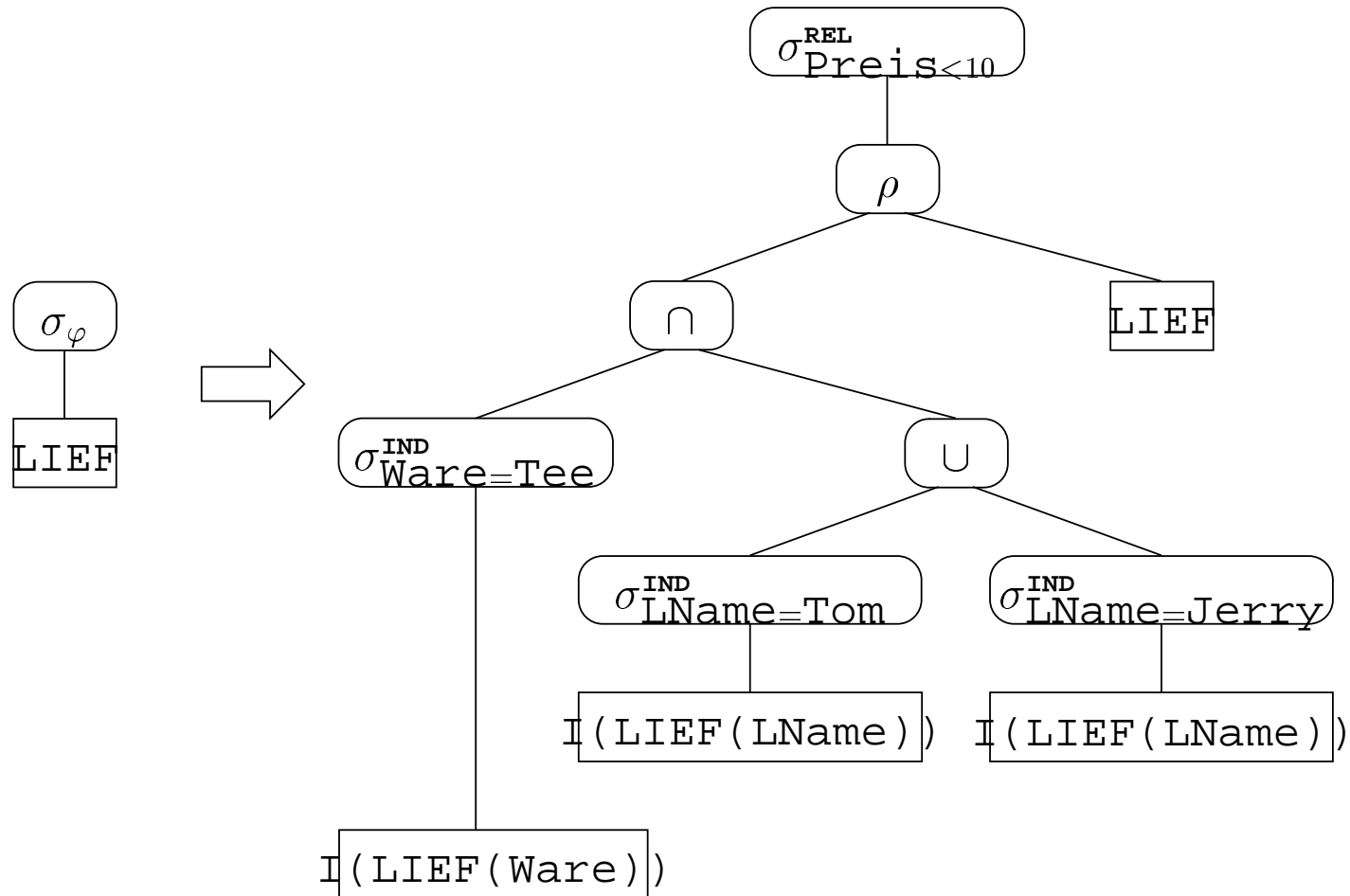
erste Variante Aufwand  $O(|R|)$ , zweite Variante  $O(|R| \times \log |R|)$

# Beispiele für interne Zugriffspläne

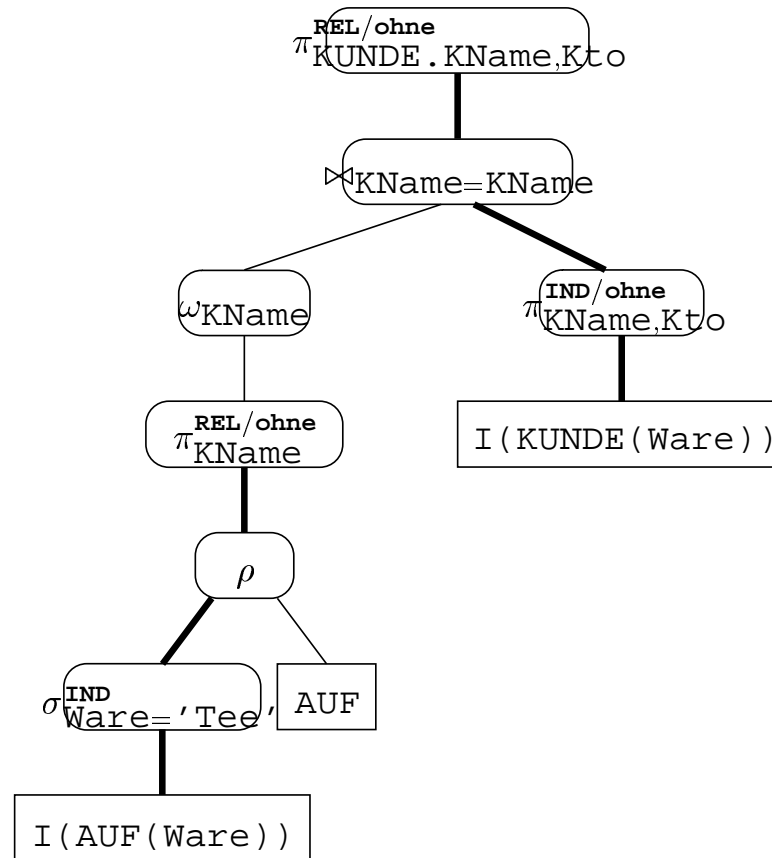
---

```
select *  
from LIEFERANT  
where Ware = 'Tee' and  
      ( LName = 'Tom' or LName = 'Jerry' )  
and Preis < 10
```

# Zwei Zugriffspläne



# Pipelining von Operationen



$$\pi_A(\sigma_\varphi(r(R))) \longleftrightarrow \langle \pi_A \circ \sigma_\varphi \rangle(r(R))$$

# Pipelining von Operationen II

---

Typische Verschmelzungen:

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

```
select K.KName, Kto  
from KUNDE K, AUFTRAG A  
where K.KName = A.KName and A.Ware = 'Tee'
```

# Gemeinsame Teilanfragen

---

- Aufgabe: Erkennung *gemeinsamer Teilanfragen*  $\rightsquigarrow$   
Gleichsetzen der zugehörigen Teilbäume des  
Operatorbaums
- Probleme:
  - ◆ unterschiedliche syntaktische Form:  $r_1 \cup r_2$   
identisch zu  $r_2 \cup r_1$
  - ◆ *Überdeckung* ( $\sigma_\varphi$  überdeckt  $n$   $\sigma_{\varphi \wedge \psi}$ )

# Kostenbasierte Auswahl

---

Berücksichtigung:

- tatsächliche Größe der Datenbankrelationen
- Existenz von Indexen (Primär-, Sekundär-) und ihre Größe
- Clustering mehrerer Relationen
- Selektivität eines Attributs, über das ein Index aufgebaut wurde

# Relevante Datenbankparameter

---

- *Systemparameter* aus Katalog:  $s$ : Länge einer Seite (nutzbarer Seitenbereich in Byte)
- Größe der Datenbank: Anzahl  $S$  der belegten Seiten (wird benötigt, wenn Tupel von Relationen gestreut gespeichert werden)
- statistische Daten über Relationen und Indexe:
  - ◆  $T_R$ : Anzahl der Tupel in Relation  $R$
  - ◆  $L_R$ : durchschn. Länge eines Tupels in  $R$
  - ◆  $W_{A,R}$ : Anzahl der verschiedenen Werte des Attributes  $A$  in  $R$  (Indexinformation oder Statistik)

Statistiken → Aktualisierung bei Änderungsoperation oder Aufruf entsprechender Kommandos

# Selektivität von Attributen

---

Anzahl der verschiedenen Werte des Attributs  $A$  in  $R$ :  $W_{A,R}$

- Gleichheit:

$$sel(A=c, R) = \frac{1}{W_{A,R}}$$

- Ungleichheit:

$$sel(\mathbf{not} A=c, R) = 1 - sel(A=c, R) = 1 - \frac{1}{W_{A,R}}$$

- Vergleich mittels  $<$ ,  $>$ , ...:

$$sel(A < c, R) = sel(A > c) = \frac{1}{2}$$

# Selektivität von Attributen II

---

Verfeinerung:

$$sel(\mathbf{A} \leq c, R) = \frac{A_{max} - c}{A_{max} - A_{min}}$$

Bereichsanfragen:

$$sel(c_u \leq \mathbf{A} \leq c_o, R) = \frac{c_o - c_u}{A_{max} - A_{min}}$$

Selektivitäten für Verbunde:

$$sel_{\bowtie}(\varphi, R, S) \approx \frac{|R \bowtie_{\varphi} S|}{|R \times S|}$$

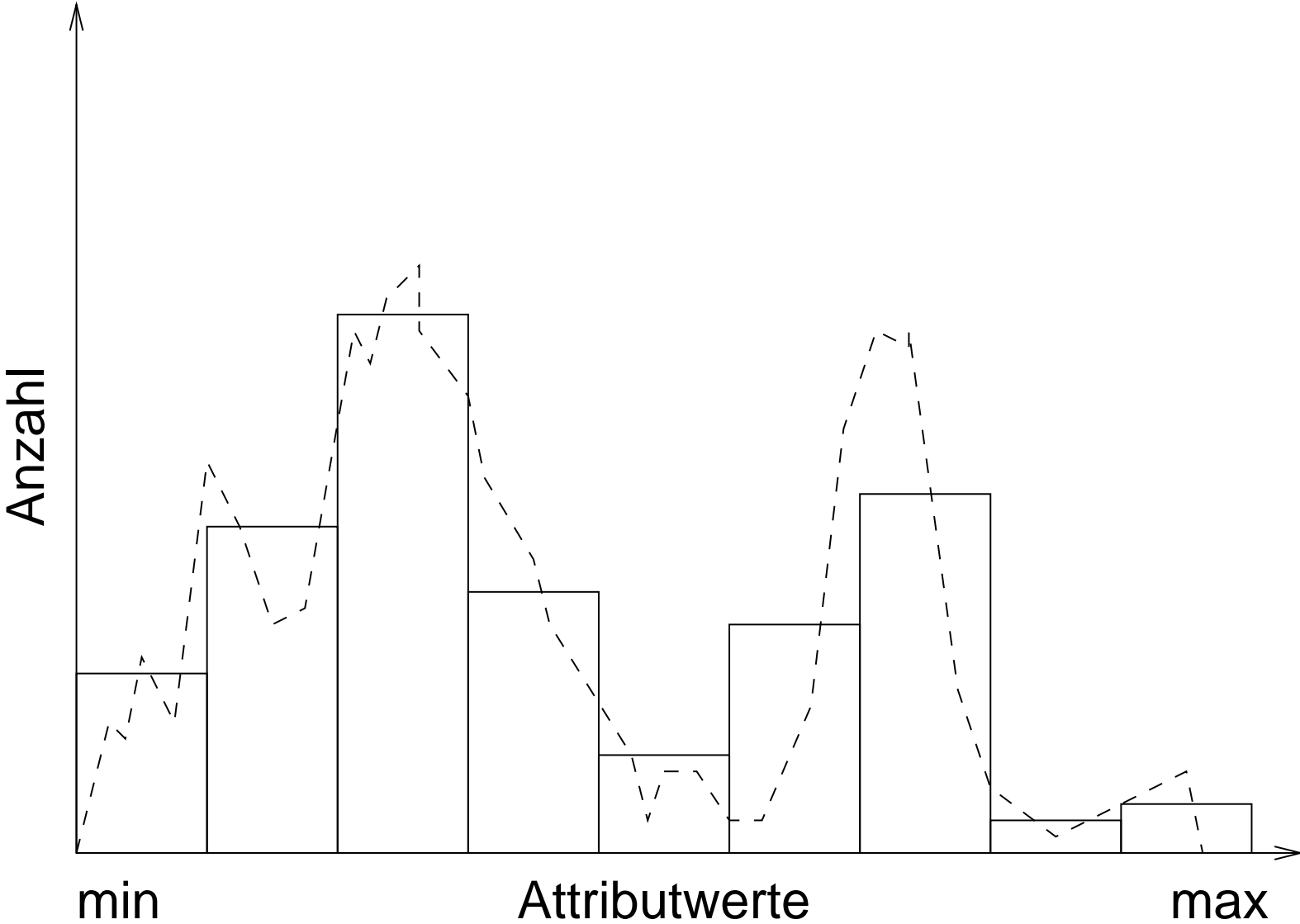
# Selektivitätsabschätzung

---

1. *Parametrisierte Funktionen*  
Parametrisierung einer Funktion, die die Datenverteilung gut widerspiegelt, möglichst genau angeben (etwa Normalverteilung)
2. *Histogramme*  
Wertebereich in Unterbereiche aufteilen und die tatsächlich in diese Unterbereiche fallenden Werte zählen
3. *Stichproben*  
Selektivität anhand einer zufälligen Stichprobe der gespeicherten Datensätze bestimmen

# Histogramme für Attributwerte

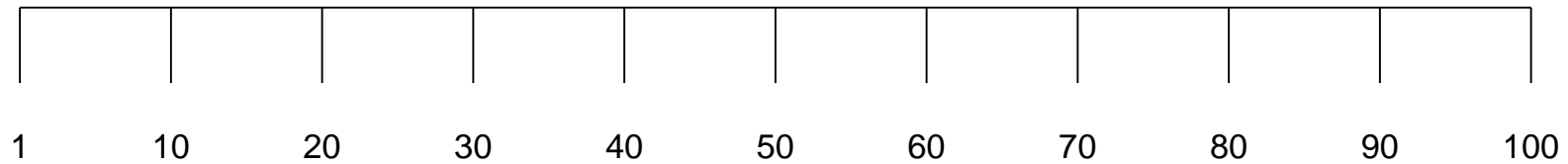
---



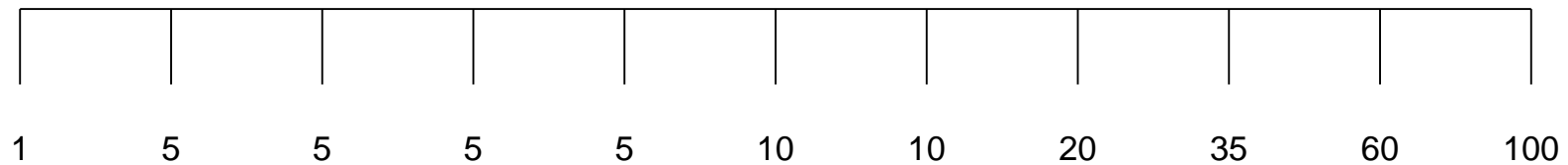
# Höhenbalancierte Histogramme

---

a) gleichverteilte Werte



b) ungleichverteilte Werte



# Kostenberechnung am Beispiel

---

Selektionen über Relation `AUFTRAG` mit Attribut `Ware`

- $S = 4.000$  (Die Datenbank belegt insgesamt 4000 Seiten)
- $T_{\text{AUFTRAG}} = 10.000$  (in der Relation `AUFTRAG` sind insgesamt 10.000 Tupel gespeichert)
- $s/L_{\text{AUFTRAG}} = 10$  (auf eine Seite passen durchschnittlich 10 Auftrags-Tupel)
- $W_{\text{Ware,AUFTRAG}} = 50$ . (50 verschiedene Waren als Wert des `Ware`-Attributs)

Selektion  $\sigma_{\varphi}$  mit  $\varphi = (A\theta a \wedge \psi)$

# Kostenberechnung: Variante A

---

- Index  $I(R(A))$  über Selektionsattribut  $A$

$$\langle \sigma_{\psi}^{\mathbf{REL}} \circ \rho \circ \sigma_{A\theta a}^{\mathbf{IND}} \rangle (I(R(A)), r(R))$$

A1 Index mit Cluster-Bildung:  $S_R \times sel(A\theta a, R)$ .

A2 Index ohne Cluster-Bildung:  $T_R \times sel(A\theta a, R)$ .  
(Maximalwert, wenn die Tupel jeweils auf verschiedenen Seiten liegen)

# Kostenberechnung: Variante B

---

- Index  $I(R(B))$  mit  $B \neq A$

$$\langle \sigma_{A\theta a \wedge \psi}^{\mathbf{REL}} \circ \rho \circ \sigma_{\mathbf{true}}^{\mathbf{IND}} \rangle (I(R(B)), r(R))$$

B1 Index mit Cluster-Bildung:  $S_R$ .

B2 Index ohne Cluster-Bildung:  $T_R$ .

Kosten ergeben sich, da die Selektivität des Prädikates **true** 1 ist

# Kostenberechnung: Variante C

---

- Relationen-Scan
- Annahme: alle Seiten der Datenbank werden gelesen und alle auf den Seiten vorhandenen Tupel werden gefunden
- Kosten: durch die Anzahl  $S$  gegeben

# Kostenberechnung: Anfragen

---

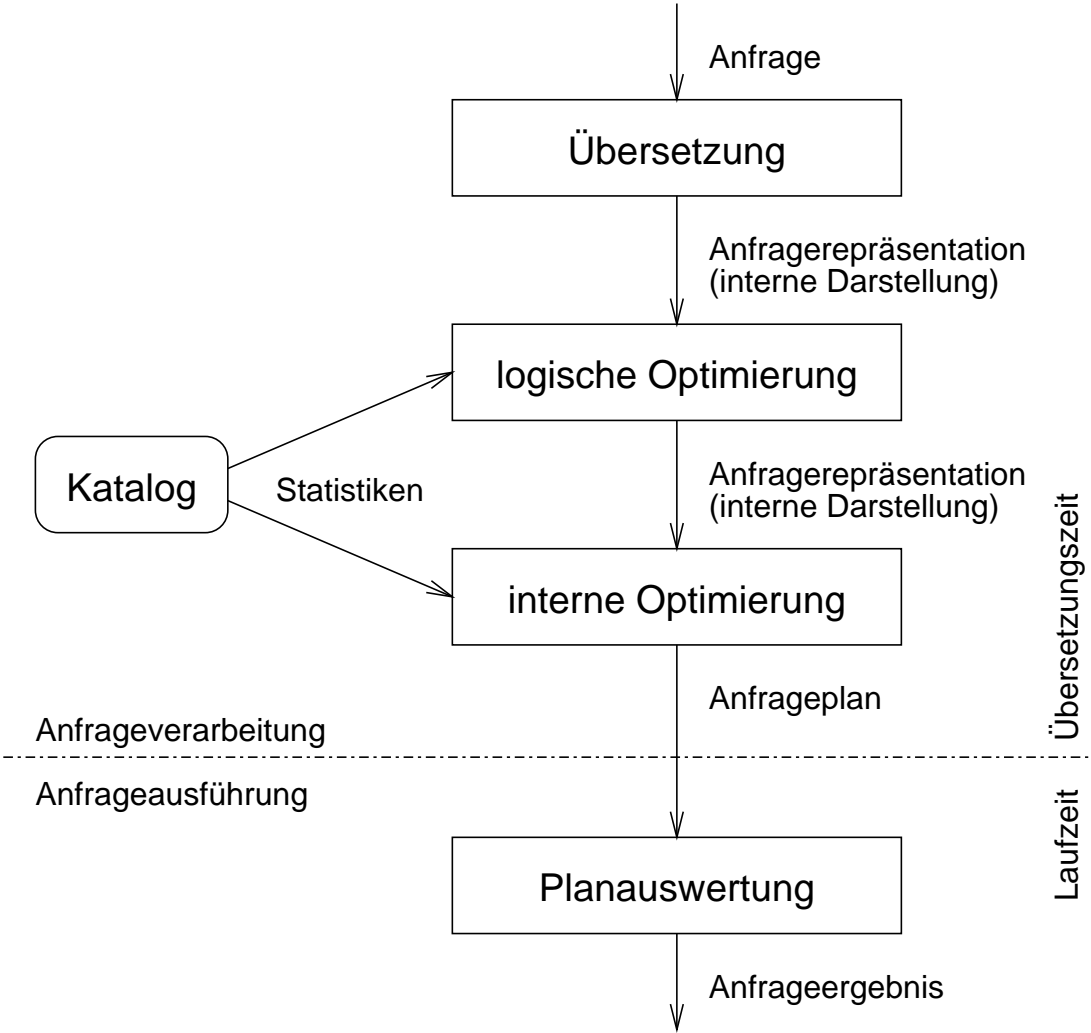
1.  $\sigma_{\text{Ware}='Tee'}(r(\text{AUFTRAG}))$ .  
Selektivität  $sel: \frac{1}{50}$
2.  $\sigma_{\text{Ware}>'Tee'}(r(\text{AUFTRAG}))$ .  
Selektivität  $sel: \text{Annahme} \rightsquigarrow \frac{1}{2}$ .

# Kosten für Ausführungsvarianten

---

Variante	Kostenformel	Ware = 'Tee'	Ware > 'Tee'
A1	$S_R \times sel(A\theta a, R)$	20	500
A2	$T_R \times sel(A\theta a, R)$	200	5.000
B1	$S_R$	1.000	1.000
B2	$T_R$	10.000	10.000
C	$S$	4.000	4.000

# Optimierer-Architektur



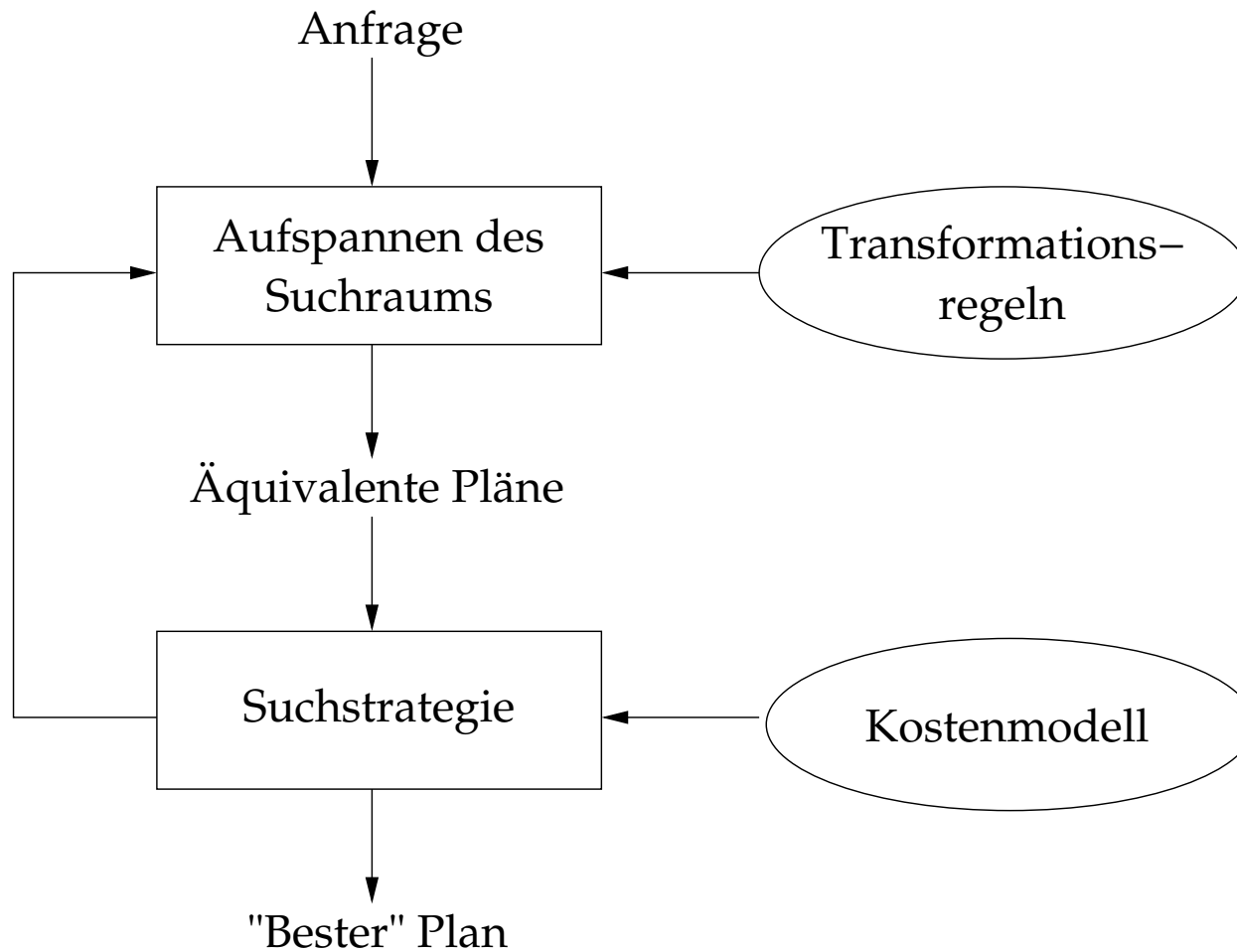
# Optimierer-Varianten

---

- *heuristische, regelbasierte Optimierer:*
  1. Erzeugung einer interner Anfragerepräsentation durch logische Optimierung
  2. mit interne Optimierung einen Anfrageplan erzeugen
  
- *kostenbasierte (Zwei-Phasen-) Optimierer:*
  1. Erzeugung verschiedener Anfragerepräsentation durch logische Optimierung
  2. Übergabe an interne Optimierung (Plangenerierung)
  3. Kostenbewertung
  4. Auswahl des besten Plans

# Optimierung: Überblick

---



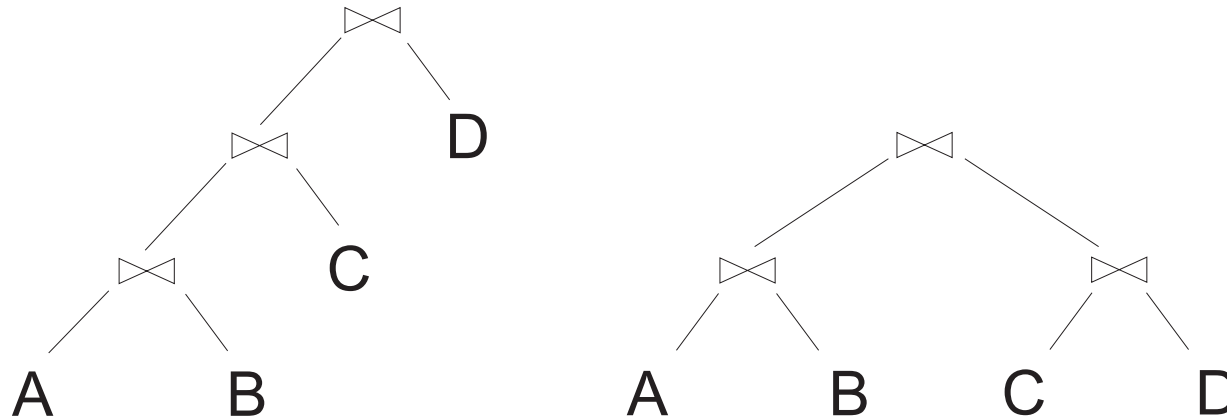
# Optimierung: Suchraum

---

- Suchraum: Menge aller äquivalenten Anfragepläne
- Aufspannen durch *Transformationsregeln* (algebraische Regeln)
- Schwerpunkt: *Join-Trees*
- für  $n$  Relationen:  $n!$  verschiedene Join-Trees!
- daher: Beschränkung des Suchraums durch
  - ◆ Heuristiken (algebraische Optimierungen)
  - ◆ vorgegebene „Form“ des Baumes

# Optimierung: Join-Trees

---



- lineare Folgen von Operatorbäumen
  - ◆ nur  $2^n$  Varianten
  - ◆ *left deep tree* und *right deep tree*  $\rightsquigarrow$  alle inneren Knoten des Baums besitzen mindestens einen Blattknoten (Basisrelation) als Kind
- *bushy trees*
  - ◆ höheres Parallelisierungspotential, jedoch großer Optimierungsaufwand

# Optimierung: 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

# Optimierung: Suchstrategien /2

---

- deterministisch:
  - ◆ systematisches Generieren von Plänen
  - ◆ beginnend bei Plänen für Zugriff auf Basisrelationen
  - ◆ Konstruktion komplexer Pläne durch Verbund einfacherer Pläne
  - ◆ erschöpfende Suche; garantiert besten Plan
  - ◆ Beispiel: *Dynamic Programming* (Breitensuche)
  - ◆ State of the art

# Optimierung: Suchstrategien /3

---

- zufallsbasiert:
  - ◆ ein oder mehrere Startpläne durch Greedy-Strategie (Tiefensuche)
  - ◆ Verbesserung der Startpläne durch Untersuchung von „Nachbarn“
  - ◆ Nachbar: Anwendung von Transformationsregeln, z.B. Vertauschen zweier zufällig gewählter Operationen
  - ◆ Beispiel: *Simulated Annealing*
  - ◆ bessere Performance bei großer Anzahl von Relationen
  - ◆ keine Garantie für besten Plan

# Optimierung in INGRES

---

- dynamisches Verfahren zur Optimierung
- rekursives Zerlegen einer Kalkülanfrage in Sequenzen von *Mono-Relationen-Anfragen* (Ein-Tupelvariablen-Anfragen)
- Verarbeitung der Mono-Relationen-Anfragen durch „One Variable Query Processor“ (OVQP)
- OVQP wählt beste Zugriffsmethode (Indexauswahl)

# Optimierung in INGRES /2

---

## ■ Prinzip:

### ◆ Anfrage $q$ :

```
select  $R_2.A_2, R_3.A_3, \dots, R_n.A_n$   
from  $R_1, R_2, \dots, R_n$   
where  $P_1(R_1.A'_1)$  and  $P_2(R_1.A_1, R_2.A_2, \dots, R_n.A_n)$ 
```

### ◆ Zerlegung in $q'$ :

```
select  $R_1.A_1$  into  $R'_1$   
from  $R_1$   
where  $P_1(R_1.A'_1)$ 
```

### ◆ und $q''$ :

```
select  $R_2.A_2, R_3.A_3, \dots, R_n.A_n$   
from  $R'_1, R_2, \dots, R_n$   
where  $P_2(R_1.A_1, R_2.A_2, \dots, R_n.A_n)$ 
```

# Optimierung in INGRES /3

---

- Behandlung von nicht reduzierbaren Multi-Relationen-Anfragen (insb. Verbunde)
  - ◆ Konvertierung in Mono-Relationen-Anfragen durch *Tupelsubstitution*
  - ◆ für Anfrage  $q$ : Auswahl einer Relation  $R_1$  und Ableitung von  $\text{card}(R_1)$  Anfragen  $q'$  mit  $n - 1$  Relationen

$q(R_1, R_2, \dots, R_n)$  ersetzen durch

$$\{q'(t_{1i}, R_2, R_3, \dots, R_n); t_{1i} \in R_1\}$$

# Optimierung in System R

---

- Dynamic Programming
- Bottom-Up-Konstruktion eines Plans
  1. Generierung einfacher Pläne (Zugriff auf eine Relation)
  2. Generierung komplexerer Pläne (2 Relationen, 3 ...) durch Kombination (Verbund) einfacher Pläne
- dabei: *Pruning*
  - ◆ Begrenzung des Lösungsraums durch Löschen von „schlechten“ Plänen für die äquivalente Pläne existieren
    1. Permutationen mit kartesischen Produkten
    2. kommutative Strategien mit höchsten Kosten

# Optimierung in System R /2

---

**Input:** SPJ-Anfrage  $q$  auf den Relationen  $r_1, \dots, r_n$

**Output:** Anfrageplan für  $q$

**for**  $i := 1$  **to**  $n$  **do**

$optPlan(\{r_i\}) := accessPlans(r_i)$

$prunePlans(optPlan(\{r_i\}))$

**end**

**for**  $i := 1$  **to**  $n$  **do**

**forall**  $s \subseteq \{r_1, \dots, r_n\}$  **such that**  $|s| = i$  **do**

$optPlan(s) := \emptyset$

**forall**  $t \subset s$  **do**

$optPlan(s) := optPlan(s) \cup$

$joinPlans(optPlan(t), optPlan(s - t))$

$prunePlans(optPlan(s))$

**end end end**

**return**  $optPlan(\{r_1, \dots, r_n\})$

# Optimierung in System R /3

---

- Beispielanfrage:

```
select M.MName  
from Mitarbeiter M, Zuordnung Z, Projekt P  
where M.MNr = Z.MNo and Z.PNr = P.PNr  
and P.PName = 'DB-Entwicklung'
```

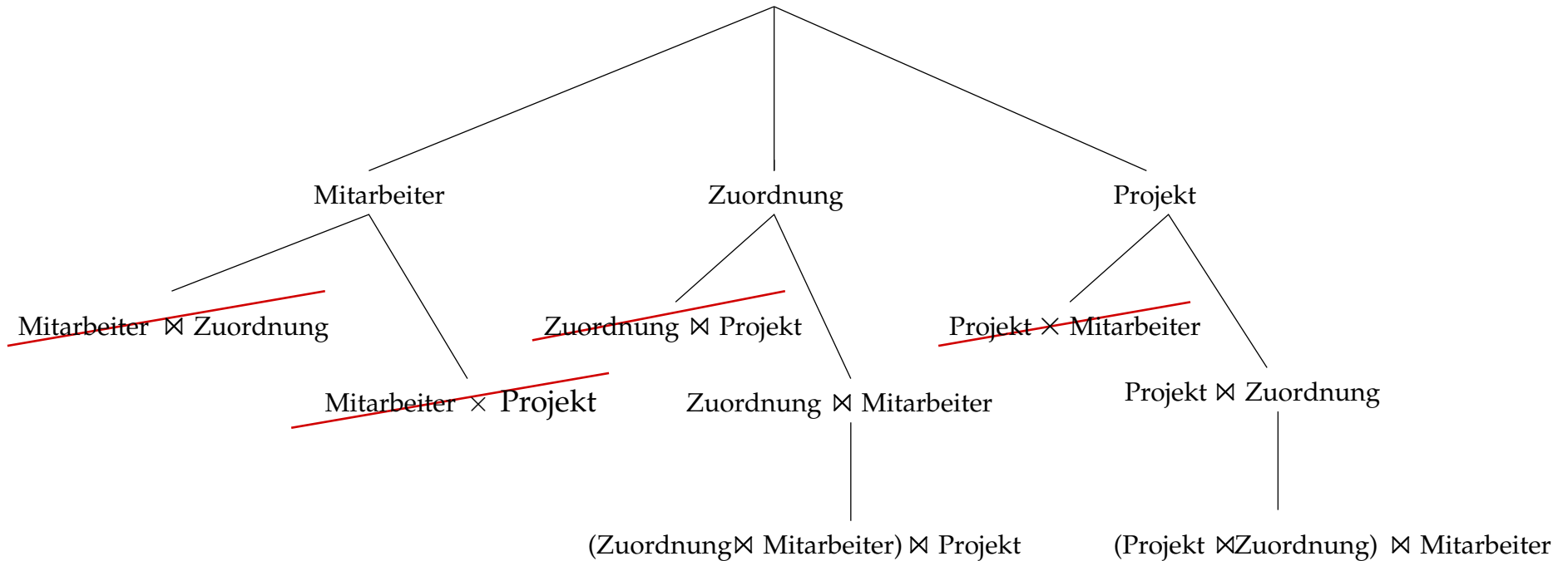
- **Indexe:** Mitarbeiter (MNr), Zuordnung (PNr),  
Projekt (PNr), Projekt (PName)

- **Zugriffspläne nach 1. Iteration:**

```
Mitarbeiter: Full-Table-Scan  
Zuordnung:   Full-Table-Scan  
Projekt:     Index-Scan auf PName
```

# Optimierung in System R /4

## ■ mögliche Verbundreihenfolgen



# Oracle9i: Statistiken

---

- Tabellen: Anzahl Tupel und Blöcke, durchschnittliche Tupellänge
- Spalten: Anzahl der verschiedenen Werte, Anzahl der **NULL**-Werte, Datenverteilung (Histogramme)
- Indexe: Anzahl der Blattseiten, Höhe des Baums, Clustering-Faktor
- System: I/O- bzw. CPU-Performance und -Auslastung

# Gewinnung von Statistiken

---

- Gewinnung durch
  - ◆ Abschätzung auf Basis von Stichproben (*row sampling, block sampling*)
  - ◆ exakte Berechnung (Aufwand: Table-Scan + Sortierung)
  - ◆ benutzerdefiniert
- Werkzeuge
  - ◆ **ANALYZE TABLE ...**
  - ◆ Package `DBMS_STATS`: Prozeduren für erweiterte Behandlung von Statistiken (von Oracle empfohlen)

# Pflege von Statistiken

---

- Aktualisierung der Statistiken von Hand (evtl. als Job)
- automatische Aktualisierung (Monitoring)
  - ◆ Beobachtung der Anzahl der Änderungsoperationen
  - ◆  $\geq 10\%$  betroffen  $\rightsquigarrow$  veraltete Daten  $\rightsquigarrow$  Aktualisierung

# Fehlende Statistiken

---

- Default-Werte für fehlende Statistiken
  - ◆ Tabellen
    - durchschnittl. Tupelgröße: 100 Bytes
    - Anzahl Blöcke: 1
    - Kardinalität:  $num\_of\_blocks * (block\_size - cache\_layer) / avg\_row\_len$
  - ◆ Indexe
    - Höhe: 1
    - Anzahl Blattseiten: 25
    - Anz. unterschiedl. Schlüsselwerte: 100

# Erzeugen von Statistiken: ANALYZE

---

- Aufruf:

```
analyze table tabelle statistik-art
```

- Formen

- ◆ **estimate statistics**: Abschätzung durch Stichprobe; optionaler Parameter spezifiziert Stichprobengröße (**sample groesse**  
**rows | percent**)

- ◆ **compute statistics**: exakte Bestimmung

- Beispiel:

```
analyze table emp  
  estimate statistics sample 10 percent;
```

# Erzeugen von Statistiken: DBMS\_STATS

---

- Aufruf der Package-Prozeduren:
  - ◆ u.a. `gather_index_stats`,  
`gather_table_stats`, `gather_schema_stats`,  
...
- Beispiel:

```
execute dbms_stats.gather_table_stats(  
    ownname => 'scott',  
    tabname => 'emp',  
    estimate_percent => NULL,  
    method_opt => NULL);
```

# DBMS\_STATS (II)

---

## ■ Parameter:

- ◆ `estimate_percent`: Abschätzung auf Basis der gegebenen Stichprobengröße (in Prozent)
- ◆ `method_opt`: Spezifikation der zu erzeugenden Statistiken, u.a.
  - `FOR ALL [INDEXED ] COLUMNS`: alle Spalten
  - `FOR COLUMNS Spaltenliste`: für gegebene Spalten
  - `AUTO`: automatische Auswahl der Spalten für Histogramme

# Anzeige von Statistiken

---

- Statistikdaten im Data Dictionary
- Views: dba\_-, user\_-, all\_tables, -tab\_col\_statistics
- Beispiel: Tabellenstatistik (user\_tables)

TABLE_NAME	NUM_ROWS	BLOCKS	AVG_ROW_LEN
EMP	14	1	37

# Anzeige von Statistiken (II)

---

- Beispiel: Spaltenstatistik  
(`user_tab_col_statistics`)

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS
EMPNO	14	0	13
ENAME	14	0	13
JOB	5	0	4
MGR	6	1	5
HIREDATE	13	0	12
SAL	12	0	11
COMM	4	10	3
DEPTNO	3	0	2

# Oracle: Histogramme

---

- Arten
  - ◆ höhenbasiert (equi-height)
  - ◆ wertbasiert (frequency)
    - jeder Wert der Spalte hat korrespondierendes Bucket
    - Bucketnummer entspricht Häufigkeit des Wertes
    - Anwendung, wenn Anzahl der verschiedenen Werte der Spalte  $\leq$  Anzahl der Buckets
- Auswahl in Abhängigkeit von Häufigkeit der Werte

# Histogramme: Erzeugung

---

- **ANALYZE TABLE**

```
analyze table emp compute statistics  
for column sal size 10;
```

- **DBMS\_STATS**

```
execute dbms_stats.gather_table_stats(  
    ownname => 'scott',  
    tabname => 'emp',  
    method_opt => 'for columns size 10 sal');
```

- **Default-Anzahl der Buckets: 75**

# Ausgabe von Histogrammen

---

- Data-Dictionary-Views: dba\_-, user\_-, all\_histograms

- Anfrage:

```
select endpoint_number, endpoint_value
from user_histograms
where table_name='EMP' and column_name='SAL' ;
```

```
ENDPOINT_NUMBER  ENDPOINT_VALUE
-----
                1                800
                2                950
                3               1100
                ...
               13               3000
               14               5000
```

# Oracle: Ausgabe von Plänen

---

- Speichern eines Ausführungsplans (inkl. Kosten) zu einer Anfrage in einer Tabelle `plan_table`

```
explain plan set statement_id = 'MPLAN'  
for select title from movie, budget  
where id between 2000 and 40000  
       and movie.id = budget.movie  
       and budget < 100000 and year = 1998;
```

# Oracle: Ausgabe von Plänen (II)

---

- Auslesen des Plans durch Anfrage oder spezielle Tools

```
select substr(lpad(' ', 2*(level-1)), 1, 8) ||  
       substr(operation, 1, 16) "OPERATION",  
       substr(options, 1, 12) "OPTIONS",  
       substr(object_name, 1, 12) object_name,  
       id, parent_id, cost, cardinality, bytes  
from plan_table  
start with id=0 and statement_id = 'MPLAN'  
connect by prior id = parent_id and  
              statement_id = 'MPLAN';
```

# Oracle: Ausgabe von Plänen (III)

---

## ■ Ausgabe

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID
-----	-----	-----	----	----- . . .
SELECT STATEMENT			0	
NESTED LOOPS			1	0
TABLE ACCESS	BY INDEX ROW	MY_MOVIE	2	1
INDEX	RANGE SCAN	MOVIE_PK	3	2
TABLE ACCESS	BY INDEX ROW	BUDGET	4	1
INDEX	UNIQUE SCAN	SYS_C006658	5	4

# Oracle: Spalten der Plan-Tabelle

---

Spalte	Bedeutung
statement_id	ID aus <code>explain plan</code>
operations	Planoperator (1. Zeile: Statement)
options	Details zum Planoperator
object_name	Tabelle, Index etc.
id, parent_id	IDs der Operationen
position	(1. Zeile: Gesamtkosten, sonst: relative Pos.)
cost	Kosten der Operation (Funktion über CPU/IO-Kosten)
cardinality	Anzahl der verarbeiteten Tupel
bytes	Größe der verarbeiteten Bytes
cpu_cost, io_cost	CPU/IO-Kosten (proportional zu tats. Werten)
temp_space	benötigter temp. Speicher in Bytes

# Oracle: Operationen in der Plan-Tabelle

---

operation	option	Bedeutung
filter		Selektion bzgl. Bedingung
hash join		Hash-Join
merge join		Merge-Join
merge join	(outer, cartesian)	
nested loops		Nested-Loops-Join
index	unique scan	Zugriff auf einzelne Rowid
index	range scan	Bereichszugriff auf Index
sort	group by	Sortierung für Gruppierung
sort	unique	... für Duplikateliminierung
sort	join	... für Merge-Join
table access	full	table scan
table access	by index rowid	Tabellenzugriff über Rowid von Indexzugriff

# Oracle: Optimizer Hints

---

- gezielte Beeinflussung der Optimierung von Anfragen
- Aspekte:
  - ◆ Ziel: Durchsatz / Antwortzeit
  - ◆ Anfragetransformation (z.B. Star-Queries, materialisierte Sichten etc.)
  - ◆ Zugriffspfade
  - ◆ Verbundreihenfolge
  - ◆ Verbundoperation
- Angabe: durch Kommentare
  - /\* +hint \*/*
  - +hint*

# Hints: Durchsatz vs. Antwortzeit

---

- Optimierungsziel: größter Durchsatz (geringste Ressourcennutzung)

```
select /*+ all rows */ *  
from emp;
```

- Optimierungsziel: beste Antwortzeit, d.h.  $n$  erste Tupel möglichst schnell liefern  
(nicht für **group by**, **order by**, Mengenoperationen und **distinct**-Anfragen)

```
select /*+ first rows(10) */ *  
from emp;
```

# Hints: Zugriffspfade

---

- Auswahl verschiedener Strategien: `full`, `index`, ...
- Beispiel: Nutzung eines Index für `ename`

```
select /*+ index(emp ename_idx) */ *  
from emp  
where ename = 'JONES' ;
```

# Hints: Verbunde

---

- Angabe der Join-Implementierung
  - ◆ `/* +use_nl(inner_tbl) */`: Nested-Loops-Join
  - ◆ `/* +use_merge(tbl1 tbl2) */`: Merge-Join
  - ◆ `/* +use_hash(tbl1 tbl2) */`: Hash-Join
- Angabe der Verbundreihenfolge
  - ◆ `/* +ordered */`: Reihenfolge wie in **from**-Klausel
- Beispiel:  

```
select /*+ ordered */ *  
from tab1, tab2, tab3  
where tab1.col = tab2.col  
and tab1.col = tab3.col;
```