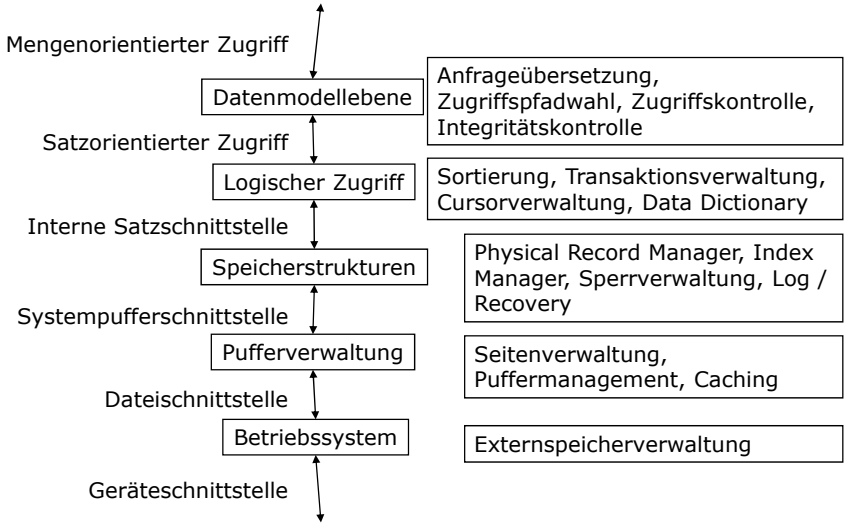
Kurzüberblick aus erster VL

1. Einführung
2. Physische Speicherstrukturen (2 → 2)
3. Physische Repräsentation von Daten (1 → 1)
4. Indexstrukturen (5 → 5)
5. Anfrageausführung (2 → 2)
6. Optimierung (3 → 4)
7. Benchmarking (1 → 1)
8. Recovery (2 → 2)
9. Transaktionsmanagement (3 → 2)

Einführung: Architektur

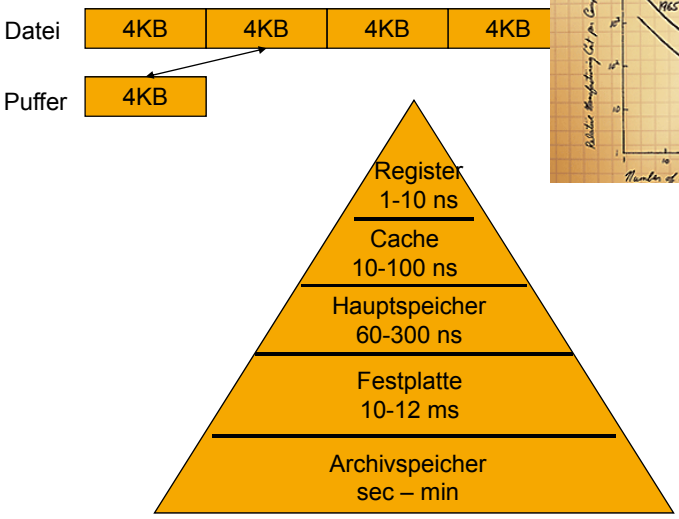


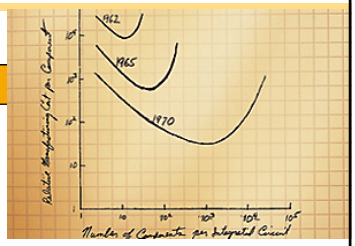
Einführung: Das Megatron 2002 DBMS

- Naive Sicht
  - DBMS sollte recht einfach sein
- Megatron 2002
  - Unter UNIX verfügbar
  - Relationales Datenmodell
  - SQL Anfragen

Physische Speicherstrukturen: Speicherhierarchie

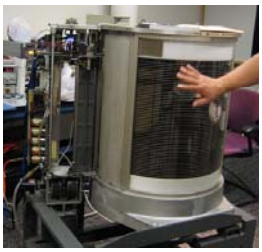
5

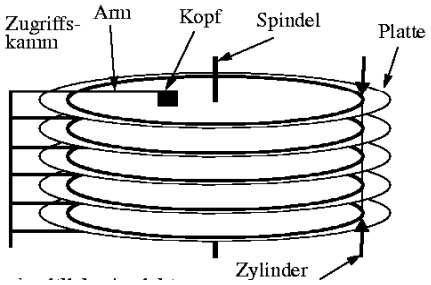


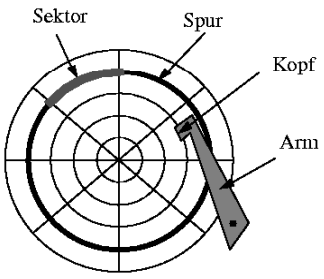


Physische Speicherstrukturen: Disks

6







Latenzzeit
1. Kommunikationszeit
2. Seektime
3. Rotationslatenzzeit
4. Transferzeit

Physische Speicherstrukturen: TPMMS

- Phase 1
  - Sortiere Teilstücke, die in Hauptspeicher passen.
  - => Viele sortierte Teillisten
- Phase 2
  - Mische sortierte Teilliste in einzige große Liste.
- Berechnungsgrenzen

Physische Speicherstrukturen: Zugriffsbeschleunigung

- Zwei Arten von Anwendungen
  1. Viele Blöcke werden in bekannter Folge gelesen oder geschrieben. Nur ein Prozess. (TPMMS, Phase 1)
  2. Viele kleine, parallele, unvorhersagbare Prozesse (ähnlich TPMMS, Phase 2)
- Fünf Tricks
  - Zylinderorganisation
  - Mehrere Disks
  - Spiegelung
  - Scheduling mit Elevator Algorithmus
  - Prefetching

Physische Speicherstrukturen: RAID

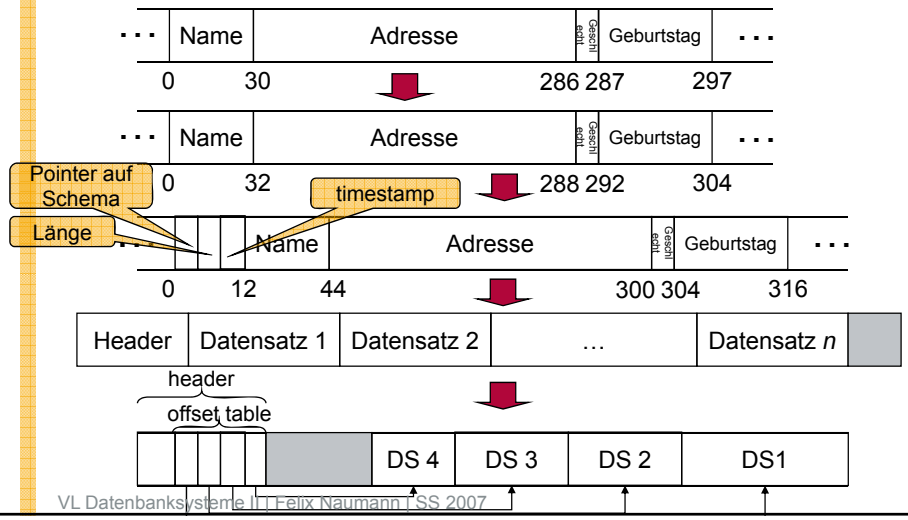
9

	Block-Striping	Bit-Striping	Kopie	Parität	Parität, dedizierte Disk	Verteilte Parität	Erkennen mehrerer Fehler
RAID 0	X						
RAID 1			X				
RAID 0+1	X		X				
RAID 2		X					
RAID 3		X		X	X		
RAID 4	X			X	X		
RAID 5	X			X		X	
RAID 6	X			X			X

Physische Repräsentation von Daten

10

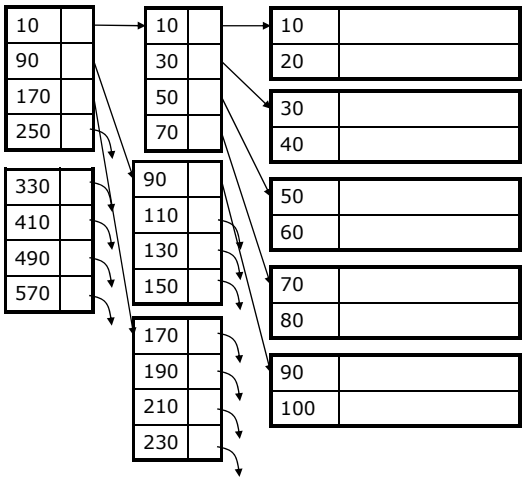
- CREATE TABLE Schauspieler (Name CHAR(30), Adresse VARCHAR(255), Geschlecht CHAR(1), Geburtstag DATE);



Indexstrukturen: Indizes auf sequenziellen Dateien

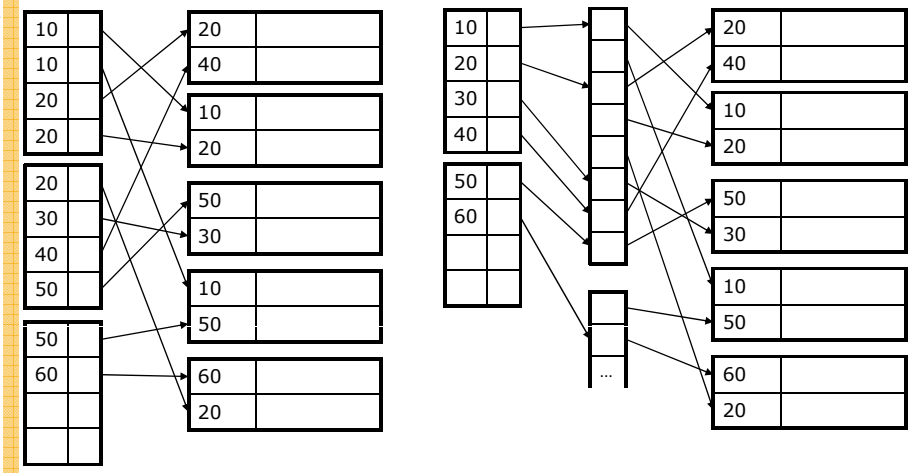
11

- Sequentielle Datei
- Dicht-besetzte Indizes
- Dünn-besetzte Indizes
- Mehrstufige Indizes
- Indizes für Nicht-eindeutige Suchschlüssel
  - 4 Varianten
- Änderungsoperationen
- Tombstones



Indexstrukturen: Sekundärindizes

12



- Indirektion
- Invertierter Index

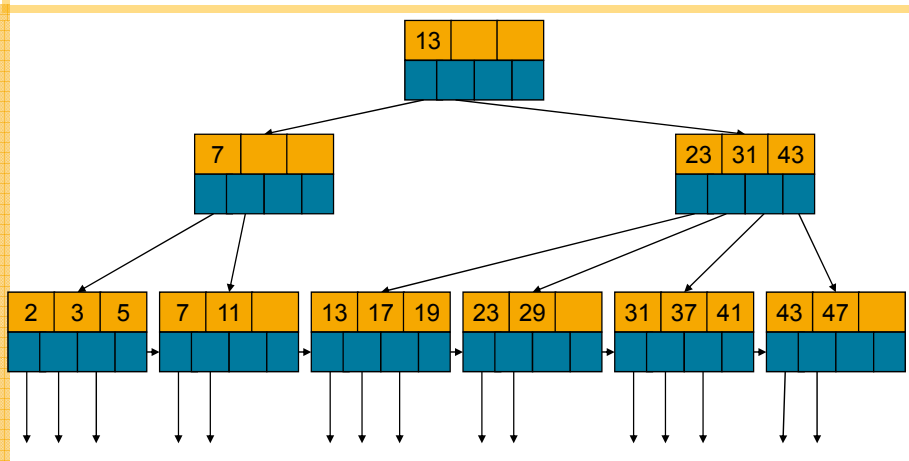
Indexstrukturen: B⁺-Bäume

13

- Schlüssel in Blätter sind Schlüssel aus den Daten
  - Sortiert über alle Blätter verteilt (von links nach rechts)
- **Wurzel:** mindestens zwei verwendete Pointer.
- **Blätter:** Der letzte Pointer zeigt auf das nächste Blatt (rechts)
  - Von den übrigen n Pointern werden mindestens $\lfloor (n+1)/2 \rfloor$ verwendet.
  - Zeigen auf Datenblöcke
- **Innere Knoten:** Pointer zeigen auf B-Baum Blöcke darunterliegender Ebenen
  - Mindestens $\lceil (n+1)/2 \rceil$ sind verwendet
  - Falls j Pointer verwendet werden, gibt es j-1 Schlüssel in dem Block
    - $K_1, \ldots, K_{j-1}$
  - Erster Pointer zeigt auf Teilbaum mit Schlüsselwerten < K_1.
  - Zweiter Pointer auf Teilbaum mit Schlüsselwerten zwischen K_1 und K_2...

Indexstrukturen: B⁺-Bäume

14

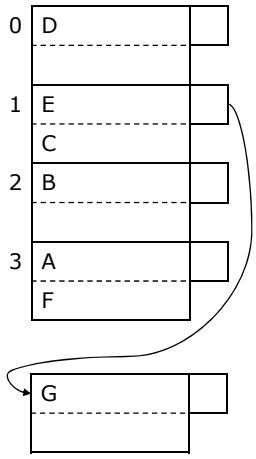


Suchen, Einfügen, Löschen im B-Baum

Sekundärschlüssel, Range Queries, B-Baum Varianten

Indexstrukturen: Hashtabellen

15



- Erweiterbare Hashtabellen
  - Berechnet (zu) großes Bitarray (k bit; z.B. $k = 32$)
  - Bucketarray verwendet nur die ersten i Bits ($i \leq k$)
  - => 2^i buckets
- Lineare Hashtabellen
  - $\log_2 n$ Bits zur Identifizierung der Buckets
    - Wähle die jeweils letzten Bits des Hashwerts

Indexstrukturen: Lineare Hashtabellen

16

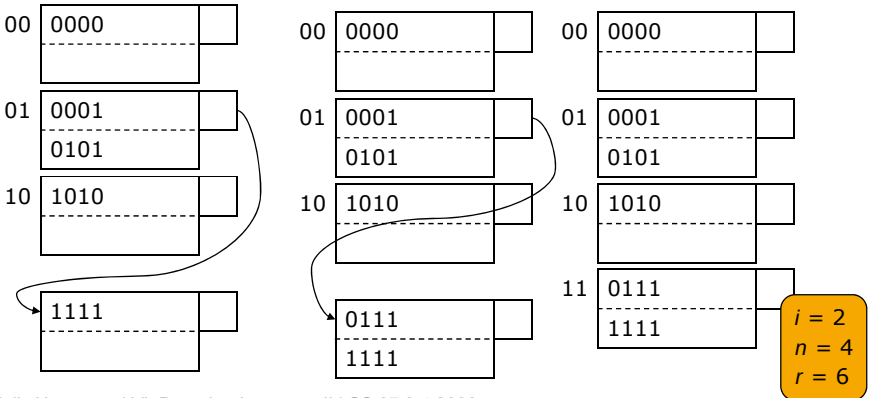
Füge ein: 0111

$i = 2$
$n = 3$
$r = 5$

Bucket 11 existiert noch nicht.
Deshalb wähle Bucket 01.

$i = 2$
$n = 3$
$r = 6$

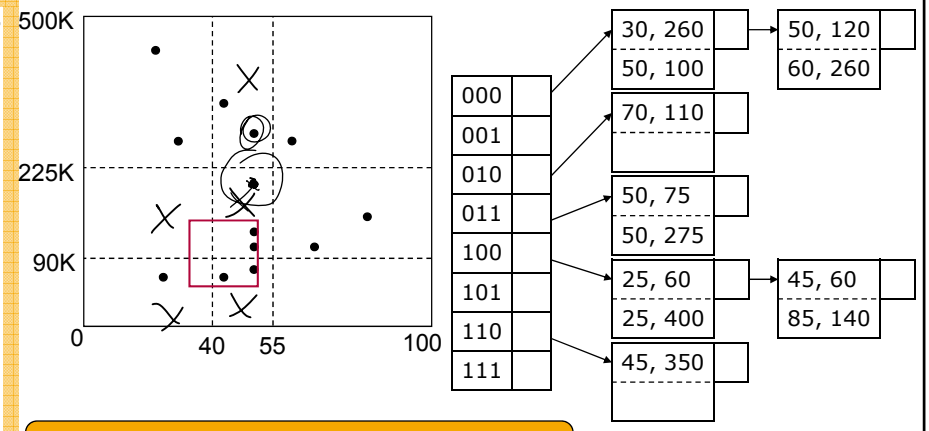
Problem: 6/3 > 1,7
=> Neuer Block 11
=> Split Block 01



$i = 2$
$n = 4$
$r = 6$

Indexstrukturen: Multidimensionale Indizes

17



Data Cubes, GIS, Bereichsanfragen in mehreren Dimensionen, Nearest Neighbor Anfragen

Grid files, Partitioniertes Hashing

1 Bit für Alter: MOD(Alter/2)
2 Bits für Gehalt: MOD(Gehalt/4)

Anfrageausführung: Physische Operatoren

18

- Table-scan, Index-scan, Sort-scan
- Kostenparameter: B, T, V(R,a)
- Iteratorkonzept
- Pipelining vs. Blockierend
- One-Pass Algorithmen
  - Daten nur einmal von Disk lesen
  - Mindestens ein Argument passt in Hauptspeicher
    - außer Selektion und Projektion
- Two-Pass Algorithmen
  - Meist einmal lesen, einmal schreiben, nochmal lesen
  - TPMMS
  - Gewisse Größenbeschränkung auf Input
- Multipass Algorithmen
  - Unbeschränkt in Inputgröße
  - Rekursive Erweiterungen von Two-Pass

Anfrageausführung: One-pass und NLJ

Operator	Nötiger Hauptspeicher M	I/O	Algorithmus
σ, π	1	B	1-pass, tupel-basiert
γ, δ	$\approx$ B	B	1-pass, relationen-basiert
$\cup$, $\cap$, $-$, $\times$, $\bowtie$	min(B(S), B(R))	B(R) + B(S)	1-pass, relationen-basiert binär
$\bowtie$	M $\geq$ 2	B(R)B(S)/M	Block-basierter NLJ

- Nested Loop Join
  - Tupel-basiert, Block-basiert
  - Innere und äußere Schleife
  - Kostenschätzung

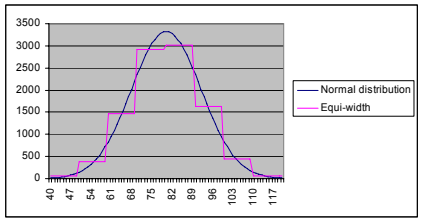
Anfrageausführung: Two-pass und Multi-pass

- Zwei Phasen
  - Einlesen der Daten
  - Verarbeitung der Daten (hier: Sortierung von Teillisten)
  - Schreiben der Daten
  - Wiedereinlesen der Daten (hier: Merging der Teillisten)
    - Hier unterscheiden sich die Algorithmen
- Sort-basierte Two-Pass Algorithmen
  - Meist TPMMS als Basis
  - Duplikateliminierung, Gruppierung und Aggregation, Mengenoperationen, Join, Sort-merge-Join
- Hash-basierte Two-Pass Algorithmen
  - Tupel, die gemeinsam betrachtet werden müssen, erhalten gleichen Hashwert; landen also in einem Bucket
  - Hybrid Hash Join – Idee
- Index-basierte Algorithmen
  - Selektion und Joins

Optimierung

21

- I/O Kostenschätzung für Operationen
  - Von Tupel-Kardinalitäten zu I/O-Kosten
  - Insbesondere Join-Kosten
    - $T(R \bowtie S) = T(R) \cdot T(S) / \max[V(R,Y1),V(S,Y1)]\max[V(R,Y2),V(S,Y2)]$
- Histogramme
- Enumerierung physischer Anfragepläne



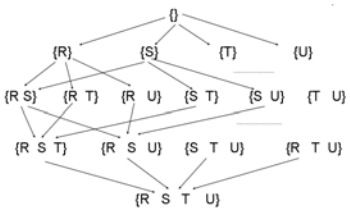
Optimierung: Dynamische Programmierung

22

	{R,S}	{R,T}	{R,U}	{S,T}	{S,U}	{T,U}
Kardinalität	5000	1M	10000	2000	1M	1000
Kosten	0	0	0	0	0	0
opt. Plan	R ⋈ S	R ⋈ T	R ⋈ U	S ⋈ T	S ⋈ U	T ⋈ U

	{R,S,T}	{R,S,U}	{R,T,U}	{S,T,U}
Kardinalität	10000	50000	10000	2000
Kosten	2000	5000	1000	1000
opt. Plan	(S⋈ T)⋈ R	(R⋈ S)⋈ U	(T⋈ U)⋈ R	(T⋈ U)⋈ S

Interesting Orders



Benchmarking

- Anforderungen
  - Relevanz zum Anwendungsgebiet
  - Portabilität
  - Skalierbarkeit
  - Einfachheit
- Komponenten
  - Schema
  - Daten bzw. Datengenerator
  - Workload
    - Anfragen und Häufigkeiten
  - Metrik

Recovery

- Fehlerarten (etwas feiner)
  - Fehlerhaften Dateneingabe
  - Medienfehler
  - Katastrophe
  - Systemfehler
- Transaktionen
  - **`INPUT(X), READ(X, t), WRITE(X, t), OUTPUT(X)`**
- Allgemein: Log Datei, Flush-Log, Ablauf
- Undo Logging / Redo Logging / Undo-Redo-Logging
  - Regeln
  - Vorteile / Nachteile
  - Recovery
  - Checkpointing
    - Blockierend / Nicht-blockierend
- Archivierung / Dumps

Recovery

25

1. <START T1>
2. <T1, A, 4, 5>
3. <START T2>
4. <COMMIT T1>
5. <T2, B, 9, 10>
6. <START CKPT (T2)>
7. <T2, C, 14, 15>
8. <START T3>
9. <T3, D, 19, 20>
10. <END CKPT>
11. <COMMIT T2>

CRASH

12. <COMMIT T3>

- Beispielfragen
  - Führen Sie ein Recovery durch.
  - Setzen Sie einen Checkpoint
  - Unterschied Undo/Redo

Transaktionsmanagement: Schedules

26

- Basics zum Transaktionsmanagement
  - ACID, Schedules, Locking, Rollback
- Rücksetzbare Schedules
  - Ein Schedule heißt „rücksetzbar", falls jede Transaktion erst committed nachdem alle Transaktionen committed haben, von denen sie gelesen hat.
- ACR Schedules
  - Ein Schedule ist ACR, falls Transaktionen nur Werte von committed Transaktionen lesen.
- Strikte Schedules
  - Eine Transaktion gibt keine xl Sperren frei, bis sie committed oder aborted und entsprechendes COMMIT oder ABORT im Log auf Disk geschrieben wurde.
- Aufgaben
  - Gegeben Schedule, welche Eigenschaften hat er?
  - Gegeben Transaktionen, entwerfen Sie einen Schedule, der ACR, aber nicht strikt ist.

Transaktionsmanagement: Sicht-Serialisierbarkeit

- Hypothetische Transaktionen T_0 und T_f
- Quelle einer Lese-Aktion
- Wenn für S_1 und S_2 für jede Lese-Aktion die Quelle gleich ist, sind S_1 und S_2 Sicht-äquivalent.
- Wenn Schedule S Sicht-äquivalent zu einem seriellen Schedule ist, ist S Sicht-serialisierbar.
- Aufgaben
  - Gegeben Schedule, ist er sicht-äquivalent?
    - Graphbasierter Test
  - Gegeben Tas, entwerfen Sie einen Schedule der Sicht- aber nicht Konflikt-serialisierbar ist
    - Oder umgekehrt

Organisatorisches

- Lehrevaluation
- Notsprechstunde
  - Jederzeit, wenn Tür offen ist
- Klausur: HS 1
  - Mittwoch 9:45 Uhr
    - Beginn 10:00 Uhr
    - Ende: 11:30 Uhr
  - Same procedure as last semester

Viel Erfolg!