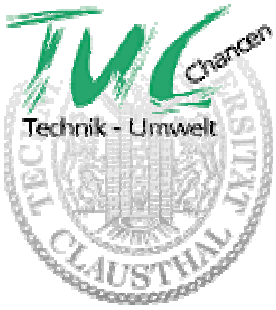


Technische Universität Clausthal

Institut für Informatik

Wintersemester 2004/05

Ausarbeitung zum Hauptseminarvortrag: „Einführung in das Decomposed Storage Model“

im Themenkomplex: „Databases on Modern CPU and Memory Architectures“

bei Herrn Grust

Robert Hartmann
(Matrikel Nr: 302300)

Inhaltsverzeichnis

1) Grundlagen

Definition: Datenbank (aus [3])

»Eine Datenbank (*database*), kurz *DB*, ist eine integrierte und strukturierte Sammlung persistenter Daten, die allen Benutzern eines Anwendungsbereiches als gemeinsame und verlässliche Basis aktueller Informationen dient. «

Definition: Datenbankmanagementsystem (aus [3])

»Ein Datenbankmanagementsystem (*database management system*),
kurz *DBMS*, ist ein All-Zweck-Softwaresystem, das den Benutzer bei der Definition, Konstruktion und Manipulation von Datenbanken für verschiedene Anwendungen applikationsneutral und effizient unterstützt. «

Definition: Datenbanksystem (aus [3])

»Ein Datenbanksystem (*database system*), kurz *DBS*, fasst die beiden Komponenten Datenbank und Datenbankmanagementsystem zusammen: DBS = DB + DBMS «

Es gibt verschiedene Datenbanksysteme mit eigenen Datenmodellen;
unter anderen gibt es:

- Relationale Datenbanksysteme:
  Inhalte sind Daten-Mengen und Relationen auf diesen.

- Objektorientierte Datenbanksysteme:
  Inhalte sind Objekte im Sinne der objektorientierten Programmierung

- Objekt-relationale Datenbanksysteme:
  Inhalte sind Mengen von Objekten, die in Beziehung stehen zu anderen Daten, welche keine Objekte darstellen

Anwendungsbereiche für Datenbanksysteme:

Online transaction processing (OLTP):

- Datenanalyse minimal
- Anfragen umfassen viele Attribute aber wenig Tupel

Online analytical processing (OLAP):

- Komplexe Analyse großer Datenbestände
- Anfragen umfassen viele Tupel aber wenige Attribute

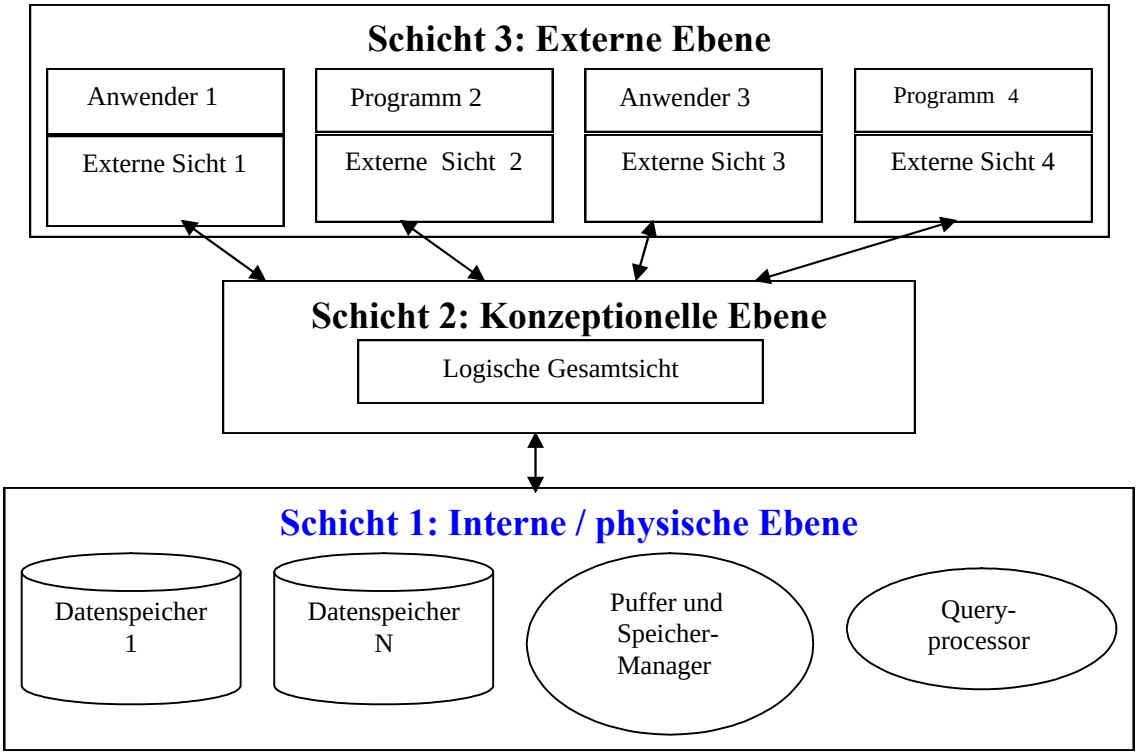
Datamining:

- Mehrere Datenbanksysteme als Teil eines Datawarehouse
- Sehr viele komplexe Anfragen auf sehr großen Datenkonglomeraten

2) Das 3 Schichten-Modell (1975)

Diese abstrakte Architektur ist im Wesentlichen Grundlage auch für moderne Datenbanksysteme.

Externe Ebene	Sicht einzelner Benutzergruppen
Konzeptuelle Ebene	Logische Gesamtsicht im konzeptuellen Schema
Interne/physische Ebene	Speicherschema der Datenbank



(3 Schichten Modell, Zeichnung: Robert Hartmann, Januar 2005)

3) Das n-äre Speichermodell (NSM)

- ist ein Speicherschema der physischen Ebene
- enthält die Tabellen des konzeptionellen Schemas
- besitzt n-äre Relationen, diese werden zusammenhängend gespeichert
- wird von sehr vielen relationalen Datenbanksystemen benutzt

Vorteil: Günstige Anzahl von Festplattenzugriffen bei OLTP

Nachteil: bei OLAP trotz der geringen Attributanzahl der Anfrage müssen alle Attribute der jeweiligen Relation geladen werden.

Natürlich gibt es Möglichkeiten zur Verbesserung der Zugriffe beim Aufsuchen (*retrieval*) von Datensätzen im n-ären Speichermodell.

Da wäre die **Clusterung** nach Attributen; jedoch stellt sich das Problem, nach welchen Attributen man sortieren soll. Zudem erzeugt man dadurch Performance-Verlust beim Modifizieren und Löschen von Datensätzen.

Eine weitere Möglichkeit wäre die Nutzung von i**nvertierten Dateien**. Jedoch auch diese Möglichkeit ist Problem behaftet. Es ist im Allgemeinen nicht klar, für wie viele Attribute invertierte Dateien angelegt werden sollten. Zudem kann es zu Speicherplatzproblemen auf dem Sekundärspeicher führen, wenn für alle Attribute einer Relation solche Dateien angelegt werden würden.

Auch die Nutzung von **Indexstrukturen** (z.B. B^+-Bäume, B^*-Bäume , Hash-Strukturen) besitzt die gleichen Probleme wie die oberen beiden Möglichkeiten.

Alle diese Möglichkeiten können nur dann „günstig“ umgesetzt werden, wenn Wissen über die konkrete Anwendung vorhanden ist.

Um eine problemunabhängige Möglichkeit zu finden hat man sich die Technik der **Datenaufsplittung** zu nutze gemacht.
Es existieren genau zwei Möglichkeiten der Aufsplittung, beide werden am Beispiel dieser n-ären Relation vorgeführt:

OID	Name	Tier	Fellfarbe	Kastration	Tätowierung	RFID-chip
1	Charlie	Kater	schwarz-weiß	Ja	Ja	Ja
2	Flo	Kater	getigert	Ja	Ja	Nein
3	Lucy	Katze	bunt	Ja	(NULL)	(NULL)
4	Alfons	Hund	grau	(NULL)	Ja	(NULL)

Die erste Möglichkeit ist die **horizontale Aufteilung**:

- z.B. zum Verteilen ganzer Relationen auf mehrere Datenträgern

OID	Name	Tier	Fellfarbe	Kastration	Tätowierung	RFID-chip
1	Charlie	Kater	schwarz-weiß	Ja	Ja	Ja
2	Flo	Kater	getigert	Ja	Ja	Nein

OID	Name	Tier	Fellfarbe	Kastration	Tätowierung	RFID-chip
3	Lucy	Katze	bunt	Ja	(NULL)	(NULL)
4	Alfons	Hund	grau	(NULL)	Ja	(NULL)

Die zweite Möglichkeit ist die **vertikale Aufteilung**:

- z.B. zum Verringern der Relationsgröße

OID	Name	Tier	Fellfarbe
1	Charlie	Kater	schwarz-weiß
2	Flo	Kater	getigert
3	Lucy	Katze	bunt
4	Alfons	Hund	grau

OID	Kastration	Tätowierung	RFID-Chip
1	Ja	Ja	Ja
2	Ja	Ja	Nein
3	Ja	(NULL)	(NULL)
4	(NULL)	Ja	(NULL)

4) Das Decomposed Storage Model (DSM)

- Das DSM ist ein komplett vertikal zerlegtes Speichermodell.
- Für jedes Attribut aus dem konzeptionellen Schema existiert eine binäre Tabelle, also eine binäre Relation.
- Jede Relation besitzt einen eindeutigen Namen,
  z.B. den Namen des Attributes
- Linker Wert in der Relation ist der Stellvertreter (*surrogate*)
  zur Identifikation von Spalten aus dem konzeptionellen Schema
- Rechter Wert ist der zugehörige Attributwert

Beispiel:

Konzeptionelles Schema entspricht dem n-ären Speicherschema:

```
Serien(Name, Folgenanzahl, Erstausstrahlungszeitraum)
```

Für das DSM müssen n-wertige Relationen in 2-wertige Relationen aufgespaltet werden:

```
Serien-Name(OID, Name);

Serien-Folgenanzahl(OID, Folgenanzahl);

Serien-Erstausstrahlungszeitraum(OID, Erstausstrahlungszeitraum);

Entitäten(OID);
```

Datenspeicheraufwand:

- 2 Kopien jedes Datums/jeder binären Relation (einmal sortiert nach OID-Werten, und einmal nach Attributwerten)
- Kopie jedes OID-Wertes pro zugehörigem Attributwert

- Speicherkompression ist bei DSM möglich, d.h. Relationen sind nach Attributwerten geclustert und können abgespeichert werden als Attributwert und Liste mit zugehörigen OIDs.

Es ergibt sich somit die Speicherverhältnisformel (aus [1]):

$$\frac{DSM}{NSM} = \underbrace{2}_{\text{Verdopplung der Relationen}} \cdot \underbrace{\frac{A(AS+SS)}{A \cdot AS+SS}}_{\text{Konzeptionelles Schema}} \cdot \underbrace{\frac{\left(\frac{AS}{RV}+SS\right)+(AS+SS)}{2 \cdot (AS+SS)}}_{\frac{\text{kompressiert}}{\text{nicht kompressiert}}}$$

wobei

A	…	Anzahl der Attribute
AS	…	Durchschnittsgröße der Attribute
SS	…	Durchschnittsgröße der OIDs
RV	…	Anzahl der durchschnittlich wiederholten Felder

Zur Verdeutlichung des Verhältnisses möge das Zahlenbeispiel aus [1] genügen:

A=10; AS=15; SS=5; RV=2

$$\rightarrow \quad \frac{DSM}{NSM} = 2 \cdot \frac{10(15+5)}{10 \cdot 15+5} \cdot \frac{\left(\frac{15}{2}+5\right)+(15+5)}{2 \cdot (15+5)} \approx 2,097$$

Typische Verhältniswerte sind zwischen 0,5 und 4

Aufsuch-Güte (retrieval performance):

Es gibt nach [1] genau 5 Faktoren, welche die Güte beeinflussen:

DSM	**NSM**
Clusterbildung nach allen Attributen	Anzahl der invertierten Dateien
Relationsgröße: max. binär	Relationsgröße: kein Maximum (Theorie)
Attributanzahl, die in den Bedingungen stehen	
Anzahl der Attribute, auf die projiziert werden soll	
Anzahl der Join-Relationen im konzeptuellen Schema	

Positive Merkmale von DSM:

- **Unterstützt mehrwertige Attribute**
- **Unterstützt Entitäten (auch ohne Attribute)**
- **Unterstützt Mehrfachbeziehungen**
- **Keine NULL-Wert-Problematik**
- **Kleine Relationsgröße → Update & Delete nicht zeitkritisch**
- **Daten leicht auf Sekundärspeicher verteilbar**
- **Möglichkeiten zur parallelisierbaren Anfragetechnik**

Negative Merkmale von DSM:

- **Erhöhter Speicheraufwand**
- **Mehr „Rechenoperationen“ für einen Join notwendig**
- **Implementierungsfehler schwerer zu erkennen**

5) Pivot-Algorithmus für DSM

Der Verbund (*Join*) von Relationen ist eine sehr zeitintensive Operation.
$O(join) > O(selection) \ \wedge \ O(join) > O(projection)$

➔Um also die Ausführzeit einer Anfrage zu reduzieren kann man die Anzahl der Joins reduzieren, was aber nicht immer möglich ist, oder man benutzt ein System welches die Join-Parameter geschickt wählen kann. So ein System findet man im **Pivot-Algorithmus**.

Der Pivot-Algorithmus besteht aus 4 Phasen [2]:

1. Phase der Selektion (*Select Phase*)
2. Pivot-Join-Phase (*Pivot Phase*)
3. Phase der Projektionsrelationen (*Value Materialization Phase*)
4. Antwortgenerierung (*Composition Phase*)

Die Arbeitsweise dieses Algorithmus zeigt ein Beispiel, wobei ich bewusst darauf verzichte, die einzelnen Werte der Relationen anzugeben, damit die Struktur des Algorithmuses sichtbar bleibt. Werte der Relationen konnten auf den Folien meines Vortrages gesehen werden.
Und sie können bei mir erfragt werden: Robert.Hartmann@tu-clausthal.de

Die Notation des Algorithmuses lehnt sich an Prolog an, dahingehend, dass der Ausdruck

```
New-Relation(Attribut1, Attribut2)
   <== Relation-A(Attribut1, Attribut2) , Relation-B(Attribut2,Attribut3);
```

bedeutet: In der Ergebnisrelation New-Relation stehen die Werte von Attribut1 und Attribut2, genau dann wenn sie in der Relation-A vorkommen und die Werte von Attribut2 gleichzeitig in der Relation-B existieren.

Ein OR aus dem Sprachumfang von SQL wird als senkrechter Strich | dargestellt.

Beispiel:

Gegeben sind die Relationen im konzeptionellen Schema:

```
Katze(KatzenID, Name, Geschlecht)
frisst(KatzenID, FutterID, Symptome)
Futter(FutterID, Name, Art, Geschmacksrichtung)
```

Gewünscht ist eine zur folgenden SQL-Anweisung äquivalente Lösung für DSM mit Pivotstrategie:

```
Select
       K.Name as Name, F.Art as Futterart,
       F.Name as Produkt,
       F.Geschmacksrichtung as Geschmack
from
       Katze K, Futter F, frisst fri
```

```
where
       // Join-Bedingungen:
            K.KatzenID = fri.KatzenID
       AND  fri.FutterID = F.FutterID
       // Selektionsbedingungen:
       AND (   fri.Symptome like "%Durchfall%"
            OR fri.Symptome like "%verweigert%");
```

Lösungsweg

1) Umformulieren der n-ären Relationen in binäre Relationen:

```
Katze(KatzenID, Name, Geschlecht)
=>   Katze-Name(KatzenID, Name);
     Katze-Geschlecht(KatzenID, Geschlecht);

frisst(KatzenID, FutterID, Symptome)

=> frisst-KatzenID(Key, KatzenID)
   frisst-FutterID(Key, FutterID)
   frisst-Symptome(Key, Symptome)

Futter(FutterID, Name, Art, Geschmacksrichtung, Inhalt)
=>   Futter-Name(FutterID, Name);
     Futter-Art(FutterID, Art);
     Futter-Geschmacksrichtung(FutterID, Geschmacksrichtung);
```

2) Anwendung des Pivot-Algorithmuses

I. Select-Phase

a)(Parallelisierbare Auswertung der Selektionsbedingungen)

```
S-Durchfall(Key)  <== frisst-Symptom(Key, "%Durchfall%");
S-verweigert(Key) <== frisst-Symptom(Key, "%verweigert%");
```

b)(parallelisierbare Erstellung der Join-Indizes)

{*Es gibt keine "natürlichen Join-Indizes", daher müssen nun die Join-Indizes passend zur SQL-Anfrage generiert werden.*

– Ein "natürlichen Join-Index" ist eine binäre Relation zwischen zwei Objektvertretern (Surrogates).}

```
JoinIndex(KatzenID,FutterID)
  <== frisst-KatzenID(Key, KatzenID),
      frisst-FutterID(Key, FutterID);
```

II. Pivot-Phase

(Bestimmung des Pivot-Elementes und Ausführung des Joins)

{*Da ein Verfahren zur deterministischen Wahl des optimalen Pivot-Elements noch unbekannt ist: Sei* ***KatzenID*** *das Pivot-Element.*}

```
I1(KatzenID, FutterID), I2(KatzenID)
 <== S-Durchfall(Key),
     frisst-KatzenID(Key, KatzenID),
     JoinIndex(KatzenID, FutterID);

I3(KatzenID, FutterID), I4(KatzenID)
 <== S-verweigert(Key),
     frisst-KatzenID(Key, KatzenID),
     JoinIndex(KatzenID, FutterID);
```

III. Value Materialization Phase

(Relationen der Projektionen werden erstellt, diese Relationen beinhalten schon die einzelnen Werte der Ergebnisspalten der Antwort auf die Anfrage.)

```
V-Temp1(KatzenID, FutterID)
 <==(I1(KatzenID, FutterID)| I3(KatzenID, FutterID));

V-Temp2(KatzenID)
 <==(I2(KatzenID)| I4(KatzenID));

V-Tiername(KatzenID, Name)
 <== V-Temp2(KatzenID),
     Katze-Name(KatzenID, Name);

V-Futterart(FutterID, Art)
 <== V-Temp1(KatzenID, FutterID),
     Futter-Art(FutterID, Art);

V-Futtername(FutterID, Name)
 <== V-Temp1(KatzenID, FutterID),
     Futter-Name(FutterID, Name);

V-Futtergeschmacksrichtung
  (FutterID, Geschmacksrichtung)
 <== V-Temp1(KatzenID, FutterID),
     Futter-Geschmacksrichtung(FutterID, Geschmacksrichtung);
```

IV. Composition Phase

(Generierung der Antwort)

```
Antwort(Name, Futterart, Produkt, Geschmack)
 <== V-Tiername(KatzenID, Name),
     V-Futterart(FutterID, Futterart),
     V-Futtername(FutterID, Produkt),
     V-Futtergeschmacksrichtung(FutterID, Geschmack);
```

6) Schlusswort

- Noch offenes Such-Problem: Festlegung des Pivot-Elementes
- Ohne Wissen über die Aufgabe hat DSM gegenüber NSM *Retrieval*-Vorteile
- Relationen sind 2-wertig
- Bei *Update* und *Delete* benötigt DSM mehr Schritte als NSM
- Veränderungen von Objekten im konzeptionellen Schema wirken sich nur inhaltlich auf das DSM aus.

7) Literaturnachweis

[1] G. P. Copeland, S. N. Khoshafian:
„A Decomposition Storage Model“;
1985 ACM 0-89791-160-1/85/005/0268S.

[2] N. Khoshafian, G. P. Copeland, T. Jagodits:
„A Query Processing Strategy For the Decomposed storage model“;
1987 IEEE CH2407-5/87/0000/0636

[3] Schneider:
„Implementierungskonzepte für Datenbanksysteme“
Springer-Verlag Berlin Heidelberg 2004, ISBN 3-540-41962-4