



LUDWIG-
MAXIMILIANS-
UNIVERSITY
MUNICH


DEPARTMENT
INSTITUTE FOR
INFORMATICS


DATABASE
SYSTEMS
GROUP

Skript zur Vorlesung:

Datenbanksysteme II

Sommersemester 2016

Kapitel 6

Anfragebearbeitung

Vorlesung: PD Dr. Peer Kröger

http://www.dbs.ifi.lmu.de/cms/Datenbanksysteme_II

© Peer Kröger 2016

Dieses Skript basiert in Teilen auf den Skripten zur Vorlesung Datenbanksysteme II an der LMU München von

Prof. Dr. Christian Böhm (SoSe 2007),

PD Dr. Peer Kröger (SoSe 2008, 2013, 2014, 2015) und

PD Dr. Matthias Schubert (SoSe 2009)



6. Anfragebearbeitung

6.1 Einleitung

6.2 Indexstrukturen

6.3 Grundlagen der Anfrageoptimierung

6.4 Logische Anfrageoptimierung

6.5 Kostenmodellbasierte Anfrageoptimierung

6.6 Physische Anfrageoptimierung

6.1 Einleitung

6.2 Indexstrukturen

6.3 Grundlagen der Anfrageoptimierung

6.4 Logische Anfrageoptimierung

6.5 Kostenmodellbasierte Anfrageoptimierung

6.6 Physische Anfrageoptimierung

HW-Grundlagen

- Von-Neumann Rechner Architektur: Flaschenhalse

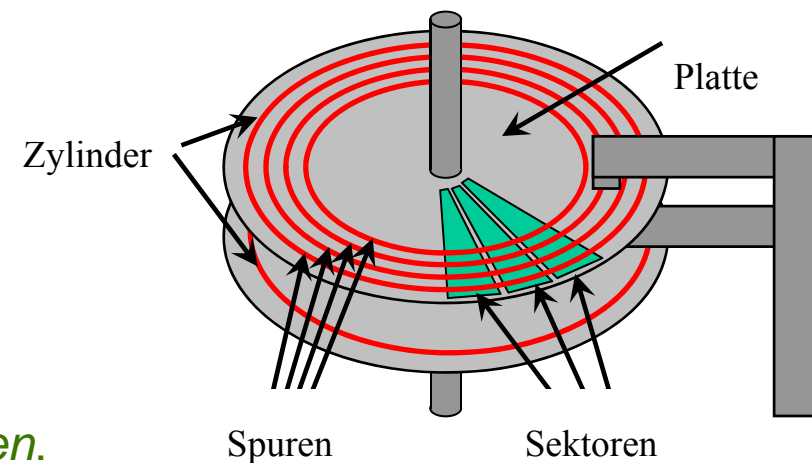


- Zur Vereinfachung unterscheidet man meist nur zwischen
 - **CPU-bound**
CPU, Arbeitsspeicher und Bus bilden den Hauptengpass
 - **I/O-bound**
Hintergrundspeicher und I/O bilden den Hauptengpass

HW-Grundlagen (cont.)

– Schematischer Aufbau einer Festplatte

- Ein Plattenspeichersystem besteht aus *Platten*
- Die Oberfläche der Platten besteht aus *Spuren*
- Die Spuren bestehen aus *Sektoren*.
- *Zylinder* = alle Spuren mit konstantem Radius
- Platten rotieren um gemeinsame Achse, der Arm ist in radialer Richtung bewegbar



HW-Grundlagen (cont.)

- Zugriff auf eine Seite:
 - Setze den Arm auf den gewünschten Zylinder (*Suchen*)
 - Warte bis die Platte so rotiert ist, dass sich der Anfang der Seite unter dem Arm befindet (*Latenz*)
 - Übertrage die Seite in den Hauptspeicher (*Transfer*)
- **Zugriffszeit = Suchzeit + Latenzzeit + Transferzeit**
- I/O-Rate = erwartete Anzahl von Zugriffen pro Sekunde
- Übertragungsrate = maximale Anzahl übertragener Bytes pro Sekunde (Bandbreite)

Speichermedien

- I/O-Engpass auch mit modernen Platten nicht überwindbar
- Lösungsansatz: Verwende statt einer großen Festplatte mehrere kleine, die parallel betrieben werden können

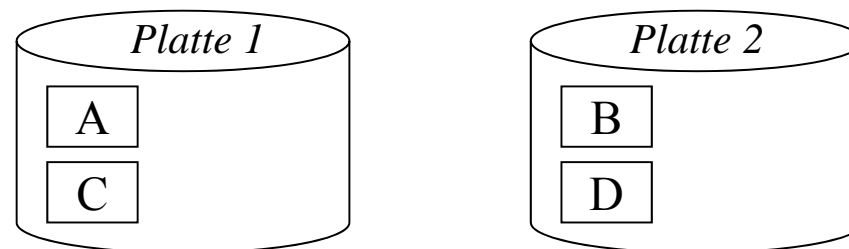
=> RAID-Systeme

- Die Komplexität wird durch den RAID-Controller nach Außen verborgen => es gibt nur ein (virtuelles) Laufwerk
- Acht verschiedene RAID-Level die unterschiedliche Zugriffsprofile optimieren

Speichermedien (cont.)

– RAID 0

- Datenmenge wird durch blockweise Rotation auf die Platten verteilt (**Striping**)
- Beispiel: Striping von 4 Blöcken (A,B,C,D) auf zwei Platten

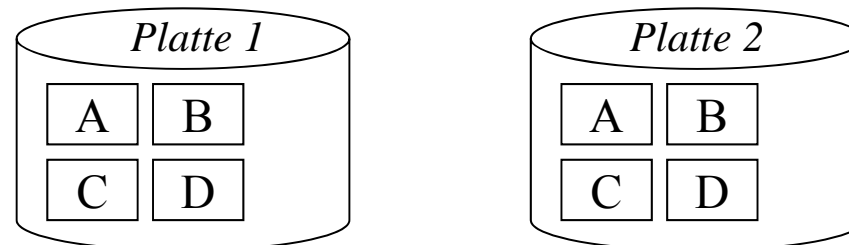


- Größtmögliche Beschleunigung: Anfrage an aufeinanderfolgende Blöcke kann parallel bearbeitet werden
- Fehleranfällig:
 - besteht eine Datei aus vielen Blöcken werden diese über die entsprechenden Platten verteilt
 - Ausfall einer Platte führt zur Beschädigung der Datei

Speichermedien (cont.)

– RAID 1

- Jedes Laufwerk besitzt eine Spiegelkopie (**Mirror**)
- Durch Redundanz ist Fehlerfall eines Laufwerks kein Problem
- Beispiel:

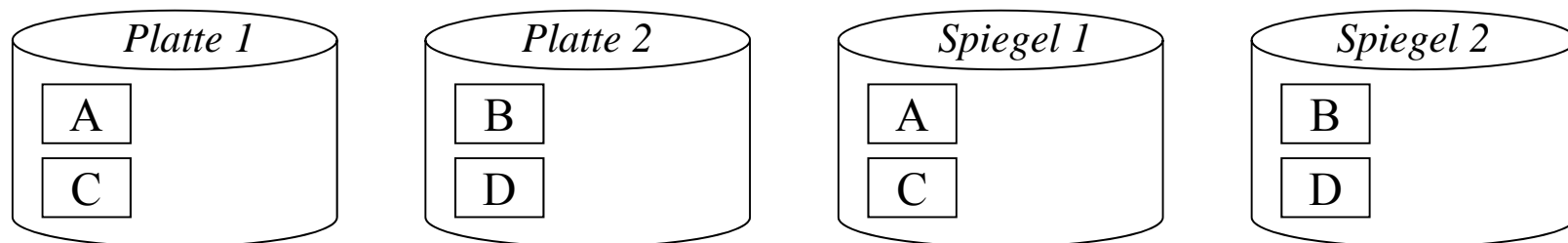


- Leseoperationen parallelisierbar (wie RAID 0)
- Schreiboperationen müssen auf beiden Mirrors (parallel) durchgeführt werden

Speichermedien (cont.)

– RAID 0+1

- Kombination aus RAID 0 und RAID 1
- Verteilung der Datenblöcke wie bei RAID 0
- Spiegelung der Platten wie bei RAID 1



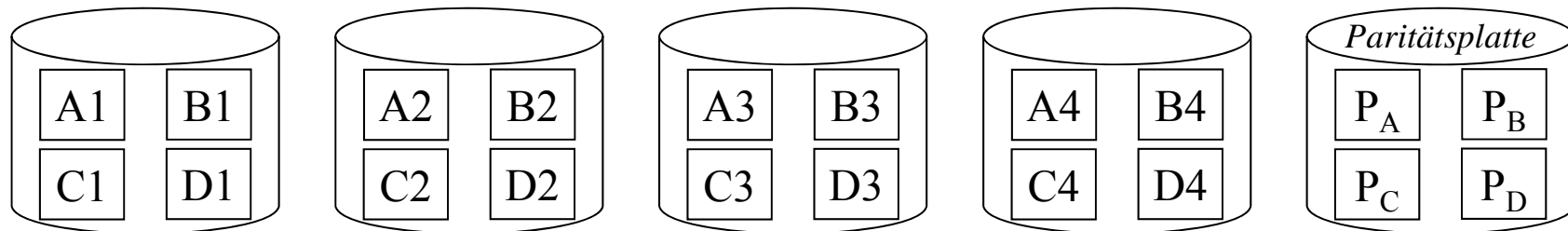
- Vereint Vorteile von RAID 0 und RAID 1
- ABER: Anzahl der benötigten Platten steigt!!!

Speichermedien (cont.)

- Ab RAID 2 wird Datensicherheit ökonomisch günstiger umgesetzt
- Hilfsmittel: Paritätsinformationen
 - Prüfsumme für mehrere Daten
 - Verwendbar, um Daten auf Korrektheit zu überprüfen
 - Verwendbar, um Daten im Fehlerfall zu rekonstruieren
 - Vorgehen:
 - Speichere zu N Datenbereichen auf unterschiedlichen Platten zusätzlich deren Prüfsumme auf einer anderen Platte
 - Ist einer der N Datenbereiche defekt kann dieser aus der Prüfsumme und den $N-1$ übrigen (intakten) Datenbereichen wiederhergestellt werden

Speichermedien (cont.)

- RAID 2
 - Striping auf Bitebene
 - Paritätsinformationen auf separaten Platten
 - In der Praxis meist nicht eingesetzt
- RAID 3 und RAID 4
 - Striping auf Bit- oder Byte-Ebene (RAID 3) bzw. blockweise (RAID 4)
 - Paritätsinformationen auf einer speziellen Platte

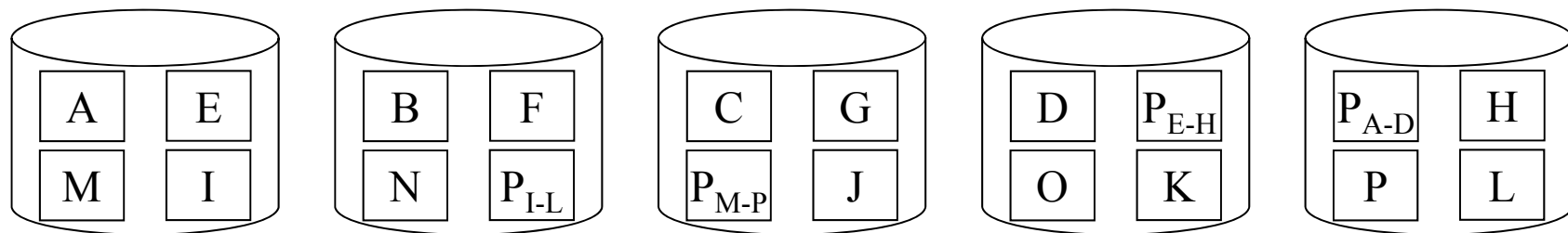


- Nachteil: jede Schreiboperation muss auf Paritätsplatte zugreifen

Speichermedien (cont.)

– RAID 5

- Striping blockweise (wie RAID 4)
- Verteilung der Paritätsinformationen auf alle Platten



- Damit ist der Flaschenhals der Paritätsplatte beseitigt
- Schreibeoperationen setzt aber nach wie vor die Neuberechnung und das Ablegen des neuen Paritätsblocks voraus
- ACHTUNG: Paritätsblock P_x kann nur ein Fehler in den Daten X korrigieren!

Speichermedien (cont.)

- Abschließende Bemerkungen
 - Wahl des RAID-Levels hängt vom Anwendungsprofil ab (z.B. Anteil der Leseoperationen im Vergleich zu Schreiboperationen)
 - Kommerzielle RAID-Systeme erlauben typischerweise eine flexible Konfiguration
 - Viele DBS unterstützen Striping von Tupeln auf unterschiedliche Platten auch ohne Einsatz von RAID-Systemen
 - Trotz der Fehlertoleranz von RAID-Systemen ist der Einsatz von Recovery-Techniken (Kapitel 4) wichtig

Anfragebearbeitung

- Zu bearbeitende Seiten müssen vom HGSP in den DB-Puffer geladen werden
- Problem: Verwaltung der Daten auf einem Speichermedium sequentiell
 - Zeitaufwand für Bearbeitung einer Suchanfrage: $O(n)$
(im ungünstigsten Fall alle n Datensätze durchsuchen)
- Wird ein bestimmter Datensatz anhand eines Suchkriteriums gesucht, kann über eine Indexstruktur eine aufwändige Suche vermieden werden
- Der Index erlaubt es, die Position des Datensatzes innerhalb des Mediums schnell zu bestimmen.

6. Anfragebearbeitung

6.1 Einleitung

6.2 Indexstrukturen

6.3 Grundlagen der Anfrageoptimierung

6.4 Logische Anfrageoptimierung

6.5 Kostenmodellbasierte Anfrageoptimierung

6.6 Physische Anfrageoptimierung

Indexstrukturen

- Um Anfragen und Operationen effizient durchführen zu können, setzt die interne Ebene des Datenbanksystems geeignete Datenstrukturen und Speicherungsverfahren (***Indexstrukturen***) ein.
- Aufgaben
 - Zuordnung eines ***Suchschlüssels*** zu denjenigen physischen Datensätzen, die diese Wertekombination besitzen, d.h. Zuordnung zu der oder den Seiten der Datei, in denen diese Datensätze gespeichert sind; z.B.:
(VW, Golf, schwarz, M-ÜÄ 40) => (logische) Seite 37
 - Organisation der Seiten unter dynamischen Bedingungen, d.h. bei Überlauf einer Seite: Aufteilen der Seite auf zwei Seiten

Klassen von Indexstrukturen

- Datenorganisierende Strukturen
 - Organisiere die Menge der tatsächlich auftretenden Daten (Suchbaumverfahren)
- Raumorganisierende Strukturen
 - Organisiere den Raum, in den die Daten eingebettet sind (dynamische Hash-Verfahren)
- Anwendungsgebiete:
 - Primärschlüsselsuche (B-Baum und lineares Hashing)
 - Sekundärschlüsselsuche (invertierte Listen)
- ***Hier: Fokus auf Suchbaumverfahren***

Was müssen Indexstrukturen können?

– **Effizientes Suchen:**

- Häufigste Operation in einem DBS: Suchanfragen.
- Insbesondere Suchoperationen müssen mit wenig Seitenzugriffen auskommen.
- Beispiel: **unsortierte sequentielle Datei**
 - Einfügen und Löschen von Datensätzen werden effizient durchgeführt.
 - Suchanfragen müssen ggf. die gesamte Datei durchsuchen.
- Eine Anfrage sollte daher mit Hilfe der Indexstruktur möglichst schnell zu der Seite oder den Seiten geführt werden, auf denen sich die gesuchten Datensätze befinden.
- Bei Suchbäumen: **binäre Suche** in einer sortierten Datei

Was müssen Indexstrukturen können?

- ***Dynamisches Einfügen, Löschen und Verändern von Datensätzen***
 - Der Datenbestand einer Datenbank verändert sich im Laufe der Zeit.
 - Verfahren, die zum Einfügen oder Löschen von Datensätzen eine Reorganisation der gesamten Datei erfordern, sind nicht akzeptabel.
 - Beispiel: ***sortierte sequentielle Datei***
 - Das Einfügen eines Datensatzes erfordert im schlechtesten Fall, dass alle Datensätze um eine Position verschoben werden müssen.
 - Folge: auf alle Seiten der Datei muss zugegriffen werden.
 - Das Einfügen, Löschen und Verändern von Datensätzen darf daher nur lokale Änderungen bewirken.

Was müssen Indexstrukturen können?

– *Ordnungserhaltung*

- Datensätze, die in ihrer Sortierordnung direkt aufeinander folgen, werden oft gemeinsam angefragt.
- In der Ordnung aufeinander folgende Datensätze sollten in der gleichen Seite oder in benachbarten Seiten gespeichert werden.

– *Hohe Speicherplatzausnutzung*

- Dateien können sehr groß werden.
- Eine möglichst hohe Speicherplatzausnutzung ist wichtig:
 - Möglichst geringer Speicherplatzverbrauch.
 - Im Durchschnitt befinden sich mehr Datensätze in einer Seite, wodurch auch die Effizienz des Suchens steigt und die Ordnungserhaltung an Bedeutung gewinnt.

Aufbau: Abbildung des Suchbaums auf die Festplatte

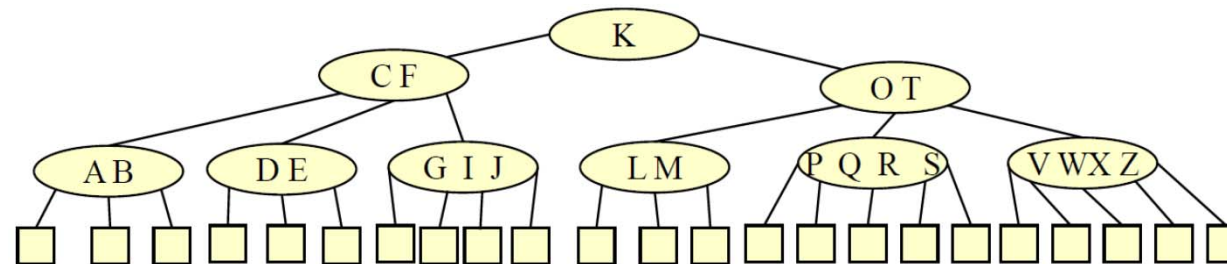
- 1. Idee: bilde jeden Knoten des Suchbaums auf einen Block auf der Platte ab
 - Problem: pro Knoten, der Erreicht wird, entsteht ein Seitenzugriff auf der Platte

=> Hohe Kosten
- 2. Idee: fasse mehrere Knoten zu einem zusammen und bilde die zusammengefassten Knoten auf einen Block ab
 - Wie viele Knoten zusammen gefasst werden können, hängt von Größe der Blöcke und Speicherbedarf der Schlüssel ab
 - Prominenteste Instanz: B-Baum (der Ordnung m)

B-Baum der Ordnung m

- Jeder Knoten enthält höchstens $2m$ Schlüssel.
- Jeder Knoten außer der Wurzel enthält mindestens m Schlüssel.
- Die Wurzel enthält mindestens einen Schlüssel.
- Ein Knoten mit k Schlüsseln hat genau $k+1$ Söhne.
- Alle Blätter befinden sich auf demselben Level.

Beispiel ($m=2$):



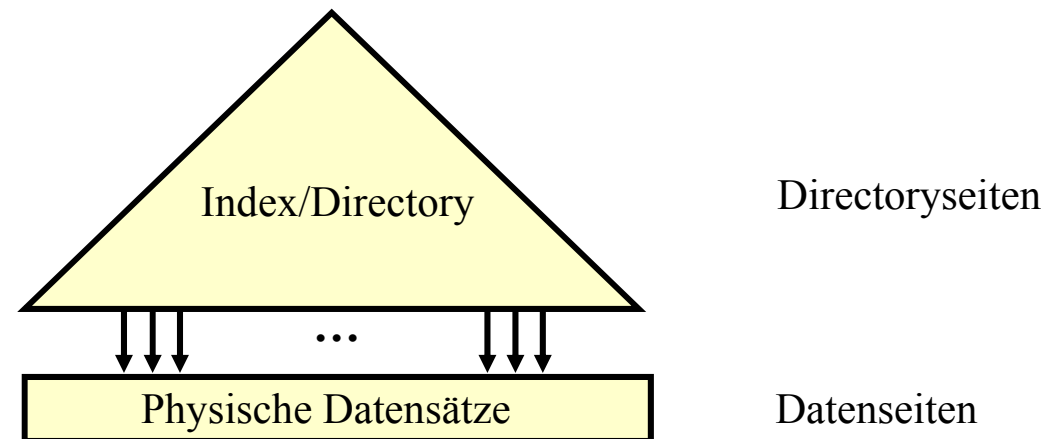
- In realen Implementierungen typ. 600-900 Schlüssel pro Seite
- Im Hauptspeicher binäre Suche innerhalb der Knoten

B-Baum: Diskussion

- Suche nach einem Schlüssel ist auf einen Pfad im Baum beschränkt
- max. Anzahl von Blockzugriffen: $O(\log n)$ für n Datensätze
- **Aber:** Zugriff auf einzelne Blöcke typischerweise *nicht sequentiell* sondern in *zufällig* Reihenfolge (WARUM?)
- Häufig tritt in Datenbankanwendungen neben der Primärschlüsselsuche aber auch sequentielle Verarbeitung auf:
 - Sortiertes Auslesen aller Datensätze
 - Bereichsanfragen: “Nenne mir alle Kunden, deren Nachname im Bereich [De ... Ku] liegt.”
- Die Indexstruktur sollte die sequentielle Verarbeitung unterstützen, d.h. die Verarbeitung der Datensätze in aufsteigender Reihenfolge ihrer Primärschlüssel

B+-Baum

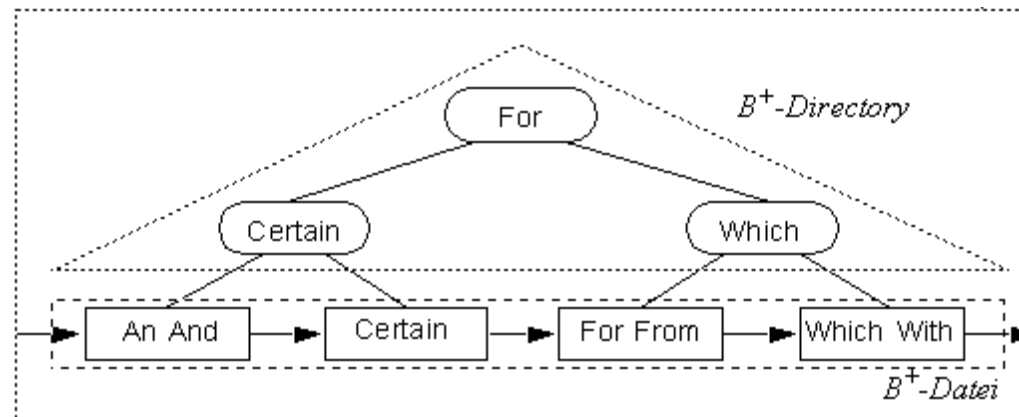
- Trennung der Indexstruktur in Directory- und Datenseiten:
 - Die eigentlichen physischen Datensätze werden in den **Datenseiten** gespeichert (Blattknoten des zugrundeliegenden Suchbaums).
 - Die **Directoryseiten** sind die inneren Knoten des Suchbaums und speichern Einträge, die aus aggregierten Zugriffsinformationen bestehen und die Navigation im Suchbaum ermöglichen (Separatoren).
- **Sequentielle Verkettung** der Datenseiten auf der Platte



B+-Baum

– Beispiel:

B⁺-Baum für die Zeichenketten: *An, And, Certain, For, From, Which, With*



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 Index vorgegeben (und typischerweise wesentlich kleiner, z.B. 4-8 KBytes).

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

6.1 Einleitung

6.2 Indexstrukturen

6.3 Grundlagen der Anfrageoptimierung

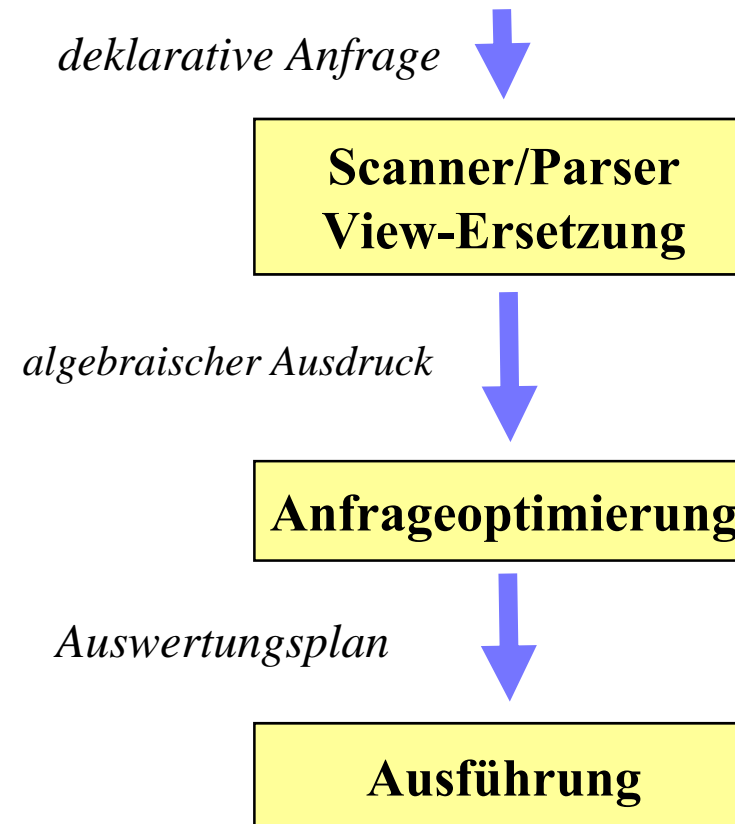
6.4 Logische Anfrageoptimierung

6.5 Kostenmodellbasierte Anfrageoptimierung

6.6 Physische Anfrageoptimierung

Aufgabe der Anfragebearbeitung

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



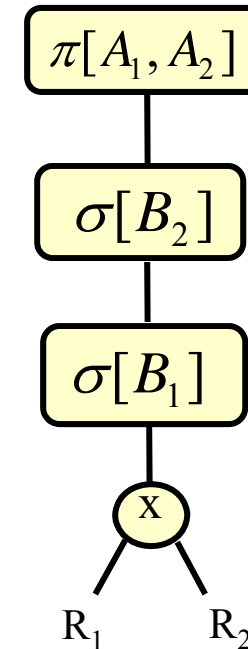
Kanonischer Auswertungsplan

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

6.1 Einleitung

6.2 Indexstrukturen

6.3 Grundlagen der Anfrageoptimierung

6.4 Logische Anfrageoptimierung

6.5 Kostenmodellbasierte Anfrageoptimierung

6.6 Physische Anfrageoptimierung

Äquivalenzregeln der Relationalen Algebra

- Join, Vereinigung, Schnitt und Kreuzprodukt sind kommutativ

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

- 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

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

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

$$\sigma_{B_1 \wedge B_2 \wedge \dots \wedge B_n}(R) = \sigma_{B_1}(\sigma_{B_2}(\dots(\sigma_{B_n}(R))\dots))$$

- Geschachtelte Projektionen können eliminiert werden

$$\pi_{A_1}(\pi_{A_2}(\dots(\pi_{A_n}(R))\dots)) = \pi_{A_1}(R)$$

Damit eine solche Schachtelung sinnvoll ist, muss gelten: $A_1 \subseteq A_2 \subseteq \dots \subseteq A_n$

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

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

Äquivalenzregeln der Relationalen Algebra (cont.)

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

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

, falls $\text{attr}(B) \subseteq \text{attr}(R)$

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

- 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 $A1 = \text{attr}(R) \cap (A \cup \text{attr}(B))$
 $A2 = \text{attr}(S) \cap (A \cup \text{attr}(B))$

- Selektionen können mit Vereinigung, Schnitt und Differenz vertauscht werden

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

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. $\neg(B_1 \wedge B_2) = \neg B_1 \vee \neg B_2$

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

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

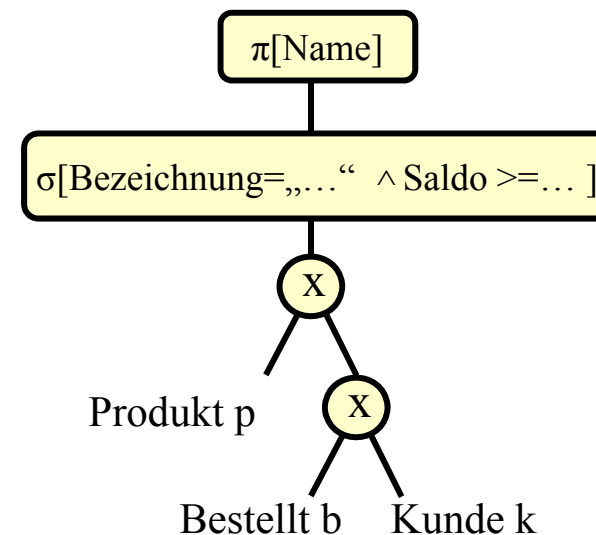
Beispiel (cont.)

– SQL Anfrage:

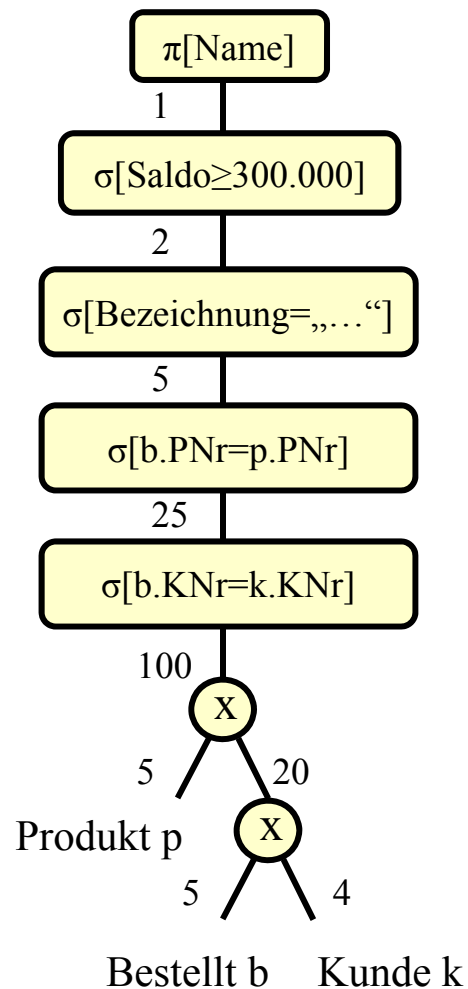
```

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
    
```

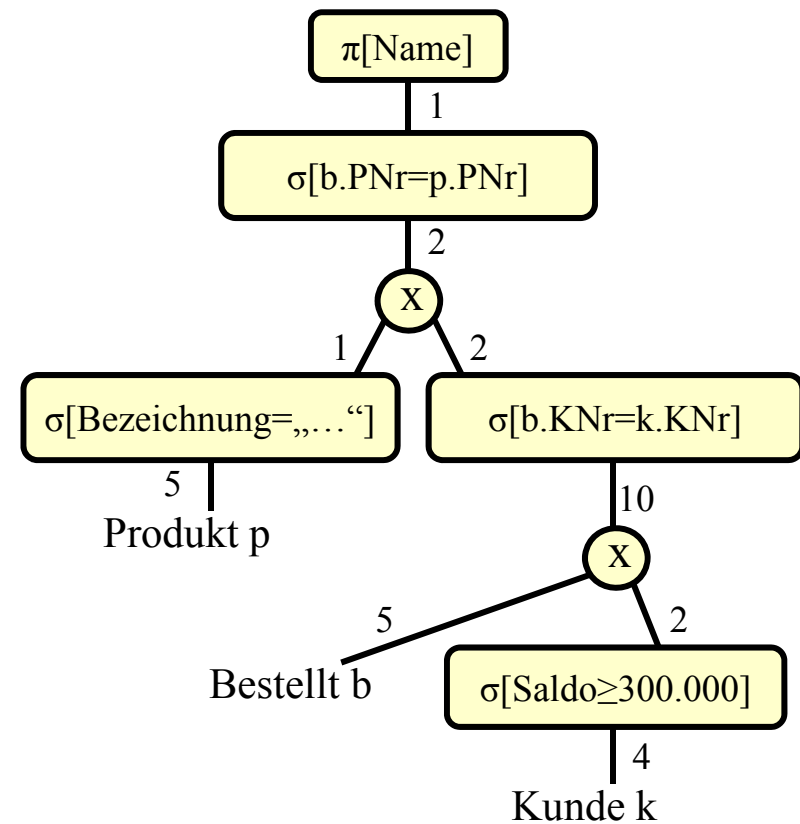
– Kanonischer Auswertungsplan:



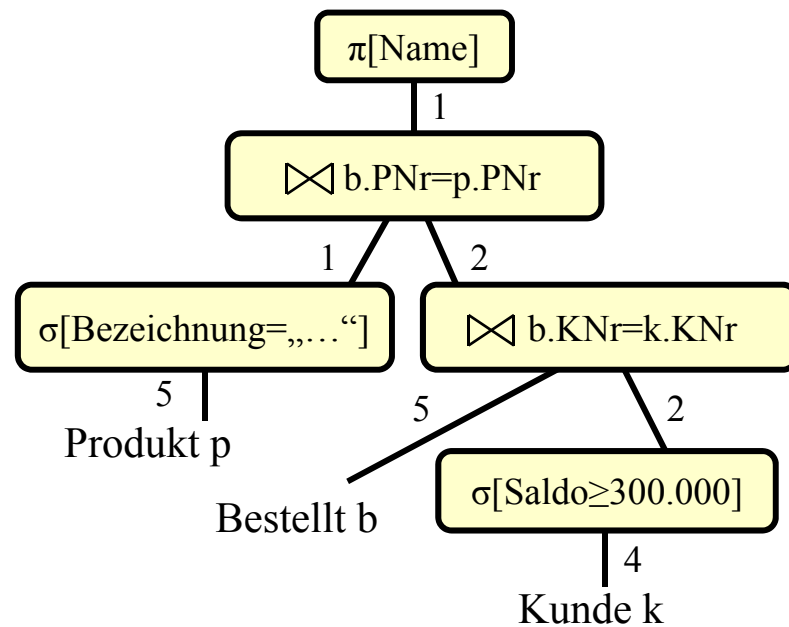
- Aufbrechen der Selektionen



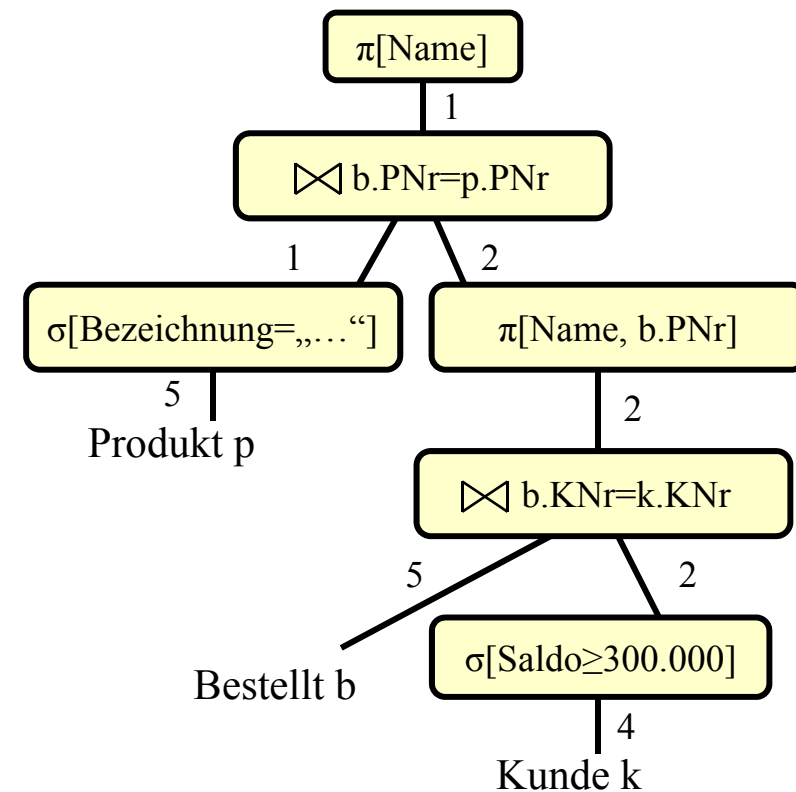
- Verschieben der Selektionen



- Zusammenfassen zu Joins



- Einfügen zusätzlicher Projektionen



6. Anfragebearbeitung

6.1 Einleitung

6.2 Indexstrukturen

6.3 Grundlagen der Anfrageoptimierung

6.4 Logische Anfrageoptimierung

6.5 **Kostenmodellbasierte Anfrageoptimierung**

6.6 Physische Anfrageoptimierung

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)

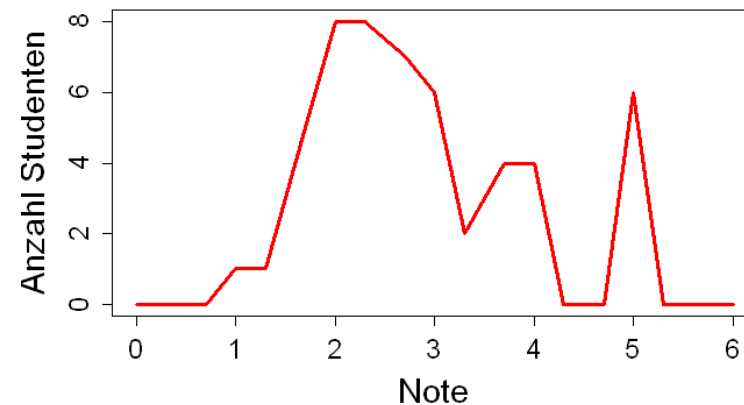
Selektivität (cont.)

- 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 $sel=1 / l$ (l 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 $sel_{RS} = 1/|R|$
 - 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.)

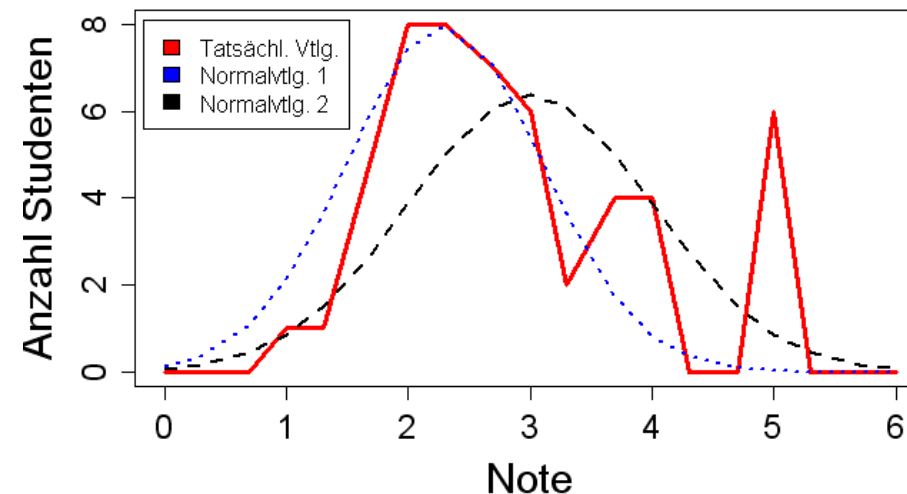
- Im Allgemeinen benötigt man anspruchsvollere Methoden um zu schätzen, wieviele Tupel sich in einem bestimmten Wertebereich befinden.
- Drei Grundsätzliche Arten von Schätzmethoden:
 - Parametrische Verteilungen
 - Histogramme
 - Stichproben

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.



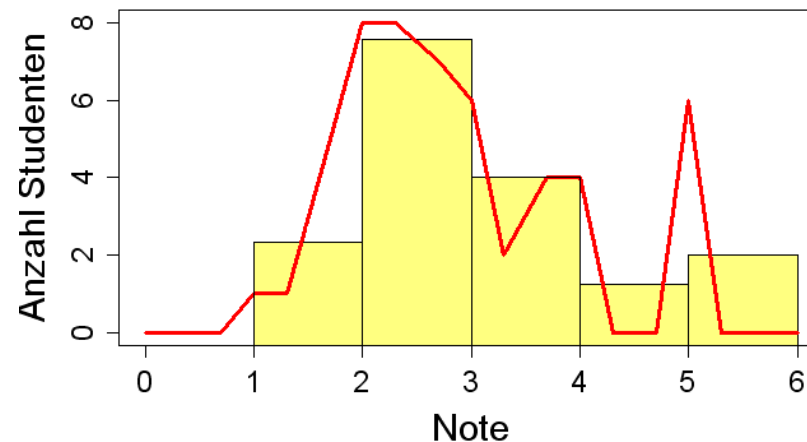
Probleme:

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

- 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

