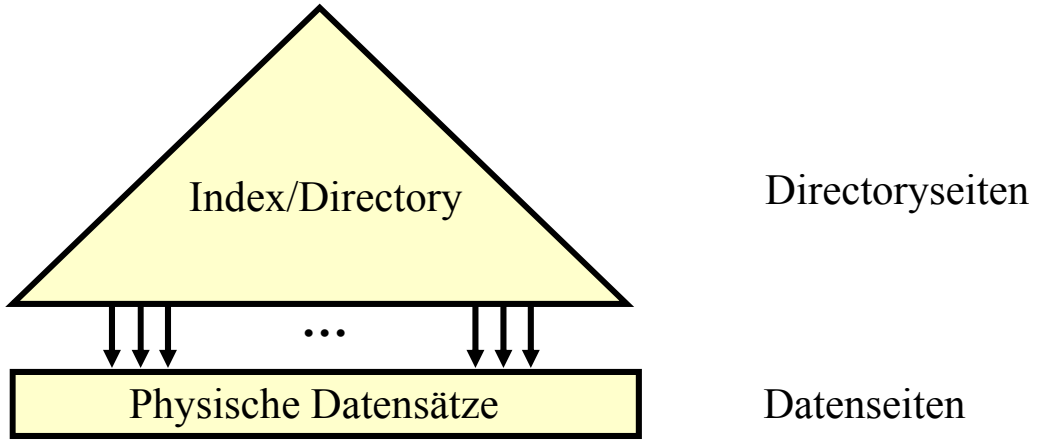
6. Anfragebearbeitung

6.2 Indexstrukturen

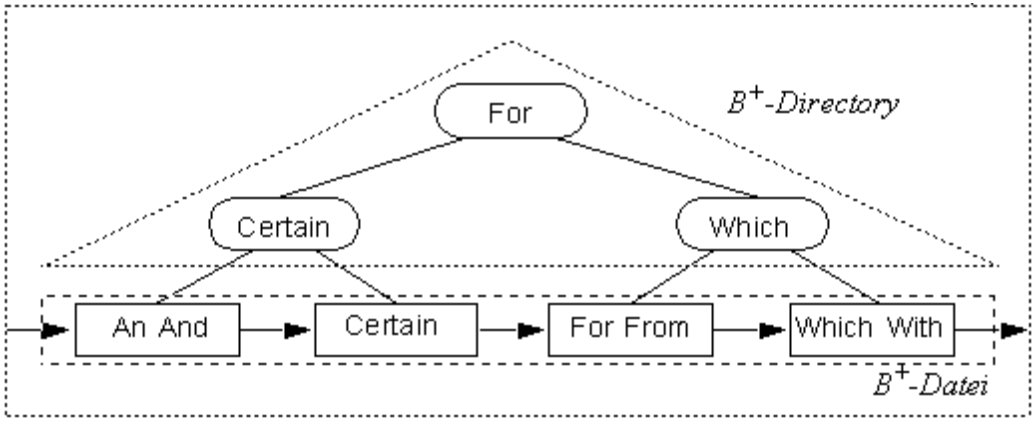
Aufbau baumartiger Indexstrukturen

- Baumartige Indexstrukturen bestehen üblicherweise aus Directory- und Datenseiten:
  - Die eigentlichen physischen Datensätze werden in den **Datenseiten** gespeichert (Blattknoten des zugrundeliegenden Suchbaums).
  - Die **Directoryseiten** sind die inneren Knoten des Suchbaums und speichern Einträge, die aus aggregierten Zugriffsinformationen bestehen und die Navigation im Suchbaum ermöglichen.



Beispiel B+-Baum

- Erweiterung des B-Baums (vgl. Vorlesungen Algorithmen und Datenstrukturen, Index- und Speicherstrukturen)
  - Datenelemente nur in den Blattknoten speichern
  - Innere Knoten enthalten lediglich Schlüssel
- Knoten entsprechen Seiten auf der Platte, d.h. Zugriff erfordert ggfs. einen Plattenzugriff
- B^+-Baum für die Zeichenketten: *An, And, Certain, For, From, Which, With*



6.2 Indexstrukturen

Modellierung der I/O-Kosten

- In DBS sind zwei verschiedene I/O-Zugriffsmuster vorherrschend:
  - *Sequentielles Lesen* einer großen Datei (Verarbeitung von Relationen ohne Index)
  - *Wahlfreies Lesen* von Blöcken konstanter Größe, wobei die einzelnen Blöcke an wahlfreien Positionen beginnen (Verarbeitung von Relationen mit Hilfe eines Index)
- Seien

f	Größe der Datei in MByte
a	Anzahl der Blockzugriffe
c_{puffer}	Größe des Puffers im Arbeitsspeicher in MByte
c_{index}	Blockgröße des Index in MByte
t_{seek}	Suchzeit in ms
t_{lat}	Latenzzeit in ms
t_{tr}	Transferleistung des Laufwerkes in ms/MByte

Modellierung der I/O-Kosten (cont.)

– Sequentielles Lesen

- Da der Arbeitsspeicher begrenzt ist, erfolgt das sequentielle Lesen einer Datei in einzelnen Blöcken, die zwischen den I/O-Aufträgen verarbeitet werden:
  - Bei der ersten Leseoperation wird der Schreib-/Lesekopf auf die entsprechende Position gesetzt.
  - Bei jeder weiteren Leseoperation fallen nur noch Latenzzeit und Transferzeit an.

$$t_{scan} = t_{seek} + f \cdot t_{tr} + \left\lceil \frac{f}{c_{puffer}} \right\rceil \cdot t_{lat}$$

- Meist wählt man den Puffer so groß, dass die Transferzeit pro Leseoperation wesentlich höher als die Latenzzeit ist. In diesem Fall können Latenz- und Suchzeit vernachlässigt werden:

$$t_{scan} \approx f \cdot t_{tr}$$

Modellierung der I/O-Kosten (cont.)

– Wahlfreies Lesen

- Bei wahlfreien Zugriffen fallen bei jedem Auftrag sowohl Transferzeit, Latenzzeit als auch Suchzeit an:

$$t_{random} = (t_{seek} + t_{lat} + c_{index} \cdot t_{tr}) \cdot a$$

- Verglichen mit der scanbasierten Verarbeitung ist die Größe c_{index} einer Transfereinheit hier nicht durch den zur Verfügung stehenden Arbeitsspeicher, sondern durch die Blockgröße des Indexes vorgegeben (und typischerweise wesentlich kleiner, z.B. 4-8 KBytes).

6.2 Indexstrukturen

Wahlfreier vs. sequentieller Zugriff

- Ein sequentieller Zugriff auf *n* Datenblöcke ist in etwa *n*-mal schneller als *n* nacheinander ausgeführte wahlfreie Zugriffe auf die Datenblöcke (für große *n*)
- Transferraten wachsen schneller als Zugriffsraten

  => Verhältnis wahlfreier zu sequentieller Zugriff

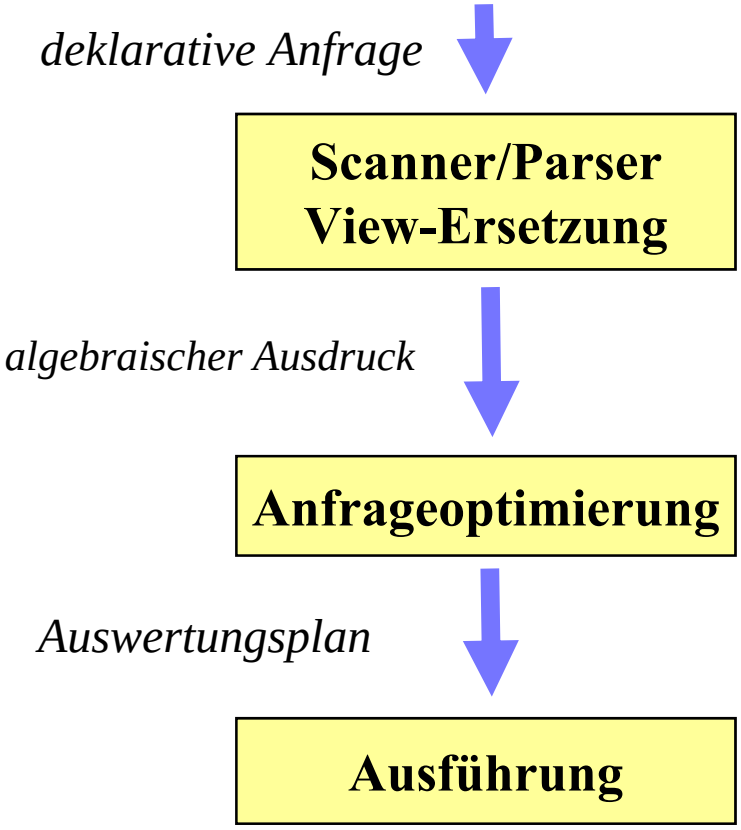
  verschlechtert sich
- Folgende Maßnahmen sind deshalb wichtig:
  - *Große Blöcke*: Die Wahl größerer Transfereinheiten verbessert das Verhältnis
  - *Clusterbildung der Daten*: Die Daten sollten von einer Indexstruktur so in Blöcken abgelegt werden, dass mit großen Blöcken in möglichst wenigen Zugriffen möglichst viele nützliche Daten übertragen werden

6. Anfragebearbeitung

6.3 Grundlagen der Anfrageoptimierung

Aufgabe der Anfragebearbeitung

– Übersetzung der *deklarativen* Anfrage in einen *effizienten, prozeduralen* Auswertungsplan



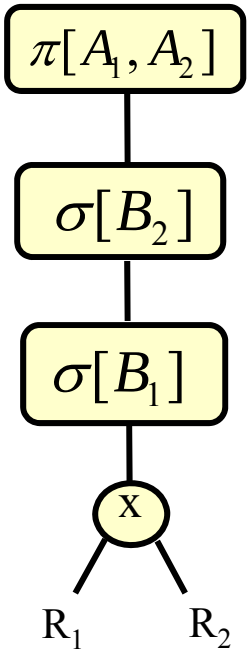
Kanonischer Auswertungsplan

SELECT A_1, A_2

FROM R_1, R_2

WHERE B_1 *AND* B_2

$$\pi_{A_1,A_2}(\sigma_{B_2}(\sigma_{B_1}(R_1 \times R_2)))$$



1. Bilde das kartesische Produkt der Relationen R_1, R_2
2. Führe Selektionen mit den Bedingungen B_1, B_2 durch.
3. Projiziere die Ergebnis-Tupel auf die erforderlichen Attribute A_1, A_2

Logische vs. physische Anfrageoptimierung

- Optimierungstechniken, die den Auswertungsplan umbauen (d.h. die Reihenfolge der Operatoren verändern), werden als **logische Anfrageoptimierung** bezeichnet.

- Unter **physischer Anfrageoptimierung** versteht man z.B. die Auswahl einer geeigneten Auswertungsstrategie für die Join-Operation oder die Entscheidung, ob für eine Selektionsoperation ein Index verwendet wird oder nicht und wenn ja, welcher (bei unterschiedlichen Alternativen). Es handelt sich also um die Auswahl eines geeigneten Algorithmus für jede Operation im Auswertungsplan.

Regel- vs. kostenbasierte Anfrageoptimierung

- Es gibt zahlreiche Regeln (Heuristiken), um die Reihenfolge der Operatoren im Auswertungsplan zu modifizieren und so eine Performanz-Verbesserung zu erreichen, z.B.:
  - Push Selection: Führe Selektionen möglichst frühzeitig (vor Joins) aus
  - Elimination leerer Teilbäume
  - Erkennen gemeinsamer Teilbäume

- Optimierer, die sich ausschließlich nach diesen starren Regeln richten, nennt man **regelbasierte** oder auch **algebraische Optimierer**.

Regel-/kostenbasierte Optimierung (cont.)

– Optimierer, die zusätzlich zu den starren Regeln die voraussichtliche Performanz der Auswertungspläne ermitteln und den leistungsfähigsten Plan auswählen, werden als **kostenbasierte** oder auch (irreführend) **nicht-algebraische Optimierer** bezeichnet.

– Die Vorgehensweise kostenbasierter Anfrageoptimierer ist meist folgende:
  - Generiere einen initialen Plan (z.B. Standardauswertungsplan)
  - Schätze die bei der Auswertung entstehenden Kosten
  - Modifiziere den aktuellen Plan gemäß vorgegebener Heuristiken
  - Wiederhole die Schritte 2 und 3 bis ein Stop-Kriterium erreicht ist
  - Gib den besten erhaltenen Plan aus

6. Anfragebearbeitung

6.4 Logische Anfrageoptimierung

Äquivalenzregeln der Relationalen Algebra

– Join, Vereinigung, Schnitt und Kreuzprodukt sind kommutativ

$$
\begin{aligned}
R \quad S &= S \quad R \\
R \cup S &= S \cup R \\
R \cap S &= S \cap R \\
R \times S &= S \times R
\end{aligned}
$$

– Join, Vereinigung, Schnitt und Kreuzprodukt sind assoziativ

$$
\begin{aligned}
R \bowtie (S \bowtie T) &= (R \bowtie S) \bowtie T \\
R \cup (S \cup T) &= (R \cup S) \cup T \\
R \cap (S \cap T) &= (R \cap S) \cap T \\
R \times (S \times T) &= (R \times S) \times T
\end{aligned}
$$

– Selektionen sind untereinander vertauschbar

$$\sigma_{Bed1}(\sigma_{Bed2}(R)) = \sigma_{Bed2}(\sigma_{Bed1}(R))$$

6.4 Logische Anfrageoptimierung

Äquivalenzregeln der Relationalen Algebra (cont.)

- Konjunktionen in einer Selektionsbedingung können in mehrere Selektionen aufgebrochen werden, bzw. nacheinander ausgeführte Selektionen können zu einer konjunktiven Selektion zusammengefasst werden

$$\sigma_{B1 \wedge B2 \wedge \ldots \wedge Bn}(R) = \sigma_{B1}(\sigma_{B2}(\ldots(\sigma_{Bn}(R))\ldots))$$

- Geschachtelte Projektionen können eliminiert werden

$$\pi_{A1}(\pi_{A2}(\ldots(\pi_{An}(R))\ldots)) = \pi_{A1}(R)$$

Damit eine solche Schachtelung sinnvoll ist, muss gelten: $A1 \subseteq A2 \subseteq \ldots \subseteq An$

- Selektion und Projektion sind vertauschbar, falls die Projektion keine Attribute der Selektionsbedingung entfernt

$$\pi_A(\sigma_B(R)) = \sigma_B(\pi_A(R)) \quad \text{, falls } attr(B) \subseteq A$$

6.4 Logische Anfrageoptimierung

Äquivalenzregeln der Relationalen Algebra (cont.)

– Selektion und Join (Kreuzprodukt) können vertauscht werden, falls die Selektion nur Attribute eines der beiden Join-Argumente verwendet

$$\sigma_B(R \bowtie S) = \sigma_B(R) \bowtie S$$
$$\sigma_B(R \times S) = \sigma_B(R) \times S$$

, falls $attr(B) \subseteq attr(R)$

– Projektionen können teilweise in den Join verschoben werden

$$\pi_A(R \bowtie_B S) = \pi_A(\pi_{A1}(R) \bowtie_B \pi_{A2}(S))$$

, falls $A1 = attr(R) \cap (A \cup attr(B))$
$A2 = attr(S) \cap (A \cup attr(B))$

– Selektionen können mit Vereinigung, Schnitt und Differenz vertauscht werden

$$\sigma_B(R \cup S) = \sigma_B(R) \cup \sigma_B(S)$$

6.4 Logische Anfrageoptimierung

Äquivalenzregeln der Relationalen Algebra (cont.)

- Der Projektionsoperator kann mit der Vereinigung, aber nicht mit Schnitt oder Differenz vertauscht werden (siehe Übung!)

$$\pi_A(R \cup S) = \pi_A(R) \cup \pi_A(S)$$

- Selektion und ein Kreuzprodukt können zu einem Join zusammengefasst werden, wenn die Selektionsbedingung eine Joinbedingung ist (z.B. Equi-Join)

$$\sigma_{R.A1=S.A2}(R \times S) = R \bowtie_{R.A1=S.A2} S$$

- Auch an Bedingungen können Veränderungen vorgenommen werden
  - Kommutativgesetze, Assoziativgesetze, z.B. $B_1 \wedge B_2 = B_2 \wedge B_1$
  - Distributivgesetze, z.B. $B_1 \vee (B_2 \wedge B_3) = (B_1 \vee B_2) \wedge (B_1 \vee B_3)$
  - De Morgan, z.B. $\neg(B_1 \wedge B_2) = \neg B_1 \vee \neg B_2$

6.4 Logische Anfrageoptimierung

Restrukturierungsalgorithmus

- Aufbrechen der Selektionen
- Verschieben der Selektionen so weit wie möglich nach unten im Operatorbaum
- Zusammenfassen von Selektionen und Kreuzprodukten zu Joins
- Einfügen und Verschieben von Projektionen so weit wie möglich nach unten
- Zusammenfassen einzelner Selektionen zu komplexen Selektionen

6.4 Logische Anfrageoptimierung

Beispiel

Fahrzeug-Datenbank

Kunde(KNr, Name, Adresse, Region, Saldo)

KNr	Name	Adresse	Region	Saldo
201	Klein	Lilienthal	Bremen	200 000
337	Horn	Dieburg	Rhein-Main	100 000
444	Berger	München	München	300 000
108	Weiss	Würzburg	Unterfranken	500 000

Bestellt(BNr, Datum, KNr, KNr, PNr)

BNr	Datum	KNr	PNr
221	10.05.04	201	12
312	11.05.04	201	4
401	20.05.04	337	330
456	13.05.04	444	330
458	14.05.04	444	98

Produkt(PNr, Bezeichnung, Anzahl, Preis)

PNr	Bezeichnung	Anzahl	Preis
12	BMW 318i	10	40.000
4	Golf 5	40	25.000
330	Fiat Uno	5	18.000
98	Ferrari 380	1	180.000
14	Opel Corsa	14	17.000

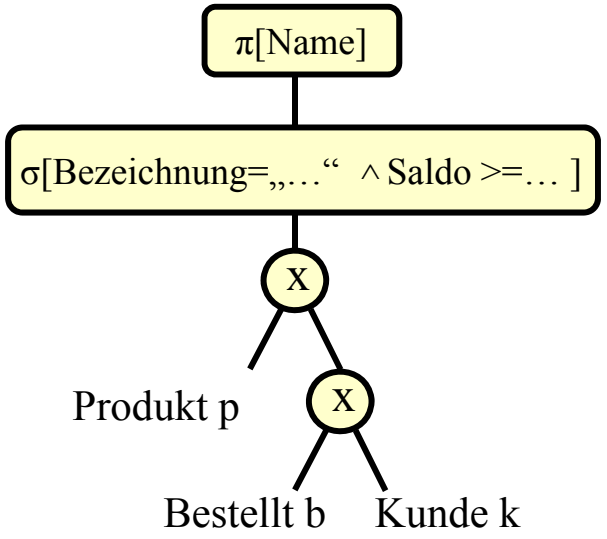
6.4 Logische Anfrageoptimierung

Beispiel (cont.)

– SQL Anfrage:

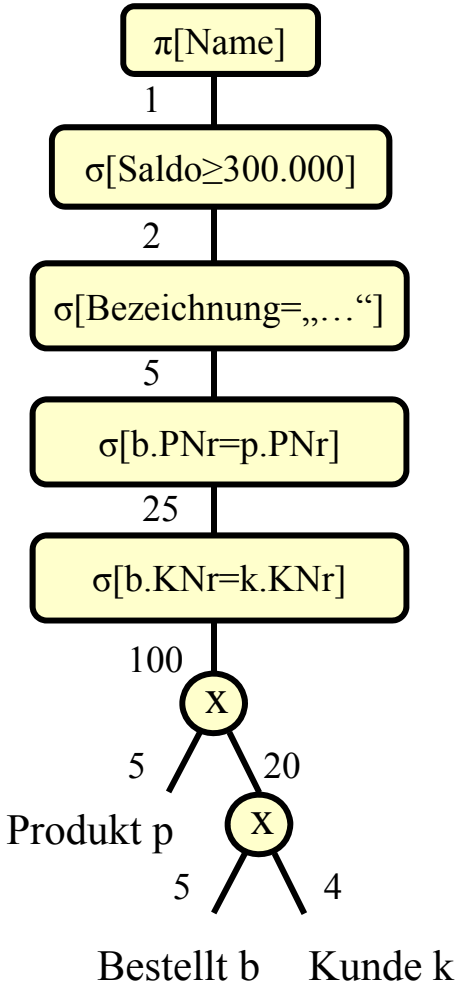
```
select      Name,
from        Kunde k, Bestellt b, Produkt p
where       b.KNr = k.KNr
and         b.PNr = p.PNr
and         Bezeichnung = „Fiat Uno“
and         Saldo ≥ 300.000
```

– Kanonischer Auswertungsplan:

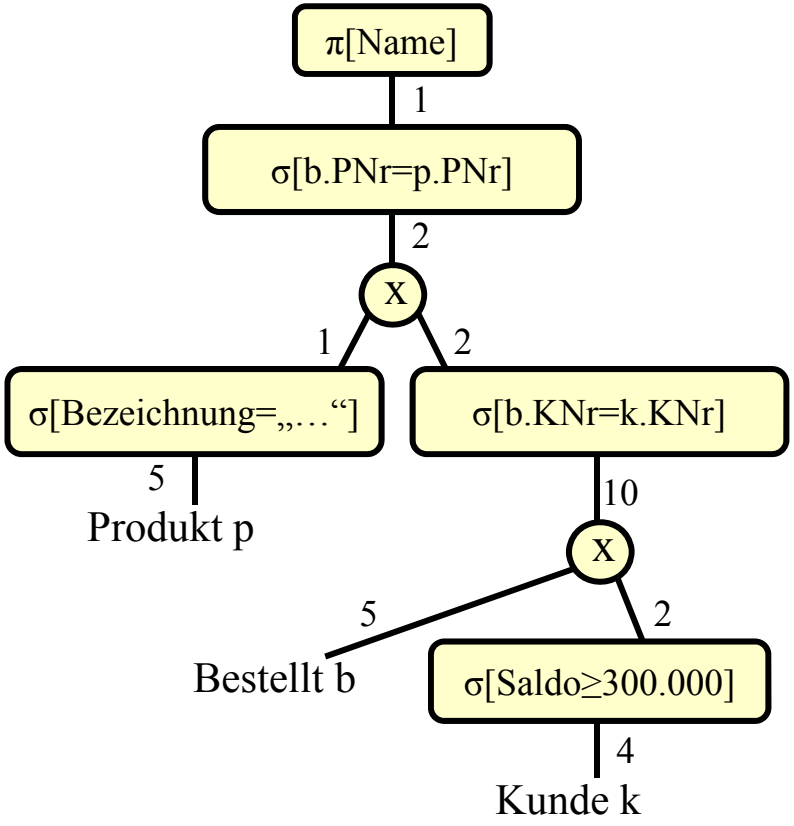


6.4 Logische Anfrageoptimierung

- Aufbrechen der Selektionen

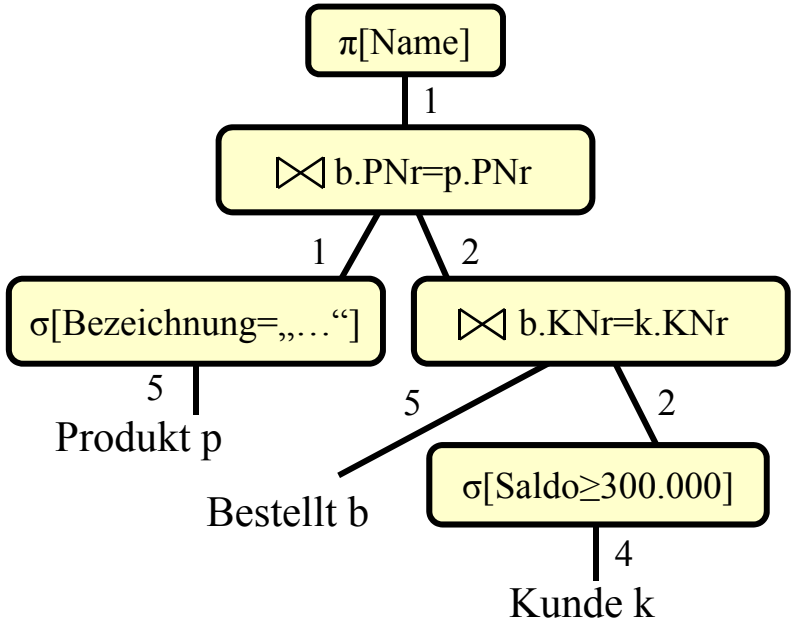


- Verschieben der Selektionen

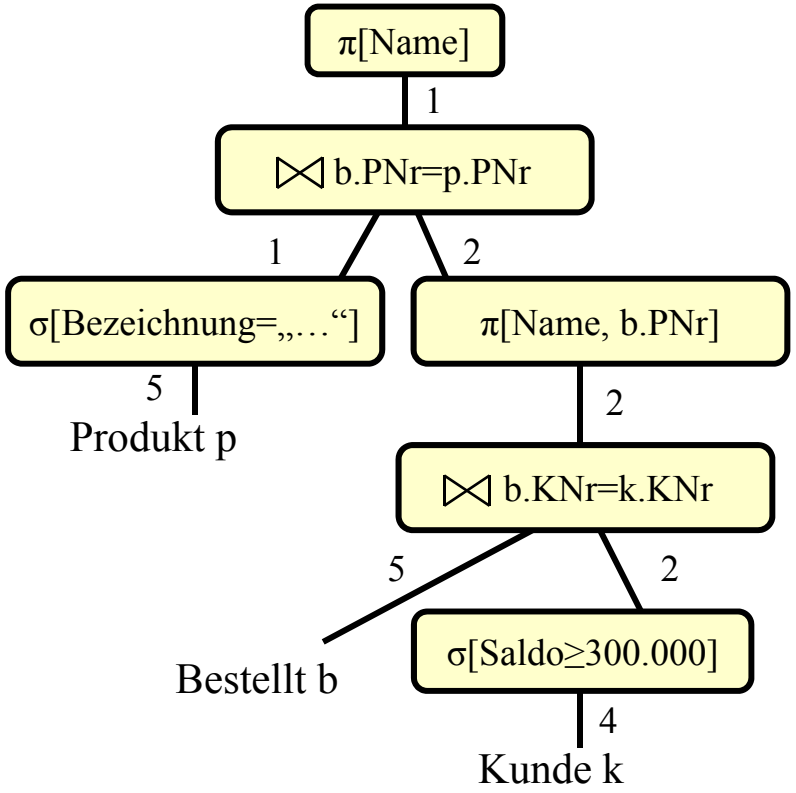


6.4 Logische Anfrageoptimierung

- Zusammenfassen zu Joins



- Einfügen zusätzlicher Projektionen



6. Anfragebearbeitung

Selektivität

- Der Anteil der qualifizierenden Tupel wird *Selektivität sel* genannt.
- Für die Selektion und den Join ist sie folgendermaßen definiert:
  - **Selektion** mit Bedingung *B*: $$sel_B = \frac{|\sigma_B(R)|}{|R|}$$

    (relativer Anteil der Tupel, die *B* erfüllen)
  - **Join** von *R* und *S*: $$sel_{RS} = \frac{|R \bowtie S|}{|R \times S|} = \frac{|R \bowtie S|}{|R| \cdot |S|}$$

    (Anteil relativ zur Kardinalität des Kreuzprodukts)

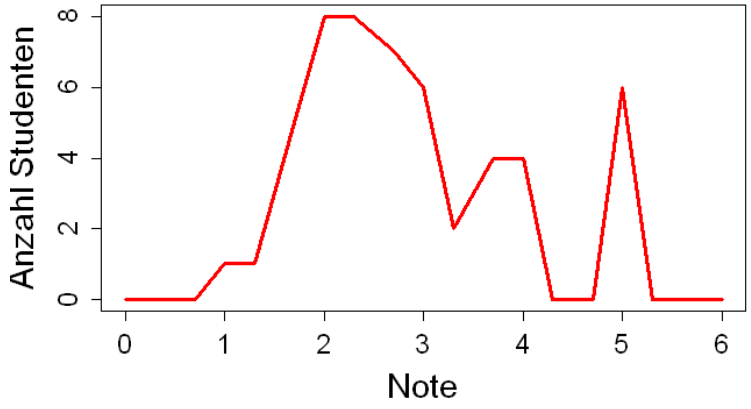
Selektivität (cont.)

- Die Selektivität muss geschätzt werden, für Spezialfälle gibt es einfache Methoden:
  - Die Selektivität von $\sigma_{R.A=c}$, also Vergleich mit einer Konstante c beträgt 1 / |R|, falls A ein Schlüssel ist
  - Falls A kein Schlüssel ist, aber die Werte gleichverteilt sind, ist sel=1 / I (I ist dabei die *image size*, d.h. die Anzahl versch. A-Werte in R)
  - Besitzt bei einem Equi-Join von R und S (Join Bed.: R.A=S.B) das Attribut A Schlüsseleigenschaft, kann die Größe des Join-Ergebnisses mit |S| abgeschätzt werden, da jedes Tupel aus S maximal einen Joinpartner findet. Die Selektivität ist also selRS = 1/|R|
  - logisches UND: $sel_B(\sigma_{B1 \wedge B2}) = sel_B(\sigma_{B1}) \cdot sel_B(\sigma_{B2})$
  - logisches ODER: $sel_B(\sigma_{B1 \vee B2}) = sel_B(\sigma_{B1}) + sel_B(\sigma_{B2}) - sel_B(\sigma_{B1}) \cdot sel_B(\sigma_{B2})$
  - logisches NICHT: $sel_B(\sigma_{\neg B1}) = 1 - sel_B(\sigma_{B1})$

Selektivität (cont.)

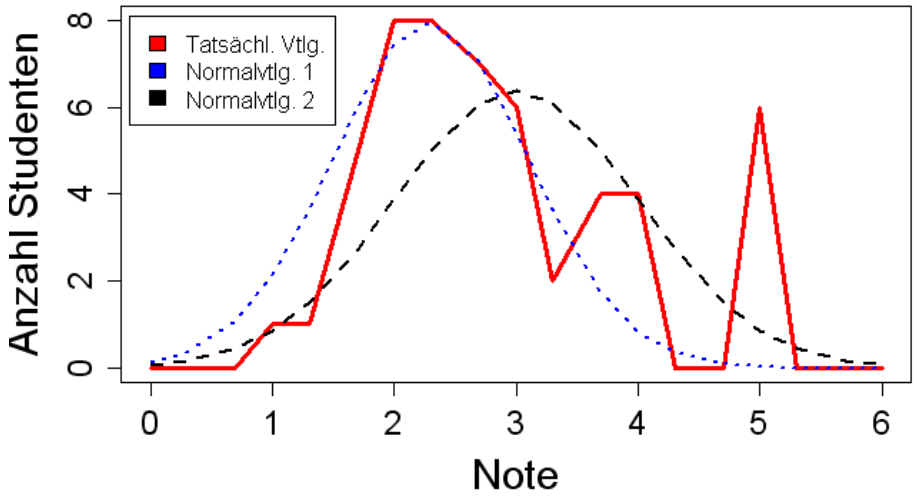
- Im Allgemeinen benötigt man anspruchsvollere Methoden um zu schätzen, wieviele Tupel sich in einem bestimmten Wertebereich befinden.
- Drei Grundsätzliche Arten von Schätzmethoden:
  - Parametrische Verteilungen
  - Histogramme
  - Stichproben

Beispiel: Schätzung der Verteilung der Noten der DBS II Klausur anhand des Ergebnisse von 2007:



Selektivität (cont.)

- Parametrische Verteilungen
  - Bestimme zu der vorhandenen Werteverteilung die Parameter einer Funktion so, dass die Verteilung möglichst gut angenähert wird.

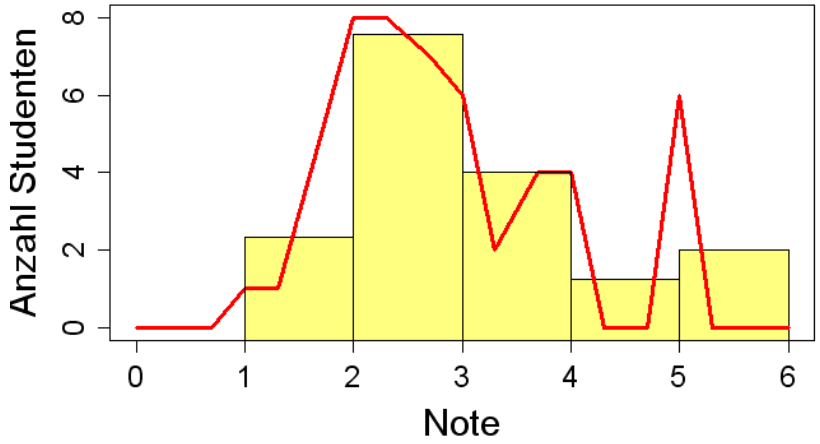


Probleme:

Wahl des Verteilungstyps (Normalverteilung, Exponentialverteilung…) und Wahl der Parameter, besonders bei mehrdimensionalen Anfragen (also z.B. bei Selektionen, die sich auf mehrere Attribute beziehen)

Selektivität (cont.)

- Histogramme
  - Unterteile den Wertebereich des Attributs in Intervalle und zähle die Tupel, die in ein bestimmtes Intervall fallen
    - *Equi-Width-Histograms:* Intervalle gleicher Breite
    - *Equi-Depth-Histograms:* Unterteilung so, dass in jedem Intervall gleich viele Tupel sind



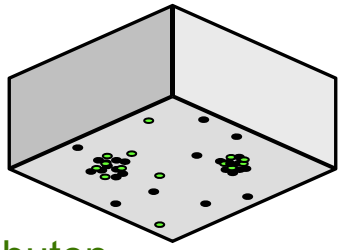
=> Flexible Annäherung an die Verteilung

Selektivität (cont.)

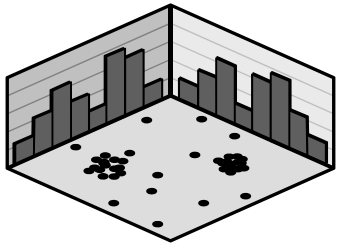
- Stichproben
  - Sehr einfaches Verfahren
  - Ziehe eine zufällige Menge von n Tupeln aus einer Relation, und betrachte deren Verteilung als repräsentativ für die gesamte Relation.
  - Problem der Größe des Stichprobenumfangs n:
    - n zu klein: Wenig repräsentative Stichprobe
    - n zu gross: Ziehen der Stichprobe erfordert zu viele „teure“ Zugriffe auf den Hintergrundspeicher

6.5 Kostenmodellbasierte Anfrageoptimierung

- Probleme bei Anfragen über mehrere Attribute (mehr-dimensionale Anfragen)
  - Sampling
    - ***Problem:*** Genauigkeit abhängig von der Samplegröße



  - 1D Histogramme
    - ***Problem:*** Annahme der Unabhängigkeit zwischen den Attributen
  - Mutli-D Histogramme
    - ***Problem:*** Anzahl der Gridzellen steigt exponentiell mit *d*



  - Parametrische Methoden
    - ***Problem:*** nur für max. 2-3 Attribute geeignet