Kapitel 4 – Werkzeuge: Objekt-relationale und Multimediale DBS

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Wie verwalte ich Multimedia-Dokumente?

- Prinzip Müllhalde
  - jedes MM-Dokument als Datei auf Betriebssystem-Ebene
  - beschreibende Daten? Organisation? Suche? Konsistenz? Gemeinsame Teilobjekte?
- Prinzip Seidener Faden
  - jedes MM-Dokument als Datei auf Betriebssystem-Ebene
  - Beschreibende Daten und Verweis auf Datei in die Datenbank
  - Organisation? (Suche?) Konsistenz? Gemeinsame Teilobjekte?
- Geht es nicht besser?

Multimedia-Datenbanksysteme

- Multimedia-Daten
  - in externen Dateien, aber mit enger Bindung an DBMS
  - innerhalb des Datenbanksystems als BLOB
  - innerhalb des Datenbanksystem im Typsystem des DBMS
- Definition: MMDBMS ist ein volles DBMS
  - mit Standard-Datentypen (und Operationen)
  - mit Multimedia-Datentypen (und Operationen)
    - Schnittstelle festgelegt, Speicherung wie oben
  - Unterstützung riesiger Datenmengen (x Millionen zu y MB, oder x Tausend zu y GB)
  - effizientes Management von Speichermedien (Disk Arrays, hierarchische Speicher) und Datenkompression
  - Information-Retrieval-Konzepte (inhaltsbasierte Suche)

Multimedia-Datenbanksysteme

- was kann Datenbanktechnik für Multimedia leisten?
  - **Datenunabhängigkeit** der Anwendungen
  - Anwendungsneutralität der Datenstrukturen (Schema)
  - Unterstützung der Suche
  - Speicherorganisation
- Basisdienst
  - für die Vielzahl der Anwendungen ("Infrastruktur")
  - nicht so sehr ein Endbenutzersystem, Programmschnittstelle!
- Speichern und Wiedergewinnen
  - von (Multi-) Media-Datenobjekten

Multimedia-DBS (2)

- Geräte- und Formatunabhängigkeit
  - beim Zugriff auf MM-Datenobjekte
- Beziehungen
  - von MM-Datenobjekten untereinander
  - wie auch zwischen MM-Datenobjekten und strukturierten Daten
  - darstellen
    und zum Auffinden benutzen:
    - Unterstützung der **Navigation**
- Inhaltsorientierte Suche
  - deskriptiver Zugriff
  - unscharf (Ähnlichkeit)

Multimedia-DBS (3)

und natürlich:

- Wiederherstellung im Fehlerfall
- Mehrbenutzerbetrieb, Synchronisation
- Zugangskontrolle
- ....

Was ist neu?

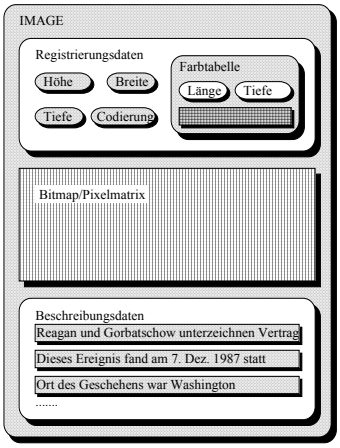
- Suche nach Ähnlichkeit
  - nicht nur exakte Übereinstimmung
- sehr spezielle Zugriffspfade
  - oft multidimensional
  - evtl. auch Graphen
- Speicherverwaltung
  - große Objekte
  - Einbeziehung von Abspielzeiten
- Auslieferung der Daten
  - zeitgesteuert
  - dauert signifikante Zeit

Multimedia-Datenbank-Technik

- Einführung der (elementaren) Datentypen
  - TEXT, GRAPHIC, IMAGE, SOUND, VIDEO, ...
    mit darauf anwendbaren Funktionen (→ Abstrakte Datentypen)
- Einbettung in existierende Datenmodelle
  - Relationenmodell (als Domains)
  - objektorientiertes Modell (als Klassen)
- Nutzung der verfügbaren Modellierungskonstrukte:
  - Relationen bzw. Klassen
  - Attribute bzw. Instanzvariablen
  - Primärschlüssel bzw. Objektidentifikator
  - Methoden, Vererbung, ...
- und der Abfragesprachen:
  - Relationenalgebra, SQL

Beispiel: der Datentyp IMAGE

- abstrakte Sicht auf eine Instanz:



Beispiel: der Datentyp IMAGE (2)

- Operationen
  - beschreiben den Datentyp vollständig
  - Funktionen, d.h. liefern Ergebnis bestimmten Typs
- Zugriff: Ausgeben

  lesender Zugriff auf Teile:

```
interface Image {
    public int height ( );
    public int width ( );
    ...
```

  ganzes Bild in ein Programm (als SUN-Pixrect):

```
    public Pixrect getPixrect ( );
```

  auf ein Gerät:

```
    public boolean display (Device d);
```

Beispiel: der Datentyp IMAGE (3)

- Auswerten, ableiten

  public int **pixelcount** (byte [ ] pixelvalue);

  zählt die Häufigkeit eines bestimmten Pixelwerts;

  public Image **window** (int x0, int y0, int x1, int y1);

  bildet einen Ausschnitt

- Modifizieren

  Prozeduren (keine Funktionen):

  ```
  public void replacePixelvalue (
    int x, int y,
    byte [ ] pixelvalue
  );
  ```

  u.v.a.

Beispiel: der Datentyp IMAGE (4)

- Erzeugen (eingeben)

  a) aus einem Programm:

  ```
  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 Systemkontext (hier: SUN) auch:

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

  b) aus einer Datei:

  ```
    public ImageClass (String filename, String format);
  ```

  c) von einem Gerät:

  ```
    public ImageClass (Device d);
  ```

Suche (Selektion)

- hierarchisch definierte Suchbereiche
- Benutzung zugeordneter Attribute mit Standard-Datentypen:
  - beschränkter Informationsgehalt
- Benutzung der Medienobjekte selbst:
  - Browsing (MINOS)
  - Pattern Matching
  - Ordnung, Vergleich (POSTGRES)
    Gleichheit von Bildern?
  - Klassifikation (Schlüsselworte)
  - Inhaltsrepräsentation
    - Statistik, Merkmale, Features
    - Text (Inhaltsangabe)
    - Semantische Netze
    - Frames
    - Prädikate (Logik)
    - .....

Anforderungen an die Datenhaltung

- Langzeitspeicher (Archiv):
  - großes Datenvolumen
  - hierarchische Organisation
  - vielfache Verknüpfungen
  - Bedeutung der effizienten Suche
  - Wissensextraktion?
- Kurzzeitspeicher (editieren, lesen, anschauen)
  - schneller Zugriff
  - Einbringen in Langzeitspeicher
  - Synchronisation bei der Ausgabe (z. B. Bild und Ton)
  - unterschiedliche Geräte
  - unterschiedliche Sichten (Tabelle oder Graphik)

Aufgaben eines MMDBS

- Organisation der Datenhaltung
  in den Multimedia-Systemen
- allgemeine Aufgaben eines DBS
  - Persistenz
    (Daten bleiben über einen Programmlauf hinaus erhalten)
  - Externspeicherverwaltung
    (Datenunabhängigkeit)
  - Mehrbenutzerbetrieb
    (Synchronisation)
  - Wiederherstellung im Fehlerfall
    (Transaktionskonzept, Recovery)
  - Anfragesprache
    (logische Sicht, einfache Handhabbarkeit)

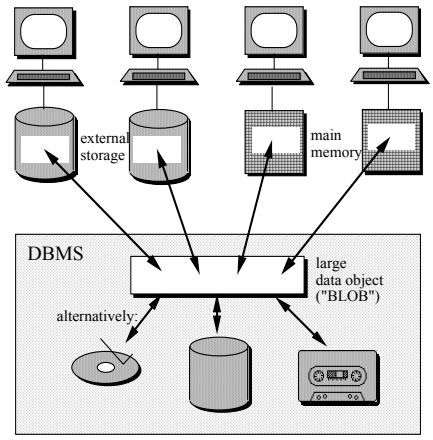
Speichern und Wiedergewinnen von Medienobjekten

- nicht mehr:
  - keine Integration komplexer Bearbeitungs- und Auswertungsalgorithmen in das DBS
  - z. B. Editoren für Medienobjekte nicht Bestandteil des DBS; sollten statt dessen auf die DB zugreifen (beim Laden und Sichern)
  - Trennung: Zugriffs- und Manipulationsoperationen
- und nicht weniger:
  - keine ununterscheidbaren "Binary Large Objects" (BLOBs)
  - Eigenschaften, Strukturen und Anforderungen der Medienobjekte zu verschieden (z. B. Zeitabhängigkeit oder Komprimierung)
- und als Ganzes
  - keine "atomisierte" Speicherung
    der einzelnen Wörter oder Bildpunkte

Geräteunabhängigkeit

- Multimedia-Datenobjekte
  - sehr groß
- verlangen nach neuen Datenträgern:
  - optischen Speichern
  - Videotapes
- Speichergeräte
  - werden anders bedient als die herkömmlichen Magnetplattenspeicher
- Anwendungsprogramme
  - sollen unabhängig sein
    von der gerade verwendeten Speichertechnologie
  - keine Programmänderung
    bei Einführung neuer (besserer) Speichergeräte
- Aufgabe des DBS:
  - spezifische Eigenschaften eines Speichergeräts so weit wie möglich verbergen

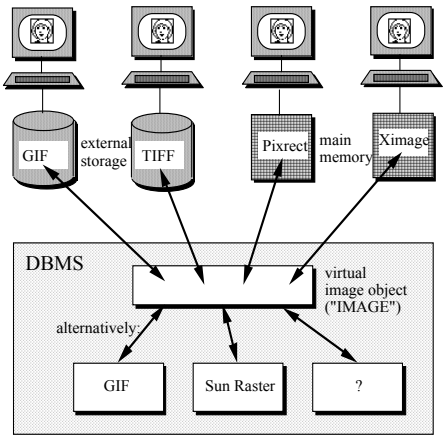
Geräteunabhängigkeit (2)



Formatunabhängigkeit

- Speicherungsformate für Medienobjekte
  - in großer Zahl vorhanden
  - darunter diverse "Standards"
    (bei Bildern: GIF, TIFF, Sun Rasterfile, FBM,
    PBM, ALV, JPEG, ...)
  - weitgehend ineinander überführbar
  - außerdem: Komprimierungstechniken, Containerformate (z. B. AVI)
- Aufgabe des DBS:
  - internes Speicherungsformat vor den Benutzern verbergen
  - Umsetzungsroutinen bereitstellen
  - Änderung des Speicherungsformats ermöglichen
    (z. B. wenn bessere Komprimierung verfügbar)
    ohne Beeinträchtigung der Anwendungsprogramme

Formatunabhängigkeit (2)



Beziehungen zwischen Daten in verschiedenen Medien

- Attributbeziehung
  - Objekt des modellierten Weltausschnitts (Entity) nicht nur durch formatierte Attribute beschrieben, sondern auch durch Bild und Geräusch (Beispiel: Auto)
- Komponentenbeziehung
  - Datenobjekt (z. B. Dokument oder Strukturbeschreibung) besteht aus Komponenten, die Medienobjekte sind

Beziehungen (2)

- Substitutionsbeziehung
  - zwei Medienobjekte mit (annähernd) gleichem Inhalt in verschiedenen Medien dargestellt (z. B. als Tabelle und als Graph)
  - Wahl der geeignetsten Darstellung je nach verfügbarem Ausgabegerät oder Benutzergeschmack
- Synchronisationsbeziehung
  - (meist in Kombination mit Komponentenbeziehung)
  - zwei Medienobjekte immer gleichzeitig darstellen (z. B. zu einem Text oder Bild Tonaufnahme abspielen); strengste Form: Lippensynchronität von Ton und Bild bei Video

Inhaltsorientierte Suche

- sehr viele Medienobjekte verwalten, z. B.
  - Röntgenbilder im Krankenhaus
  - Photoarchiv einer Zeitung
- Suche über die begleitenden formatierten Daten
  - z. B. Archivnummer, Datum
  - machbar, reicht aber nicht aus
  - auch Anfragen wie:
    - Röntgenbilder mit einer Fraktur in der rechten oberen Hälfte des Schädels?
    - Bilder eines Schneesturms?
    - Alle Bilder, auf denen Gorbatschow einen Vertrag unterzeichnet?
- Inhaltsangaben für Medienobjekte verwalten
  - z. B. Strukturinformationen
  - weitere denkbar
- unscharfe Suche

Schnittstellen und Funktionen

- im Unterschied zu bisherigen DBS:
  Programmschnittstelle deutlich anders als interaktive
  - vgl. Bild ansehen – Bild im Programm analysieren
    Tonaufnahme anhören – im Programm analysieren
- Beispiel: Relationales Datenmodell,
  erweitert um Datentypen IMAGE, TEXT, GRAPHICS, SOUND usw.

```
create table Person
    (Name  char(30),
    ....,
    Portrait  Image,
    Fingerabdruck  Image)
```

- Zugriffsfunktionen des Datentyps:
  - abhängig von der Art der Schnittstelle

Programmschnittstelle

(mit erweitertem SQL)

- Lesen eines Fingerabdrucks zur Analyse im Programm:

```
select Fingerabdruck.height(), Fingerabdruck.width()
   into :hoehe, :breite
   from Person
   where Name = "Müller";
```

(Speicherplatz anlegen für Pixel)

```
select Fingerabdruck.pixelmatrix()
   into :pixel
   from Person
   where Name = "Müller";
```

(Bearbeiten des Array pixel)

evtl. Durchreichen des Bilds an Ausgabegeräte oder Fenster (problematisch mit SQL)

Programmschnittstelle (2)

- direkte Ausgabe:

```
exec sql
   select Fingerabdruck.display(:fenster) into :fehler
   from Person
   where .... ;
if ( fehler != 0 )
    .... ;
```

- Schreiben auf Datei:

```
exec sql
   select Fingerabdruck.toFile(:datei) into :fehler
   from Person
   where .... ;
if ( fehler != 0 )
    .... ;
```

- "Seiteneffekte" in SQL-Anweisungen
  - funktionale Programmierung – LISP – besser geeignet

Interaktive Schnittstelle

- gleiches Beispiel:

```
select Fingerabdruck
   from Person
   where Name = "Müller";
```

- Ergebnis ist eine Tabelle
  - Sonderzeichen oder Bildsymbol (Icon) signalisiert: Wert ist vom Typ IMAGE
  - Mausklick oder spezielles Kommando zum Zeigen, in einem Fenster oder auf separatem Monitor
  - (andere Gestaltung möglich!)

Welche Datenbanktechnologie?

- Objektorientierte Datenbanksysteme
  - komplexe Objektstrukturen und Methoden können MM-Datenstrukturen und -operationen am besten darstellen
  - schlecht erweiterbar, instabil, wenig verbreitet, ...
- **Objektrelationale Datenbanksysteme**
  - **komplexe Objektstrukturen und Methoden in Tabellen**
  - **neue Datentypen definieren (Schnittstelle), evtl. extern realisieren**
  - **erweiterbar, relativ stabil, weit verbreitet (da Nachfolger von RDBMS)**
- Middleware-Ansatz
  - RDBMS und Medien-Server unter Middleware-Schicht
  - sehr leicht erweiterbar, sehr stabil, .., aber wenig effizient, DBMS-Features zu wenig ausgenutzt

Multimedia-Datentypen

1. Einführung neuer (Basis-) Datentypen:
   TEXT, GRAPHICS, IMAGE, SOUND, VIDEO
   mit darauf anwendbaren Operationen
   ($\rightarrow$ Abstrakte Datentypen)

2. Einbettung in existierende Datenmodelle
   - Relationenmodell
   - Objekt-relationales Modell; SQL/MM

Nutzung der verfügbaren Modellierungskonstrukte
und Anfragesprachen

Basisdatentypen

integer
- Operationen:
  $+ , - , * , / , \ldots$: integer $\times$ integer $\rightarrow$ integer
  $= , \neq , \leq , \geq , \ldots$: integer $\times$ integer $\rightarrow$ 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!

Typkonstruktoren

(„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);
```


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	**image**)
Insasse	(Ins-Nr	integer,
	… ,	
	Vorderansicht	**image**,
	Seitenansicht	**image**,
	Fingerabdrücke	**image**)
Auto	(Hersteller	varchar(50),
	Baujahr	integer,
	… ,	
	Foto	**image**,
	Motorgeräusch	**sound**)

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              varchar(50),
                Alter             integer)
Rennfoto       (Archivnr          integer,
                Datum             date,
                Ort               varchar(80),
                Aufnahme          image)
Ist_dargestellt_auf (Pferdename   varchar(50),
                Archivnr          integer,
                Position          varchar(10))
```

  - Fotos nun selbständige Entities, d. h. auch speicherbar ohne Zuordnung zu einem Gegenstand (hier: Pferd)
- Zugriff:
  - zwei (i. Allg. teure!) Verbund-Operationen

Bewertung

- 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?

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/clr02"))
  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 **Wertersetzung**, 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 () … );
```

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

SQL:1999

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)

SQL:1999 (2)

- 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
…;
```

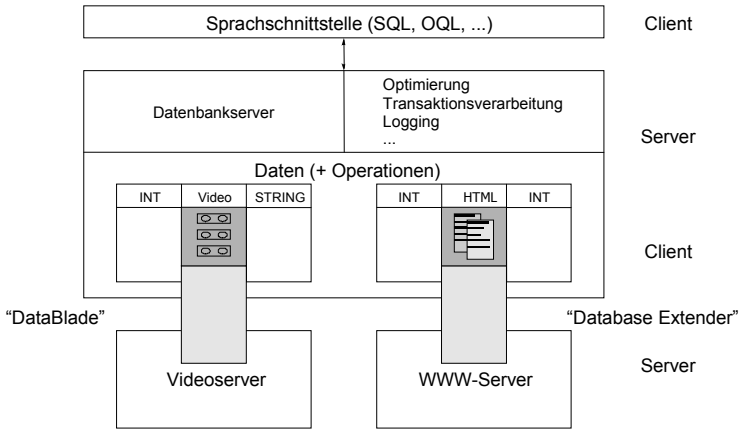

SQL:1999 (3)

  - 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")

SQL:1999 (4)

- 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

Objektrelationale Datenbanksysteme

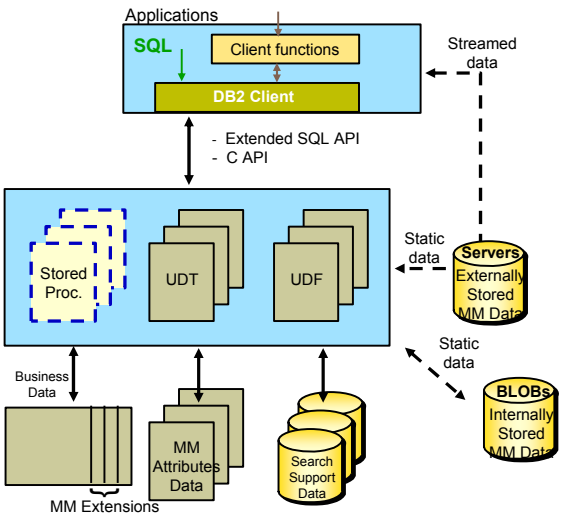


Beispiele: objekt-relationale Datenbanksysteme DB2, Oracle

Speicherung und Indexierung

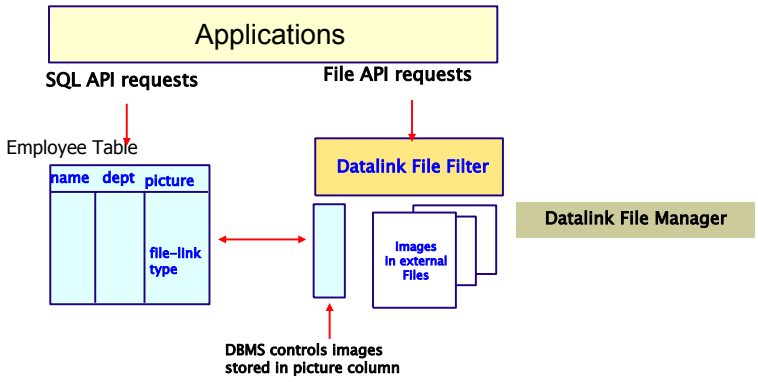
- Speicherung
  - Multimedia-Dokument ablegen in externer Datei, Byte-String oder anderer DBMS-Dateistruktur
  - sinnvoll bei Geo-Daten und anderen Bilddaten (etwa R-Baum)
- Indexierung
  - statt Multimedia-Dokument nun abgeleitete Informationen über das Dokument im DBMS ablegen
  - sinnvoll bei allen MM-Daten (etwa R-Baum, X-Baum, LSD-Baum, .. Andere Indexstrukturen für hochdimensionale Daten)
- Verbreiteter Fall
  - Multimedia-Inhalt in Byte-String (BLOB) oder externe Datei, unter Kontrolle des DBMS (Management of External Data)
  - Indexstruktur im DBMS

Architektur



Integration großer Datenbestände mittels DB2

- ... oder als Datalinks
  - Referentielle Integrität
  - Zugriffskontrolle
  - Konsistentes Backup und Recovery
  - Transaktionssicherheit



Kommerzielle Systeme

- 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

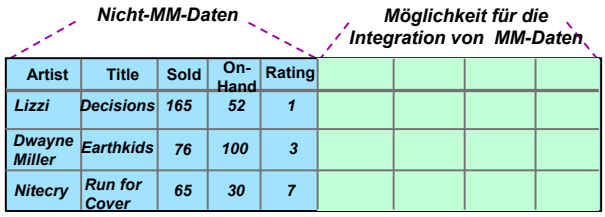
Kommerzielle Systeme (2)

- 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)

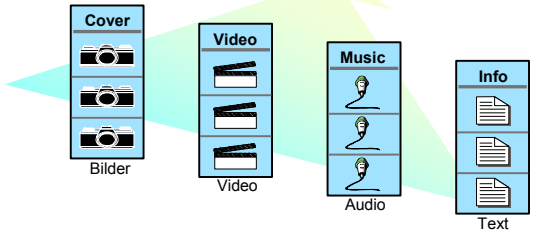
- Stand:
  - Experimentierstadium, noch keine Einheitlichkeit, Norm noch nicht umgesetzt
  - immerhin erster Versuch: SQL/MM,
    aber nur Test-Implementierungen
  - keine Echtzeit!

DB2 Extender - Applikationssicht



Artist	Title	Sold	On-Hand	Rating				
Lizzi	*Decisions*	*165*	*52*	*1*				
Dwayne Miller	*Earthkids*	*76*	*100*	*3*				
Nitecry	*Run for Cover*	*65*	*30*	*7*				

- Komplexe Datentypen
- Neue Funktionen
- Integrierte, innovative Suche
- Offenheit



Erweitertes Datenmodell (Beispiel Image Extender)

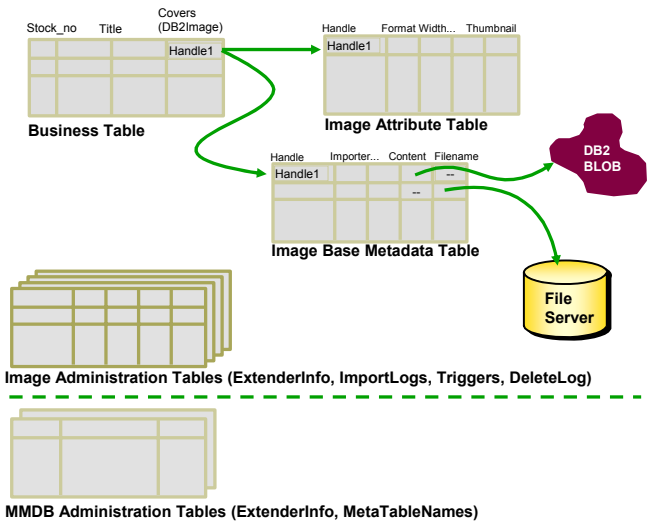


Image Extender: Übersicht

- Es werden u.a. folgende Attribute generiert:
  - format, thumbnail, Länge, Breite, ...
- Unterstützt gebräuchliche Bildformate
  - (BMP,EPS,EP2,GIF,IMG,IPS,JPG,PCX,PGM,PS,PSC,PS2,TIF,YUG, ...)
- Format-Konvertierungen
- Unterstützt sowohl interne als auch externe Speicherung von Daten
- Unterstützt Query by Image Content (QBIC)
- Bewerten der Suchergebnisse (ranking)

Image Extender: QBIC

- Nach diesen Kategorien kann gesucht werden:

- Farbdurchschnitt
  - Farbdurchschnitt des Bildes z.B. grün, rot violett

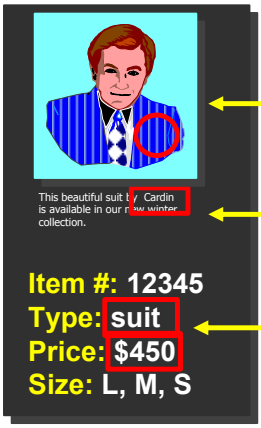
- Farb-Histogramm
  - Farbverteilung in einem Bild, z.B. 50% Rot und 20% Gelb

- Farb-Position
  - Bestimmende Farben in bestimmten Bildpositionen

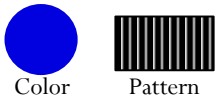
- Textur
  - Größe, Kontrast und Ausrichtung von Wiederholungsmustern in einem Bild

- Obige Daten werden in einem Katalogisierungslauf extrahiert und stehen dann für Suchprozesse zur Verfügung

Kombinierte Bild-, Text-, und Katalogsuche



Bildinhalt Suche



Textinhalt Suche

"Cardin or Dior"

Katalog Suche

Type = suit
Price < $500

Video Extender: Übersicht

- Es werden u.a. folgende Attribute generiert:
  - Format, Dauer, Frame-Nummer, ...
- Unterstützt AVI, MPEG1, MPEG2, QT
- Unterstützt DB-interne Speicherung für store-and-forward playback
- Unterstützt externe Speicherung auf Media-Server für realtime playback
- Import, Export und Update von Videos
- Suchen nach Video-Attributen
- Unterstützt Video Shot Change Detection (Szenenwechsel)

Video Extender: Scene Change Detection

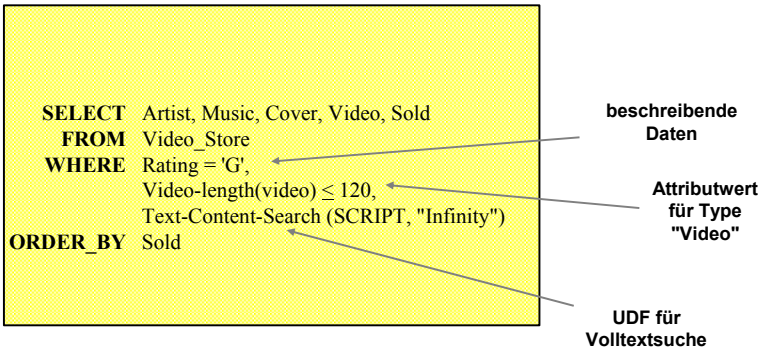
- Szenenwechsel: Signifikanter Unterschied zwischen 2 Video Frames
- Shot: Frames zwischen 2 Szenenwechsel

- Funktionen:
  - Erkennung von Szenenwechseln
  - Erzeugen und Verwalten des Shot-Katalogs in der Datenbank
  - Extrahieren von Frames
- Zusammenspiel mit QBIC
  - "Gibt es im Video eine Sonnenaufgangsszene ähnlich diesem Bild? Wenn ja, spiele bitte das Video ab dieser Szene!"

Beispielrelation mit MM-Inhalten

SOLD	ONHAND	RATING	ARTIST	TITLE	COVER	VIDEO	MUSIC	SCRIPT
234	59	PG-13	Arnold	The Exterminator				
13	45	R	Kevin	Dancing with Bulls				
1295	209	G	Glenn	101 Doll Imitations				
379	112	G	Buzz	Toy Glory				

Mischung traditioneller relationaler Daten mit MM-Daten: Anfrage



SQL/MM – Ein SQL-Standard für Medienobjekte

- Themen:
  - Motivation
  - Überblick
  - SQL/MM FullText
  - SQL/MM StillImage

Motivation

- Verwendung von Mediendaten und -operationen in vielen Anwendungen
- Ausnutzung der Erweiterbarkeit von ORDBMS für die Definition von Medienobjekten
- Bereitstellung von Erweiterungen in Paketen
  - erleichtert Verwaltung (Installation, Upgrade, Entfernen) und Wiederverwendung (ein Paket kann anderes benutzen)
- Proprietäre Pakete existieren bereits für
  - **Informix:** Excalibur Text Search DataBlade, Excalibur Image DataBlade, Informix Video Foundation Data-Blade Module
  - **DB2:** Image Extender, Audio Extender, Video Extender, Text Extender
  - **Oracle:** Visual Information Retrieval (VIR) Cartridge, ConText Cartridge, InterMedia
- Standardisierung erlaubt:
  - gemeinsame „Sprache"
  - Datenaustausch
  - Anwendungen laufen auf verschiedenen Implementierungen

Überblick SQL/MM

- gehört zum SQL-Standard, ist aber eigenständig
  - SQL: ISO/IEC 9075, SQL/MM: ISO/IEC 13249
  - Voller Name: *SQL Multimedia and Application Packages*
- besteht aus mehreren Teilen
  - Teil 1: SQL/MM **Framework** (IS Nov. 2002)
  - Teil 2: SQL/MM **Full Text** (IS Okt. 2000)
  - Teil 3: SQL/MM **Spatial** (IS Dez. 1999)
  - Teil 5: SQL/MM **Still Image** (IS Mai 2001)
  - Teil 6: SQL/MM **Data Mining** (Draft)
- Teil 1 gibt Überblick und spezifiziert Konformität
- jeder weitere Teil
  - ist ein Paket für eine Art von Mediendaten
  - besteht aus UDT's, Methoden und Funktionen gemäß SQL:1999

SQL/MM Full Text

- Version vom 10.12.2001
- spezifiziert
  - UDT **FullText** für Text-Daten und
  - UDT **FT_Pattern** für Suchmuster
- FullText:
  - vier Suchmethoden;
    - zwei unterscheiden sich jeweils nur im Parameter: Zeichenkette oder Muster vom Typ FT_Pattern (Overloading)
    - Contains-Methoden: Boolesche Suche ⇒ Ergebnis: ja/nein
    - Rank-Methoden: Ranking ⇒ Ergebnis: impl.-abh. Real-Wert
  - zwei Konstruktoren (Zeichenkette, Zeichenkette + Sprache)
  - Funktion FullText_to_Character zum Erzeugen einer Zeichenkette

SQL/MM Full Text: UDT-Definitionen

```
create type FullText as (
   Contents character varying(FT_MaxTextLength),
   Language character varying(FT_MaxLanguageLength),
   …
   )
method Contains (pattern FT_Pattern) returns integer
method Contains (
   pattern character varying(FT_MaxPatternLength)
   ) returns integer
method Rank (pattern FT_Pattern)
   returns double precision
method Rank …
```

SQL/MM Full Text: UDT-Definitionen (2)

```
method FullText (
   String character varying(FT_MaxTextLength)
   ) returns FullText
method FullText (
   String ... ,
   Language character varying(FT_MaxLanguageLength)
   ) returns FullText;
create cast (FullText as
   character varying(FT_MaxTextLength)
   with FullText_to_Character);
create type FT_Pattern as
   character varying(FT_MaxPatternLength);
```

- Werte von FT_Pattern müssen Ausdrücke einer in BNF beschriebenen Sprache sein
- Auswertung durch Regeln über Symbolen der Sprache beschrieben

SQL/MM FullText: Suchmuster für Contains und Rank

- Textbeispiel

  aText: „In diesem Abschnitt wird der Standard SQL/MM vorgestellt. Dieser Standard definiert Typen und Routinen für Medienobjekte.“
- einzelnes Wort

  ```
  aText.Contains (' "Abschnitt" ') = 1
  ```
- Menge von Worten
  - Wildcards

    ```
    aText.Contains (' "Abschnitt_" ') = 0
    ```
  - Erweiterungsmuster (ähnliche Worte, allgemeinere W., speziellere W., Synonyme, Abstammung)

    ```
    aText.Contains ('
       thesaurus "Informatik"
       expand synonym term of "Norm"
    ') = 1
    ```

SQL/MM FullText: Suchmuster für Contains und Rank (2)

- Kontextmuster

```
aText.Contains ('
   ("Abschnitt") near "Standard" within 0 sentences in order
') = 1
```

- Konzeptmuster

```
aText.Contains ('
   is about "Internationaler Standard zur Volltextsuche"
') = 1
```

  - einzelne Phrase, Aufzählung von Einzelwortmustern, Mengen von Phrasen, Muster mit Booleschen Operatoren (I, &, NOT)

- Beispielanfrage:

```
select * from myDocs
  where Doc.Rank(' "Standard" ') > 0.8
```


SQL/MM FullText: Berücksichtigung der Sprache

- zu jedem Text kann eine Sprache angegeben werden (siehe Definition)
- zu einigen Mustern kann ebenfalls eine Sprache angegeben werden
- wozu?
  - Erkennung von Wort-, Satz- und Absatzgrenzen
  - richtige Expansion, z. B. für ähnliche Worte
  - Behandlung von Stoppworten (engl. 'die' vs. dt. 'die')
  - **Textbeispiel** wieder aText: „In diesem Abschnitt wird der Standard SQL/MM vorgestellt. Dieser Standard definiert Typen und Routinen für Medienobjekte."

    ```
    aText.Contains (' ("Typen oder Routinen") ') = 1
    ```

  - Wortnormalisierung:
    „Müller" wird ersetzt durch „Mueller"

SQL/MM Still Image

- Version vom 10.12.2001
- spezifiziert
  - UDT **SI_StillImage** für Bilddaten,
  - UDT **SI_Feature** für Merkmale und
  - UDT **SI_FeatureLis**t für Listen von Merkmalen
- SI_StillImage:
  - interne Repräsentation offen gelegt (⇨ keine Datenunabhängigkeit)
  - zwei Konstruktoren (BLOB, BLOB + Format)
  - zwei Mutator-Methoden: BLOB-Ersetzung + Formatänderung
  - zwei Observer zur Erzeugung von Miniaturen („Thumbnails")

SQL/MM Still Image: UDT SI_StillImage

```
create type SI_StillImage as (
   SI_content binary large object(SI_MaxContLength),
   SI_contentLength integer,
   SI_format character varying(8),
   SI_height integer,
   SI_width integer,
   …
   )
```

- SI_content:
  - umfasst auch Registrierungsdaten (Header, Farbtabellen usw.)
  - „Container" für das ganze Bild
- SI_format:
  - unterstützte Formate (das DBS kann sie lesen und Bildeigenschaften extrahieren)
  - benutzerdefinierte Formate

SQL/MM Still Image: UDT SI_StillImage (2)

```
method SI_StillImage (
   content binary large object(SI_MaxContLength)
   ) returns SI_StillImage
method SI_StillImage (
   content binary large object(SI_MaxContLength),
   format character varying(…)
   ) returns SI_StillImage
method SI_setContent (
   content binary large object(SI_MaxContLength)
   ) returns SI_StillImage
method SI_changeFormat (
   targetFormat character varying( … )
   ) returns SI_StillImage
```

nur für unterstützte Formate

SQL/MM Still Image: Merkmale (Features)

- Basistyp SI_Feature hat die Subtypen:
  - **SI_AverageColor**: eine einzige Farbe für das ganze Bild
  - **SI_ColorHistogram**: Häufigkeiten für Gruppen von Farben (s. Kap. 4.3)
  - **SI_PositionalColor**: Zerlegung des Bildes in Rechtecke mit Durchschnittsfarbe
  - **SI_Texture**: Größe von wiederholten Elementen, Helligkeitsvariation, dominierende Richtung
- alle Merkmale haben die Methode SI_Score, die
  - die Distanz eines Bildes zum Merkmal berechnet und
  - einen Real-Wert zwischen 0 und 1 zurückgibt
- alle Subtypen von SI_Feature haben eine Funktion zur Merkmalsextraktion
- Objekte von SI_AverageColor und SI_ColorHistogram können direkt konstruiert werden (aus Konstanten)

SQL/MM Still Image: Merkmale (2)

```
create type SI_Feature
method SI_Score (image SI_StillImage)
   returns double precision
create type SI_AverageColor under SI_Feature as
   (SI_AverageColorSpec SI_Color)
method SI_AverageColor (
   RedValue integer,
   GreenValue integer,
   BlueValue integer
   ) returns SI_AverageColor
create function SI_AverageColor (image SI_StillImage)
   returns SI_AverageColor
```


SQL/MM Still Image: Merkmalsliste

- Liste von Merkmal-Wert-Paaren
- Methode SI_Score liefert gewichteten Mittelwert

```
self.SI_Features[1].SI_Score(img) * self.SI_Weights[1]
+ self.SI_Features[2].SI_Score(img) * self.SI_Weights[2] + …
/ (self.SI_Weights[1] + self.SI_Weights[2] + … )
```

```
create type SI_FeatureList as (
SI_Features SI_Feature array[SI_MaxFeatureNumber],
SI_Weights double precision array[SI_MaxFeatureNumber]
)
method SI_FeatureList (firstFeature SI_Feature, weight
double precision) returns SI_FeatureList
method SI_Append (feature SI_Feature, weight double
precision) returns SI_FeatureList
```

SQL/MM Still Image: Beispiel

```
select * from Logos where
   SI_FeatureList (
      SI_Texture (SI_StillImage(:bspLogo)), 0.8
      ).SI_Append (
         SI_ColorHistogram (SI_StillImage(:bspLogo)), 0.2
         ).SI_Score (Logo) > 0.7
```


SQL/MM – Ein SQL-Standard für Medienobjekte

- Schlussbemerkung
  - erst drei Teile standardisiert: FullText, Spatial, Still Image
  - für Video und Audio keine Teile in Sicht?
  - merkwürdige Uneinheitlichkeit
    (Rank bei FullText, Score bei StillImage) –
    warum nicht Generalisierung zu MM_Object o. ä.?
- Fragen
  - wird der Standard umgesetzt?
    - [Stol01a] zeigt, wie es bei Still Image mit DB2 gehen könnte
  - wie groß ist der Unterschied zu den bereits existierenden Paketen?