AG Datenbanken und Informationssysteme

Wintersemester 2006 / 2007

Prof. Dr.-Ing. Dr. h. c. Theo Härder
Fachbereich Informatik
Technische Universität Kaiserslautern

http://wwwdvs.informatik.uni-kl.de

2. Übungsblatt

Für die Übung am Donnerstag, **09. November 2006**, von 15:30 bis 17:00 Uhr in 13/222.

Aufgabe 1: Datenunabhängigkeit und das Drei-Schichtenmodell

Erläutern Sie den Begriff der Datenunabhängigkeit und geben Sie anschließend zu den folgenden Vorgängen jeweils an, in welchen Schichten des Drei-Schichtenmodells (Daten-, Zugriff- oder Speichersystem) Änderungen notwendig sind:

a) Ein neues Anwendungsprogramm wird geschrieben, das bestehende Daten benutzt.

b) Neue Indexstrukturen werden implementiert, um die Leistung von Anwendungsprogrammen zu verbessern.

c) Neue Indexe werden angelegt, um die Leistung von Anwendungsprogrammen zu optimieren.

d) Die Daten werden auf einem anderen und leistungsfähigeren physischen Speichertyp gespeichert.

e) Ein anderer, neuer Rechnertyp wird eingesetzt.

f) Die für Ein- und Ausgabevorgänge verwendete Seitengröße wird geändert.

g) Die alte Seitenersetzungsstrategie wird durch eine neue ersetzt.

h) Neue Methoden zur Optimierung von Anfragen werden zusätzlich eingesetzt.

i) Die Adressierungsmethoden von Sätzen werden modifiziert.

j) Neue Tabellen und Fremdschlüsselbeziehungen werden eingefügt.

Lösung:

Datenunabhängigkeit bedeutet, dass Anwendungsprogramme möglichst unabhängig von der physischen Repräsentation der Daten sind. Sie wird durch schrittweise Abstraktion von der physischen Repräsentation auf dem Externspeicher hin zu höherwertigen Operatoren auf logischen Daten erreicht.

a) keine Änderungen notwendig

b) Änderungen im Zugriffssystem für die neuen Indexstrukturen

c) keine Änderungen notwendig

d) evtl. Änderungen im Betriebssystem oder Speichersystem zum Zugriff auf den neuen Speichertyp

e) keine Änderungen (wenn das Betriebssystem und evtl. Speichersystem den neuen Rechner und dessen Komponenten unterstützt)

f) Änderungen im Speichersystem

g) Änderungen im Speichersystem

h) Änderungen im Datensystem

i) Änderungen im Zugriffssystem

j) keine Änderungen notwendig

Aufgabe 2: Ablauf einer Anfrage im Fünf-Schichtenmodell

Gegeben sei folgende SQL-Anfrage:

```
SELECT * FROM Manager_View WHERE Geburtsort='Kaiserslautern'
```

Dabei ist die Relation `Manager_View` eine Sicht auf die Basisrelation `Angestellte`, die einen Index bzgl. des Attributs `Geburtsort` besitzt. Zu Beginn liegen die DB-Katalogdaten bereits im DB-Puffer, ansonsten sind keine Datenseiten im Puffer verhanden. Beschreiben Sie basierend auf dem in der Vorlesung vorgestellten Fünf-Schichtenmodell den Ablauf in den einzelnen Schichten bei der Ausführung der obigen Anfrage.

Lösung:

a) Schicht 5: Logische Datenstrukturen

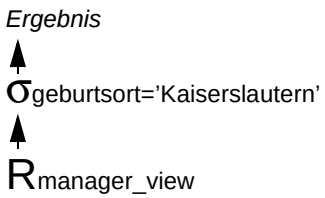
Aufgabe: Übersetzung und Ausführung von mengenorientierten DB-Operationen.

Objekte: Relationen und Sichten, Tupel

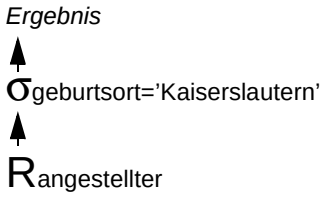
Operatoren: Relationale Operatoren + Erweiterungen

(1) Übersetzung und Optimierung der Anfrage

- Übersetzung der Anfrage:



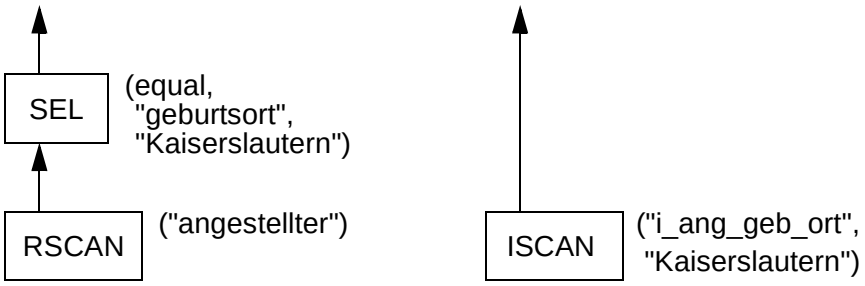
- Mit Hilfe des Metadatenkatalogs wird die Abbildung von 'manager_view' auf 'angestellte' ermittelt und eine entsprechende Anfragetransformation durchgeführt:



- Es wird eine algebraische Optimierung durchgeführt.
- Es werden alternative Zugriffspläne erstellt. Dabei wird der Metadatenkatalog benutzt, um Auskünfte über das interne Schema zu erlangen:

Gibt es einen Index auf ’angestellter(geburtsort)’?

Hat der Index eine ausreichende Selektivität für diesen Wert?



- Aus den möglichen Zugriffplänen, wird der ’beste’ Zugriffsplan ausgewählt (nicht-algebraische Optimierung). Der Indexzugriff muss nicht immer besser sein. Wenn die Selektivität nicht ausreicht (maximal 1% Treffer), kann ein Relationenscan überlegen sein.

(2) Auswerten der Anfrage

Die Operatoren des Zugriffsplans sind mit den Operationen der Schicht 4 realisiert. Für den Indexzugriff über einen mehrdeutigen Schlüssel, kann dies etwa wie folgt aussehen:

```
SCAN set = S4::OPEN_SCAN ( "i_ang_geb_ort", "geburtsort = kaiserslautern",
NULL)
while( set.NEXT() )
{
   RECORD tupel = set.FETCH();
}
set.CLOSE();
```

Der Indexscan wird mit einer Start- und einer Stoppbedingung initialisiert. Eventuell kann auch noch eine Suchrichtung angegeben werden.

b) Schicht 4: Logische Zugriffspfadstrukturen

Aufgabe: Satzweises Aufsuchen, Einfügen, Löschen und Modifizieren von Datensätzen, Transformation vom internen in das externe Format.

Objekte: Segmente, Satztypen, Satz, Index

Operatoren: Öffnen und Schließen von Segmenten, Durchlaufen aller Sätze eines Satztyps, Durchsuchen eines Index mit einem Schlüssel, Einfügen, Ändern oder Löschen eines Satzes

(1) S4::OPEN_SCAN(<indexname>, <startbedingung>, <stoppbedingung>)

- Im Metadatenkatalog nachsehen, welcher Art der Index ist, beispielsweise B*-Baum.
- Initialisiere den Index, beispielsweise mit

  BS_TREE btree = S3::INIT_BS_TREE("i_ang_geb_ort");

(2) SCAN.NEXT()

- Suche den nächsten Datensatz, der die Startbedingung erfüllt. Beachte dabei die Suchrichtung, falls eine vorgegeben ist. Dies könnte bei uns wie folgt aussehen:

  TID record_id = btree.FIND("Kaiserslautern");

- Falls ein Datensatz gefunden wird, der die Startbedingung und nicht die Stopbedingung erfüllt, dann gib ’true’ an den Aufrufer zurück, ansonsten ’false’.

(3) SCAN.FETCH()

- Satz laden

  RECORD record = S3::GET_RECORD(record_id);

c) Schicht 3: Speicherungsstrukturen

Aufgabe: Abbildung und Kontrolle der physischen Datensätze, Implementierung und Aktualisierung von Zugriffspfaden

Objekte: interne Sätze, B*-Bäume, Hash-Strukturen, Externspeicheradressierung

Operatoren: Operationen der Zugriffspfade, Operationen zur Abbildung von logischen auf physische Sätze (Lesen, Einfügen, Modifizieren, Löschen)

(1) S3::INIT_BS_TREE (...)

- Siehe im Metadatenkatalog nach, wo die Wurzelseite w des Index zu finden ist, und stelle die passende Seite im Puffer bereit.

  S2::BUFFER().FIX (w);

(2) BS_TREE.FIND (...)

- Suchalgoritmus für B*-Baum (aus "Datenstrukturen" oder "GBIS" bekannt).
- Wenn Zeiger (Externspeicheradressierung!) auf Knoten verfolgt werden, müssen die passenden Seiten im Puffer bereitgestellt werden:

  S2::BUFFER().FIX (...);

- Wenn Seiten nicht mehr benötigt werden, wird dies dem Puffer angezeigt:

  S2::BUFFER().UNFIX (...)

(3) S3::GET_RECORD (TID)

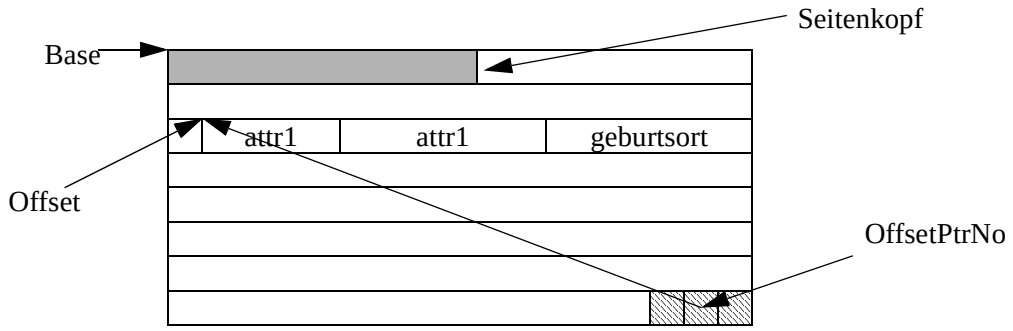
- Umsetzung der TID: In welcher Seite liegt der Datensatz?

  PAGE record_page = record_id.pageOf();

- Stelle die Seite im Puffer bereit:

  ADDRESS base = S2:BUFFER().FIX (record_page)

- Siehe im Metadatenkatalog nach, wie der Datensatz aufgebaut und wie er auf die Seite abgebildet ist.



- Lies den Datensatz aus der Seite und bringe ihn in das interne Satzformat. Dazu wird aus der Basisadresse der Seite im Hauptspeicher und einem Offset die Hauptspeicheradresse des Datensatzes ermittelt. Anhand der Metadaten kann festgestellt werden, wieviele Attribute der Datensatz hat, welchen Typs die Attribute sind und wie sie auf die Seite abge-

bildet wurden. Mit dieser Information kann der Datensatz aus der Seite ausgelesen werden.

Die Abbildung der Datensätze in die Seiten erfordert eine effiziente Freispeicherverwaltung, mit den üblichen Problemen (Fragmentierung, ..., usw.).

Die Externspeicheradressierung muss einen effizienten Zugriff erlauben und gleichzeitig stabil gegen Verschiebungen sein (uvm.).

d) Schicht 2: Seitenzuordnungsstrukturen

Aufgabe: Abstraktion von Dateien und Blöcken.

Objekte: Segmente und Seiten

Operatoren: Öffnen und Schließen von Segmenten, Anfordern & Bereitstellen von Seiten.

(1) BUFFER.FIX (page)

- Suche ’page’ im Puffer: Wie sollte der Puffer organisiert sein, damit dies effizient geschieht?
- falls die Seite noch nicht im Puffer existiert:
  -- falls der DB-Puffer voll ist, schaffe einen freien Slot:

     Dazu muss mit einer geeigneten Verdrängungsstrategie (LRU, FIFO, ...) eine Seite ausgewählt werden, die augenblicklich nicht benötigt wird. Falls diese Änderungen enthält, muss sie zurückgeschrieben werden:

     S1::write (block) (Die Abbildung von Seiten auf Blöcke ist in der Regeln nicht 1:1!)

     Dabei ist eine geeignete Einbringstrategie zu benutzen (für Logging & Recovery).

  -- Lade die Seite vom Externspeicher in den DB-Puffer:

     S1::read (block)
- Markiere ’page’ als ’fixed’ damit sie nicht verdrängt wird, solange sie benötigt wird. Übergebe dann die Hauptspeicheradresse an den Aufrufer.

(2) BUFFER.UNFIX (...)

- Markiere die Seite als ’unfixed’ und schaffe damit einen Verdrängungskandidaten.

(3) Weitere Operationen: OPEN_SEGMENT, CLOSE_SEGMENT

e) Schicht 1: Speicherzuordnungsstrukturen

Aufgabe: Verwaltung externer Speichermedien, Verbergen der Geräteeigenschaften

Objekte: Dateien, Blöcke

Operatoren: Anlegen und Löschen einer Datei, Öffnen und Schließen einer Datei, Lesen und Schreiben von Blöcken

Die Aufgaben werden häufig vom Betriebssystem übernommen. Nur wenn spezielle Eigenschaften notwendig sind, beispielsweise zur Erhöhung der Fehlertoleranz, werden eigene Mechanismen eingesetzt.

f) Schicht 0: Hardware

Aufgabe 3: Funktionale Abhängigkeiten (Herleitung)

Zeigen Sie **mit Hilfe der FA-Definition** aus der Vorlesung, dass folgendes Transitivitätsaxiom gilt:

$$X \rightarrow Y\ ,\ Y \rightarrow Z \models X \rightarrow Z$$

Wiederholen Sie aus der Vorlesung mit dem obigen Transitivitätsaxiom (A3) sowie Reflexivitäts- und Verstärkungsaxiom (A1 und A2) die Herleitung für die Vereinigungs-, Zerlegungs- und Pseudotransitivitätsregel (R4, R5 und R6).

Lösung:

Beh.: $X \rightarrow Y\ ,\ Y \rightarrow Z \models X \rightarrow Z$

Bew.:

Es gelten $X \rightarrow Y$ und $Y \rightarrow Z$. Dann gilt laut Definition für alle R aus von **R**:

sei $u \in R$ und $v \in R$ beliebig, dann gilt $(u.X = v.X) \Rightarrow (u.Y = v.Y)$ und

sei $u \in R$ und $v \in R$ beliebig, dann gilt $(u.Y = v.Y) \Rightarrow (u.Z = v.Z)$

Hieraus folgt, dass für alle R aus **R** gilt:

$\forall u \in R\ \forall v \in R\ (u.X = v.X) \Rightarrow (u.Z = v.Z)$

Damit folgt: $X \rightarrow Z$

Beh.: $X \rightarrow Y\ ,\ X \rightarrow Z \models X \rightarrow YZ$	(Vereinigung, Additivität (R4))
Bew:	
$X \rightarrow Y \models X \rightarrow XY$	(Verstärkung (A2) auf $X \rightarrow Y$ mit $X \subseteq X$)
$X \rightarrow Z \models XY \rightarrow ZY$	(Verstärkung (A2) auf $X \rightarrow Z$ mit $Y \subseteq Y$, $Y \subseteq A$)
$X \rightarrow XY\ ,\ XY \rightarrow ZY \models X \rightarrow YZ$	(Transitivität)

Beh.: $X \rightarrow YZ \models X \rightarrow Y$	(Zerlegung (R5))
Bew:	
$YZ \rightarrow Y$	(Reflexivität (A1) mit $Y \subseteq YZ$)
$X \rightarrow YZ\ ,\ YZ \rightarrow Y \models X \rightarrow Y$	(Transitivität)

Beh.: $X \rightarrow Y\ ,\ YW \rightarrow Z \models XW \rightarrow Z$	(Pseudotransitivität (R6))
Bew:	
$X \rightarrow Y \models XW \rightarrow YW$	(Verstärkung (A2) auf $X \rightarrow Y$ mit $W \subseteq W$, $W \subseteq A$)
$XW \rightarrow YW\ ,\ YW \rightarrow Z \models XW \rightarrow Z$	(Transitivität)

Bem.: Siehe [ST95] Simovici, D.A., Tenney, R.L.: Relational Database Systems, Academic Press, 1995.

Aufgabe 4: Anomalien in DB-Relationenschemata

Erläutern Sie die Begriffe *Einfüge-*, *Lösch-* und *Änderungsanomalie* und geben Sie jeweils ein Beispiel für eine Anomalie im folgenden DB-Schema an, das die Relation BESTELLUNG enthält:

BESTELLUNG (KUNDE, DATUM, ARTIKEL, LIEFERADRESSE, PREIS)

Die Attribute KUNDE, DATUM und ARTIKEL bilden den einzigen Schlüssel der Relation.

Lösung:

a) **Einfügeanomalie**
Wird die Information aus mehreren Entity-Typen in einem vermischt, so treten Einfügeanomalien auf, wenn man Information eintragen möchte, die nur zu einem Entity-Typ bzw. einem Teil davon gehört.
Beispiel:
Möchte man einen Kunden (KUNDE, LIEFERADRESSE) eintragen, der noch nie bestellt hat, so können die Daten nur eingefügt werden, indem man eine Pseudobestellung mit Pseudodatum und Pseudoartikel sowie NULL-Wert für PREIS erzeugt.

b) **Löschanomalie**
Wenn man die Information bzgl. eines von mehreren vermischten Entity-Typen löscht, dann kann es zum gleichzeitigen und unbeabsichtigten Löschen von Information der anderen Entity-Typen kommen.
Beispiel:
Hat man nur ein einziges Tupel, das eine Artikelbestellung eines Kunden repräsentiert, und man möchte die Bestellung löschen, so werden die Daten des zugehörigen Kunden ebenfalls aus der DB entfernt.

c) **Änderungsanomalie**
Sie treten auf, wenn man Änderungen auf Daten, die mehrfach in der DB gespeichert sind, ändern möchte. Damit die Konsistenz der Daten gewährleistet werden kann, müssen die mehrfach auftretenden Daten gleichzeitig aktualisiert werden. Dies führt zu erhöhtem Speicherbedarf sowie Leistungseinbußen bei Änderungen.
Beispiel:
Angenommen, in KUNDE wird der Kundenname verwaltet. Möchte man nun den Namen eines Kunden ändern, so muss man alle betroffenen Tupel abändern, sonst tritt eine Anomalie auf und es wäre ein „neuer Kunde entstanden“.

Bem.:

Siehe [KE06] Kemper, A., Eickler, A.: Datenbanksysteme, Oldenburg, 6.Auflage, 2006.