

Superviews: Virtual Integration of Multiple Databases

Amihai Motro

IEEE Transactions on Software Engineering
Vol. SE-13, No. 7, July 1987
Pages 785-798

The Problem

- ✓ A database should provide each application with a single integrated view of all the data it requires.
 - One of the original motivations behind the invention of databases.
- ✓ Eventually, applications evolve that are no longer satisfiable from a single database.
 - The existence of multiple, relevant databases negates the above advantage.
 - Many database management systems do not allow accessing more than one database at a time.

Possible Solutions

- ✓ Consolidate the multiple databases in a single database.
 - Costly; justified only when the new requirements are permanent.
 - It may be desirable to keep the independence of individual databases.
 - It is impossible, if the databases are provided by external sources.
- ✓ Develop a multidatabase language.
 - Access and combine data from individual databases.
 - Example: MALPHA [Litwin 84]
- ✓ Construct a virtual database.

Virtual Database

- ✓ Actual database: $\langle \textit{schema}, \textit{data} \rangle$
- ✓ Virtual database: $\langle \textit{schema}, \textit{mapping} \rangle$
 - Schema describes the entire multidatabase environment (*virtual* schema).
 - No data.
 - Instead, the mapping links the schema to the schemas of other databases.
 - The target databases could be virtual, as well.
 - The mapping can be interpreted as another form of data.
- ✓ Integration need not be “total.”
- ✓ With respect to permanency: virtual database schemas are somewhere between the fixed schemas of physical databases, and the transient views formed by MALPHA queries.

Superviews: Main Components

- ✓ Virtual Database Generator
 - An interactive program that creates a virtual database (schema and mapping) from individual database schemas.
 - The user “stitches” the individual schemas with integration statements.
 - The process also yields a mapping from the new schema to the individual schemas.
- ✓ Virtual Query Processor
 - Software that processes *global* queries (queries on the virtual database).
 - It employs the mapping to *decompose* each global query to a set of queries against the individual databases.
 - It recomposes the set of answers received in an answer to the original query.
 - The process is *transparent* to the user (as if an actual database existed).

Related Research

- ✓ Multidatabase systems
 - Multibase
 - ADDS
- ✓ Physical consolidation
 - Restructuring
 - Data translation
- ✓ Design methodology
 - View integration

The Data Model: A Functional Approach

- ✓ The functional approach to data modeling was first described in [Sibley and Kerschberg 1977].
- ✓ Various flavors; we define our own version.
- ✓ Basic notions:
 - *Domain*: set of values.
 - *Function*: Assigns values of one domain to another domain.
 - Two types of functions: *attribute*, *generalization*.

The Functional Model: Basic Concepts

A database is

1. A collection D of named *classes*.
2. Two relations **att** and **gen** defined on D .
 - Their intersection is empty.
 - Their union has irreflexive transitive closure.
3. Each class $S \in D$ has
 - A *domain* **dom**(S) of values.
 - A *type*: **type**(S) = { $T \mid T \text{ att } S$ }.
 - A *key*: $K \subseteq \text{type}(S)$, denoted $K \text{ key } S$.
4. Extensions
 - Every relationship $T \text{ att } S$ is supported by a function:
 $f_{st} : \text{dom}(S) \rightarrow \text{dom}(T)$
 - Every relationship $T \text{ gen } S$ is supported by an *injective* function:
 $i_{st} : \text{dom}(S) \rightarrow \text{dom}(T)$

Basic Concepts (*cont.*)

5. Compositions

1. Inheritance of attributes over generalizations:

$$S \text{ att } T, T \text{ gen } R \Rightarrow S \text{ att } R, f_{RS} = i_{RT} \circ f_{TS}$$

2. Transitivity of generalizations:

$$S \text{ gen } T, T \text{ gen } R \Rightarrow S \text{ gen } R, i_{RS} = i_{RT} \circ i_{TS}$$

6. Keys

1. If (T_1, \dots, T_K) **key** S and

$f_{ST_i} : S \rightarrow T_i$ ($i=1, \dots, k$) are the **att** functions,

then

$$f : \mathbf{dom}(S) \rightarrow \mathbf{dom}(T_1) \times \dots \times \mathbf{dom}(T_k)$$

$$f(s) = (f_{ST_1}(s), \dots, f_{ST_k}(s))$$

is an injection.

2. K **key** $S, S \text{ gen } T \Rightarrow K$ **key** T

Example

✓ **Classes:**

FACULTY, STUDENT, PERSON, PIN, NAME, RANK, GPA

✓ **Class relationships:**

| | |
|-------------------------|---------------------------|
| PIN att FACULTY | NAME att STUDENT |
| PIN att STUDENT | NAME att PERSON |
| PIN att PERSON | GPA att STUDENT |
| RANK att FACULTY | PERSON gen FACULTY |
| NAME att FACULTY | PERSON gen STUDENT |

✓ **Types:**

type(FACULTY) = (PIN, NAME, RANK)

type(STUDENT) = (PIN, NAME, GPA)

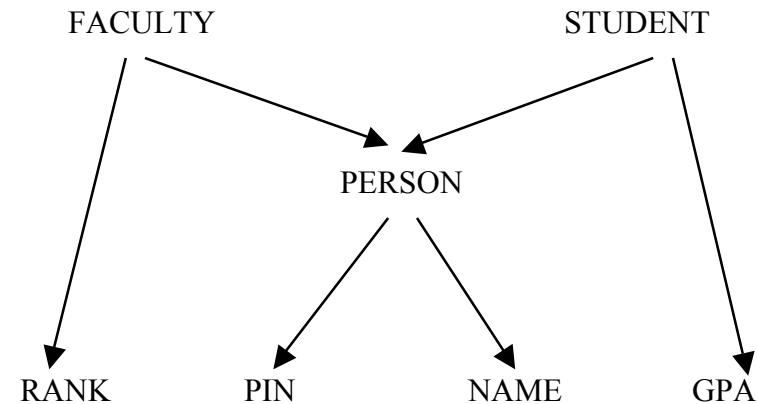
type(PERSON) = (PIN, NAME)

✓ **Keys:**

PIN **key** PERSON

PIN **key** STUDENT

PIN **key** FACULTY



Query Language vs. Access Operators

- ✓ Functional query language, similar to DAPLEX [Shipman 1981].

- Example

for each ENROLLMENT

such that COURSE(ENROLLMENT) = 'CS101'

and GRADE(ENROLLMENT) = 'A'

print NAME(STUDENT(ENROLLMENT))

- ✓ This language is implemented with a minimal set of *access operators*:

- *Domain*: Retrieve the values in the domain of S .

$$\{S\} \equiv \mathbf{dom}(S)$$

- *Function*: Retrieve the value assigned to $s \in S$ by attribute T .

$$T(S=s) \equiv f_{ST}(s)$$

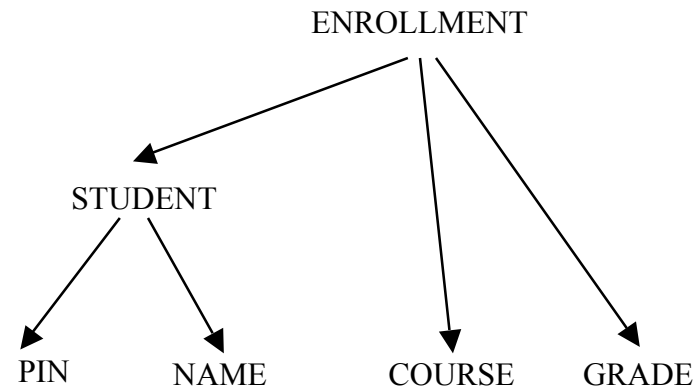
- *Inverse*: Retrieve the set of values in $\mathbf{dom}(S)$ that have the value t for attribute T .

$$\{S(T=t)\} \equiv f^{-1}_{ST}(t)$$

Example

- ✓ The previous query is translated by using these access operators in a host language:

```
for each  $x$  in {ENROLLMENT} do  
  begin  
    if COURSE(ENROLLMENT =  $x$ ) = 'CS101'  
    and GRADE(ENROLLMENT =  $x$ ) = 'A'  
    then do  
      begin  
         $y :=$  STUDENT(ENROLLMENT =  $x$ );  
        print NAME(STUDENT =  $y$ )  
      end  
    end  
  end.
```



The Integration Language

The language consists of 10 operators (2 of which may be emulated by sequences of other operators).

- ✓ Manipulate the generalization hierarchy
 - Meet
 - Join
 - Fold
 - Combine and Connect (sequences of meet and fold)
- ✓ Modify the attribute hierarchy, to iron-out structural differences between two schemas:
 - Aggregate
 - Telescope
 - Add
 - Delete
- ✓ Other
 - Rename

Constructing A Superview

- ✓ Illustrate the technique with example

Deriving the Mapping

- ✓ Associate with each final class an expression that denotes the *origins* of this class, in terms of classes of the initial databases.
- ✓ Obtained incrementally during the integration process.
- ✓ Each operation updates the expressions of the classes it creates or modifies.
- ✓ The expressions associated with the final classes constitute the *mapping*.
- ✓ Example:
 - Classes SUPPLIER and CUSTOMER are generalized to ASSOCIATE,
 - Then TEL-NO is deleted from ASSOCIATE.
 - The expression associated with the final class ASSOCIATE is:
 $((\text{SUPPLIER} \wedge \text{ASSOCIATE})_1 - \text{TEL-NO})_2$

Query Translation

When an access operator is submitted to a superview

- ✓ It is translated over each of the integration operators that were involved in the classes that it addresses.
- ✓ The order of translation is the reverse order of the application.
- ✓ The final answers (the answers to queries submitted to the actual databases) are passed back in reverse direction.
- ✓ These answers are recomposed, until an answer to the original query is obtained.
- ✓ We need rules for translating each of 3 access operators over 10 possible integration operators — total 30 rules.
- ✓ We need rules for combining answers obtained from different databases.

Query Translation (*Cont.*)

Example (3 of the 30 rules):

- ✓ Consider the operator **meet** S and T into Q .
- ✓ Assume R is attribute of Q .
- ✓ Three access operators need translation:
 - The *domain* of Q :
 $\{Q\} \equiv \{S\} \cup \{T\}$
 - A *function* operator from Q to R :
 $R(Q = x) \equiv R(S = x) \vee R(T = x)$
 \vee is the *best-value* operator: combines two values as in the “best” way.
 - An *inverse* operator from R to Q :
 $\{Q(R = y)\} \equiv \{S(R = y)\} \oplus \{T(R = y)\}$
 \oplus is the *best-set* operator: combines two sets in the “best” way.

The Best-Value Operator

Combines two *function* operators.

- ✓ $T(S = s) =$
 - [*not applicable*] if $T \notin \mathbf{type}(S)$
 - [*not-found*] if $s \notin \mathbf{dom}(S)$
 - [not-available] if $f_{ST}(s)$ is undefined
 - $f_{ST}(s)$ otherwise
- ✓ Each successive situation is more “successful”.
- ✓ The *best-value* operator combines two values with the more successful of the two.
- ✓ However, two different values (both are of the fourth type) are combined with [*not-consistent*].

The Best-Set Operator

Combines two *function* operators.

- ✓ $\{S(T = t)\} =$
 - [not applicable] if $T \notin \mathbf{type}(S)$
 - [not-found] if $s \notin \mathbf{dom}(T)$
 - $f^{-1}_{ST}(t)$ otherwise
- ✓ Each successive situation is more “successful”.
- ✓ The *best-set* operator combines two values with the more successful of the two.
- ✓ However, two different sets (both are of the third type) are combined with their *union*.

Conclusion

- ✓ Defined a new approach to database integration.
- ✓ Some issues:
 - Update of virtual databases
 - Data incompatibility
 - Merging heterogeneous databases (not in the functional model)