Vorlesung

Datenbank-Implementierungstechniken

TU Ilmenau, SomSem 2003

Kai-Uwe Sattler

kus@iti.cs.uni-magdeburg.de

Überblick

1. Aufgaben und Prinzipien von Datenbanksystemen
2. Architektur von Datenbanksystemen
3. Verwaltung des Hintergrundspeichers
4. Dateiorganisation und Zugriffsstrukturen
5. Zugriffsstrukturen für spezielle Anwendungen
6. Basisalgorithmen für Datenbankoperationen
7. Optimierung von Anfragen
8. Transaktionsmodelle und -verwaltung
9. Wiederherstellung und Datensicherheit
10. Neuere Entwicklungen und Ausblick

Nötiges Vorwissen

Datenbanken I:

- Grundprinzipien Datenbanksysteme
- Tabellen, Attribute, Schlüssel
- Relationale Algebra und SQL

Wird am Anfang der Vorlesung kurz wiederholt!

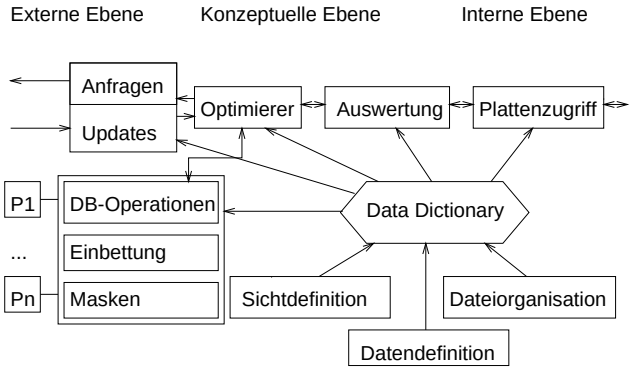
Literatur

- Saake, G.; Heuer, A.: *Datenbanken — Implementierungskonzepte.* mitp-Verlag, Mai 1999
- Härder, T.; Rahm, E.: *Datenbanksysteme — Konzepte und Techniken der Implementierung*. Springer-Verlag, 2001
- Garcia-Molina, H.; Ullman, J.; Widom, J.: *Database System Implementation*. Addison-Wesley, 1999.
- Silberschatz, A.; Korth, H. F.; Sudarshan, S.: *Database System Concepts*. Wiley & Sons, 2001.

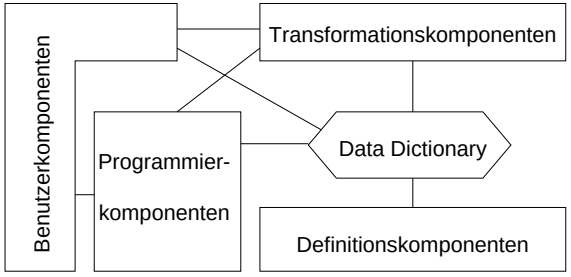
1. Aufgaben und Prinzipien von DBS

- Wiederholung Datenbankgrundbegriffe
- Überblick über behandelte Komponenten

Datenbankgrundbegriffe: Komponenten



Klassifikation von Komponenten



Neun Funktionen nach Codd

1. Integration
2. Operationen
3. *Katalog*
4. *Benutzersichten*
5. Konsistenzüberwachung
6. Datenschutz
7. *Transaktionen*
8. *Synchronisation*
9. *Datensicherung*

Datenmodelle und Datendefinition

Wichtigste Modelle in kommerziellen Systemen

- das *hierarchische Datenmodell*: Daten in Baumform als hierarchisch strukturierte Datensätze,
- das *Netzwerkmodell*: Unterstützung von Netzwerken von verzeigerten Datensätzen,
- das *relationale Datenbankmodell*: Daten in Tabellenform,
- das *objektorientierte Datenmodell*: modelliert Daten objektorientiert durch in Klassen organisierte, verzeigerte Objekte,
- das *semistrukturierte Datenmodell*: Verwaltung „schemaloser“, selbstbeschreibender Daten in Graphstrukturen (XML).

Relationale Datenbanken

Ausleih

InventarNr	Name
4711	Meyer
1201	Schulz
0007	Müller
4712	Meyer

Buch

InventarNr	Titel	ISBN	AUTOR
0007	Dr. No	3-125	James Bond
1201	Objektbanken	3-111	Heuer
4711	Datenbanken	3-765	Vossen
4712	Datenbanken	3-891	Ullman
4717	Pascal	3-999	Wirth

SQL-DDL

```
create table Buch (
   ISBN  char(10),
   Titel  varchar(200),
   Verlagsname  varchar(30),
   primary key (ISBN),
   foreign key (Verlagsname)
         references Verlage (Verlagsname) )
```


Anfragen

Grundlagen

- Relationenalgebra sowie
- Tupel- oder Bereichskalkül.

Relationenalgebra

$$\sigma_{\texttt{Name}=\texttt{'Meyer'}}(r(\texttt{Ausleih}))$$

$$\pi_{\texttt{Titel}}(r(\texttt{Buch}))$$

$$\pi_{\texttt{InventarNr},\texttt{Titel}}(r(\texttt{Buch})) \bowtie \sigma_{\texttt{Name}=\texttt{'Meyer'}}(r(\texttt{Ausleih}))$$

Änderungskomponente

Änderungskomponente eines Datenbanksystems ermöglicht es,

- Tupel einzugeben,
- Tupel zu löschen und
- Tupel zu ändern.

Sprachen und Sichten: SQL

```
select Buch.InventarNr, Titel, Name
from   Buch, Ausleih
where  Name = 'Meyer' and
       Buch.InventarNr = Ausleih.InventarNr

update Angestellte
set    Gehalt = Gehalt + 1000
where  Gehalt < 5000

insert into Buch values
   (4867,'Wissensbanken', '3-876','Karajan')

insert into Kunde
   ( select LName, LAdr, 0 from Lieferant )
```


Spracheinbettung

```
exec sql declare AktBuch cursor for
         select ISBN, Titel, Verlagsname
         from Buch
         for update of ISBN, Titel;

exec sql commit work;

exec sql rollback work;
```

Sichten in SQL

```
create view Meyers as
select Buch.InventarNr, Titel, Name
from Buch, Ausleih
where Name = 'Meyer' and
      Buch.InventarNr = Ausleih.InventarNr
```


Überblick über behandelte Komponenten

- *Optimierer*
- *Dateiorganisationen und Zugriffspfade*
- Organisation des *Sekundärspeichers*
- *Transaktionsverwaltung*
- *Recovery-Komponente*

Optimierer

- Äquivalenz von Algebra-Termen
  1. $\sigma_{A=Konst}$ *(REL1* $\bowtie$ *REL2)* und A aus REL1
  2. $\sigma_{A=Konst}$ *(REL1)* $\bowtie$ *REL2*
- allgemeine Strategie: Selektionen möglichst früh, da sie Tupelanzahlen in Relationen verkleinern
- Beispiel: REL1 100 Tupel, REL2 50 Tupel
  intern: Tupel sequentiell abgelegt
  1. 5000 ($\bowtie$) + 5000 (σ) = 10000 Operationen
  2. 100 (σ) + 10 · 50 ($\bowtie$) = 600 Operationen
     falls 10 Tupel in REL1 die Bedingung $A =$ Konst erfüllen

Joins

- Merge-Join: *Verbund durch Mischen* von R_1 und R_2
  - insbesondere dann effizient, wenn eine oder beide Relation(en) sortiert nach den Verbund-Attributen vorliegen, d.h. für Verbund-Attribute X muss gelten: $X := R_1 \cap R_2$
  - r_1 und r_2 werden nach X sortiert
  - Mischen von r_1 und r_2, d.h., beide Relationen parallel sequentiell durchlaufen und passende Paare in das Ergebnis aufnehmen
- Nested-Loops-Join: doppelt Schleife über R_1 und R_2
  - liegt bei einer der beiden Relationen ein Zugriffspfad für X vor, dann innere Schleife durch Zugriff über diesen Zugriffspfad ersetzen

Komplexität der Operationen

- Selektion
  - hash-basierte Zugriffsstruktur: $O(1)$
  - sequentieller Durchlauf: $O(n)$
  - in der Regel (baumbasierte Zugriffspfade): $O(\log n)$
- Verbund
  - sortiert vorliegende Tabellen: $O(n+m)$ (Merge Join)
  - sonst: bis zu $O(n*m)$ (Nested Loops Join)
- Projektion
  - vorliegender Zugriffspfad oder Projektion auf Schlüssel: $O(n)$
  - Duplikateliminierung durch Sortieren: $O(n \log n)$

Optimierungsarten

- *Logische Optimierung*:
  - nutzt nur algebraische Eigenschaften der Operationen
  - *keine* Informationen über die Speicherungsstrukturen und Zugriffspfade
  - Verwendung heuristischer Regeln anstelle exakter Optimierung
  - Beispiele:
    - Entfernung redundanter Operationen
    - Verschieben von Operationen derart, daß Selektionen möglichst früh ausgeführt werden

  $\rightsquigarrow$ *algebraische Optimierung*

Optimierungsarten II

- *Interne Optimierung*:
  - Nutzung von Informationen über die vorhandenen Speicherungsstrukturen
  - Auswahl der Implementierungsstrategie einzelner Operationen (Merge Join vs. Nested-Loops-Join)
  - Beispiele:
    - Verbundreihenfolge anhand der Größe und Unterstützung der Relationen durch Zugriffspfade
    - Reihenfolge von Selektionen nach der Selektivität von Attributen und dem Vorhandensein von Zugriffspfaden

Algebraische Optimierung

- *Entfernen redundanter Operationen* ($r \bowtie r = r$)

$$r(\texttt{BuchLangeWeg}) = r(\texttt{Buch}) \bowtie \pi_{\texttt{ISBN,Datum}}(\ldots \sigma_{\texttt{Datum<'31.12.1990'}}(r(\texttt{Ausleihe})))$$

  - Anfrage an Sicht:

$$\pi_{\texttt{Titel}}(r(\texttt{Buch}) \bowtie r(\texttt{BuchLangeWeg}))$$

  - Einsetzen der Sichtdefinition:

$$\pi_{\texttt{Titel}}(r(\texttt{Buch}) \bowtie r(\texttt{Bücher}) \bowtie \pi_{\ldots}(\ldots))$$

Algebraische Optimierung II

- *Verschieben von Selektionen*

$$\sigma_{\texttt{Autor='Vossen'}}(r(\texttt{Buch}) \bowtie \pi_{\texttt{ISBN,Datum}}(\ldots))$$

Verbund auf kleineren Zwischenergebnissen:

$$(\sigma_{\texttt{Autor='Vossen'}}(r(\texttt{Buch}))) \bowtie \pi_{\texttt{ISBN,Datum}}(\ldots)$$

Selektion und Verbund kommutieren

Algebraische Optimierung III

- *Reihenfolge von Verbunden*

$$(r(\texttt{Verlag}) \bowtie r(\texttt{Ausleihe})) \bowtie r(\texttt{Buch})$$

Nachteil: erster Verbund entartet zum kartesischen Produkt, da keine gemeinsamen Attribute

$$r(\texttt{Verlag}) \bowtie (r(\texttt{Ausleihe}) \bowtie r(\texttt{Buch}))$$

$\bowtie$ assoziativ und kommutativ

Dateiorganisation und Zugriffspfade

Konzeptionelle Ebene		**Interne Ebene**		**Platte**
Relationen	$\longrightarrow$	Dateien (Files)	$\longrightarrow$	
Tupel	$\longrightarrow$	Sätze (Records)	$\longrightarrow$	Blöcke
Attributwerte	$\longrightarrow$	Felder	$\longrightarrow$	

Zugriffspfade

- Primär- versus Sekundär-Index
- sequentielle Dateien, B-Bäume, Hashen
- eindimensional versus mehrdimensional
- spezielle Anwendungen

Transaktionen und Recovery

- **A**tomicity (Atomarität oder *Ununterbrechbarkeit*)
  - Transaktion wird ganz oder gar nicht ausgeführt
- **C**onsistency (Konsistenz oder *Integritätserhaltung*)
  - der von einer Transaktion hinterlassene neue Zustand genügt den Integritätsbedingungen
- **I**solation
  - Ergebnis einer Transaktion muß einem isolierten Ablauf dieser Transaktion entsprechen, auch bei mehreren nebenläufigen Transaktionen
- **D**urability (*Dauerhaftigkeit* oder *Persistenz*)
  - nach Ende einer Transaktion stehen Ergebnisse dauerhaft in der Datenbank

Transaktionen II

Das Ergebnis einer Transaktion soll so aussehen, als sei sie nach dem ACID-Prinzip abgelaufen.

Datenelemente und Sperren

Sperrmodelle:

T_1 : **lock** A;
read A;
$A := A + 1$;
write A;
unlock A;

Deadlocks:

T_1:	T_2:
lock A;	**lock** B;
...;	...;
lock B;	**lock** A;
...;	...;
unlock A;	**unlock** B;
unlock B;	**unlock** A;

Serielle Schedules

T_1:	T_2:
read A;	**read** B;
$A := A - 10$;	$B := B - 20$;
write A;	**write** B;
read B;	**read** C;
$B := B + 10$;	$C := C + 20$;
write B;	**write** C;

$T_1; T_2$ und $T_2; T_1$ sind seriell

Begriff der Serialisierbarkeit

Ein Schedule heißt *serialisierbar*, wenn sein Ergebnis äquivalent zu dem eines seriellen Schedules ist.
Methoden:

- Serialisierbarkeitsgraphen
- Zwei-Phasen-Sperr-Protokoll
- Zeitmarkenverfahren

Unterschiedliche Ablaufpläne

Schedule S_1		Schedule S_2		Schedule S_3	
T_1	T_2	T_1	T_2	T_1	T_2
read A		**read** A		**read** A	
$A - 10$			**read** B	$A - 10$	
write A		$A - 10$			**read** B
read B			$B - 20$	**write** A	
$B + 10$		**write** A			$B - 20$
write B			**write** B	**read** B	
	read B	**read** B			**write** B
	$B - 20$		**read** C	$B + 10$	
	write B	$B + 10$			**read** C
	read C		$C + 20$	**write** B	
	$C + 20$	**write** B			$C + 20$
	write C		**write** C		**write** C

Kaskadierende Transaktionsabbrüche

	T_1	T_2	
	lock A		
	read A		
	$A := A - 1$		
	write A		
	lock B		
	unlock A		
		lock A	
		read A	
		$A := A \times 2$	
	read B		
		write A	
		unlock A	← **commit** T_2
abort T_1 →	$B := B/A \downarrow$		

Recovery

- stabiler vs. *instabiler* Speicher
- Log-Buch / Journal
- *Backward Recovery*: Änderungen *rückgängig* machen ⇝ UNDO
- *Forward Recovery*: Änderungen nachziehen ⇝ REDO
- Schattenspeicher