

# Multimedia-Datenbanken

## Kapitel 10: Relationale Multimedia-Datenbank-Verwaltungssysteme

---

Friedrich-Alexander-Universität Erlangen-Nürnberg  
Technische Fakultät, Institut für Informatik  
Lehrstuhl für Informatik 6 (Datenbanksysteme)

**Prof. Dr. Klaus Meyer-Wegener**

Wintersemester 2002 / 2003

Technische Universität Kaiserslautern  
Fachbereich Informatik  
AG Datenbanken und Informationssysteme

**Dr. Ulrich Marder**

Wintersemester 2003 / 2004

---

# 10. Relationale MMDBVS

## □ zur Erinnerung: Speichern und Wiedergewinnen

- Ziele dabei:
  - Geräte- und Formatunabhängigkeit
  - Beziehungen (für navigierenden Zugriff)
  - inhaltsorientierte Suche
  - Echtzeit

## □ zu lösende Teilprobleme (nach [Chri85b]):

1. Definition der Datentypen mit Operationen
  - abstraktes Modell / Schema der Medienobjekte
  - Zugriff, Änderung
  - Extrahieren von Information (in formatierte Daten)
  - Transformationen (Medien-Umsetzung)
2. Inhaltsadressierung
  - indirekte Suche (in zugeordnetem Text)
  - Ähnlichkeit statt Gleichheit
  - räumliche Beziehungen
  - automatische Inhaltserschließung?
  - Vergleichsoperationen

3. Techniken des Information Retrieval
  - Integration mit DB-Techniken
4. Mehrbenutzerbetrieb, Recovery, Zugriffskontrolle, Unterstützung von Versionen
  - Granulate
5. Speichergeräte mit großer Kapazität
  - optische Platten
  - Überwachung von Kopien, Komprimierung
6. Performance (Leistungsfähigkeit)
  - Zugriffsmethoden für die neuen Datentypen
  - Echtzeit
  - Hardware-Architektur, physischer DB-Entwurf, Optimierung von Anfragen
7. System-Architektur
  - Erweiterung existierender DBVS
  - separate Spezial-DBVS mit gemeinsamer Benutzerschnittstelle
  - komplett neues DBVS
8. Einsetzbare Prototypen
  - Erfahrungen sammeln

# 10.1 Multimedia-Datentypen

## 1. Einführung neuer (Basis-) Datentypen:

TEXT, GRAPHICS, IMAGE, SOUND, VIDEO  
mit darauf anwendbaren Operationen  
(→ Abstrakte Datentypen)

## 2. Einbettung in existierende Datenmodelle

- Relationenmodell
- Objekt-relationales Modell; SQL/MM (Kapitel 11)
- objektorientiertes Modell (Kapitel 12)

## Nutzung der verfügbaren Modellierungskonstrukte und Anfragesprachen

# Basisdatentypen

## integer

- Operationen:  
 $+, -, *, /, \dots : \text{integer} \times \text{integer} \rightarrow \text{integer}$   
 $=, \neq, \leq, \geq, \dots : \text{integer} \times \text{integer} \rightarrow \text{boolean}$

## real / float

- Operationen:  
analog zu integer

## char

- Operationen:  
Umsetzung in integer und umgekehrt, Ausgabe (drucken), ...

## boolean / bit

- Operationen:  
and, or, ...

**d. h. Typen bestimmt durch ihre Operationen!**

(„generische“ oder „parametrisierbare“ Typen)

## □ **listOf Typ (min, max)**

- Operationen:  
Länge feststellen, Zugriff auf einzelne Elemente, Konkatenation, Teilliste, reduce wie in LISP, ....
- Beispiele:  
byte = listOf boolean (8,8)  
string = listOf char (0,\*)
- kanonische Fortsetzung aller Operationen auf dem Elementtyp:  
Liste3 := Liste1 \* Liste2  
elementweise ausgeführt

## □ **setOf Typ (min, max)**

- Operationen:  
Anzahl Elemente, for each, Vereinigung, Differenz, Element hinzufügen oder entfernen, ....

# Der Datentyp Text

## ❑ Was ist das abstrakte Modell von Text?

- nur das, was „gleichen“ Texten gemeinsam ist – darstellungsunabhängig!
- mehr als nur listOf char!

## ❑ anwendbare Funktionen (in Java-Notation):

- lesender Zugriff:

```
interface Text {  
    public int length ();  
    public int alphabet (); // 0 == ISO Latin-1, ...  
    public int alphabetSize ();  
    public int language (); // 0 == English, 1 == German, ...  
    public char charAt (int n);  
    public byte [ ] getASCII ();  
    public byte [ ] getEBCDIC ();  
    public String getUnicode ();  
    ... }
```

## Der Datentyp Text (2)

- mit Worttrenner (Leerzeichen, „White Space“) und Zeilenende:  
public byte [ ] **word** (int wordNo);  
public byte [ ] **line** (int lineNo);  
public int **wordCount** ();  
public int **lineCount** ();
- ganzheitlich, z. B. Anzeigen und Ausdrucken:  
public boolean **print** (Printer p);  
public boolean **display** (Window w);
- ändernder Zugriff (mit Konsistenzerhaltung!):  
public void **replaceLine** (int lineNo, byte [ ] newLine);  
public void **insertLine** (int lineNo, byte [ ] newLine);  
public void **concatenate** (Text t2);



# Der Datentyp Text (3)

## ❑ generelles Problem:

### Prozedur oder Funktion?

- Prozedur ändert direkt (Beispiele oben)
- Funktion erzeugt neues Objekt:

```
public Text replaceLine (int lineNo, byte [] newLine);
```

## ❑ im Kontext von SQL (siehe unten)

- derzeit nur Funktionen nutzbar

# Der Datentyp Text (4)

- Erzeugen:

```
class TextClass implements Text {  
    public TextClass (  
        int length,  
        int charLength,  
        int code, // 0 == ASCII, 1 == EBCDIC, ...  
        int formatter, // 0 == none, 1 == PostScript, ...  
        byte endOfLine,  
        byte [ ] characters  
    ) { ... };  
    ...  
}
```

oder in einem spezifischen Kontext (Unix) auch:

```
public TextClass (String filename) { ... };
```

# Der Datentyp Image

- lesender Zugriff :

```
interface Image {  
    public int height ();  
    public int width ();  
    public int pixelcount (byte [] pixelvalue);  
    public Pixrect getPixrect ();  
    public boolean display (Device d);  
    ...  
}
```

- ändernder Zugriff:

```
public Image window (int x0, int y0, int x1, int y1); // (crop-Semantik)  
public Image replaceColormap ( ... );  
public Image replacePixelvalue ( ... );
```

## Der Datentyp Image (2)

- Erzeugen:

```
class ImageClass implements Image {  
    public ImageClass (  
        int height, int width, int depth,  
        float aspectRatio,  
        Code encoding,  
        int colormapLength, int colormapDepth,  
        int [ ] [ ] colormap,  
        byte [ ] pixelmatrix  
    );  
    ...  
}
```

in einem spezifischen Kontext (SUN) auch :

```
public ImageClass (Pixrect pr, Colormap cm);
```

## 10.2 Erweiterung des Relationenmodells

Text, Image, ... sind zugelassene atomare **Domänen** (Wertebereiche),  
d. h. Attribute können vom Typ Text, Image, ... sein

### □ **Beispiele:**

Angestellter	(Pers-Nr	integer,
	...	,
	Passbild	<b>image</b> )
Insasse	(Ins-Nr	integer,
	...	,
	Vorderansicht	<b>image</b> ,
	Seitenansicht	<b>image</b> ,
	Fingerabdrücke	<b>image</b> )
Auto	(Hersteller	varchar(50),
	Baujahr	integer,
	...	,
	Foto	<b>image</b> ,
	Motorgeräusch	<b>sound</b> )

**Relationenschema-Typ 1** (1 : 1-Beziehung, Attributbeziehung)

# Relationenschema-Typ 2

## □ 1 : N-Beziehung

- bei variabler Anzahl von Texten, Bildern etc. zu einem Entity: separate Relation einführen (Erste Normalform)

```
Patient (Name          varchar(100),
        ...,
        Passbild      image)
Röntgenbild (Patientenname varchar(100),
             Datum         date,
             Ansicht       varchar(30),
             Körperteil    varchar(40),
             Aufnahme      image)
```

- immer noch Attributcharakter
- Patientenname Teil des Primärschlüssels, d. h. nur Röntgenbilder speicherbar, zu denen Patient bekannt

## □ Zugriff:

- zur gemeinsamen Ausgabe von Patientendaten und Röntgenbildern: Verbund-Operation (Join)

# Relationenschema-Typ 3

## □ N : M-Beziehung

- Bilder können mehrere Entities zugleich zeigen:  
weitere Relation einführen

Pferd	(Name Alter	varchar(50), integer)
Rennfoto	(Archivnr Datum Ort Aufnahme	integer, date, varchar(80), <b>image</b> )
Ist_dargestellt_auf	(Pferdename Archivnr Position	varchar(50), integer, varchar(10))

- Fotos nun selbständige Entities, d. h. auch speicherbar ohne Zuordnung zu einem Gegenstand (hier: Pferd)

## □ Zugriff:

- zwei (i. Allg. teure!) Verbund-Operationen

## ❑ **Beziehungen**

- alle drei Typen (1 : 1, 1 : N, N : M) zwischen Medienobjekten und Entities darstellbar
- allerdings ohne besondere Semantik

## ❑ **Medienobjekte**

- können als Attribute oder als Entities verwaltet werden

## ❑ **umständlich:**

- verschiedene Typen von Entities zu einem Medienobjekt, z. B. Schiff, Auto und Flugzeug auf einem Bild
- mehrere Ist-dargestellt-auf-Relationen, dadurch noch mehr Joins

## ❑ **stabile Umgebung:**

- relationale Datenbanksysteme verbreitet im Einsatz,
- inkrementeller Lernaufwand für Benutzer,
- aufwärtskompatible Ergänzung existierender Datenbanken

## ❑ **in dieser Form (leidlich) realisierbar in objektrelationalen DBS (s. unten)**

## ❑ **geeignet zum Testen der neuen Datentypen:**

- welche Operationen braucht man wirklich?



## 10.3 Erweiterung der Datenbank-Anfragesprache

### □ hier: Benutzung von SQL (Standard)

- Beispielrelation:

Luftbildaufnahmen	(Nr	integer,
	Bild	image)

- **Eingabe**

- vom Programm aus:  
insert into Luftbildaufnahmen  
values (:nr, **image** (:pr, :cm));
- interaktiv (aus einer Datei):  
insert into Luftbildaufnahmen  
values (14537, **image** ("Aufnahme8.neu"));
- Typen der angegebenen Werte müssen zu den Domains der entsprechenden Attribute passen (Type Checking zur Übersetzungszeit)

# Erweiterung der DB-Abfragesprache (2)

- **Modifikation**

```
update Luftbildaufnahmen  
  set Bild = Bild.replaceColormap(:cm)  
  where Nr = 1286;
```

- **Suche und Ausgabe**

- vom Programm aus:

```
select Bild.getPixrect(), Bild.getColormap() into :pr, :cm  
  from Luftbildaufnahmen  
  where Nr = :k;
```

```
select Bild.height(), Bild.width() into :hoehe, :breite  
  from Luftbildaufnahmen  
  where Bild.pixelcount (:dunkelbraun) < 1000  
  and Bild.noOfColors () >= 4095;
```

- interaktiv (direkt auf ein Gerät):

```
select Bild.display (Device ("/dev/cir02"))  
  from Luftbildaufnahmen  
  where Nr = 715;
```

# Probleme mit der SQL-Einbettung

## ❑ nicht unterstützt:

- wiederholter Zugriff auf dasselbe Attribut  
(erst Höhe und Breite des Bildes, dann Farbtabelle, ... )

## ❑ nicht erlaubt:

- Kombination von values und subselect im insert  
(Ausschnitt eines Bildes woanders speichern)

## ❑ update

- Semantik des update ist **Wertesetzung**, nicht Modifikation
- besser wäre etwas wie:  
update Luftbildaufnahmen  
  apply Bild.replaceColormap ( ... )

## ❑ Programmierspracheneinbettung als Anweisung

- nicht als funktionaler Ausdruck:  
var := select ... from ... where ... ;  
writeScreen (select Bild.getPixrect () ... );

## 10.4 Objektrelationale DBS

### □ bieten im Strukturteil:

- Typen, Typkonstruktoren, ADTs
- Objektidentitäten für komplexe Tupel in Relationen (Komponenten über Identität dargestellt)
- Klassen- und Typhierarchie (getrennt), Klassen entsprechen Relationen (Tabellen)
- Vererbung und evtl. auch Overriding

### □ und im Operationenteil:

- generische, relationale Operationen (SQL)
- Methoden

### □ Beispiele:

- am Anfang: (University-) Ingres, 1984
- dann das kommerzielle Ingres und Postgres
- UniSQL
- Illustra und Informix Universal Server
- DB2
- Oracle 8

ISO- und ANSI-Norm,  
Weiterentwicklung von SQL-89 und SQL-92

Merkmale:

- Abstrakte Datentypen, mit create type definiert
- Objekt-Identifikatoren für einige ADTs („Objekt-ADTs“) neben Tupel-Identifikatoren für Tupel in Tabellen
- ADT-Hierarchien (ähnlich den Typhierarchien), die Substituierbarkeit von Objekten zusichern
- Tabellenhierarchien (ähnlich der Inklusion von Extensionen zwischen Unter- und Oberklassen, hier bezogen auf Tabellen); alle Attributnamen und Schlüssel werden geerbt, dürfen aber auch verändert werden
- Definition von Funktionen für ADTs
- Überladen des Funktionsnamens mit Möglichkeiten zur dynamischen Auswahl der Implementierung (ähnlich dem Overriding)
- komplexe Datentypen wie Arrays und Tupel (Mengen bzw. Multimengen für SQL:2003 geplant)

- ❑ **Tupel und Tabelle spielen unverändert Sonderrolle:**
  - Persistenz an das Einfügen von Tupeln in Tabellen gebunden, andere Objekte oder Werte nicht persistent
  - Anfragen nur an Tabellen möglich, ergeben auch wieder Tabellen

- ❑ **Abstrakte Datentypen**

- für Objekte (Objekt-ADT, mit Identität) oder für Werte (Wert-ADT)
- Attribute und Funktionen:

```
create type Student
    under Person
    as (
        MatrNr integer,
        Studienfach varchar(30),
        ...
    )
    instance method Durchschnittsnote ()
        returns real
        language SQL
        deterministic
        contains SQL
...;
```

- zu jedem Attribut eines ADTs zwei Funktionen automatisch generiert:
  - Observer: lesender Zugriff, Anfrage
  - Mutator: Ersetzung des Attributwerts
- ebenfalls equals als Vergleichsoperation zu jedem ADT
- Konstruktor mit Namen des ADTs,  
Destruktor destroy (nicht für Werte)

### □ **Einkapselung:**

- **keine** Sichtbarkeitsstufen wie public, protected, private
- bei Objekt-ADTs automatisch gekapseltes und nicht änderbares Surrogat-Attribut erzeugt;  
mit "with oid visible" sichtbar zu machen
- "equals oid" erlaubt dann auch Test auf Identität,  
im Gegensatz zur normalen Zustandsgleichheit ("equals state")

## ❑ Unterscheidung von

- Funktionen (Rückgabewert, Anfrage) und
- Prozeduren (kein Rückgabewert, Änderung)

## ❑ Parameter

- von Prozeduren können mit in, out und inout gekennzeichnet werden
- bei Funktionen nur in

## ❑ Funktionsdeklaration

- mit SQL:1999 selbst beschreiben,
- als "stored procedure" im System selbst abgelegt



## □ Database Extenders (IBM DB2 V2)

- Datentypen und Funktionen
  - Text Extender:
    - Funktionen zum Speichern von und Suchen in großen Texten
    - Suche nach im Text vorkommenden Stichworten, Synonymen von Stichworten und Phrasen
    - Ranking
    - intern Attribute für Sprache und Format des Texts
  - Image Extender
  - Audio Extender
  - Video Extender
  - Fingerprint Extender

# Kommerzielle Systeme (2)

## □ **Data Blades (Illustra, Informix Universal Server)**

- Sammlungen von Datentypen und zugehörigen Funktionen
- Speicherung in der DB (BLOB) oder in separater Datei
- zugeschnittene Indexstrukturen (z. B. R-Baum)
  - Text
  - Spatial
  - Image
  - Visual Information Retrieval (Virage)
  - weitere für WWW, Statistik, Zeitreihen u. a.
- von Fremdfirmen erstellt, von Informix zertifiziert

## □ **Data Cartridges (Oracle 8)**

- ConText, Virage, usw. wie bei den anderen

# Kommerzielle Systeme (3)

## □ **Stand:**

- Experimentierstadium, noch keine Einheitlichkeit, Norm noch nicht umgesetzt
- immerhin erster Versuch: SQL/MM (s. nächstes Kapitel), aber nur Test-Implementierungen
- keine Echtzeit!