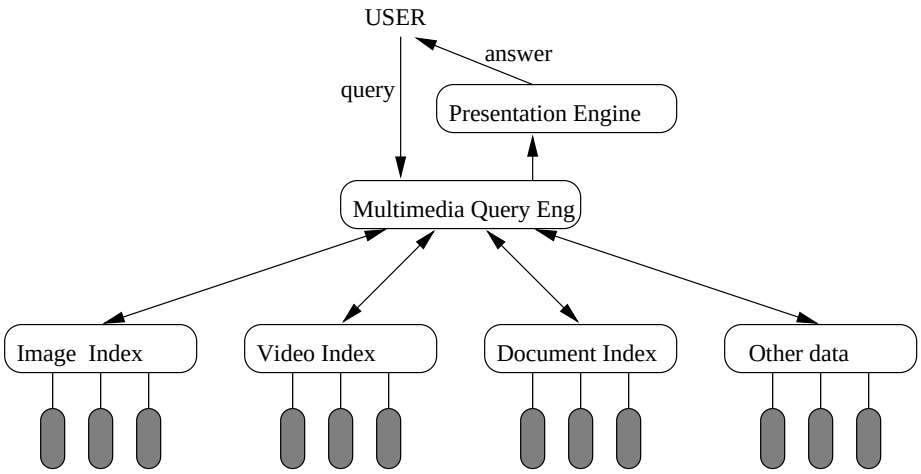
Multimedia Databases

- Thus, far we have shown how to represent and organize the content of different types of *individual* media data.
- What about databases containing a mix of media types?
- Such databases can arise in one of three ways:
  - All the media data is "legacy" data.
  - Some of the media data is "legacy" data, others are being specially crafted.
  - None of the media data is "legacy" data, all of it is being specially crafted.

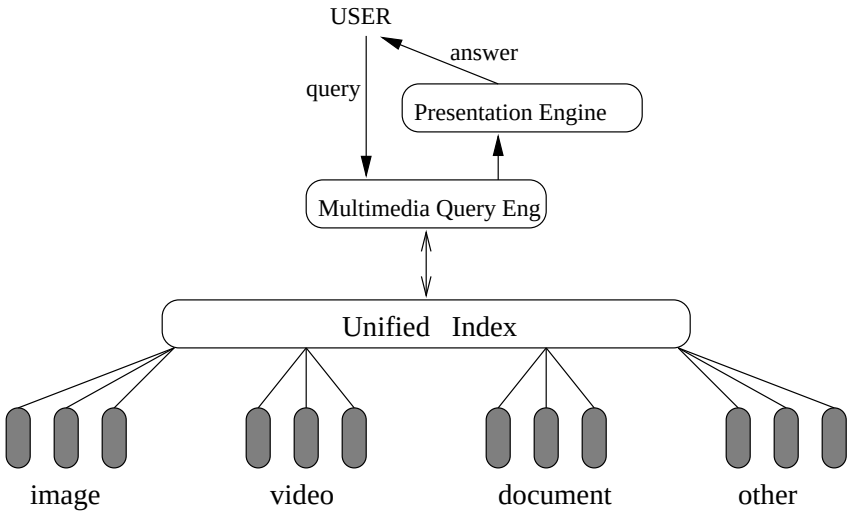
Multimedia Database Architectures

- **(The Principle of Autonomy)** Should we choose to group together all images, all videos, all documents, and index each of them in a way that is maximally efficient for the expected types of accesses we plan to make on those objects ?
- **(The Principle of Uniformity)** Should we try to find a single abstract structure $\mathcal{A}$ that can be used to index *all* the above media types, and that can thus be used to create a "unified index" that can then be used to access the different media objects ?
- **(The Principle of Hybrid Organization)** A third possibility is to use a hybrid of the previous two principles. In effect, according to this principle, certain media types use their own indexes, while others use the "unified" index.

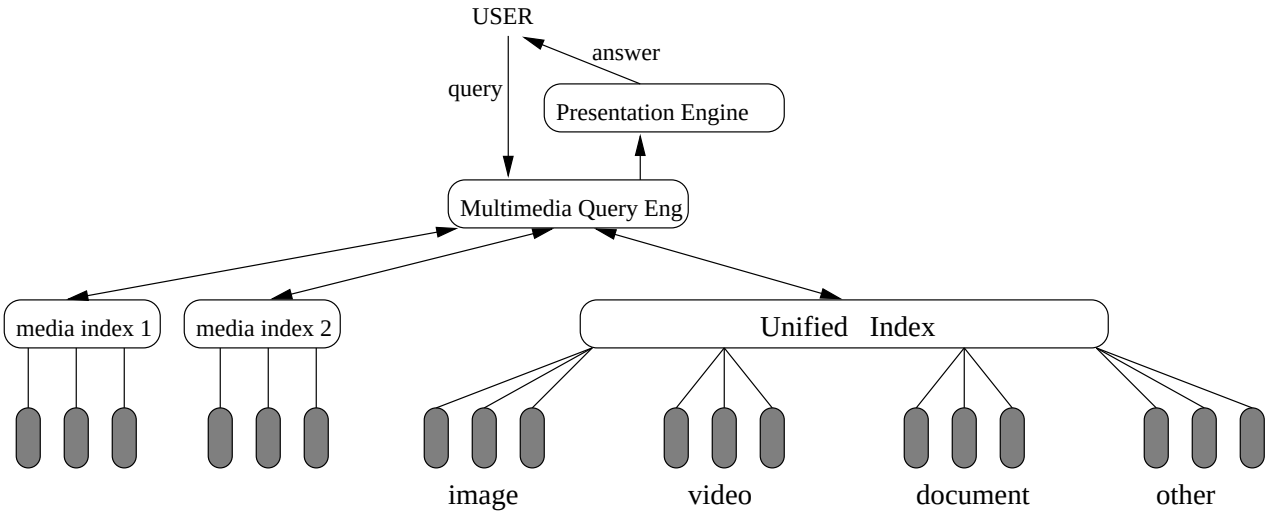
The Principle of Autonomy



The Principle of Uniformity



The Principle of Hybrid Organization



Organizing Multimedia Data based on the Principle of Uniformity

Suppose we consider the following statements about media data. These statements may be made by a human or may be produced as the output of an image/video/text content retrieval engine.

- The image `photo1.gif` shows Jane Shady, Denis Dopeman and an unidentified third person, in Medellin, Colombia. The picture was taken on January 5, 1997.
- The video-clip `video1.mpg` shows Jane Shady giving Daniel Dopeman a briefcase (in frames 50-100). The video was obtained from surveillance set up at Denis Dopeman's house in Rockville, Maryland, in October, 1996.
- The document `dopeman.txt` contains background information on Denis Dopeman, obtained from Mr. Dopeman's FBI file.

All the above statements are *Metadata* statements.

- Associate, with each media object o_i, some meta-data, $\mathsf{md}(o_i)$.
- $\mathsf{md}(o_i)$ may be produced by a human or a program.
- If our archive contains objectx $o_1, \ldots, o_n$, then index the metadata $\mathsf{md}(o_1), \ldots, \mathsf{md}(o_n)$ in a way that provides efficient ways of implementing the expected accesses that users will make.

Media-Abstractions

- If we consider the content of media data of different types, what is it that is common to all these media types, and what is it that is different ?
- We would like a single data structure that can represent different instances of this common "core".
- Media abstractions are mathematical structures representing such media content.
- Media abstractions may be implemented through a single data structure.

Media-Abstractions continued

- A *media-abstraction* is an 8-tuple

$$(\mathcal{S}, \underline{\mathsf{fe}}, \mathsf{ATTR}, \lambda, \Re, \mathcal{F}, \mathsf{Var}_1, \mathsf{Var}_2)$$

where:

  - $\mathcal{S}$ is a set of objects called *states*, and
  - $\underline{\mathsf{fe}}$ is a set of objects called *features*, and
  - ATTR is a set of objects called *attribute values*, and
  - $\lambda : \mathcal{S} \rightarrow 2^{\underline{\mathsf{fe}}}$ is a map from states to sets of features, and
  - $\Re$ is a set of relations on $\underline{\mathsf{fe}}^i \times \mathsf{ATTR}^j \times \mathcal{S}$ for $i, j \geq 0$, and
  - $\mathcal{F}$ is a set of relations of $\mathcal{S}$ and
  - Var_1 is a set of objects, called *variables*, ranging over $\mathcal{S}$, and
  - Var_2 is a set of variables ranging over $\underline{\mathsf{fe}}$.

Media-Abstractions: Intuition

- A "state" is the smallest "chunk" of media data that we wish to reason about.
- A "feature" is any object in a state that is deemed to be of "interest." Features can in principle include both objects and activities.
- A feature may have one or more attributes associated with it.
- The feature extraction map specifies what features occur in which states. In some cases, feature extraction maps are implemented as content extraction programs, while in other cases, they may involve humans manually specifying content.
- $\Re$ is a set of state-dependent and state-independent relations, e.g. `left-of` is state dependent, while `age` is not.
- $\mathcal{F}$ contains inter-state relations, e.g `before`(s1,s2) may say that state s1 occurred before state s2.

Image Data Viewed as a Media-Abstraction

- The set of states consists of {pic1.gif, . . . , pic7.gif}.
- **Features:** The set of features consists of the names of the people shown in the photographs. Let us call this list: Bob, Jim, Bill, Charlie, and Ed.
- **Extraction Map λ:** This map tells us, for each state, which features occur in that state. The table below contains this description:

State	Feature
pic1.gif	bob,jim
pic2.gif	jim
pic3.gif	bob
pic4.gif	bill
pic5.gif	charlie
pic6.gif	ed, bill
pic7.gif	ed

- The set of relations may contain just two relations: a state-dependent relation called `left_of` and a state-independent relation called `father`, with the obvious meanings.
- The set of inter-state relations may be empty.

Video Data Viewed as a Media-Abstraction

- **States:** The set of states consists just of frames 1 through 5.
- **Features:** The set of features consists of: Jane Shady, Denis Dopeman, Dopeman_house and briefcase,
- **Extraction Map λ:** The extraction map, λ, is described by the following simple table:

State	Feature
frame1	Dopeman_house, briefcase, Jane Shady
frame2	Dopeman_house, briefcase, Jane Shady, Denis Dopeman
frame3	Dopeman_house, briefcase, Jane Shady, Denis Dopeman
frame4	Dopeman_house, briefcase, Jane Shady, Denis Dopeman
frame5	Dopeman_house, Jane Shady

- **Relations:**

1. `have` is a state-dependent relation specifying who has an object in a given state. This may be given by the following simple table:

Person	Object	State
Jane Shady	briefcase	1
Jane Shady	briefcase	2
Jane Shady	briefcase	3
Denis Dopeman	briefcase	4

2. `spouse` is a state-independent relation specifying the name of the spouse of an individual. This may be given by the simple table:

Person	Spouse
Jane Shady	Peter Shady
Jane Shady	Peter Shady
Denis Dopeman	Debra Dopewoman

3. **Inter-State Relations:** This may consist of just one relation called `before(s1,s2)` which holds iff state `s1` occurs before state `s2`.

Simple Multimedia Database

- A *simple multimedia database* is a finite set, $\mathcal{M}$, of media-abstractions.
- Simple multimedia databases are naive.
- A a media abstraction may list "Church" as a feature; however, when searching for "Cathedrals" or "Monuments", we may not find the church, because the system does not know that cathedrals and churches are (more or less) synonymous, and that all churches are monuments (but not vice-versa).
- Also, users often search for media objects containing one or more features, and then "refine" the search later when they find that the media-objects returned by their query, though correct, do not correspond precisely to what they wanted.

Structured Multimedia Database

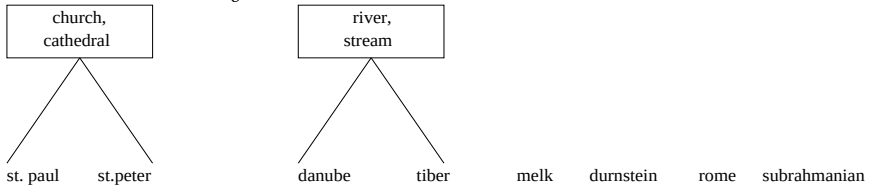
A *structured multimedia database system,* SMDS, is a 5-tuple $(\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \equiv, \leq, \mathsf{inh}, \mathsf{subst})$ where:

- $\mathcal{M}_i = (\mathcal{S}^i, \underline{\mathsf{fe}}^i, \mathsf{ATTR}^i, \lambda^i, \Re^i, \mathcal{F}^i, \mathsf{Var}_1^i, \mathsf{Var}_2^i)$ is a media-abstraction, and
- $\equiv$ is an equivalence relation on $\mathcal{F} = \cup_{i=1}^n \underline{\mathsf{fe}}^i$;
- $\leq$ is a partial ordering on the set $\mathcal{F}/\equiv$ of equivalence classes on $\mathcal{F}$, and
- $\mathsf{inh} : \mathcal{F}/\equiv \longrightarrow 2^{\mathcal{F}/\equiv}$ such that $[f_1] \in \mathsf{inh}([f_2])$ implies that $[f_1] \leq [f_2]$. Thus, inh is a map that associates with each feature f, a set of features that are "below" f according to the $\leq$-ordering on features.
- subst is a map from $\cup_{i=1}^n \mathsf{ATTR}^i$ to $2^{\cup_{i=1}^n \mathsf{ATTR}^i}$.

Example SMDS

Media	Object	Part/frame	Feature(s)
image	photo1.gif	-	church, durnstein, danube, subrahmanian
image	photo2.gif	-	cathedral, melk, subrahmanian
image	photo3.gif	-	church, st. paul, rome
video	video1.mpg	1-5	church, durnstein, stream
video	video1.mpg	6-10	stream
audio	audio1.wav	1-20	st. peters, tiber, rome

- three media-abstractions, one each associated with image, video, and audio data.
- the set of features, $\mathcal{F}$ contains: `church, durnstein, danube, subrahmanian, cathedral, melk, st. paul, rome, stream, restaurant, st. peters, tiber.`
- $\equiv$ says that:
  - `church` $\equiv$ `cathedral`.
  - `river` $\equiv$ `stream`.
- the $\leq$ relation says that:



Querying SMDSs (Uniform Representation)

Basic SMDS functions are:

- `FindType(Obj)`: This function takes as input, a media object `Obj` as input, and returns the output type of the object. For example,

$$\texttt{FindType(im1.gif)} = gif.$$
$$\texttt{FindType(movie1.mpg)} = mpg.$$

- `FindObjWithFeature(f)`: This function takes as input, a feature `f` and returns as output, the set of all media-objects that contain that feature. For example,

`FindObjWithFeature(john)` = {im1.gif,im2.gif,im3.gif, video1.mpg:[1,5].
`FindObjWithFeature(mary)` = {video1.mpg:[1,5], video1.mpg:[15,50].

- `FindObjWithFeatureandAttr(f,a,v)`: This function takes as input, a feature `f`, an attribute name `a` associated with that feature, and a value `v`. It returns as output, all objects o that contain the feature and such the value of the attribute `a` in object o is `v`. For example,

  1. `FindObjWithFeatureandAttr(Jane Shady,suit,blue)`: This query asks to find all media objects in which Jane Shady appears in a blue suit.

2. `FindObjWithFeatureandAttr(Elephant,bow,red)`: This query asks to find all media objects in which an elephant wearing a red bow appears.

- `FindFeaturesinObj(Obj)`: This query asks to find all features that occur within a given media object. It returns as output, the set of all such features. For example,

  1. `FindFeaturesinObj(im1.gif)`: This asks for all features within the image file `im1.gif`. It may return as output, the objects John, and Lisa.
  2. `FindFeaturesinObj(video1.mpg:[1,5])`: This asks for all features within the first 5 frames of the video file `video1.mpg`. The answer may include objects such as Mary and John.

- `FindFeaturesandAttrinObj(Obj)`: This query is exactly like the previous query except that it returns as output, a relation having the scheme:

  $$(\texttt{Feature}, \texttt{Attribute}, \texttt{Value})$$

  where the triple (f, a, v) occurs in the output relation iff feature f occurs in the query `FindFeaturesinObj(Obj)` and feature f's attribute a is defined and has value v. For example,

  `FindFeaturesandAttrinObj(im1.gif` may return as answer, the table:

Feature	Attribute	Value
John	age	32
John	address	32 Pico Lane, Mclean, VA 22050.
Mary	age	46
Mary	address	16 Shaw Road, Dumfries, VA 22908.
Mary	employer	XYZ Corp.
Mary	boss	David

SMDS-SQL

- All ordinary SQL statements are SMDS-SQL statements. In addition:
- The SELECT statement may contain *media-entities*. A *media entity* is defined as follows:
  1. If m is a continuous media object, and i, j are integers, then $m : [i, j]$ is a media-entity denoting the set of all frames of media object m, that lie between (and inclusive of) segments i, j.
  2. If m is not a continuous media-object, then m is a media entity.
  3. If m is a media-entity, and a is an attribute of m, then $m.a$ is a media-entity.
- The FROM statement may contain entries of the form

  $$\langle media \rangle \langle source \rangle \langle M \rangle$$

  which says that only all media-objects associate with the named media type and named data source are to be considered when processing the query, and that M is a variable ranging over such media objects.
- The WHERE statement allows (in addition to standard SQL constructs), expressions of the form:

  $$term \text{ IN } func\_call$$

  where:

1. *term* is either a variable (in which case it ranges over the output type of *func_call*) or an object having the same output type as *func_call* and
2. *func_call* is any of the five function calls listed above.

SMDS-SQL Examples

- *Find all image/video objects containing both Jane Shady and Denis Dopeman.* This can be expressed as the SMDS-SQL query:

```
SELECT      M
FROM        smds source1 M
WHERE       (FindType(M)=Video OR FindType(M)=Image)
            AND
            M IN FindObjWithFeature(Denis Dopeman)
            AND
            M IN FindObjWithFeature(Jane Shady).
```

- *Find all image/video objects containing Jane Shady wearing a purple suit.* This can be expressed as the SMDS-SQL query:

```
SELECT      M
FROM        smds source1 M
WHERE       (FindType(M)=Video OR FindType(M)=Image)
            AND
            M IN FindObjWithFeatureandAttr(Jane Shady,s
```

- *Find all images containing Jane Shady and a person who appears in a video with Denis Dopeman.* Unlike the preceding queries, this query involves computing a "join" like operations across different data domains. In order to do this,

we use existential variables such as the variable "Person" in the query below, which is used to refer to the existence of an "unknown" person whose identity is to be determined.

```
SELECT      M,Person
FROM        smds source1 M,M1
WHERE       FindType(M)=Image) AND
            FindType(M1)=Video) AND
            M IN FindObjWithFeature(Jane Shady) AND
            M1 IN FindObjWithFeature(Denis Dopeman)
            AND
            Person IN FindFeaturesinObj(M)AND
            Person IN FindFeaturesinObj(M1)AND
            Person≠Jane Shady AND Person≠Denis Dopeman.
```

Querying Hybrid Representations of Multimedia Data

- SMDS-SQL may be used to query multimedia objects which are stored in the uniform representation.
- "What is it about the hybrid representation that causes our query language to change?"
- In the uniform representation, all the data sources being queried are SMDSs, while in the hybrid representation, different (non-SMDS) representations may be used.
- A hybrid media representation basically consists of two parts – a set of media objects that use the uniform representation (which we have already treated in the preceding section), and a set of media-types that use their own specialized access structures and query language.
- To extend SMDS-SQL to Hybrid-Multimedia SQL (HM-SQL for short), we need to do two things:
  - First, HM-SQL must have the ability to express queries in each of the specialized languages used by these non-SMDS sources.
  - Second, HM-SQL must have the ability to express "joins" and other similar binary algebraic operations between SMDS-sources, and non-SMDS sources.

HM-SQL

HM-SQL is exactly like SQL except that the SELECT, FROM, WHERE clauses are extended as follows:

- the SELECT and FROM clauses are treated in exactly the same way as in SMDS-SQL.
- The WHERE statement allows (in addition to standard SQL constructs), expressions of the form:

  $$term \text{ IN } MS : func\_call$$

  where:

  1. *term* is either a variable (in which case it ranges over the output type of *func_call*) or an object having the same output type as *func_call* as defined in the media-source MS and
  2. either MS =SMDS and em func_call is one of the five SMDS functions described earlier, or
  3. MS is not an SMDS-media source, and *func_call* is a query in QL(MS).

- Thus, there are 2 differences between HM-SQL and SMDS-SQL:

  1. *func_call*s occurring in the WHERE clause must be explicitly annotated with the media-source involved, and

2. queries from the query languages of the individual (non-SMDS) media-source implementations may be embedded within an HM-SQL query. This latter feature makes HM-SQL very powerful indeed as it is, in principle, able to express queries in other, third-party, or legacy media implementations.

HM-SQL Examples

- *Find all video clips containing Denis Dopeman, from both the video sources, video1, and video2.*

```
SELECT      M
FROM        smds video1, videodb video2
WHERE       M IN smds:FindObjWithFeature(Denis Dopeman) OR
            M IN videodb:FindVideoWithObject(Denis Dopeman).
```

- *Find all people seen with Denis Dopeman in either video1, video2, or idb.*

```
(SELECT     P1
FROM        smds video1 V1
WHERE       V1 IN smds:FindObjWithFeature(Denis Dopeman) AND
            P1 IN smds:FindFeaturesinObj(V1) AND
            P1≠ Denis Dopeman)
UNION
(SELECT     P2
FROM        videodb video2 V2
WHERE       V2 IN smds:FindObjWithFeature(Denis Dopeman) AND
            P2 IN videodb:FindObjectsinVideo(V1) AND
            P2≠ Denis Dopeman)
UNION
(SELECT     *
FROM        imagedb idb I2
WHERE       V2 IN image:getpic(Denis Dopeman) AND
            P2 IN imagedb:getfeatures(V1) AND
            P2≠ Denis Dopeman).
```

Indexing SMDSs with Enhanced Innverted Indexes

Suppose $(\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \equiv, \leq, \mathsf{inh}, \mathsf{subst})$ where

$$\mathcal{M}_i = (\mathcal{S}^i, \underline{\mathsf{fe}}^i, \mathsf{ATTR}^i, \lambda^i, \Re^i, \mathcal{F}^i, \mathsf{Var}_1^i, \mathsf{Var}_2^i)$$

is an SMDS.

1. FEATURETABLE: This is a hash table whose entries are features in $\cup_{i=1}^n \underline{\mathsf{fe}}^i$. Each hash table location i contains a bucket of "featurenodes" that hash to location i.
2. STATETABLE: This is a hash table whose entries are states in $\cup_{i=1}^n \mathcal{S}^i$. Like the FEATURETABLE, the STATETABLE contains a bucket of "statenodes" that hash to the specified location.
3. FEATURENODEs: Each featurenode contains:
   - the name of the feature (e.g. "Denis Dopeman"),
   - a list of children nodes (if f_1, f_2 are features in $\cup_{i=1}^n \underline{\mathsf{fe}}^i$, we say that f_2 is a child of f_1 iff $f_2 \leq f_1$ and there is no other feature $f_3 \in \cup_{i=1}^n \underline{\mathsf{fe}}^i$ such that $f_2 < f_3 < f_1$)
   - a list of pointers to statenodes (see below) that contain that feature (e.g. in the case of an image database, this would be a pointer to the nodes associated with images that contain the feature in question, e.g. Denis Dopeman)

- a list of pointers to other featurenodes that are appropriate substitutes for the featurenode in question. Specifically, if we consider a feature node associated with feature f, then a pointer to feature g is in this list iff $g\mathsf{inh}(f)$.

4. STATENODEs: A statenode consists of just two components: a pointer to a file containing the media-object (image, video, audio, document, etc.) that the state in question refers to, and a linked list whose members point to featurenodes – intuitively, there is a pointer to a featurenode f iff the feature in question is in the state.

Data Structure

```
type featurenode = record of  /* nodes in feature graph */
     name : string;  /* nameof feature */
     children: ^node1; /* points to a list of pointers to the children */
     statelist: ^node2; /* points to a list of pointers to states that*/
                          /* contain this feature */
     replacelist: ^node3; /* points to a list of descendants whose */
                         /* associated states can be deemed to have the */
                         /* the feature associated with this node */
     end record;

type node1 record of
     element: ^featurenode; /* points to a child of a featurenode */
     next :   ^node1; /* points to next child */
     end record;

type node2 record of
     state: ^statenode; /* pointer to the list associated with a state */
     link:  ^node2;  /* next node */
     end record;

type node3 record of
     feat: ^featurenode; /* pointer to a node that can be deemed to have */
                        /* the feature associated with the current node */
     link1: ^node3;
     end record;

type statenode record of
     rep:  ^framerep;
     flist: ^node4;
     end record

type node4 record of
     f : ^featurenode;
     link2: ^node4;
     end record:
```

Example

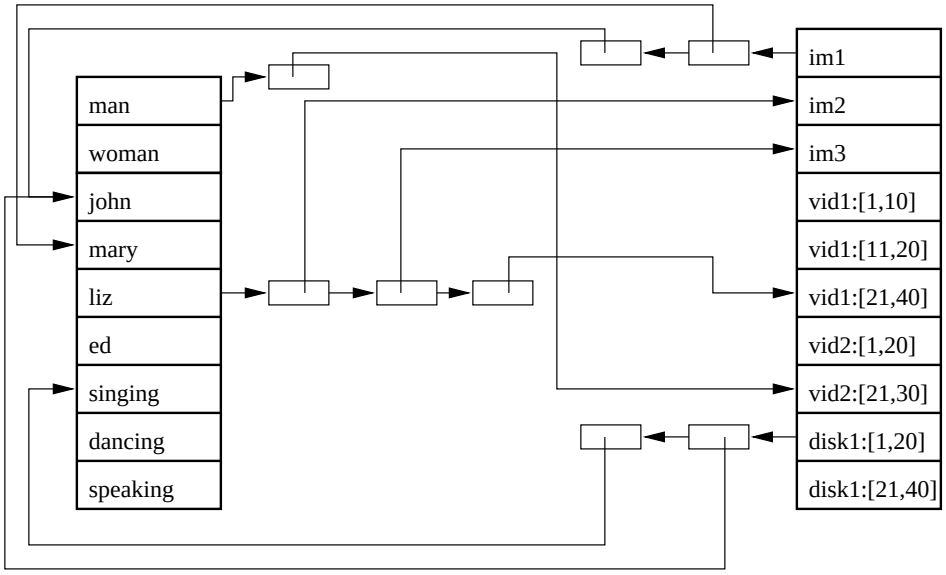
- Consider a very simple toy SMDS containing three media-abstractions: images, video, and audio, with following content:

Media-Abstraction	State	Features
image	im1	john, mary
	im2	john, mary, liz
	im3	liz, mary
video	vid1:[1,10]	john, singing
	vid1:[11,20]	john,mary, dancing
	vid1:[21,40]	john, mary, liz, singing, dancing
	vid2:[1,20]	ed, speaking
	vid2:[21,30]	man, speaking
audio	disk1:[1,20]	john, singing
	disk1:[21,40]	woman, speaking

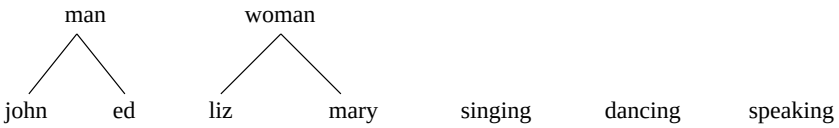
- Next slide shows one possible STATETABLE (without hashing) and a possible FEATURETABLE.
- Slide after that shows the $\leq$ ordering on features.
- We have shown the `featurelist` associated with the states `im1` and `disk1:[1,20]`. Other feature lists are not shown for the sake of simplicity.
- Likewise, we have shown the `statelists` associated with the features `man` and `liz`.

- The relations associated with these media-abstractions may be stored within standard relational databases.

Example of STATETABLE



Example of $\leq$ Ordering



Example Algorithm FindObjwithFeature(f)

Algorithm 6 *FindObjwithFeature(f);*

1. *SOL* $= \emptyset$*;*
2. *Hash feature f and resolve collisions (if any) till feature f is located (at location i say) in the FEATURETABLE;*
3. *SOL = append(SOL,Featuretable[i].Statelist);*
4. **Forall** *children f′ of some member of* $[f]$ **do** *SOL = append(SOL,FindObjwithFeature(f'))*
5. **Return** *SOL.*

This assumes that if f' is a descendant of f then f' occurs in every state in which f occurs. We may easily modify the above algorithm if this assumption is not valid.

Find features in state s

1. Hash state s and resolve collisions (if any) till state s is located (at location i say) in the STATETABLE;
2. **Return** STATETABLE[i].Featurelist.

Check if state s contains feature f

1. Hash state s and resolve collisions (if any) till state s is located (at location i say) in the STATETABLE;
2. **If** f is in STATETABLE[i].Featurelist, then **Return** true **else Return** false.

Query Relaxation/Expansion

- The inh and subst components of an SMDS are used to determine how queries must be relaxed.
- When a user poses a query Q, we *somehow modify* that query Q into a set $\{Q_1, \ldots, Q_k\}$ of queries.
- This set is partially ordered and contains the original query Q as the maximal element of the ordering.
- Intuitively, if Q' is a child of Q'' then this means that Q' is obtained from Q'' by making a modification or relaxation.
- Thus, query relaxation depends on two aspects:
  - what are the "modification" operations that are allowed to modify queries?
  - what is the ordering on the set of modified queries?

FindObjwithFeatureandInh(f,inh)

Algorithm 7 *FindObjwithFeatureandInh(f,*inh*);*

1. *SOL* = $\emptyset$;
2. *Hash feature f and resolve collisions (if any) till feature f is located (at location i say) in the FEATURETABLE;*
3. *SOL = append(SOL,Featuretable[i].Statelist);*
4. **Forall** *children f′ of some member of* inh([*f*]) **do** *SOL = append(SOL,FindObjwithFeatureandInh(f', $\emptyset$))*
5. **Return** *SOL.*

Query Relaxation, Continued

- Q_1 is a *feature-relaxation* of Q_2, denoted $Q_1 \sqsubseteq Q_2$ iff there exist features $f_1, f_2 \in \cup_{i=1}^{n} \underline{\mathsf{fe}}^i$ such that
  - $Q_1 \preceq Q_2[f_1/f_2]$ (where the notation $Q_2[f_1/f_2]$ denotes the replacement of all occurrences of f_1 in Q_2 by f_2) and
  - $f_2 \in \mathsf{inh}([f_1])$.
- **EX:** Q_2 is:

```
SELECT      M
FROM        smds video1, videodb video2
WHERE       M IN smds:FindObjWithFeature(Denis Dopeman) OR
            M IN videodb:FindVideoWithObject(Denis Dopeman)
```

- Suppose David Johns in in inh(Denis Dopeman).
- Q_1 is:

```
SELECT      M
FROM        smds video1, videodb video2
WHERE       M IN smds:FindObjWithFeature(David Johns) OR
            M IN videodb:FindVideoWithObject(David Johns)
```

- Q_1 is a feature relaxation of Q_2.

Relaxation Examples

– Q_4:

```
SELECT      M
FROM        smds
WHERE       M IN smds:FindObjWithFeature(Denis Dopeman) AND
            M IN smds:FindObjWithFeatureandAttr(briefcase,color,black).
```

– Q_5:

```
SELECT      M
FROM        smds
WHERE       M IN smds:FindObjWithFeature(Denis Dopeman) AND
            M IN smds:FindObjWithFeatureandAttr(package,color,grey).
```

– Q_6:

```
SELECT      M
FROM        smds
WHERE       M IN smds:FindObjWithFeature(Denis Dopeman) AND
            M IN smds:FindObjWithFeatureandAttr(briefcase,color,grey).
```

– (Feature Replacement) obtained Q_4 by replacing occurrences of "package" in Q_3 with "briefcase"

– (Attribute Replacement) obtained Q_5 by replacing occurrences of "black" by "grey"

– (Feature-cum-Attribute Replacement) obtained Q_6 by replacing occurrences of "black" by "grey" and occurrences of "package" with "briefcase."

Relaxations, Continued

- Q_1 is said to be a *attribute-relaxation* of Q_2, denoted $Q_1 \sqsubseteq_a Q_2$ iff there exist attributes $a_1, a_2 \in \cup_{i=1}^{n} \mathsf{ATTR}^i$ such that $Q_1 = S_2[a_1/a_2]$ and $a_2 \in \mathsf{subst}(a_1)$.
- *Relaxation* of queries is defined inductively as follows:
  - If Q_1 is a feature-relaxation of Q_2, then Q_1 is a relaxation of Q_2;
  - If Q_1 is a attribute-relaxation of Q_2, then Q_1 is a relaxation of Q_2;
  - If Q_1 is a relaxation of Q_3 and Q_3 is a relaxation of Q_2, then Q_1 is a relaxation of Q_2.
- $\mathsf{Relax}(Q)$ denotes the set of all relaxations of query Q (w.r.t. some SMDS).