

# 8. Anfrageoptimierung

- Vorgehensweise
- Übersetzung vs. Interpretation von DB-Operationen
- Anfragedarstellung
- Anfragetransformation
- Erstellung und Auswahl von Zugriffsplänen
- Kostenbewertung

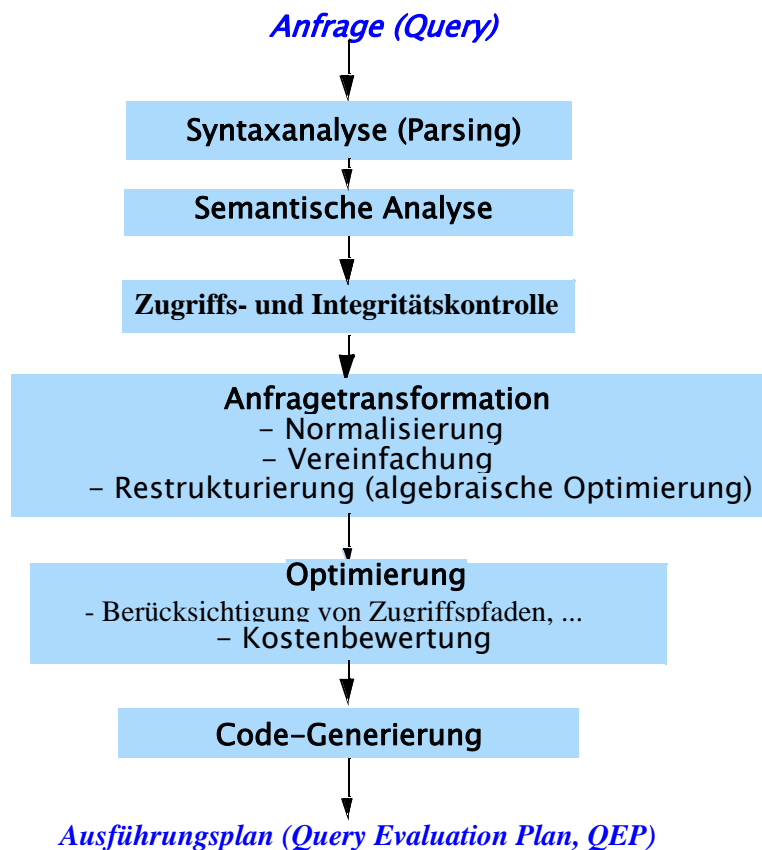


## Anfrageoptimierung

- zentrales Problem
  - Umsetzung deskriptiver Anfragen in eine zeitoptimale Folge interner DBVS-Operationen
  - Anfrageübersetzer/-optimierer des DBVS ist im wesentlichen für eine effiziente Abarbeitung verantwortlich, nicht der Programmierer
- hohe Komplexität wegen großer Auswahlmächtigkeit von Sprachen wie SQL
  - mengenorientierte Operationen auf 1 oder mehreren Tabellen, inkl. Joins
  - Prädikate wie EXISTS, NULL, LIKE u. a.
  - geschachtelte Anfragen beliebiger Tiefe (unabhängig oder korreliert)
  - Built-in- und Sortier-Funktionen auf Partitionen der Satzmenge
  - auch Änderungsoperationen sind mengenorientiert
  - referentielle Integrität ist aktiv mit Hilfe referentieller Aktionen zu wahren
  - Operationen können sich auf Sichten von Relationen beziehen
  - vielfältige Optionen der Datenkontrolle sind zu berücksichtigen
- Formulierung von 'nicht angemessenen' Anfragen wird durch deskriptive Sprachen sehr einfach
- oft extreme Kostenunterschiede zwischen funktional äquivalenten Zugriffsplänen
  - mit / ohne Indexnutzung
  - unterschiedliche Verfahren für Join, Sortierung, ...
  - unterschiedliche Reihenfolge (z.B. Selektion vor Join)

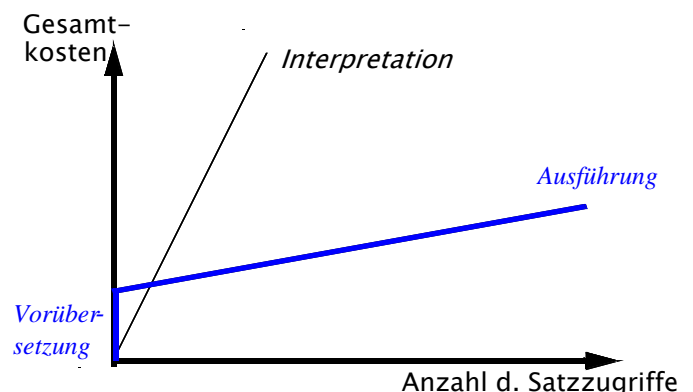


# Übersetzung von DB-Anweisungen



## Übersetzung vs. Interpretation

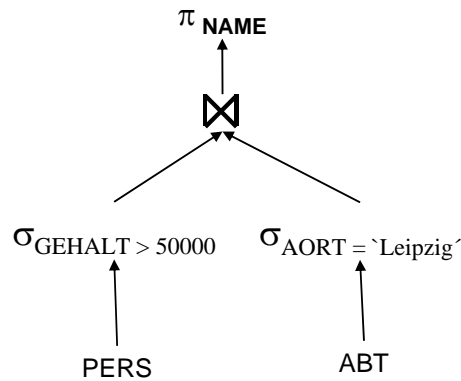
- Anfrageanalyse und -optimierung können zur Übersetzungszeit des AP oder zur Laufzeit (Interpretation) erfolgen
- Vorübersetzung:
  - erweiterter Compiler bzw. Präcompiler führt Abbildungen aus (statische Namensbindung)
  - aufwändige Optimierung möglich (Berücksichtigung mehrerer Ausführungsalternativen) mit zugeschnittenem Programm pro DB-Operation
  - effiziente Ausführung
  - Änderungen des DB-Zustandes nach der Übersetzung werden nicht berücksichtigt (neue Zugriffspfade, geänderte Statistiken etc.)
    - => Invalidierung des Zugriffsmoduls und Neuübersetzung
- Interpretation:
  - Interpreter erzeugt zur Laufzeit Einzelschritte zur Query-Ausführung
  - Berücksichtigung des aktuellen DB-Zustands bei Auswertungsstrategie
  - sehr hohe Ausführungskosten v.a. bei mehrfacher Ausführung derselben DB-Operationen (Programmschleifen) sowie durch häufige Katalogzugriffe
  - am ehesten noch akzeptabel für Ad-Hoc-Anfragen bzw. dynamische SQL-Anweisungen (EXECUTE IMMEDIATE)



# Anfragedarstellung

## ■ Darstellung der Auswertungsstrategie durch Operatorgraph

- Blätter: Eingaberelationen
- Knoten stellen Operatoren (z. B. der Relationenalgebra) dar
- Kanten beschreiben operator-kontrollierten Datenfluss
- Verfeinerung um innere Operatoren möglich



# Anfragetransformation

## ■ Ziele der Anfragetransformation (algebraische Optimierung)

- standardisierte Ausgangsdarstellung
- Elimination der Redundanz
- Verbesserung der Auswertbarkeit

## ■ Standardisierung

- Wahl einer Normalform, z.B. konjunktive Normalform  
( $A_{11}$  OR ... OR  $A_{1n}$ ) AND ... AND ( $A_{m1}$  OR ... OR  $A_{mn}$ )

## ■ Elimination der Redundanz / Vereinfachung

- Behandlung/Eliminierung gemeinsamer Teilausdrücke

$$(A_1 = a_{11} \text{ OR } A_1 = a_{12}) \text{ AND } (A_1 = a_{12} \text{ OR } A_1 = a_{11})$$

- Ausdrücke, die an "leere Relationen" gebunden sind, können vereinfacht werden
- Konstanten-Propagierung:  $A \text{ op } B \text{ AND } B = \text{const.}$
- nicht-erfüllbare Ausdrücke, z.B.:  $A \geq B \text{ AND } B > C \text{ AND } C \geq A$



# Anfragetransformation (2)

## ■ Verbesserung der Auswertbarkeit durch Query-Restrukturierung (query rewrite)

- Nutzung von Äquivalenzbeziehungen für relationale Operatoren

$$\sigma_{P_1} (\sigma_{P_2} (R)) \Leftrightarrow \sigma_{P_1 \wedge P_2} (R)$$

$$\pi_A (\pi_{A, B} (R)) \Leftrightarrow \pi_A (R)$$

$$\sigma_P (\pi_A (R)) \Leftrightarrow \pi_A (\sigma_P (R)) \quad \text{falls } P \text{ nur Attribute aus } A \text{ umfaßt}$$

$$\sigma_P (\pi_A (R)) \Leftrightarrow \pi_A (\sigma_P (\pi_{A, B} (R))) \quad \text{falls } P \text{ auch Attribute aus } B \text{ umfaßt}$$

$$\sigma_{P(R_1)} (R_1 \ R_2) \Leftrightarrow \sigma_{P(R_1)} (R_1) \ R_2 \quad P(R_1) \text{ sei Prädikat auf } R_1$$

$$\pi_{A, B} (R_1 \ R_2) \Leftrightarrow \pi_A (R_1) \ \pi_B (R_2) \quad A / B \text{ seien Attributmengen aus } R_1 / R_2$$

$$\sigma_P (R_1 \cup R_2) \Leftrightarrow \sigma_P (R_1) \cup \sigma_P (R_2) \quad \text{inkl. Join-Attribut}$$

$$\pi_A (R_1 \cup R_2) \Leftrightarrow \pi_A (R_1) \cup \pi_A (R_2)$$

- Zusammenfassung von Operationsfolgen
- Minimierung der Größe von Zwischenergebnissen
- selektive Operationen ( $\sigma, \pi$ ) vor konstruktiven Operationen ( $\bowtie, \times, \cup$ )

## ■ Nutzung von Integritätsbedingungen (semantic query processing)

- Bsp.: A ist Primärschlüssel:  $\pi_A \rightarrow$  keine Duplikateliminierung erforderlich
- Integritätsbedingungen sind wahr für alle Tupel der betroffenen Relation: Hinzufügen einer Integritätsbedingung zur WHERE-Bedingung verändert den Wahrheitswert nicht



# Beispiel zur algebraischen Optimierung

## ■ Relationen:

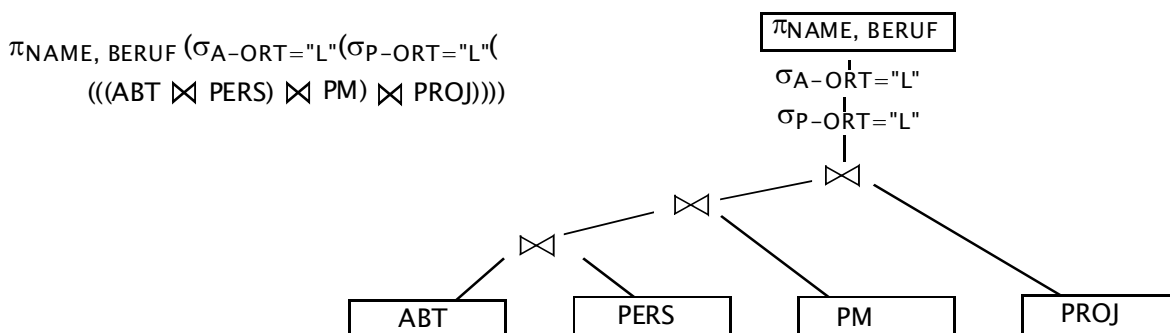
- ABT ( ANR, BUDGET, A-ORT )
- PERS ( PNR, NAME, BERUF, GEHALT, ALTER, ANR )
- PROJ ( PRONR, BEZEICHNUNG, SUMME, P-ORT )
- PM ( PNR, PRONR, DAUER, ANTEIL )

## ■ Annahmen

- ABT: N / 5 Tupel, PERS: N Tupel, PM: 5 · N Tupel, PROJ: M Tupel
- Gleichverteilung der Attributwerte A-ORT: 10 Werte; P-ORT: 100 Werte

## ■ Query: Finde Name und Beruf von Angestellten, deren Abteilung in L ist und die in L Projekte durchführen

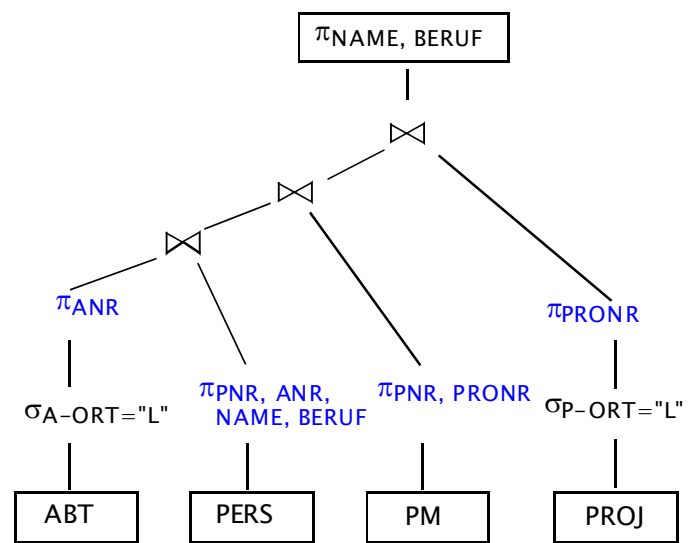
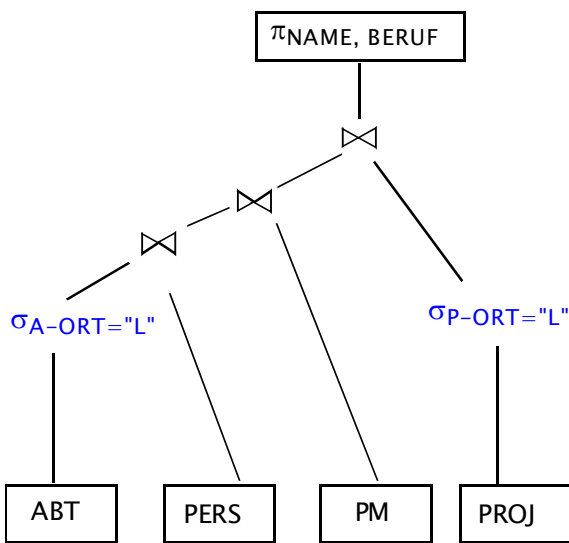
- Ausgangslösung für Operatorbaum



## Beispiel (2)

Verschieben der Selektion zu den Blattknoten

Verschieben der Projektion



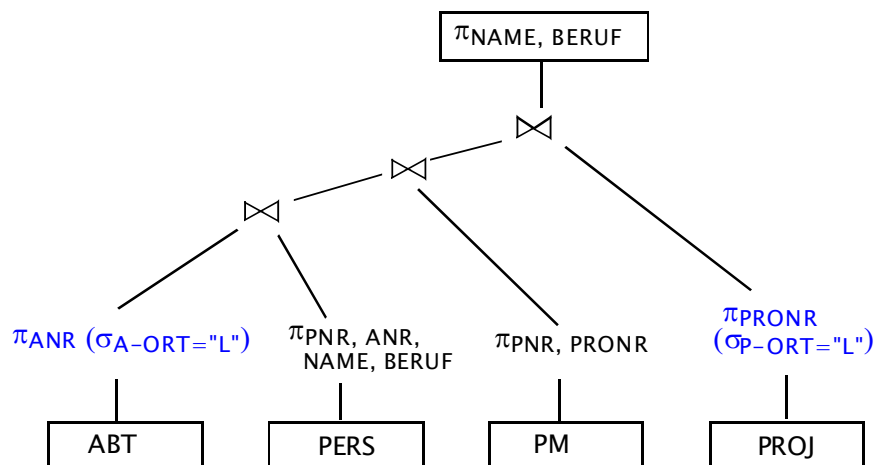
⇒ Führe Selektion so früh wie möglich aus !

⇒ Führe Projektion (ohne Duplikateliminierung) frühzeitig aus



## Beispiel (3)

■ Optimierter Operatorbaum:



⇒ Verknüpfe Folgen von unären Operationen wie Selektion und Projektion  
(wenn diese tupelweise abgewickelt werden können)

■ Alternative Möglichkeit: Zusammenfassen von

$$((\pi_{\text{PNR, PRONR}}^{\text{PM}}) \bowtie (\pi_{\text{PRONR}}^{\sigma_{\text{P-ORT}=\text{"L"}}} \cdot \text{PROJ})))$$



# Erstellung und Auswahl von Ausführungsplänen

## ■ Eingabe:

- transformierte Anfrage
- existierende Speicherstrukturen und Zugriffspfade
- Kostenmodell

## ■ Ausgabe: optimaler bzw. "guter" Ausführungsplan (Query Evaluation Plan)

## ■ Vorgehensweise:

1. Generiere alle "vernünftigen" logischen Zugriffspläne zur Auswertung der Anfrage
2. Zerlege komplexere Operationen in Folge von Ein- und Zwei-Variablen-Ausdrücke
3. Wähle für jeden logischen Operator Implementierungsstrategie unter Berücksichtigung der Zugriffspfade und Speicherstrukturen (Clusterung, Sortierreihenfolge etc.)
4. Wähle den billigsten Zugriffspfad gemäß dem vorgegebenen Kostenmodell aus

## ■ Suchstrategien

- voll-enumerativ
- beschränkt-enumerativ
- zufallsgesteuert

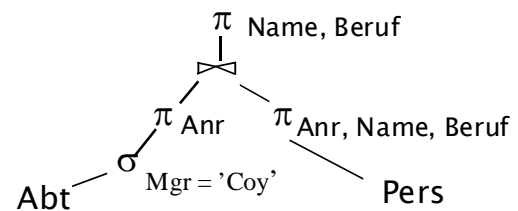
Reduzierung: bestimmte Suchpfade zur Erstellung von Ausführungsplänen werden nicht mehr verfolgt



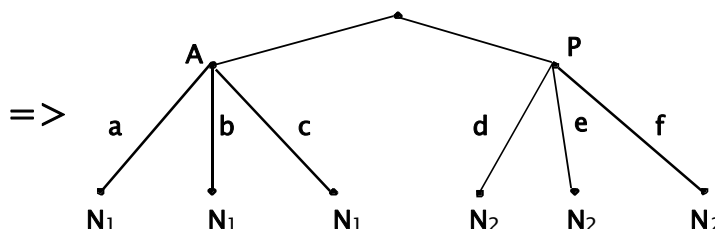
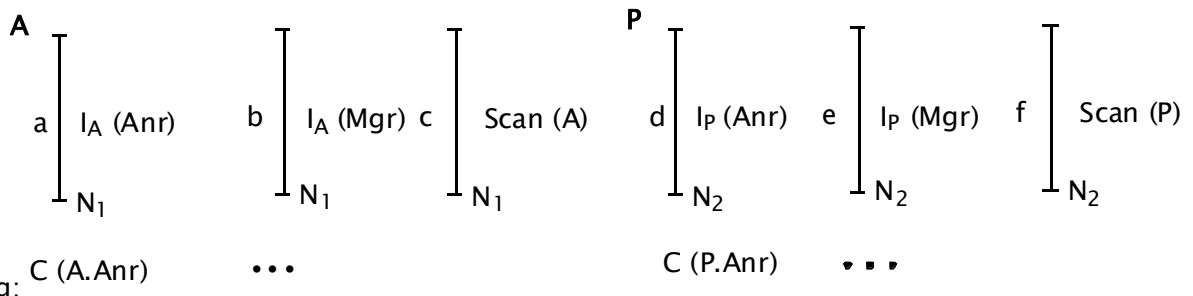
## Bestimmung von Ausführungsplänen: Beispiel

```

SELECT Name, Beruf
FROM   Pers P, Abt A
WHERE  P.Anr = A.Anr
AND    A.Mgr = 'Coy'
    
```



Mögliche Zugriffspfade (Annahme):



**Reduzierung:**  
Abschneiden von Teilbäumen



# Berechnung der Zugriffskosten

- Optimizer erstellt Kostenvoranschlag für jeden Zugriffsplan (möglicher Lösungsweg)
- Welche Kosten sind zu berücksichtigen?
  - Berechnungskosten (CPU-Kosten, Pfadlängen)
  - E/A-Kosten (# der physischen Referenzen)
  - Speicherkosten (temporäre Speicherbelegung im DB-Puffer und auf Externspeichern)
  - im verteilten Fall: Kommunikationskosten (# der Nachrichten, Menge der zu übertragenden Daten)

- Gewichtete Kostenformel:

$$C = \# \text{physischer Seitenzugriffe} + W * (\# \text{Aufrufe des Zugriffssystems})$$

- gewichtetes Maß für E/A- und CPU-Auslastung
- W ist das Verhältnis des Aufwandes für einen ZS-Aufruf zu einem Seitenzugriff

- Ziel der Gewichtung: Minimierung der Kosten in Abhängigkeit des Systemzustandes

System "I/O-bound":  $\Rightarrow$  kleines W:

$$W_{I/O} = \frac{\# \text{Instr. pro ZSAufruf}}{\# \text{Instr. pro E/A} + \text{Zugriffszeit} \cdot \text{MIPS-Rate}}$$

System "CPU-bound":  $\Rightarrow$  relativ großes W:

$$W_{CPU} = \frac{\# \text{Instr. pro ZSAufruf}}{\# \text{Instr. pro E/A}}$$



## Kostenmodell – statistische Werte

- statistische Größen:

$M_S$  Anzahl der Datenseiten des Segmentes S

$L_S$  Anzahl der leeren Seiten in Segment S

$N_R$  Anzahl der Tupeln der Relation R (Card(R))

$T_{R,S}$  Anzahl der Seiten in S mit Tupeln von R

$C_R$  Clusterfaktor (Anzahl Tupel pro Seite)

$j_I$  Anzahl der Attributwerte / Schlüsselwerte im Index I für Attribut A (=Card( $\pi_A(R)$ ))

$B_I$  Anzahl der Blattseiten (B\*-Baum) für Index I

...

- Statistiken müssen im Katalog gewartet werden

- Aktualisierung bei jeder Änderung zu aufwendig (zusätzliche Schreib- und Log-Operationen, Katalog wird zum Sperr-Engpass)
- Alternative:
  - Initialisierung der statistischen Werte zum Lade- oder Generierungszeitpunkt von Relationen und Indexstrukturen
  - periodische Neubestimmung der Statistiken durch eigenes Kommando/Dienstprogramm



# Kostenmodell - Berechnungsgrundlagen

Mit Hilfe der statistischen Werte kann der Optimizer jedem Verbundterm im Qualifikationsprädikat einen **Selektivitätsfaktor** ( $0 \leq SF \leq 1$ ) zuordnen (erwarteter Anteil an Tupeln, die das Prädikat erfüllen):

$$\text{Card}(\sigma_p(R)) = SF(p) \cdot \text{Card}(R)$$

## ■ Selektivitätsfaktor SF bei:

$$A_i = a_i \quad SF = \begin{cases} 1/j_i & \text{wenn Index auf } A_i \\ 1/10 & \text{sonst} \end{cases}$$

$$a_i \leq A_i \leq a_k \quad SF = \begin{cases} (a_k - a_i) / (\text{high-key} - \text{low-key}) & \text{wenn Index auf } A_i \\ 1/4 & \text{sonst} \end{cases}$$

$$A_i = A_k \quad SF = \begin{cases} 1 / \text{Max}(j_i, j_k) & \text{wenn Index auf } A_i, A_k \\ 1 / j_i & \text{wenn Index auf } A_i \\ 1/10 & \text{sonst} \end{cases}$$

$A_i$  IN (Liste von Werten)

$$A_i \geq a_i \quad SF = \begin{cases} (\text{high-key} - a_i) / (\text{high-key} - \text{low-key}) & \text{bei linearer Interpolation} \\ 1/3 & \text{sonst} \end{cases}$$

$$SF = \begin{cases} r / j_i & \text{bei } r \text{ Werten auf Index} \\ 1/2 & \text{sonst} \end{cases}$$

## ■ Berechnung von Ausdrücken

- $SF(p(A) \wedge p(B)) = SF(p(A)) \cdot SF(p(B))$
- $SF(p(A) \vee p(B)) = SF(p(A)) + SF(p(B)) - SF(p(A)) \cdot SF(p(B))$
- $SF(\neg p(A)) = 1 - SF(p(A))$

## ■ Join-Selektivitätsfaktor (JSF): $\text{Card}(R \bowtie S) = \text{JSF} * \text{Card}(R) * \text{Card}(S)$

- bei (N:1)-Joins (verlustfrei):  $\text{Card}(R \bowtie S) = \text{Max}(\text{Card}(R), \text{Card}(S))$



## Grundsätzlich Probleme

### ■ Anfrageoptimierung beruht i.a. auf zwei "fatalen" Annahmen

1. Alle Datenelemente und alle Attributwerte sind gleichverteilt
2. Suchprädikate in Anfragen sind unabhängig

### ■ beide Annahmen sind jedoch im allgemeinen Fall falsch !

### ■ Beispiel

(GEHALT  $\geq$  '100K') AND (ALTER BETWEEN 20 AND 30)

Bereiche:      10K - 1M                      20 - 65

-> lineare Interpolation, Multiplikation von Wahrscheinlichkeiten

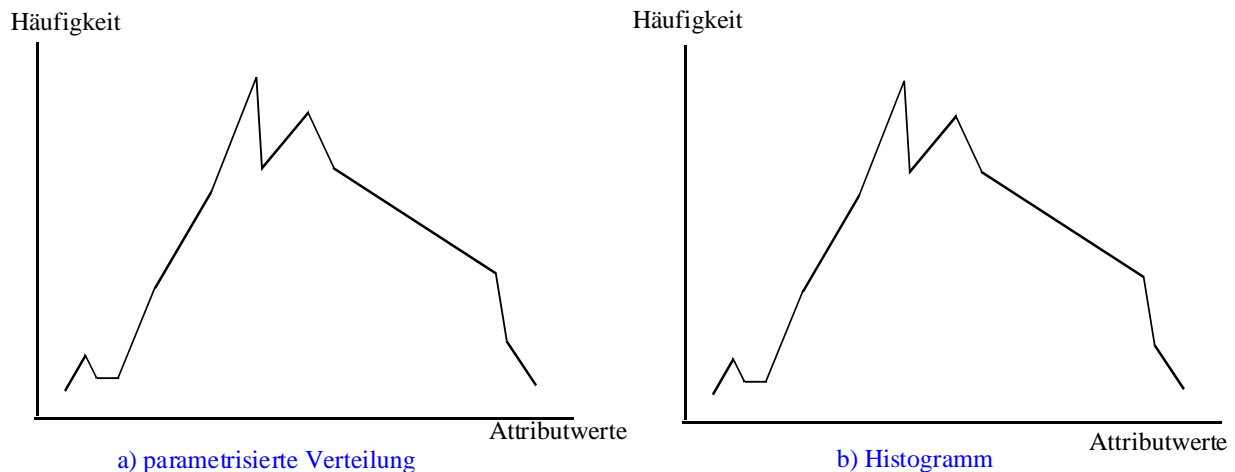
### ■ Lösung: Verbesserung der Statistiken / Heuristiken





# Verfeinerte Kostenmodelle

- verbesserte Ansätze zur Schätzung der Verteilung von Attributwerten
  - parametrisierte Verteilungen (z.B. Normalverteilung)
  - Histogramme
  - Stichproben
- Histogramme
  - Unterteilung des Wertebereichs in Intervalle; Häufigkeitszählung pro Intervall
  - äquidistante Intervalle vs. Intervalle mit etwa gleicher Häufigkeit von Werten (Equi-Depth-Histogramme)



## Tuning-Aspekte

- die meisten DBS nutzen mittlerweile kostenbasierten Optimierer
- Erzeugung bzw. Auffrischung der notwendigen Statistiken explizit durch DBA
  - Oracle: **analyze table PERS compute statistics for table;**
  - DB2: **runstats on table ...**
- Analyse generierter Auswertungspläne durch EXPLAIN-Anweisung

```
EXPLAIN PLAN FOR
SELECT DISTINCT S.Semester
FROM Student S, Hoert H,
      Vorlesung V, Prof P
WHERE P.Name="Rahm"
      AND V.liest = P.PNR
      AND V.VNR=H.VNR
      AND H.Matnr=S.Matnr
```



```
SELECT STATEMENT      Cost = 78340
  SORT UNIQUE
    HASH JOIN
      TABLE ACCESS FULL STUDENT
        HASH JOIN
          TABLE ACCESS BY ROWID PROF
            INDEX RANGE SCAN
              PROFNAMEINDEX
                TABLE ACCESS FULL VORLESUNG
                  TABLE ACCESS FULL HOERT
```

- graphische Darstellung von Auswertungsplänen



# Zusammenfassung

- Interpretation vs. Übersetzung
  - Interpretation: hoher Aufwand zur Laufzeit (v.a. bei wiederholter Ausführung einer Anweisung)
  - Übersetzung: pro DB-Anweisung wird zugeschnittenes Programm zur Übersetzungszeit erstellt  
-> hohe Laufzeiteffizienz
- Anfrageoptimierung: Kernproblem der Übersetzung mengenorientierter DB-Sprachen
  - Analyse
  - Anfragetransformation / Algebraische Optimierung
  - Optimierung unter Berücksichtigung von Zugriffspfaden und Operatorimplementierungen (Verwendung von Heuristiken)
  - Kostenbewertung und Auswahl des günstigsten Plans
  - Code-Generierung
- Kostenvoranschläge für Zugriffspläne:
  - CPU-Zeit und E/A-Aufwand
  - Anzahl der Nachrichten und zu übertragende Datenvolumina (im verteilten Fall)
- gute Optimierung erfordert genaue Statistiken
  - "fatale" Annahmen: Gleichverteilung aller Attributwerte, Unabhängigkeit aller Attribute
- EXPLAIN-Funktion zur Erklärung von Ausführungsplänen

