

# 7. Anfrageoptimierung

- Vorgehensweise
- Übersetzung vs. Interpretation von DB-Operationen
- Anfragedarstellung
- Logische Optimierung / Anfragetransformation
- Physische Optimierung
  - Erstellung alternativer Ausführungspläne
  - Auswahl eines kostengünstigen Plans
- Kostenbewertung
- Explain

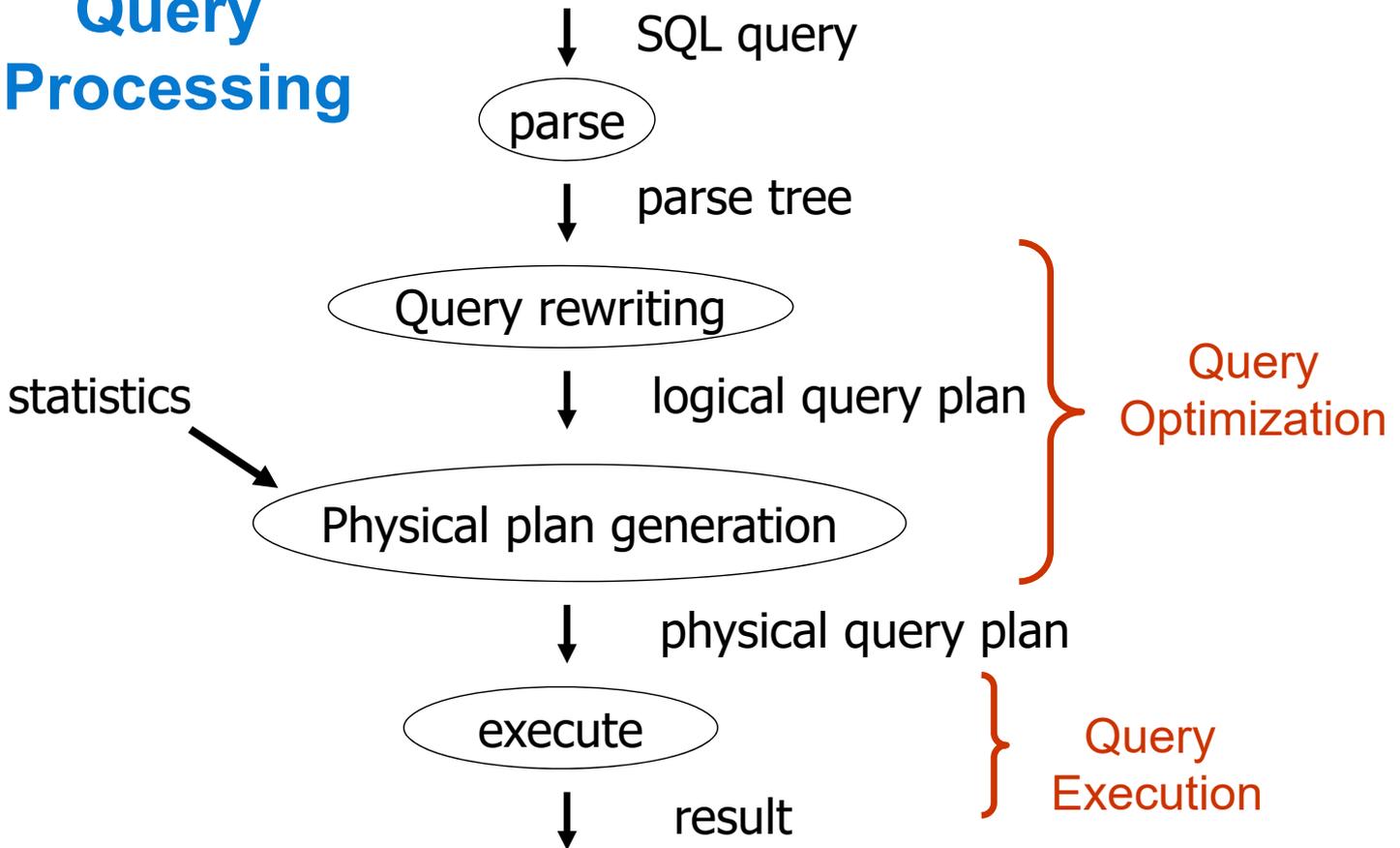


## Anfrageoptimierung

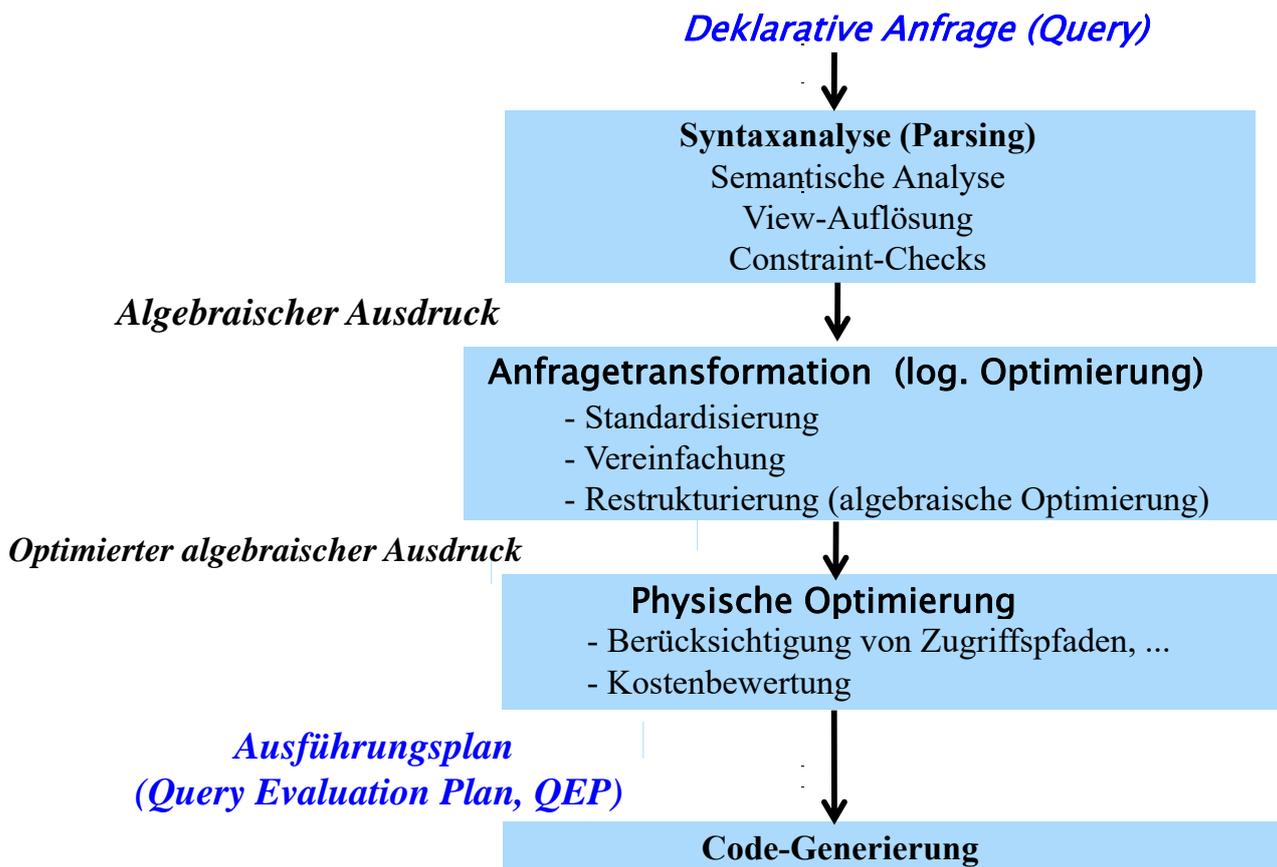
- zentrales Problem
  - Umsetzung deskriptiver Anfragen in eine zeitoptimale Folge interner DBS-Operationen
  - Effizienz ist Aufgabe des DBMS, nicht des Programmierers
- hohe Komplexität wegen großer Auswahlmächtigkeit von Sprachen wie SQL
  - mengenorientierte Operationen auf 1 oder mehreren Tabellen, inkl. Joins
  - Prädikate wie EXISTS, NULL, LIKE u. a.
  - geschachtelte Anfragen beliebiger Tiefe (unabhängig oder korreliert)
  - Operationen auf Views
  - Built-in- und Sortier-Funktionen auf Partitionen der Satzmenge
  - mengenorientierte Änderungsoperationen
  - Überwachung von Integritätsbedingungen
- oft extreme Kostenunterschiede zwischen funktional äquivalenten Zugriffsplänen
  - mit / ohne Indexnutzung
  - unterschiedliche Verfahren für Join, Sortierung, ...
  - unterschiedliche Reihenfolge (z.B. Selektion vor Join)



# Query Processing



# Übersetzung von DB-Anweisungen



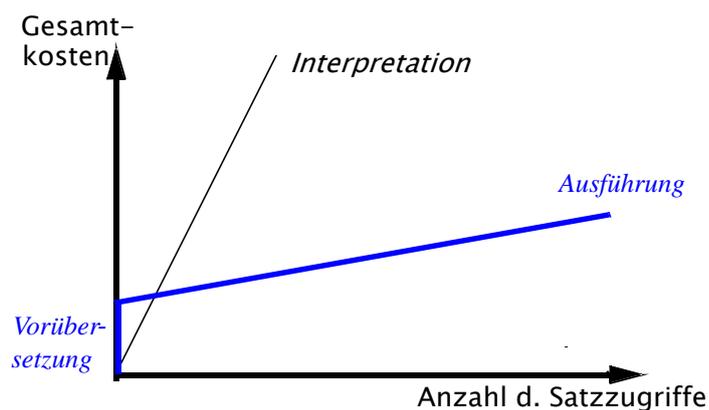
# Interpretation vs. Übersetzung (1)

- Anfrageanalyse und -optimierung können erfolgen
  - zur Übersetzungszeit des AP oder
  - zur Laufzeit (Interpretation)
- 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)



# Interpretation vs. Übersetzung (2)

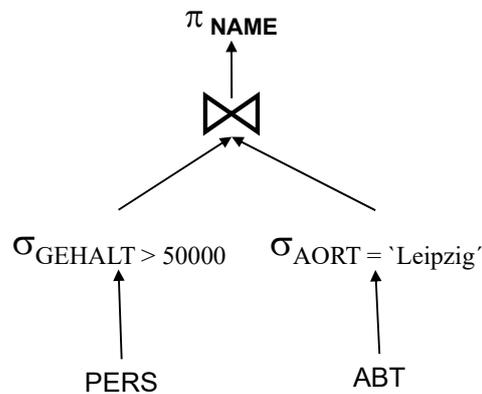
- Vorübersetzung:
  - Präcompiler führt Abbildungen aus (statische Namensbindung)
  - aufwändige Optimierung für jede DB-Operation möglich (Berücksichtigung mehrerer Ausführungsalternativen)
  - effiziente Ausführung
  - Änderungen des DB-Zustandes nach der Übersetzung werden nicht berücksichtigt (neue Indexstrukturen, geänderte Statistiken etc.)
    - => Invalidierung des ausführbaren Codes und Neuübersetzung



# Anfragedarstellung

## ■ Darstellung der Auswertungsstrategie durch Operatorgraph

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



# Anfragetransformation

## ■ Ziele der Anfragetransformation (algebraische bzw. logische 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

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

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

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

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

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

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

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

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

- 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)

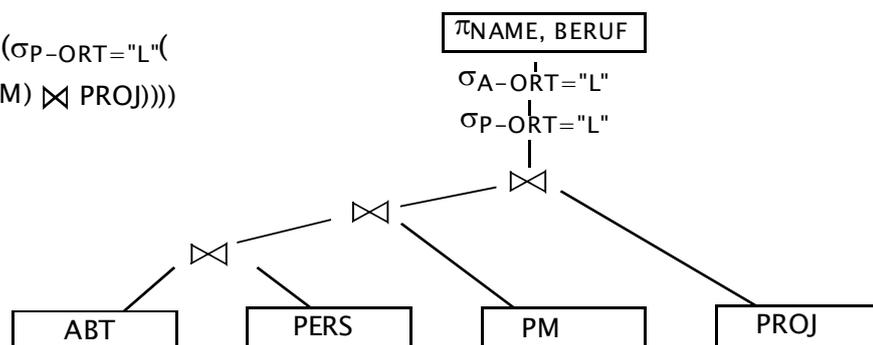
## ■ Annahmen

- 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

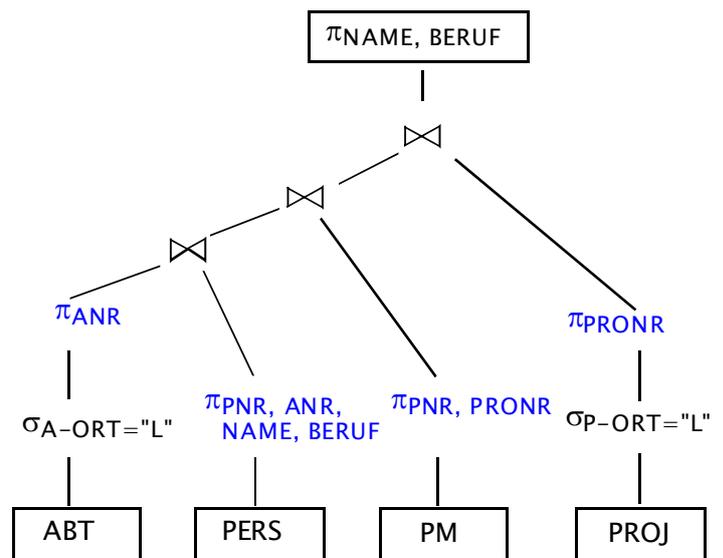
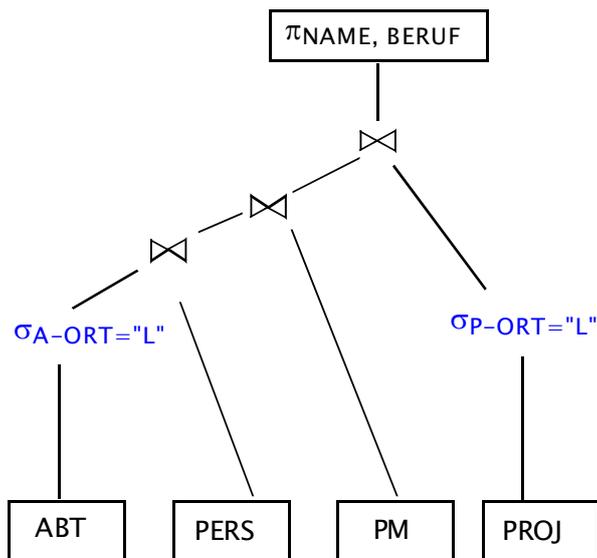
$$\pi_{\text{NAME, BERUF}}(\sigma_{\text{A-ORT}=\text{"L"}}(\sigma_{\text{P-ORT}=\text{"L"}}(((\text{ABT} \bowtie \text{PERS}) \bowtie \text{PM}) \bowtie \text{PROJ))))$$



## Beispiel (2)

Verschieben der Selektion zu den Blattknoten

Verschieben der Projektion

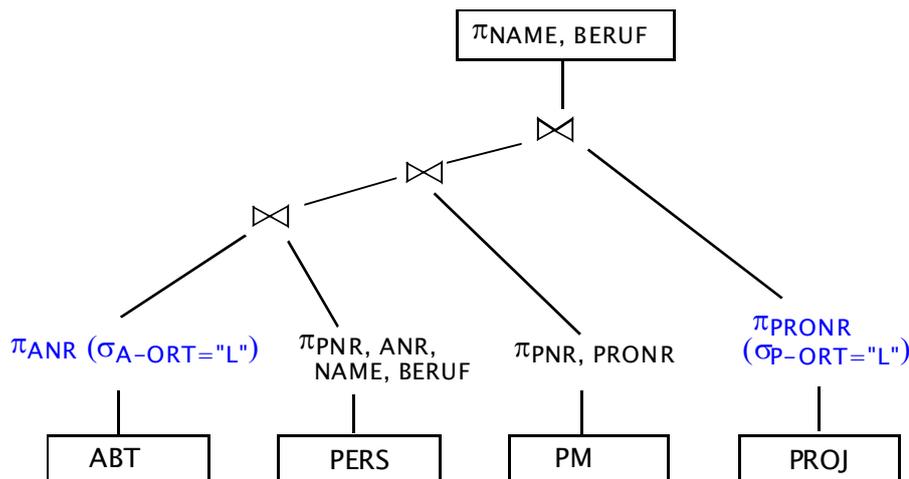


⇒ Führe Selektion so früh wie möglich aus !

⇒ 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)

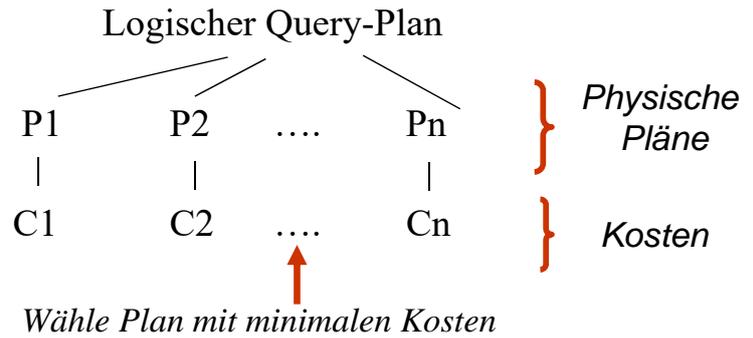
■ alternative Möglichkeit: Zusammenfassen von

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

# Physische Query-Optimierung

## ■ Eingabe:

- transformierte Anfrage
- existierende Speicherstrukturen und Zugriffspfade
- Kostenmodell



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

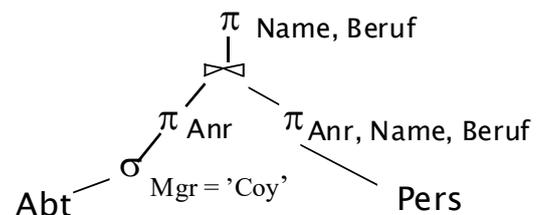
## ■ Vorgehensweise zur Generierung alternativer Pläne:

1. Generiere alle "vernünftigen" Zugriffspfade für Eingaberelationen
2. Berücksichtige unterschiedliche Reihenfolgen für Operatoren (z.B. für N-Wege-Join)
3. Wähle für jeden logischen Operator Implementierungsstrategie unter Berücksichtigung der Zugriffspfade und Speicherstrukturen (Clustering, Sortierreihenfolge etc.)
4. Wähle den billigsten Zugriffsplan gemäß dem vorgegebenen Kostenmodell aus

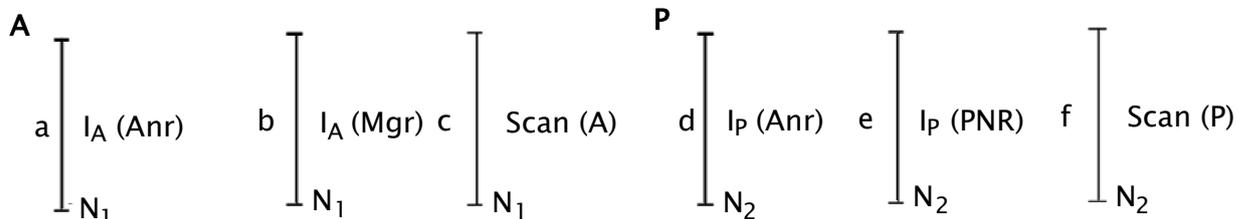


## Bestimmung von Ausführungsplänen: Beispiel

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

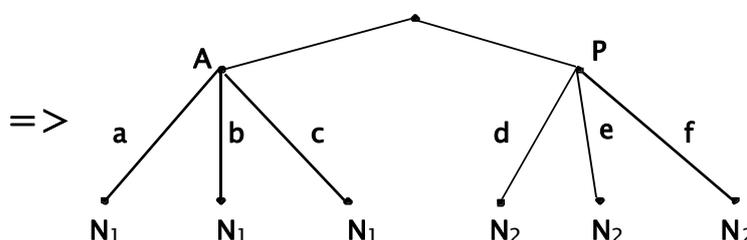


mögliche Zugriffspfade (Annahme):



Kosten-  
ermittlung: C (A.Anr) ...

C (P.Anr) ...



Reduzierung:  
Abschneiden von Teilbäumen



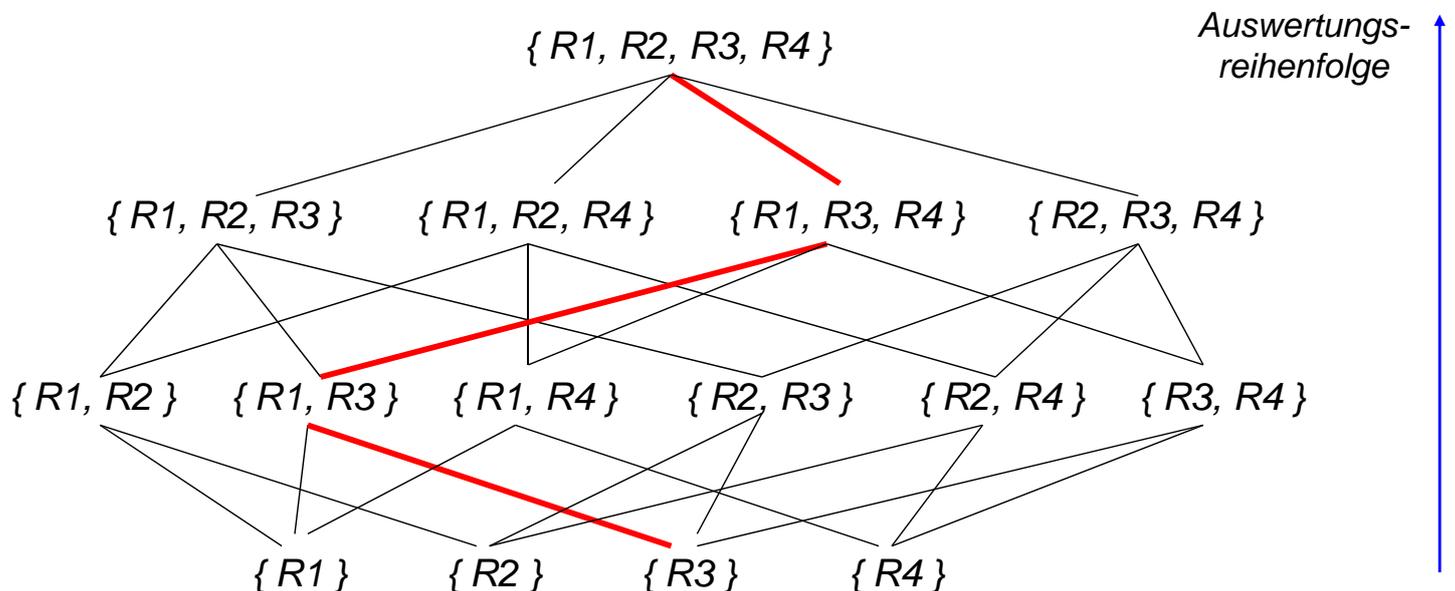
# Suchstrategien

- voll-enumerative bzw. teil-enumerative Auswertung alternativer Pläne
  - verbreiteter Einsatz: „dynamische Programmierung“
  - Pioniereinsatz in System R (Pat Selinger)
  - Annahme jeder Teilplan eines optimalen Plans ist auch optimal
  - Bottom-Up-Vorgehensweise: wähle für jeden Operator/Teilbaum billigsten Ansatz und vervollständige Teillösung iterativ
- zufallsgesteuert
- Reduzierung des Suchraums
  - bestimmte Pfade zur Erstellung von Ausführungsplänen werden nicht mehr verfolgt
  - Nutzung von Heuristiken: z.B. vermeide Berechnung des kartesischen Produkts, nur lineare Join-Ordnungen, etc.



## Beispiel Selinger-Algorithmus

Query:  $R1 \bowtie R2 \bowtie R3 \bowtie R4$



# 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 für zentralen Fall:

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

- gewichtetes Maß für E/A- und CPU-Auslastung
- $W$  ist das Verhältnis des Aufwandes für einen ZS-Aufruf zu einem Seitenzugriff
- kleines  $W$  falls E/A-Kosten dominieren („I/O bound“)
- relativ großes  $W$  bei CPU-beschränkten Umgebungen („CPU bound“)



## Kostenmodell – statistische Werte

- statistische Größen:

- $M_S$  Anzahl der Datenseiten des Segmentes  $S$
- $N_R$  Anzahl der Tupeln der Relation  $R$  (Card( $R$ ))
- $T_{R,S}$  Anzahl der Seiten in  $S$  mit Tupeln von  $R$
- $B_R$  Blockungsfaktor (Anzahl Tupel pro Seite)
- $j_I$  Anzahl der Attributwerte / Schlüsselwerte im Index  $I$  für Attribut  $A$  (=Card ( $\pi_A(R)$ ))
- $L_I$  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

- Optimizer kann jedem Term im Selektionsprädikat auf Basis statistischer Werte **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:
  - $A_i = a_i$   $SF = \begin{cases} 1/j_i & \text{wenn Index auf } A_i \\ 1/10 & \text{sonst} \end{cases}$
  - $A_i = A_k$   $SF = \begin{cases} 1 / \text{Max}(j_i, j_k) & \text{wenn Index auf } A_i, A_k \\ 1 / j_i & \text{wenn Index auf } A_i \\ 1/10 & \text{sonst} \end{cases}$
  - $A_i \geq a_i$   $SF = \begin{cases} (\text{high-key} - a_i) / (\text{high-key} - \text{low-key}) & \text{bei linearer Interpolation} \\ 1/3 & \text{sonst} \end{cases}$
  - $a_i \leq A_i \leq a_k$   $SF = \begin{cases} (a_k - a_i) / (\text{high-key} - \text{low-key}) & \text{wenn Index auf } A_i \\ 1/4 & \text{sonst} \end{cases}$
  - $A_i \text{ IN (Liste von Werten)}$ 

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



## Kostenmodell (2)

- 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 (verlustfreien) Fremdschlüssel-Joins (N:1):
  - »  $\text{Card}(R \bowtie S) = \text{Max}(\text{Card}(R), \text{Card}(S))$





# 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**
  - meist auch graphische Darstellung von Auswertungsplänen
- Beispiel für Oracle



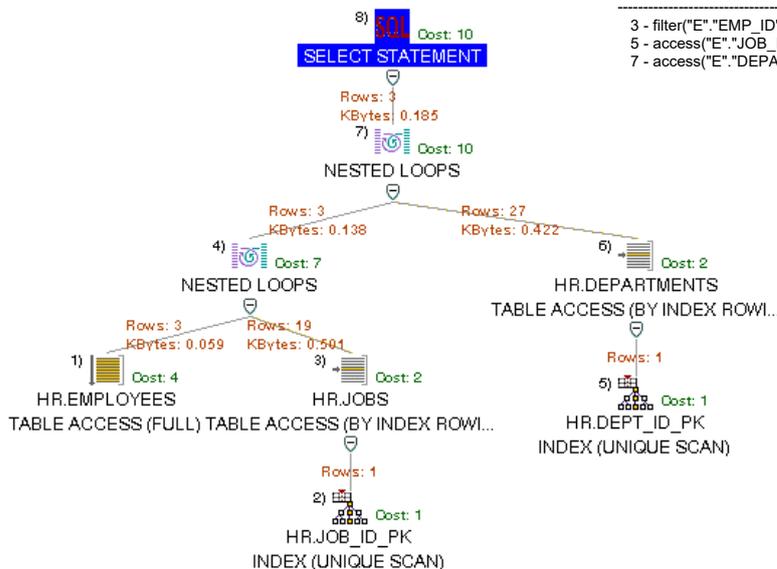
## Oracle Explain-Beispiel

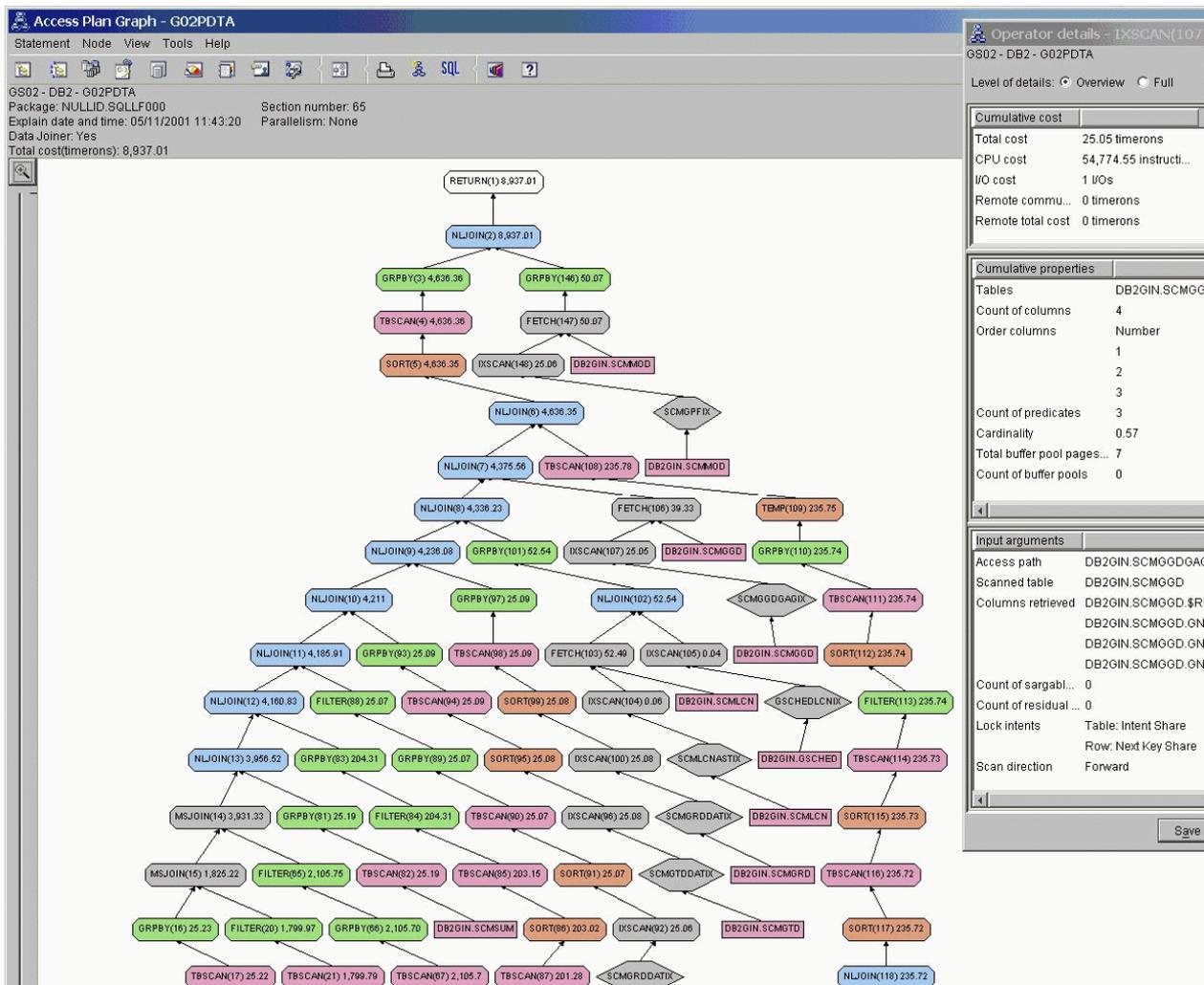
**EXPLAIN PLAN FOR**  
**SELECT** e.emp\_id, j.job\_title,  
 e.salary, d.department\_name  
**FROM** employees e, jobs j,  
 departments d  
**WHERE** e.employee\_id < 103  
 AND e.job\_id = j.job\_id  
 AND e.department\_id = d.department\_id;

Id	Operation (%CPU)	Name	Rows	Byte	Cost
0	SELECT STATEMENT		3	189	10 (10)
1	NESTED LOOPS		3	189	10 (10)
2	NESTED LOOPS		3	141	7 (15)
* 3	TABLE ACCESS FULL	EMPLOYEES	3	60	4 (25)
4	TABLE ACCESS BY INDEX ROWID	JOBS	19	513	2 (50)
* 5	INDEX UNIQUE SCAN	JOB_ID_PK	1		
6	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	27	432	2 (50)
* 7	INDEX UNIQUE SCAN	DEPT_ID_PK	1		

Predicate Information (identified by operation id):

- 3 - filter("E"."EMP\_ID"<103)
- 5 - access("E"."JOB\_ID"="J"."JOB\_ID")
- 7 - access("E"."DEPARTMENT\_ID"="D"."DEPARTMENT\_ID")





## 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
  - logische Optimierung / Query Rewrite
  - kostenbasierte physische Optimierung unter Berücksichtigung von Zugriffspfaden und Operatorimplementierungen (Verwendung von Heuristiken)
  - Code-Generierung
- Kostenvoranschläge für Ausführungspläne:
  - 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

