Chapter 8 – Data Models for Media Objects

Overview

- Requirements and general approach
  - support for large media objects
  - media object types
  - data and relationship modeling
- (Object-) Relational DBMS
  - LOBs vs. external storage
  - user-defined types and routines
  - SQL/MM
- Object-oriented DBMS
Support for Multimedia Data Types

Main aspects regarding data model support
1. Support for managing large media objects
   - managing large data objects inside DBMS
   - file-based storage of media objects
     - managed by file system or DBMS?
     - only as "infrastructure", not sufficient!
2. Introduction of new data types
   - TEXT, GRAPHICS, IMAGE, SOUND, VIDEO
   - including applicable operations
     - "abstract data type" (ADT)
3. Inclusion in existing data models
   - relational
   - object-relational
   - object-oriented
   ➤ Usage of existing modeling constructs and query languages

Basic Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>+, -, *, /, ... : integer × integer → integer</td>
</tr>
<tr>
<td></td>
<td>=, ≠, ≤, ≥, ... : integer × integer → boolean</td>
</tr>
<tr>
<td>char</td>
<td>operations:</td>
</tr>
<tr>
<td></td>
<td>conversion to/from integer, output (print), ...</td>
</tr>
<tr>
<td>boolean / bit</td>
<td>operations:</td>
</tr>
<tr>
<td></td>
<td>and, or, ...</td>
</tr>
</tbody>
</table>

i.e., types are defined through supported operations!
Composite Types / Type Constructors

(*"generic" or "parameterized" types*)

- **listOf Typ (min, max)**
  - operations:
    - determine length, access elements, concatenate, sublist, ...
  - examples:
    - byte = listOf boolean (8,8)
    - string = listOf char (0,*)
  - canonical continuation of all element-level operations:
    - List3 := List1 * List2

- **setOf Typ (min, max)**
  - operations:
    - element count, for each, union, difference, add/remove element, ...

Text Data Type

- Applicable operations (in Java notation):
  - read access:
    ```java
    interface Text {
        public int length ();
        public int alphabet ();
        // 0 == ISO Latin-1, ...
        public int alphabetSize ();
        public int language ();
        // 0 == English, 1 == German, ...
        public int charAt (int n);
        public byte [] getASCII ();
        public byte [] getEBCDIC ();
        public String getUnicode ();
    }
    ```
  - with whitespace, end of line:
    ```java
    public byte [] word (int wordNo);
    public byte [] line (int lineNo);
    public int wordCount ();
    public int lineCount ();
    ```
  - complete text (e.g., print, display):
    ```java
    public boolean print (Printer p);
    public boolean display (Window w);
    ```
  - modification (preserving consistency):
    ```java
    public void replaceLine (int lineNo, byte [] newLine);
    public void insertLine (int lineNo, byte [] newLine);
    public void concatenate (Text t2);
    ```
  - General problem: procedure or function?
    ```java
    public Text replaceLine (int lineNo, byte [] newLine);
    ```
    ```java
    ```
Text (2)

- Create:
  
  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
      } { ... };
      ...
  }

  or in a specific context:
  
  public TextClass (String filename) { ... };

- Similar model/interfaces for image, audio, video

Description Data and Comparisons

- Content description
  - dependent component, constituents of the media object
  - requires additional operations
    - add, extend
    - read length, content
    - content search (over description)
  - Example: Image
    
    interface Image {
        ...
        public void newDescr (String descr);
        public void extendDescr (String descr);
        public int descrLength ();
        public String getDescr ();
        public boolean contains (String query);
    }
Generalization

- Building a generalization hierarchy
  - generic operations defined in MediaObject supertype
  - refinement of media object types
- Example: Image
  - specialization as Bitmap, Greyscale, ColorImage
  - operations for conversion
- Additional aspects
  - disjointness, completeness
  - subtype-specific constructors
- Other, application-oriented refinements possible

Object-Relational DBMS

- Major development to extend relational DBMS to introduce object-oriented concepts into the relational data model and query language
- Main concepts
  - support for large objects and external data
  - composite data types (row types, collection types)
  - user-defined data types
    - distinct types: strong typing
    - structured data types for complex, nested data structures
    - type hierarchies with inheritance
    - typed table hierarchies
    - restricted notion of object identity
  - user-defined routines
    - stored procedures, user-defined functions, methods for structured types
    - overloading, overriding, dynamic binding
    - implementation using procedural SQL (PSM) or external programming language
- Systems: (University-)Ingres (1984), Postgres, Starburst, Illustra/Informix, DB2, Oracle
Large Object Data Types

- LOBs store strings of up to gigabytes maintained directly in the database
- There are 2 LOB data types
  - BLOB - Binary Large Object
  - CLOB - Character Large Object
- LOB size can be specified at column definition time (in terms of KB, MB, GB)
- CREATE TABLE Booktable
  - title VARCHAR(200),
  - book_id INTEGER,
  - summary CLOB(32K),
  - book_text CLOB(20M),
  - movie BLOB(2G))
- Internally, LOBs are usually stored in a separate storage space
  - i.e., out-of-line storage
- LOBs may be retrieved, inserted, updated like any other type
  - excluded from some operations
- Functions that support LOBs
  - CONCATENATION string1 || string2
  - SUBSTRING (string FROM start FOR length)
  - LENGTH (expression)
  - POSITION (search-string IN source-string)
  - NULLIF/COALESCE
  - TRIM
  - OVERLAY
  - Cast
  - LIKE predicate

LOBs and Application Programs

- LOBs may be unmanageable in application programs
  - huge amounts of storage may be needed to buffer their values
  - applications may want to deal with LOBs a piece at a time
- LOB locators
  - 4-byte value stored in a host variable that a program can use to refer to a LOB value
    - LOB still resides in the SQL server
    - a locator may be used anywhere a LOB value can be used
  - allows application to work with LOBs a piece at a time
- Example:
  - EXEC SQL BEGIN DECLARE SECTION;
  - SQL TYPE IS BLOB AS LOCATOR movie_loc;
  - EXEC SQL END DECLARE SECTION;
  - EXEC SQL
    - SELECT movie
      INTO :movie_loc
      FROM BOOKTABLE
    WHERE title = 'Moby Dick'
Locators on LOB Expressions

Locators may also represent LOB expressions
- A LOB expression is any expression that refers to a LOB column or results in a LOB data type
  - May include LOB functions
  - May even reference other locators

Implementation
- "Smart LOB" support will avoid unnecessary operations/copies on LOBs
- Only store a "recipe" (i.e., script of operations)
- Materialize only if required
  - E.g., Chapt1Loc is used in UPDATE, INSERT, or retrieved into memory buffer

Example: select chapter 1 into locator

```sql
SELECT SUBSTRING(book_text,
    POSITION('Chapter 1' IN book_text),
    POSITION('Chapter 2' IN book_text) -
    POSITION('Chapter 1' IN book_text))
FROM Booktable
INTO :Chapt1Loc
WHERE title = 'Moby Dick';
```

Managing External Data: Datalinks

Applications

SQL API requests

File API requests

Employee Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Birth Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ted</td>
<td>1950</td>
</tr>
<tr>
<td>Peter</td>
<td>1961</td>
</tr>
<tr>
<td>Abby</td>
<td>1956</td>
</tr>
<tr>
<td>John</td>
<td>1948</td>
</tr>
</tbody>
</table>

Datalinker
DataLinks in SQL/MED

- **Goal**
  - preserve external storage, manipulation of files
  - synchronize integrity control, recovery, and access control of files and associated SQL data

- **Concepts**
  - datalink is an instance of the DATALINK data type
    - references a file (URL) that is not stored by the SQL server, but maintained by an external file manager
  - datalink options (per column)
    - define the amount of management and control the SQL server has over the datalink values of a column
    - integrity, read/write access, recovery
    - specifies the semantics of link/unlink behavior
  - datalinker
    - implementation-dependent
    - implements a number of mechanisms for guaranteeing datalink properties such as integrity control, recovery, access control

Functions and Operations

- **New SQL functions for datalinks**
  - constructor: DLVALUE, ...
  - (components of) URLs: DLURLCOMPLETE, ...

- **SQL statements (examples)**
  - insert ("link")
    ```sql
    INSERT INTO Movies (Title, Minutes, Movie)
    VALUES ('My Life', 126, DLVALUE('http://my.server.de/movies/mylife.avi'))
    ```
  - select (incl. URL access token)
    ```sql
    SELECT Title, DLURLCOMPLETE(Movie)
    FROM Movies
    WHERE Title LIKE '%Life%'
    ```
Data Link Options

- Link control (NO, FILE)
  - NO LINK CONTROL
    - URL-Format of datalink
      - no further control, file is not "linked"
  - FILE LINK CONTROL
    - file is "linked", file has to exist!
    - level of control can be specified using further options

- Integrity control option (ALL, SELECTIVE, NONE)
  - INTEGRITY ALL
    - linked files cannot be deleted or renamed
  - INTEGRITY SELECTIVE
    - linked files can only be deleted or modified using file manager operations, if no datalinker is installed
  - INTEGRITY NONE
    - referenced files can be deleted or modified using file manager operations
      - not compatible with FILE LINK CONTROL

Data Link Options (continued)

- Read permission option (FS, DB)
  - READ PERMISSION FS
    - read access is determined by file manager
  - READ PERMISSION DB
    - read access is controlled by SQL server, based on access privileges to the datalink value
      - involves read access tokens
        - encoded into the URL by the SQL server
        - verified by external file manager/data linker

- Write permission option (FS, ADMIN, BLOCKED)
  - WRITE PERMISSION FS
    - write access controlled by file manager
  - WRITE PERMISSION BLOCKED
    - linked files cannot be modified
  - WRITE PERMISSION ADMIN [NOT] REQUIRING TOKEN FOR UPDATE
    - write access governed by SQL server (and datalinker)
      - requires READ PERMISSION DB
      - involves write access token for modifying file content
        - may have to be presented to the SQL server again
Functions and Operations (continued)

- “Update-in-place”
  ```
  SELECT Title, DLURLCOMPLETEWRITE(Movie)
  INTO :t, :url ...
  open using URL, modify ...
  UPDATE Movies SET Movie = DLNEWCOPY(:url, 1)
  WHERE Title = :t
  ```

- DLNEWCOPY
  - indicates to the SQL server that the file content has changed and should be managed appropriately
  - alternative: DLPREVIOUSCOPY – file content may have changed, but the application is not interested in keeping the changes, original file is restored

Data Link Options (continued)

- RECOVERY YES/NO
  - indicates whether SQL server coordinates recovery (jointly with datalinker) or not

- Unlink option (RESTORE, DELETE, NONE)
  - ON UNLINK RESTORE
    - original properties (ownership, permissions) restored as well
  - ON UNLINK DELETE
    - file is deleted when unlinked
  - ON UNLINK NONE
    - ownership and permissions are not restored

- SQL statement (example)
  ```
  "Unlink/Replace"
  UPDATE Movies SET Movie = DLVALUE('http://my.newserver.de/mylife.avi')
  WHERE Title = "My Life"
  ```
  RESTORE or DELETE for ".../movies/mylife.avi"
Valid Combinations

<table>
<thead>
<tr>
<th>Integrity</th>
<th>Read permission</th>
<th>Write permission</th>
<th>Recovery</th>
<th>Unlink</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>FS</td>
<td>FS</td>
<td>NO</td>
<td>NONE</td>
</tr>
<tr>
<td>ALL</td>
<td>FS</td>
<td>BLOCKED</td>
<td>NO</td>
<td>RESTORE</td>
</tr>
<tr>
<td>ALL</td>
<td>FS</td>
<td>BLOCKED</td>
<td>YES</td>
<td>RESTORE</td>
</tr>
<tr>
<td>ALL</td>
<td>DB</td>
<td>BLOCKED</td>
<td>NO</td>
<td>RESTORE</td>
</tr>
<tr>
<td>ALL</td>
<td>DB</td>
<td>BLOCKED</td>
<td>NO</td>
<td>DELETE</td>
</tr>
<tr>
<td>ALL</td>
<td>DB</td>
<td>BLOCKED</td>
<td>YES</td>
<td>RESTORE</td>
</tr>
<tr>
<td>ALL</td>
<td>DB</td>
<td>BLOCKED</td>
<td>YES</td>
<td>DELETE</td>
</tr>
<tr>
<td>ALL</td>
<td>DB</td>
<td>ADMIN</td>
<td>NO</td>
<td>RESTORE</td>
</tr>
<tr>
<td>ALL</td>
<td>DB</td>
<td>ADMIN</td>
<td>NO</td>
<td>DELETE</td>
</tr>
<tr>
<td>ALL</td>
<td>DB</td>
<td>ADMIN</td>
<td>YES</td>
<td>RESTORE</td>
</tr>
<tr>
<td>ALL</td>
<td>DB</td>
<td>ADMIN</td>
<td>YES</td>
<td>DELETE</td>
</tr>
<tr>
<td>SELECTIVE</td>
<td>FS</td>
<td>FS</td>
<td>NO</td>
<td>NONE</td>
</tr>
</tbody>
</table>

User-defined Types: Key Features

- **New functionality**
  - Users can indefinitely increase the set of provided types
  - Users can indefinitely increase the set of operations on types and extend SQL to automate complex operations/calculations
- **Flexibility**
  - Users can specify any semantics and behavior for a new type
- **Consistency**
  - Strong typing insures that functions are applied on correct types
- **Encapsulation**
  - Applications do not depend on the internal representation of the type
- **Performance**
  - Potential to integrate types and functions into the DBMS as "first class citizens"
User-defined Structured Types (ST)

- User-defined, complex data types
  - definition of state (attributes) and behavior
  - can be used as data type wherever predefined data types can be used
    - type of domains or columns in tables
    - attribute type of other structured types
    - type of parameters of functions, methods, and procedures
    - type of SQL variables
  - strong typing
- Structured Types can be used to define typed tables
  - types and functions for rows of tables
    - for modeling entities with relationships & behavior
  - explicit object identifier column to support flavor of "object identity"

```sql
CREATE TYPE employee AS (id INTEGER, name VARCHAR (20))
```

Creating and Using Structured Types

- Create structured type:
  ```sql
  CREATE TYPE address AS (street CHAR (30), city CHAR (20), state CHAR (2), zip INTEGER) NOT FINAL
  ```
- Create table using type for column:
  ```sql
  CREATE TABLE properties (price DECIMAL(9,2), owner VARCHAR (50), addr address)
  ```
- Insert structured values using the NEW operator
  ```sql
  INSERT INTO properties VALUES (... , NEW address , ...)
  ```
- Access attributes using dot-notation
  ```sql
  SELECT p.addr.street, p.addr.city, p.addr.state, p.addr.zip
  FROM properties p
  WHERE price < 100000
  ```
- Update attributes
  ```sql
  UPDATE properties
  SET addr.city = 'Los Angeles'
  WHERE addr.city = 'LA'
  ```
Type Hierarchies and Value Substitutability

- Create type hierarchy:
  ```
  CREATE TYPE german_addr UNDER address (family_name VARCHAR(30)) NOT FINAL
  CREATE TYPE brazilian_addr UNDER address (neighborhood VARCHAR(30)) NOT FINAL
  CREATE TYPE us_addr UNDER address (area_code INTEGER, phone INTEGER) NOT FINAL
  CREATE TYPE us_bus_addr UNDER us_address (bus_area_code INTEGER, bus_phone INTEGER) NOT FINAL
  ```

- Each row can have a column value of a different subtype!
  ```
  INSERT INTO properties (price, owner, location)
  VALUES (100000, 'Mr.S.White', NEW us_addr ('1654 Heath Road', 'Heath', 'OH', 45394, ...))
  INSERT INTO properties (price, owner, location)
  VALUES (400000, 'Mr.W.Green', NEW brazilian_addr ('245 Cons. Xavier da Costa', 'Rio de Janeiro', ..., 'Copacabana'))
  INSERT INTO properties (price, owner, location)
  VALUES (150000, 'Mrs.D.Black', NEW german_addr ('305 Kurt-Schumacher Strasse', 'Kaiserslautern', 'Schwarz'))
  ```

Routines: Procedures, Functions and Methods

- Procedure
  ```
  CREATE PROCEDURE getPropertiesCloseTo
  (IN addr VARCHAR(50),
   IN distance INTEGER,
   OUT results INTEGER) ...
  ```

- Functions
  ```
  CREATE FUNCTION distance(loc1 address, loc2 address)
  RETURNS INTEGER ...
  ```

- Methods
  ```
  CREATE TYPE address AS (street CHAR (30),
                           city CHAR (20),
                           state CHAR (2),
                           zip INTEGER)
  METHOD longitude() RETURNS DECIMAL(5, 2)
  METHOD latitude() RETURNS DECIMAL(5, 2)
  ```

- Invocation similar to function, but using method invocation syntax
  ```
  SELECT price, addr, addr.longitude(), addr.latitude()
  FROM properties
  ```
Combined Search Using UDTs/UDFs

- User-defined types can be for representing media object types
  - user-defined functions or methods for content search predicates
- Query may range over "traditional" and multimedia data at the same time

Methods That Modify Object State

- In OO-programming
  - a method that wants to modify the state of its object simply assigns new values to an attribute
  - changes are reflected in the identical object
- In SQL
  - value-based operations
    - expressions (including method invocations) always return (copies of) values
    - persistent data can only be updated by the respective DML operations (e.g., UPDATE), assigning the results of expressions to the columns to be modified
    - a method will always operate on a copy of a complex value (i.e., instance of a structured type)
    - modification of state as a pure side-effect of a method is not possible
    - modifying the object state will require an UPDATE statement
    - method returns a modified copy of the original complex value
    - separate UPDATE replaces old value with the new copy
SQL Collection Types

- Two kinds of collection types
  - Array (with optional maximum length)
  - Multiset
- Collections are typed
  - all elements are instances of the specified element type
  - any element type admissible (including user-defined types and collection types)
- Construction of collections
  - by enumeration
  - by query
- UNNESTing of collections to access elements
- Manipulation of collections
  - general: cardinality
  - arrays: element access, concatenation
  - multisets: turn singleton into element, turn into set (eliminate duplicates), multi-set union, intersection, difference
- Multiset predicates (member, submultiset, is a set)
- Collections can be compared, assigned, cast

Working with Collection Types and Values

- Example (for multisets):
  ```sql
  CREATE TABLE properties ( ..., owners VARCHAR(50) MULTISET, ...)
  ```
- Constructing multisets
  - by enumeration
  - by query
  ```sql
  UPDATE properties
  SET owners = MULTISET
      (SELECT name
       FROM people WHERE ...)
  WHERE ...
  ```
- Using multisets as table references (UNNEST)
  ```sql
  SELECT p.price, p.addr.print(), o.name FROM properties p,
  UNNEST(p.owners) AS o(name)
  WHERE o.name LIKE %Schmidt%
  ```
SQL/MM – an SQL-Standard for Media Objects

- Media objects and operations are used in many applications
- ORDBMS extensibility concepts can be used to define media object types
- Extensions can be provided in "packages"
  - easier management (installation, upgrade, removal) and reuse (package can use another package)
- Proprietary packages offered for numerous ORDBMS products
  - 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
- Standardization, based on SQL:1999, SQL:2003:
  - common "language"
  - portability of applications
  - data exchange

SQL/MM Overview

- Refers to SQL-Standard, but stand-alone
  - full name: SQL Multimedia and Application Packages
- Consists of multiple parts
  - Part 5: SQL/MM Still Image (2001)
  - ... 
- Part 1 provides overview, conformance details
- Every other part
  - represents a "package" for specific type of media data
  - contains UDTs, methods, functions based on SQL:1999
SQL/MM Full Text

- Version as of December 2001
- specifies
  - UDT FullText for text data and
  - UDT FT_Pattern for search patterns
- FullText:
  - four search methods
    - two method pairs, each pair differing only regarding search parameter type: character string or pattern if type FT_Pattern (Overloading)
    - Contains methods: boolean search ⇒ result: true/false
    - Rank methods: ranking ⇒ result: implementation-defined value of type REAL
  - two constructors
    - character string
    - character string, language
  - function FullText_to_Character two produce character string from FullText value
- Language can be defined for FullText and some of the search patterns
  - used for language-specific processing during text preprocessing (see chapter 3)

UDT Definitions

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 ...

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

- FT_Pattern values have to comply with BNF of pattern language
- Search semantics "constrained" by a set of rules
Search Patterns for Contains and Rank

- Text example
  
  aText: "This paragraph introduces the SQL/MM standard. This standard defines types and routines for media objects."

- Single word
  
  aText.Contains('"paragraph"') = 1

- Sets of words
  
  - wildcards
    
    aText.Contains('"media_"') = 0
  
  - thesaurus-based extension
    
    aText.Contains('thesaurus "CompSci"
    
    expand synonym term of "norm"
    
    ') = 1

Search Patterns for Contains and Rank (2)

- Context pattern
  
  aText.Contains('"paragraph" near "standard" within 0 sentences in order

  ') = 1

- Concept pattern
  
  aText.Contains('is about "International Standard for Fulltext Search"

  ') = 1

- Single phrase, combination of word/phrase search, arbitrary patterns using boolean operators (|, &, NOT)

- Example query:
  
  select * from myDocs
  
  where Doc.Rank('"standard" ') > 0.8
SQL/MM Spatial

- Version as of December 2001 (581 pages)
- Corresponds to graphics media type
- Specifies UDTs for
  - 2D data (point, line, polygon)
  - collections thereof
- Defines routines
  - manipulation, search, comparison of spatial data
  - conversion among UDTs and character/binary representations
- Geometry objects (ST_Geometry) are associated with SRIDs (spatial reference system identifiers) that identify a spatial reference system
  - based on well-known reference systems
    - geographic coordinate system (long/lat)
    - projection coordinate systems: X, Y
    - geo-centric coordinate system: X, Y, Z

SQL/MM Spatial: Types

- 0-dim: ST_Point
- 1-dim: ST_Curve
  - subtypes differ in the interpolation between element points
    - ST_LineString: linear interpolation
    - ST_CircularString: circular interpolation
    - ST_CompoundString: mix of both
- 2-dim: ST_Surface
  - ST_CurvePolygon: 1 external + n internal ST_Compound-String boundaries
  - ST_Polygon: only ST_LineString boundary
- collection objects
  - same reference system for all elements
    - ST_MultiPoint
    - ST_MultiCurve, ST_MultiLineString
    - ST_MultiSurface, ST_MultiPolygon
SQL/MM Spatial Methods

- ST_Geometry methods:
  - intersection (sets of points), difference, union
  - distance
  - tests (contains, overlaps, touches, crosses, ...)
  - determine reference system
- additional methods on subtypes
  - ST_Curve: length
  - ST_Surface: area, perimeter

SQL/MM Still Image

- Version as of December 2001
- Specifies
  - UDT SI_StillImage for image data,
  - UDT SI_Feature for image features
  - UDT SI_FeatureList for lists of features
- SI_StillImage:
  - internal representation is revealed (∞ no format independence)
  - two constructors (BLOB, BLOB + format)
  - two mutator (modification) methods: BLOB replacement + format change
  - two observer (read) methods for generating thumbnail images
**SQL/MM Still Image: UDT SI_StillImage**

```sql
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:**
  - also contains registration data (header fields, color map, etc.)
  - "container" for the complete image

- **SI_format:**
  - built-in formats (DBS can read and change format, extract properties and features)
  - user-defined formats

```n

**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**

**SQL/MM Still Image: Features**

- Types SI_Feature has the following subtypes (also see chapter 4):
  - **SI_AverageColor**: single color for complete image
  - **SI_ColorHistogram**: percentages for groups of colors
  - **SI_PositionalColor**: grid segments with average segment color
  - **SI_Texture**: texture information

- Features have a methods **SI_Score** for
  - computing the distance of an image to the feature, and
  - returning a REAL value in the range 0 to 1

- Subtypes of SI_Feature have functions for performing feature extraction
- Instances of SI_AverageColor and SI_ColorHistogram can also be constructed explicitly using literal values
SQL/MM Still Image: UDTs for Features

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

List of feature-value-pairs
SI_Score returns weighted average:
self.SI_Features[1].SI_Score(img) * self.SI_Weights[1]
+ …

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

Example:
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 – Closing Remarks

- Three parts standardized: FullText, Spatial, Still Image
  - no ongoing standardization work for video, audio
- Some inconsistencies (Rank for FullText, Score for StillImage)
- No full generalization pursued
  - MM_Object type with media object subtypes?
- Questions
  - are vendors implementing the standard?
  - how big are the differences compared to existing vendor packages?
Object-Relational Schema

- Text, Image, ... are possible atomic domains, i.e. attributes can be of type Text, Image, ...
- Example:
  - Employee (EmpNo integer, ...
    Photo image, Fingerprint image)
  - Inmate (I-Nr integer, ...
    Front image, Side image)
  - Car (Make varchar(50), Year integer, ...
    Photo image, EngineSound audio)
- 1:1-relationship, attribute relationship

1:N Relationship Involving Media Objects

- Alternative 1: based on foreign key
  - use separate relation (1NF) for variable number of texts, images, etc. per entity
    Patient (Name varchar(100), ...
    XRay (PName varchar(100), date, ...
      Position varchar(30), BodyPart varchar(40), Picture)
  - PName is a primary key component, i.e., XRays only for known patients
- Access:
  - to retrieve patient data with patient x-rays: requires join operation

- Alternative 2: using collection types, structured types
  - Patient (Name varchar(100), ...
    Picture image, XRays XRay MULTISET)
  - CREATE TYPE XRay ...
  - Again, XRays without a patient cannot exist
- Access:
  - to work with individual images: requires UNNEST
N:M-Relationships

- Example: picture may show more than one entity: relation for images, relationship
  
  | Horse       | (Name varchar(50), Age integer) |
  | RacePhoto   | (Archivnr integer, Date date, Location varchar(80), Picture image) |
  | Is_deicted_on | (Horsename varchar(50), Archivnr integer, Position varchar(10)) |

- Photos are separate entities, i.e., they can be stored without an association to a subject (here: horses)

- Access:
  - two (usually expensive) join operations

- Alternative approach for binary relationships: use collection types
  - e.g., multiset of archive numbers, multiset of horses

Summary for Object-Relational DBMS

- LOBs vs. file-based storage (datalinks)
  - management of large amounts of (raw) data
    - datalinks preserve file-based access, but allow gradual control/management by DBMS
    - no media object type semantics, operations

- User-defined, structured types
  - important concept for defining media object types (structure and behavior)
  - enablement for SQL/MM standardized types

- Relationships
  - different types of relationships (1 : 1, 1 : N, N : M) between MM objects and entities can be represented
  - Media objects can be represented as attributes or entities
    - may be complicated if different types of entities are involved (ship, car, airplane, ...)
    - may require multiple relationship tables

- Advantages
  - stable, proven environment
    - smooth learning curve, upward compatibility
Object-oriented DBMS Extensions

- MM data objects are instances of classes
- Class hierarchy and inheritance
  - simplified data modeling
  - more integrity control
  - inheritance, specialization, overriding
- Applications
  - may pursue definition of subclasses (extensibility)
  - “benefit” from MM methods available in the MM classes

Example: ORION

- Overview:
  - MCC (Austin, Texas)
  - since 1985: development of a Multimedia-DBMS
  - early decision for fully object-oriented architecture
  - prototype implementation in Common LISP on Symbolics and SUN, commercial product (ITASCA), but little traction
- Multimedia Information Manager (MIM)
  - packages of classes and methods ("class library" under ORION)
  - extensible:
    - developers can provide subclasses for special formats
- Unique approach:
  - even devices (I/O, storage) modeled as objects
Output Devices

Instances of classes in the hierarchy also describe
- presentation position on the device (e.g., position on screen)
- which part of the MM object is presented

There may be multiple instances representing the same physical device
- "presentation formats" for media objects

Input Devices

Instances are again more than specific devices:
- which part of the MM object is captures
- device configuration
**Input and Output Devices – More Details**

- **spatial-pres-device**
  - attributes: `upper-left-x`, `upper-left-y`, `width`, `height`
  - (section of MM object)

- **screen-window** (subclass of spatial-pres-device)
  - attributes: `win-upper-left-x`, `win-upper-left-y`, `win-width`, `win-height`
  - (area on the screen)
  - methods: `present`, `capture`, `persistent-pres`

- **spatial-capture-device**
  - attributes: `upper-left-x`, `upper-left-y`, `width`, `height`

- **image-capture-device** (subclass of spatial-capture-device)
  - attributes: `cam-width`, `cam-height`, `bits-per-pixel`
  - methods: `capture`

---

**Stored Objects**

- **Attributes of captured-object:**
  - storage-object: refers to instance of class `storage-device`
  - logical-measure: elementary unit for raw data from user perspective, e.g., seconds for audio, frame for video
  - phys-logic-ratio: bytes per second, etc.

- **Attributes of spatial-captured-object:**
  - width
  - height
  - row-major: row/column storage
  - bits-per-pixel
  - (registration data)
Storage Devices

- Describe only storage aspects used by MM objects
- Attributes of mag-disk-storage-device:
  - block-list: block number of all physical blocks occupied by MM objects
  - allocated-block-list: blocks actually allocated (see versions)
  - min-object-size-in-disk-pages: number of blocks to be added when MM object has to grow
  - seg-id: disc segment hosting new blocks

Input/Output Streams

- Instances represent a read or write operation
  - generated dynamically
- Attribute of disk-stream:
  - storage-object: references an instance of storage-device
- Attribute of read-disk-stream:
  - read-block-list: cursor; next block to read for the MM objects
  (similar for write-disk-stream)
Details of Output Operation (Example)

Application: Car

Orion:
- image-pres-device
- captured-image
- read-disk-stream
- storage-object

Mapping:
- present
- start-offset
- stream
- address

Application:
- Orion: show-picture

Summary for Object-Oriented DBMS

- Complete proposal for a MMDBS
- Application developer extends system by adding new subclasses
- Open questions:
  - Is the class hierarchy adequate? Methods?
    - e.g., classification of MM objects as 1D (linear) and 2D (spatial)
    - why not visual vs. acoustic
    - or time-dependent vs. static?
  - many alternatives:
    - device x : show image Y
    - image y : present on device X
    - evaluation criteria?
  - Is differentiation of magnetic vs. optical disc required?
    - more abstract: random access memory, sequential storage, write-once-storage, ...
  - Search?