January 1989 Report No. STAN-CS-89-1245

2

Managing Objects in a Relational Framework

by

Gio Wiederhold, Thierry Barsalou, and Surajit Chaudhuri

Department of Computer Science
Stanford University
Stanford, California 94305

DARPA INSERT SHEET

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp. Date Jun 30, 1986

1a REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release: Distribution Unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) STAN-CS-89-1245	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Computer Science Department 6b OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS *(City, State, and ZIP Code)* Stanford University Stanford, CA 94305	7b. ADDRESS *(City, State, and ZIP Code)*
8a NAME OF FUNDING/SPONSORING ORGANIZATION DARPA 8b OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)* 1400 Wilson Blvd. Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11 TITLE *(Include Security Classification)*
Managing Objects in a Relational Framework

12 PERSONAL AUTHOR(S)
Gio Wiederhold, Thierry Barsalou, and Surajit Chaudhuri

13a TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
	FROM ________ TO ________	1989 - January	102

16 SUPPLEMENTARY NOTATION

17 COSATI CODES

FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The papers collected in this report present ongoing research within the KBMS and PENGUIN projects at Stanford's Computer Science Department and Section on Medical Informatics. They have been issued together in response to a large number of requests.

The work presents the motivations for dealing with complex objects and our implementation of the machinery necessary for creating object instances in a relational database. The object generator obtains the data and can create a variety of object configurations, as appropriate to the specific application. The knowledge needed for object generation is obtained from three sources: the structural database schema, application views applied to it, and object prototypes in the application programs. Keywords: INFORMATION SCIENCE, Data Bases, Medical Info. (TLS)

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION

22a NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE *(Include Area Code)*	22c OFFICE SYMBOL
Gio Wiederhold	(415) 723-2273	

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Managing Objects in a Relational Framework

Gio Wiederhold, Thierry Barsalou, and Surajit Chaudhuri
Stanford University
Departments of Computer Science and Medicine
Stanford CA 94306-2140

Introduction and Summary

The papers collected in this report present ongoing research within the KBMS and PENGUIN projects at Stanford's Computer Science Department and Section on Medical Informatics. They have been issued together in response to a large number of requests.

The work presents the motivations for dealing with complex objects and our implementation of the machinery necessary for creating object instances in a relational database. The object generator obtains the data and can create a variety of object configurations, as appropriate to the specific application. The knowledge needed for object generation is obtained from three sources: the structural database schema, *application views applied to it*, and object prototypes in the application programs.

The significant achievement of this work is that

1 Information leading to different object configuration can be stored non-redundantly. The applications will have distinct views over the data. The base information is shared.

2 Sets of instances of complex objects can be generated automatically. By freeing the application programmer from instantiating objects one-by-one a considerable reduction of database access effort is achieved. Optimization of access schedules becomes feasible.

3 The representation of objects in the application is now independent of the storage representation. Computationally efficient data structures are chosen for the application. The organization of the persistent storage can be adjusted to reflect the most frequent or critical access pattern. When the access pattern changes the storage can be reorganized without affecting the application data structure.

This model of computation fits well into the database-server and workstation model.

The first paper listed [1] presents the critical ideas and motivation. It argues specifically that it is unwise to store objects in persistant form if the information they represent must be shared by applications having different views of the data. The figures are taken from a slide presentation.

The second paper provides an introduction to a wide array of motivations seen in the literature for object management in databases. It provides a background for the choices we made.

The two subsequent papers [2,3] track the development of a demonstration system, where data about a complex medical instrument must be shared to serve a variety of users. The database needed for this application is relatively small and moderately complex. Yet, this application is real, and also presents an excellent environment for demonstrating the concepts within an academic setting.

The programs use a visual metaphor. A strctural model of the available relations and connections is displayed. The user begins by identifying a pivot relation the basis for a new object descriptor. The model provides the information to determine which other relations can be combined with the pivot relation to fashion complex objects. Objects from this set of descriptors can now be selected for actual instantiation, display, and processing.

Some engineering usage issues of the system are stressed in [4], but this paper is not included in this report because of excessive overlap. A PROLOG-based implementation, OBJ1 is reported in [5]. Here the view-objects remain defined through predicates, do not have identity, and are not directly updateable; only base relations are to be updated, if possible, through the views.

¡¡An appendix summarizes the connections types, which complement a normalized relational model to make up the structural model.¿¿ An annotated bibliography is attached to provide further pointers for work related to objects and databases.

Acknowledgement

[1] Wiederhold, Gio: "Views, Objects, and Databases"; *IEEE Computer*, Vol.19 no.12, December 1986, pages 37-44; reprinted version in Dittrich, Dayal, Buchmann (eds.) *Object-oriented Database Systems*, Springer Verlag, 1988.

[2] Barsalou, Thierry: "An Object-based Architecture for Biomedical Expert Database Systems"; to appear in the *Proceedings of the Twelfth Symposium on Computer Applications in Medical Care*, IEEE Computer Society Press, Washington DC, November 1988.

[3] Barsalou, Thierry and Wiederhold, Gio: "Applying a Semantic Model to an Immunology Database"; in W.W. Stead, editor, *Proceedings of the Eleventh Symposium on Computer Applications in Medical Care*, pages 871-877, IEEE Computer Society Press, Washington DC, November 1987.

[4] Barsalou, Thierry and Gio Wiederhold: "Knowledge-based Mapping of Relations into Objects"; to be published in The International Journal of Artificial Intelligence in Engineering, Computational Mechanics Publ., UK, 1988.

[5] Cohen, Benjamin C.: "Views and Objects in OB1: A PROLOG-based View-Object-Oriented Database"; David Sarnoff Res.Ctr., Princeton, TR.PRRL-88-TR-005, March 1988.

Views, Objects, and Databases

Gio Wiederhold
Stanford University, Computer Science Dep.

October, 1986

Abstract
The major point of this note is that it appears to be unwise to store persistent objects, which are to be shared, in object format. There appears to be a major conflict between the database paradigm: making information available for sharing using non-procedural access, and the programming paradigm: devising data structures which are effective representations for procedural access. We illustrate the problem with a simple example taken from electronic design applications, where both sharing and object-oriented access is desirable.

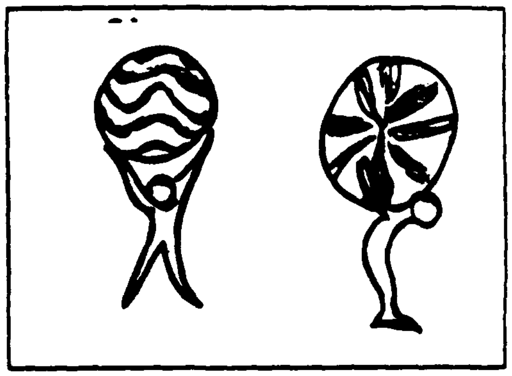
We then suggest a procedural approach which permits object-oriented access to use information stored in a relational database. A implementation of the sketched approach will have severe performance penalties unless storage transformations are applied to the database and its interface optimized for object-oriented access. However, having a defined interface will permit strorage reconfiguration and access optimization, notions inherent in non-procedural data access. A direction for further development, leading towards data independence for object access functions, is indicated.

1. Introduction

The intention of this article is two-fold. First of all we show intrinsic differences in the underlying concepts of access to persistent storage in databases and current extensions of object-oriented programming systems intended to incorporate persistence [Asilomar 86] [Lochovsky 85]. Since the objectives of both paradigms are similar we develop a connection between *object* concepts in programming languages and *view* concepts in database systems.

Secondly, we propose an architecture which exploits their commonality, and indicate research and development directions which are needed to make the bridge viable. Such an architecture seems to be especially suitable for computer-aided design tasks.

Engineering Information Systems (EIS) and systems for similar applications must provide suitable abstractions of real-world objects for their users and support long-lived transactions [Katz 83]. The complexity of the design process, and the number of specialized individuals needed to bring major projects to completion is driving the search for more systematized solutions than those provided by the file mechanisms now in use in operational precursors of EISs [Lorie 83] [Udagawa 84]. The issues being raised here pertain to systems with large quantities of data, and long lifetimes. Design tasks which can be handled by single individuals are not likely to benefit from EIS technolgy.



```
We all love to carry
objects around, they
are so much more natural
than tuples.
```

```
What's the problem

Owning objects is
nice, and we CONTROL
them.
```

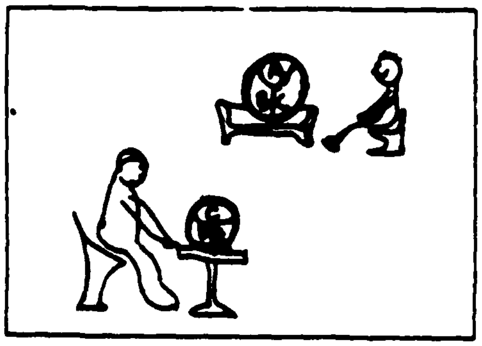


Figure 1a: Objects.

Databases and Views

Database concepts provide independence of storage and user data formats. The schema describes the form of the database contents, so that a variety of users can satisfy their data requirements by querying the shared database using non-procedural declarations of the form:

```
SELECT what-I-want WHERE some-set-of-conditions-is-true
```

A database administrator integrates the requirements of diverse users. The shared database can be changed as long as the schema is changed to reflect such changes. Concepts of database normalization help avoid redundancy of storage and anomalies that are associated with updates of redundantly stored information.

The principal formal database mechanism to obtain selected data for an application is a view specification. A view on a database consists of a query which defines a suitably limited amount of data. A database administrator is expected to use predefined views as a management tool. Having predefined views simplifies the user's access and can also restrict the user from information which is to be protected. We are interested here only in the first objective, and not in the protection aspects associated with views.

Views have seen formal treatment in relational databases [SQL], although subset definitions without structural transformations are common in other commercial database systems [DBTG]. We will hence describe views from a relational point of view, without implying that a relational implementation is best for the underlying EIS databases.

A view is defined in a relational model as a query over the base relations, and perhaps also over other views. Current implementations do not materialize views, but transform user operations on views into operations over the base data. The final result is obtained by interpreting the combination of view definition and user query, using the description of the database stored in the database schema.

The view, as any relational query result, is a relation. However, even when the base database is fully normalized, say to Boyce-Codd normal form, the view relation is, in general, only in first normal form. Views are in that sense already closer to objects: related data has been brought together.

The issue that views present data not in second or third normal form seems ignored in database research, except for the update complications that result [Keller 85]. We have no evidence that the unnormalized views are uncomfortable to a user expecting a normalized relational representation. Acceptance of unnormalized views can be taken as a partial validation of the aceptability of structures more suitable to represent objects than normalized tables. Some current research is addressing unnormalized relations independent from the view aspect [Roth 84].

Objects

Object-oriented programming languages help to manage related data having complex structure by combining them into objects. An object instance is collection of data elements and operations which is considered an entity. Objects are typed, and the format and operations of an object instance are inherited from the object prototype.

The prototype description for the object type is predefined and the object instances are instantiated as needed for the particular problem. The object prototype provides then a meta description, similar to a schema provided for a database. That description is, however, fully accessible to the programmer. Internally, an object can have an arbitrary structure, and no user-visible join operations are required to bring data elements of one object instance together.

The object concept covers a range of approaches. One measure is the extent to which messages to external operation interfaces are used to provide access and manipulation functions. Objects may be active as in the ACTORS paradigm [Hewitt 77] or passive, as in CLU [Liskov77], or somewhere in between, as in SIMULA or SMALLTALK in terms of initiating actions.

The use of objects permits the programmer to manipulate data at a higher level of abstraction. Convincing arguments have been made for the use of object-oriented languages and some impressive demonstrations exist. Especially attractive is the display capability associated with objects as seen in THINGLAB [Borning 79]. Object concepts can of course be implemented using non-specialized languages, for instance in LISP [Moreau 85] or PROLOG [Mermet 85]. The tasks in EIS seem to match object-oriented concepts well and many experiments have been conducted [Afsarmanesh 85].

Objects and Databases

Let us assume a database is used for persistent storage of shared objects. A database query can obtain the information for an object instance, and an object-oriented programming language can instantiate that object. An interface is needed, since neither can perform the task independently. A view can define the information required for a collection of objects, but the data will not be arranged as expected for a collection of objects. Linkage to the object prototype and its operations is performed in the programming system. The program queries the database non-procedurally to remain unaffected by database changes made to satisfy other users.

The query needed to instantiate an object may seem quite complex to a programmer. A relation is a set of tuples, and, from an idealized point-of-view, each tuple provides the data for some object. However, normalization often requires that information concerning one object be distributed over multiple relations, and brought together as needed by join operations. The base relations must contain all the data required to construct the view tuples; the composition knowledge is encoded into the SELECT expressions used to construct the views. An ideal composition of an object should allow its data to be managed as a unit, but unless non-first-normal form relations — supporting repeating groups — are supported for views. multiple tuples are still required to represent one entity in a relational view [Stonebraker 83].

Hence storage of objects is not easy in databases, as indicated by the extensions proposed to INGRES for such tasks [Stonebraker 85]. A further complexity is that objects themselves may be composed from more primitive objects. In hierarchical databases records may be assembled from lower level tuples, but in relational databases the programmer has to provide join specifications externally to assemble more comprehensive relations from basic relations.

There is some hope that the formal techniques being developed for databases can permit the management of the information required to manage objects. Performance remains a bottleneck, and we will consider this issue later with our proposal. The database community has to be careful not to promise too much, too soon to the object-oriented folk.

Sharing Information in Objects

More serious are the problems we see in the management of shared information within the object-oriented paradigm. We consider two problems:

1 The growth of objects which contain information for multiple design tasks
2 The conflict when object configurations differ for successive design tasks.

Let us first consider the simpler case, where multiple users deal with the same configuration of objects. We will draw our examples from EIS, and consider that the objects are components of a circuit.

In using an EIS the level of abstraction for the objects changes during the process of design. First the object are simple logic elements, the process of design refines these objects to circuit components, and eventually to simple geometric elements projected from each layer of the chip [Spooner 86]. The final objects will carry many data elements not needed at the higher levels. The sketch in Figure 1 symbolizes how objects grow, and loose their vitality.

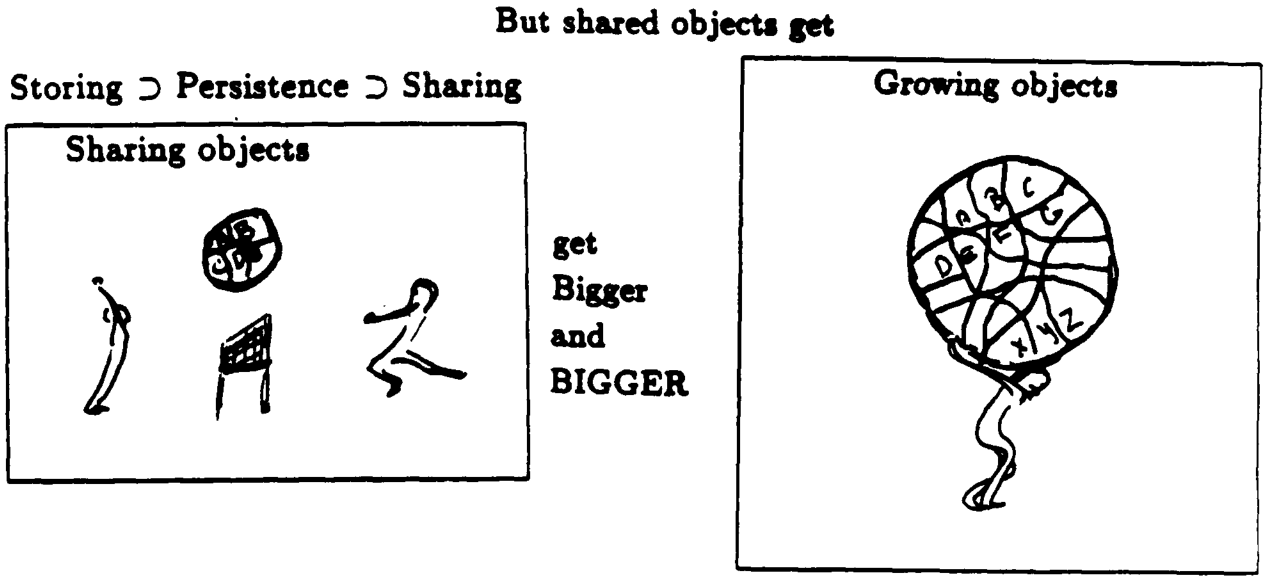


Figure 1b: Growth of Objects As They Try To Satisfy Multiple Design Tasks.

As the design process moves from one subtask to the next successor objects are constructed out of earlier objects. Each new generation has new data fields appended.

Unfortunately, since design often requires cycles, old information cannot disappear. An unacceptable geometry may require a change at the circuit level, say adding an inverter to change a polarity. If design is ıterative then successive transformations of objects must not lose information. This means that objects suitable for one task must contain information relevant to all tasks that may be successors, although much information may be irrelevant to the task at hand. The information may be hidden within the object, but must be passed on correctly, so it remains available when needed.

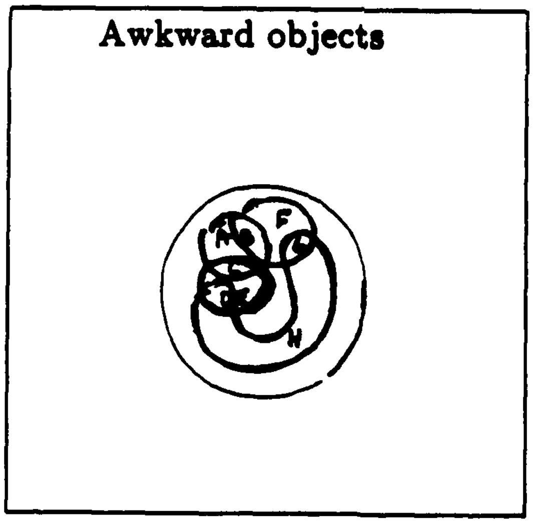
The objects become big, and no longer have the qualities associated with the object-oriented paradigm. A solution to this problem of overloaded objects is to have super-objects, owned by an object czar. Objects for each user task type are created by projection from the super-object. Updating privileges must be well defined.

This solution does not solve the second class of problems, namely sharing object information when the object configuration differs. Now, to create objects for a user task objects may have to be created from combinations of several, and perhaps different objects or super-objects.

Complexity rears its head

Especially when iterations are made
over the information

Objects get



unwieldly.

The benefits of having objects are lost.

- **understandability**
- **naturalness**

Figure 1c: Complexity.

We show in the next section an EIS example having components and nets connecting.

These present different configurations of the same information. In aircraft design the objects serve design tasks as aerodynamics, structures, power, fuel, control, etc., and it is obvious that their configuration is quite different. Even in a simple commercial credit card operation there is the customer as an object for some tasks and the establishments are the objects wanting to be paid in other tasks.

Because of these problems we present later a proposal which will not try to store to objects directly in persistent form. We prefer an new approach to satisfy the demands of database style sharing and object concepts.

2. Looking at some Examples

To clarify the introduction we will take a simple device, a D-type latched flip-flop. At some level of abstraction it is an object, at a lower level it is composed of several gate objects of only three types: **AND**, **INV**, **NOR**, and contacts to the external world. The graphical representation of Figure 2 shows the component objects of the flip-flop at the gate level and the interconnection nets. The components are capitalized and the nets have lowercase labels.

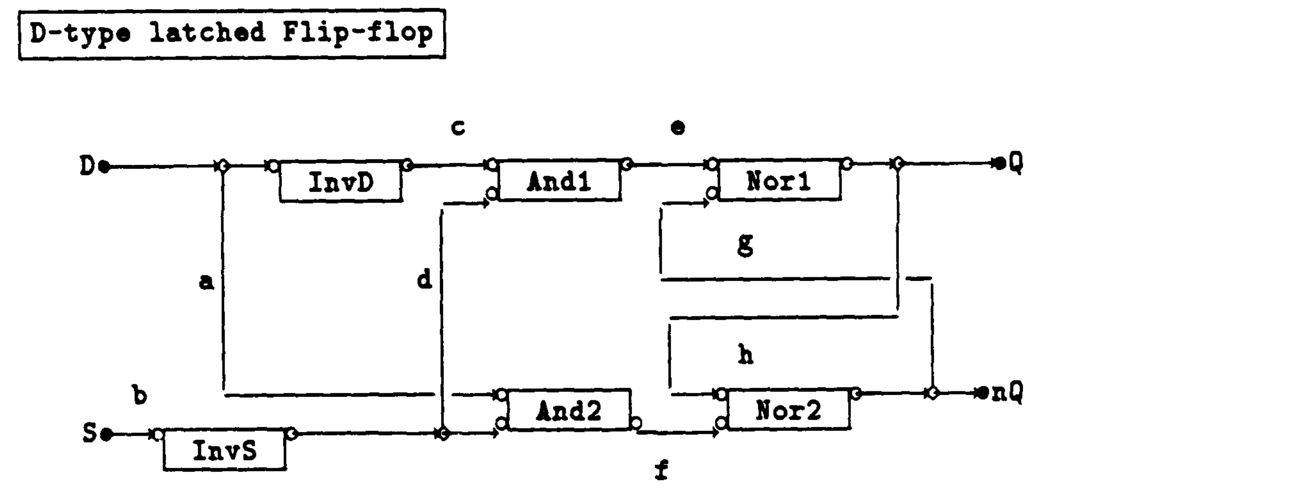


Figure 2: Latched D flip-flop [From Mukherjee 86]

A fully normalized database storing the information describing this circuit requires several relations. We show in Figure 3 the two library relations, which describe the types of gates (`Gates`) and their connection points (`Gate-connects`). Many other values will be kept in such a library. The ruling part or key attributes of each relation are placed ahead of a separator symbol :⟩.

Relation **Gates:**

Gate-type :)	Function,	Area,	No-pins,	Power,	Delay,	. . . ;
Inv	$x = \bar{a}$	30	2	. . .		
And	$x = a \wedge b$	40	3	. . .		
Nor	$x = \overline{a \vee b}$	35	3	. . .		
Cntct	a	900	1	. . .		

Relation **Gate-connects:**

Gate-type,	C-no :)	C-id,	IO;
Inv	1	a	in
Inv	2	x	out
And	1	a	in
And	2	b	in
And	3	x	out

(more)

Gate-type,	C-no :)	C-id,	IO;
Nor	1	a	in
Nor	2	b	in
Nor	3	x	out
Cntct	1	a	inout

Figure 3: Relations describing the gates library.

Figure 4 presents a non-redundant representation for the specific circuit using two more relations, one for each gate instance and one giving the net connections for each gate. Other design-specific information can be kept within these relations.

Relation **Components:**

Id :)	Type,	Role,	Position,	. . . ;
D	Cntct	Data in	1/1	
S	Cntct	Sense in	3/1	
InvD	Inv	Data inverter	1/2	
InvS	Inv	Sense inverter	3/2	
And1	And	And data	1/3	
And2	And	And sense	3/3	
Nor1	Nor	Nor to Q	1/4	
Nor2	Nor	Nor to nQ	3/4	
Q	Cntct	State out	1/5	

Relation **Connections:**

Id,	Pin :)	net;
D	1	a
S	1	b
InvD	1	a
InvD	2	c
InvS	1	b
InvS	2	d
And1	1	c
And1	2	d
And1	3	e
And2	1	a
And2	2	d

(more)

Id,	Pin :)	net;
And2	3	f
Nor1	1	e
Nor1	2	g
Nor1	3	h
Nor2	1	h
Nor2	2	f
Nor2	3	g
Q	1	h
nQ	1	g

Figure 4: A Fully Normalized Description

The representation of the design shown is quite complete, but also fairly unclear. We need to create views which place all relevant data into coherent tuples. A view is needed to analyze the components and their loads; another view is needed to look at the nets, and other views will be needed for timing analysis, heat dissipation, etc.

In Figure 5 we extract two views, `ComponentsView` and `NetsView` for the database of Figure 4, using also the library relations shown in Figure 3. The `ComponentsView` is intended to be appropriate for checking components and their sources and sinks. It primarily accesses the `Components` relation, and joins with its tuples data about the connected components and from the libraries.

The `NetView` is intended to generate data on the interconnection nets for an application doing checking. The primary relation for the `NetView` view is the `Connections` relation, augmented with library information for the connected components.

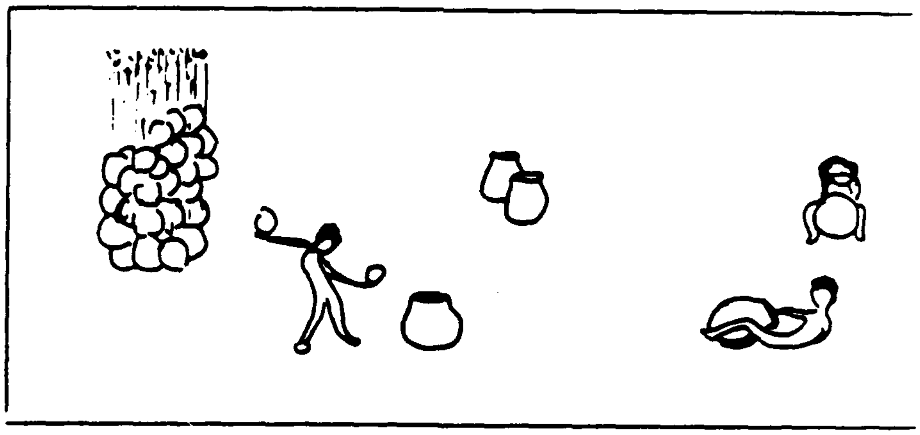
The number of objects in `ComponentsView` is equal to the cardinality of the component relation (10), but the augmentation makes the result much larger (30). The eight nets are represented in the primary relation and in the view by twenty tuples, one per connected point (∘) or external contact (•). In both views we present the tuples in a logical order.

A solution?

Database paradigm

Shared information is NORMALized

Individual information is obtained by a VIEW



for EIS example,

using fully normalized database

two library relations,

two design relations

Many views can be derived from a base relation.

Figure 1d: Creating Objects From A Database.

```
CREATE VIEW ComponentsView (Id, Pin, IdC :) Type, N, IO, IOC)
     AS SELECT C.Id, C.Pin, CC.Id, CM.Type, T.No-pins, G.IO, GC.IO
     FROM Connections C CC, Components CM, Gate-connects G GC, Gates T,
     WHERE C.Id = CM.Id AND C.Net = CC.Net AND C.Type = T.Gate-type
     AND C.Type = G.Gate-type AND CC.Type = GC.Gate-type;
CREATE VIEW NetsView (Net, Dev, Pd :) IOd)
     AS SELECT C.net, C.Id, C.Pin, GC.IO
     FROM Connections C, Components CM, Gate-connects G GC, Gates T,
     WHERE C.Id = CM.Id AND CM.Type = T.Gate-type;
```

ComponentsView :=

Id,	Pin,	IdC :)	Type,	P,	IO,	IOC;
D	1	InvD	Cntct	1	inout	in
D	1	And2	Cntct	1	inout	in
S	1	Inv1	Cntct	1	inout	in
InvD	1	D	Inv	2	in	inout
InvD	2	And1	Inv	2	out	in
InvS	1	S	Inv	2	in	inout
InvS	2	And1	Inv	2	out	in
InvS	2	And2	Inv	2	out	in
And1	1	InvD	And	3	in	out
And1	2	InvS	And	3	in	in
And1	2	And2	And	3	in	in
And1	3	Nor1	And	3	out	in
And2	1	D	And	3	in	inout
And2	1	InvD	And	3	in	out
And2	2	InvS	And	3	in	out
And2	2	And1	And	3	in	in
And2	3	Nor2	And	3	out	in
Nor1	1	And1	Nor	3	in	out
Nor1	2	Nor2	Nor	3	in	out
Nor1	2	nQ	Nor	3	in	inout
Nor1	3	Nor2	Nor	3	out	in
Nor1	3	Q	Nor	3	out	inout
Nor2	1	Nor1	Nor	3	in	out
Nor2	1	Q	Nor	3	in	inout
No[illegible]	2	And2	Nor	3	in	out
Nor2	3	Nor1	Nor	3	out	in
Nor2	3	nQ	Nor	3	out	inout
Q	1	Nor1	Cntct	1	inout	out
Q	1	Nor2	Cntct	1	inout	in
nQ	1	Nor2	Cntct	1	inout	out
nQ	1	Nor1	Cntct	1	inout	in

NetsView :=

Net,	Dev,	Pd :)	IOd;
a	D	1	inout
a	InvD	1	in
a	And2	1	in
b	S	1	inout
b	InvS	1	in
c	InvD	2	out
c	And1	1	in
d	InvS	2	out
d	And1	2	in
d	And2	2	in
e	And1	3	out
e	Nor1	1	in
f	And2	3	out
f	Nor2	2	in
g	Nor2	3	out
g	Nor1	2	in
g	nQ	1	inout
h	Nor1	3	out
h	Nor2	1	in
h	Q	3	inout

Figure 5: Component View and Net View

This view is still awkward, since component entities require multiple tuples. A more reasonable presentation would delete redundancies and add implicit non-first-normal-form bindings by rearranging columns according to the source relations. The Nor1 component shown within Figure 5 would then have a object data structure as shown in Figure 6. We add a column N which gives the number of connected components. The computation of N using SQL requires GROUP BY and COUNT operations.

Id,	Type,	P,	Pin,	N,	IO,	IdC,	IOC;
Nor1	Nor	3	1	1	in	And1	out
			2	2	in	Nor2	out
						nQ	inout
			3	2	out	Nor2	in
						Q	inout

Figure 6: Reduced Datastructure for a View Object.

We now describe in Figure 7 this structure in the object-oriented extension of C, namely C++ [Stroustrup 86]. In C++ structures have to be mappable at compile-time, so arrays of dynamic extent cannot be embedded:

```
enum io { IN, OUT, INOUT };
class Cpin;
class CCpin;
class ComponentsView
{
    char     Id [8];
    char     Type [6];
    short    P;  //  Component pin count
    Cpin* Pin;
};
class Cpin
{
    io       IO;
    short    Q;  // Connected component count
    CCpin* C;
};
class CCpin
{
    char*   IdC;
    io      IOC;
};
```

Figure 7: C++ Datastructure for Component Objects.

Since it is permissible in C++ to have a dynamic array as the last element of a class by writing an appropriate constructor, it is possible to bring the CCpin array inboard.

Views and Objects

Let us now reconsider the similarities between views and object concepts. Both are intended to provide a better level of abstraction to the user, although the database user is seen to manipulate sets of objects in a non-procedural notation while the objects are manipulated procedurally and iteratively [SmSm 77].

The *collection* of tuples of a view is defined by the query which generates the set, and described by the relation-schema associated with the view query. The set of objects is defined by the user initiated action of generation and described by the prototype. Object-oriented languages may have a `ForAll` primitive to rapidly generate collections of objects of some type.

The description of the relation has to be available to the the relational user, since no implied operations can be kept in the relation-schema. There are proposals to store object defining procedures in relational schemas, but these have not yet been tested [Stonebraker 85].

Both tuples and objects can be selected based on any of multiple criteria, and interrogated to yield data for processing. View update may be severely restricted due to ambiguities in base relation update [Dayal 82]. Object update can be restricted by having only limited access functions, but otherwise no constraints are imposed on the programmer, although consistency problems can easily arise among users sharing objects.

Basic, of course, is the difference in persistence. Databases, and hence views over them, persist between program invocations. Objects must be explicitly written to files to gain persistence. Related to persistence is the critical issue to be addressed, namely the multiplicity of views that can be derived from a base relation. Figure 5 showed a view of **components** and a view of **nets** derived from the relations of Figure 3 and 4.

3. The Proposed Architectural Unit: View-Objects

We now propose to combine the concepts of views and objects, as discussed above, into a single concept: view-objects. This proposal is motivated by noting that systems, suitable for engineering information problem support require both sharability and complex abstract units, i.e., both views and objects. We use the concept of views to generate a working set of objects corresponding to projections of the entities described in the database [Carter 86]. The generators may need to access multiple relations to reconstruct the entities hidden to the relational representation.

Components of the architecture are:

1. A set of **base relations.**
2. A set of **view-object generators.**

In a complete version of this approach we also require the inverse function of 2., namely,

3. A set of **view-object decomposers** and archivers.

The conceptual base relations serve as the persistent database for applications under this architecture. They contain all the data needed to create any specified object type. Conventional database technology should be adequate for development, but eventually performance demands may drive users to specialized databases. We will consider the options for physical organization later.

Concurrent access by long-lived transaction will require a capability to conveniently access prior versions of the data. Where database management systems do not serve that need intrinsically the database must be configured with additional time-based attributes. As experience is gained this service will be included in specialized systems being built.

The view-object generator is the new component in this architecture. It performs the following steps:

a. The view-object generator extracts out of the base database data in relational form as needed — as now done by a query corresponding to a view. A view tuple or set of view tuples will contain projected data corresponding to an object. A view relation will cover a selected set of objects.

b. The view-object generator assembles the data into a set of objects. The objects will be made available to the program by attaching them to the predefined object prototype.

The view-object generator needs information other than the data, i.e., knowledge to perform its functions:

a1. The query portion identifies the base data.

a2. The specification of the primary relation identifies the object-entities.

a3. The object prototype specifies the structure and linkages of the data elements within an object, and the access functions for the object.

Initially the view-object generators will be implemented as code, closely following the translation programs in use now to convert persistent storage representations of engineering data files into representations suitable for specific tools. For experiments a relational database can be used for storage of the base data, but expect that ongoing developments in EIS will provide systems with more appropriate functionality and performance [IDA:86].

The goal is to eventually provide non-procedural access for object-oriented approaches as well. By formalizing the semantics of the objects required by the tools, and interpreting the object type description we believe that the view-object g^enerators may be automated. The programmer then only provides the object type declaration and the WHERE clause defining the desired set of object instances.

With increased storage of data semantics in extended schemas one may advance further. With structural or dependency information about the base relations automatic generation of the internal structure of an object type may become feasible. Since access to the objects by the tools is still procedural, the performance benefits of automated object generation would be minor. The major benefit is in the control of access and consistency which may be gained [Neuman 82].

Justification

We are proposing a major extension to database and object-oriented concepts, with the intent to obtain the joint benefits that each approach provides separately today. The effort must justified by these benefits.

3.0.1 Sharable access to objects

The proposed architecture supports procedural access to objects, as expected by object-oriented programming systems. Access to the base data is automatic, and non-procedural. This permits base data to be effectively shared, since no single application or combination of applications imposes a structure on the base relations.

3.1 Growth of the system

New object types can be defined by adding new view-object generators. As is expected in databases, new data instances, types, and entire relations can be added without affecting other users' programs. Data can be reorganized without changing the object generator code since only the views will be involved. Such flexibility is essential for growth and multi-user access.

3.2 Support for a wide variety of representations

When simple, tabular results are to be obtained from the database, a view-object generator can easily generate tables for direct inspection and manipulation. A graphics object-generator can project the attributes needed for visual display.

3.3 High performance access from the database

To achieve this goal new database interfaces must be developed, which make the benefits of the set-oriented database retrieval concepts available to programming languages. Current database interfaces create a bottleneck when delivering data to programs. A common method for relational systems is to accept a query which specifies a set, but then require repeated invocations which deliver only single values or records at a time. This access mode, due to conventional programming techniques, is clearly inadequate.

The data from the databases is to be inserted into sets of objects at a time. For the object-oriented programmer such a set is best defined as a super-object. A program typically can not proceed until the set is complete. The object generators need only be invoked once for all the data needed to compose an super-object. An effective view-object generator hence needs an interface at a deeper level into the existing relational functions.

3.4 Updating from objects

The programs can freely update the contents of the objects. Some applications require that these changes be made persistent and hence be moved from its object representation to the base database.

To achieve update we envisage a third architectural component, the view-update generator. This component can be invoked at commit-time, when results effecting the objects are to be made persistent. We envisage that such a process will only update from objects which have changed, and only replace values which were changed.

Where views have been constructed using joins, aggregation operators and the like, automatic view update can be ambiguous. Restrictions on object update are one solution [Rossoupolous 86]. However, as shown by [Keller 86] such ambiguities can be enumerated and a choice can be made when the view-update is generated. The view update generator can also take advantage of the semantic knowledge available to an advanced view-object generator. An important source of knowledge constraining updates is authorization information.

4. Cost Tradeoffs

What costs and benefits will this architecture provide other than what we see as its structural advantage? We will now review areas where performance may be gained versus one of the two underlying approaches — database use or object-oriented programming — alone.

Set-based data access

A single invocation of a view-object generator will instantiate all specified objects. The overhead of programmed access to relations, typically requiring an intial CALL giving the query and then iterated CALLs to obtain the result piece by piece is avoided. Also no sequence of object instance generating calls, as seen in object-oriented programming is needed. Of course, one invocation of the object generator will be major operation, but only one call should be needed per object type.

The object generator may also perform the so-called macro-generator function, where instances of design objects are generated based on a general prototype and parameters. The source of such information is a library of cell descriptions, permitting, say, the generation of a series of cells making up a register.

The view-object generator will be more complex than either a database retrieval alone or an object prototype. It will provide a very clear definition of the mapping from base data to objects and do so in localized manner. A partial example was given, using the SQL approach to describe the view in Figure 5 and then using C++ to define the storage structure for the objects in Figure 7.

Binding of retrieved data

The ability of the view-object generator to bind the object will pay off in processing time. No joins are needed at execution to bind relational tuples, and no search operations are needed to assemble tuples belonging to the same object. Since the required information exists at view generation time no search cost is expended. The only requirement is that the object's internal data structure permits retention of the information.

Task allocation which matches hardware system components
Implicit, but not essential to our proposal, are the complementary concepts of database servers and design workstations. They will be linked by a powerful, but often still bandwidth limited communication network [Roussopoulos 86].

In such an architecture we expect that most of the retrieval and restriction operations will be carried out on the machine serving the database and the object generation and use will occur in the workstation.

Optimization in the file server The file server can select optimum retrieval strategies and reduce the data volume needed to convey the information. Keeping data in sorted order and removing redundant fields as shown in Figure 6 is straightforward and effective.

All operations are specified non-procedurally and can be rearranged and interleaved as needed to handle requests from the users. Concurrency control and version management are tasks best handled in the file server.

Communication minimization The data volume is reduced in the file server. Communication bandwidth between server and workstation can be used fully for information, rather than for data to be ignored in the workstation.

High performance on workstations The workstation only needs to assemble the information obtained into the desired object configuration.

It is desirable that all objects for an application task can be placed into the real memory of the working processor [Holland 85]. The objects now do not contain data fields irrelevant to the process at hand. Since these working objects are now compact, the probability that they will all fit into a modern workstation is increased.

Virtual memory management can be invoked if needed, but the capacities of modern machines are such that there should be adequate memory space for working storage of the required objects. However, we can envisage several techniques to exploit having the data independence provided by view generators to improve locality for virtual memory management.

5. Physical Organization of the Database

Performance issues are considered critical in Engineering Information Systems, and experiments with relational databases have not given us confidence that adequate performance is easy to obtain [Milton 83], [Narfelt 85]. Performance will furthermore be impacted by the extensions, such as version support, which are needed in the engineering environment [Katz 84]. Many of these extensions will also be useful for broader objectives, so that there is a motivation to share the development and maintenance costs of relational database systems extensions.

Objects are seen as a means to solve the performance problem, because data is bound in a user-sensible manner [Ketabchi 86]. We believe that such binding can indeed be significant in workstations, although our own experiments have not given proof of that assumption when external storage is used [Winslett 85]. There are obviously many more factors to be considered simultaneously when building comprehensive systems [Zara 85], but storing data in relational tables is often seen as a critical issue.

The fully normalized model of the representation has as its objective the minimization of redundancy and the avoidance of a number of anomalies that can occur. It also supports a very simpler storage structure, as seen in our example that can lead to a large number of relations. Retrieval from such an organization will require a large number of joins, as seen in the example leading to Figure 5.

Updates seem to require less work in a relational database than in our proposal. However, when the database has to obey interrelational contraints, expensive joins must also to be executed when the data are updated. For instance, it is neccessary to assure that the components exist and that the connections are valid when a net connection is to be changed.

There is, of course, no requirement for the physical storage structure to mimic literally the logical relational structure. Specifically, information from multiple tuples may be stored in one record. Denormalization has been formally considered [Schkolnick 80], but is not supported by current DBMS. It is commonly employed in practical relational databases [Hardwick 80]. Preserving correctness in a denormalized database requires often additional transformation and care: records may have repeated fields and at other times some fields may be replicated in multiple records. Sometimes here work is required as well: dependent tuples, as connection points for a component, will automatically be deleted if they were implemented as a repeating group. We expect to profit here from ongoing research on non-first-normal-form databases [Roth 84].

In our architecture, where we expect that the object generator is the primary means for accessing data, complexity issues discouraging use of a non-normalized storage structure are moot. The object generators can be adjusted, probably eventually automatically, to any storage structure chosen. It is likely that the most efficient storage structure will be similar to the dominant object structure.

With such an organization we have provided the two primary objectives favoring the concept of object-oriented programs *for computer-aided design:*

1 The view-objects provides the desired clean and compact user interface.

2 The flexibility of the storage structure can provide the locality needed to achieve high performance in source data access.

We can now provide these objectives for multiple users, and share selected information in a controlled fashion. We gain greatly from the flexibility obtained by interposing the object generator operating on a persistent base storage structure.

We are no longer bound to initial conceptulization of an the optimal data storage structure bound to an optimal object format. It is likely that in any design project the data retrieval loads will change over time. The object types which dominate use during initial design phases are not likely to be the object types used during the maintenance phase of the designed devices [Tichy 82]. A physical storage reorganization is now possible, as long as the view-object generators are adjusted correspondingly.

Replication, the primary tool to improve performance, can be utilized as well. A high demand subset of the database can be replicated and made available in a performance effective manner to the object generators. Object update functions can use primary copy token concepts [Minoura 82] to update all copies consistently and synchronously.

6. Conclusion

We argue that direct storage of objects, to make them persistent, is not appropriate for large, multi-user Engineering Information Systems. We propose storage using relational concepts, and an interface which generate objects. Initial versions of the interface may be coded directly, to test the validity of the concept. In the longer range we look towards knowledge-driven transformations.

Generation of objects from base data brings the advantages of sharing persistent information to an object-oriented program. It also provides a better interface from programs to databases, recognizing that access by programs, large or small, will remain essential for data analysis and complex transactions.

The separation of storage and working representation will also simplify the development of new approaches to engineering design [Brown 83], [Hartzband 85]. The object generator can be viewed as a system component utilizing knowledge to select and aggregate data for the information objectives.

Acknowledgement

This paper has benefited from interaction with a VHSIC program sponsored project to define information models for Engineering Information Systems, although it certainly does not reflect the consensus of the group. Comments by Tony Gadient of the USAF, WPAFB, Arnold Goldstein of MCC, and Prof. Alice Parker of USC were constructive and helped focus the issues. Earlier presentations of these concepts at Workshops for Integrated Design and Manufacturing Processes at the Naval Postgraduate School in Monterey CA, April 1986 and at the object-oriented Database Workshop engendered useful feedback. David Beech, from Hewlett-Packard, visiting the Stanford Center for Integrated Systems, provided motivation to consider the object-oriented approach and gave constructive criticism to earlier versions of this paper. He also provided the C++ example for Figure 7. Support for this research was provided in part by DARPA contract N39-84-C-0211 for Knowledge Based Management Systems and in part by the Air Force through a contract with UES F33615-82-C-1716.

References

We can cite here only a few of the many publications which have provided concepts leading to this paper. We include some recent global references to provide access to other work. A more complete list ca.. be obtained from the author.

[Afsarmanesh 85] Afsarmanesh,H., Knapp,D., McLeod,D., and Parker,A.: "An Extensible Object-Oriented Approach to Databases for VLSI/CAD"; Univ.of Southern Cal., CSD, TR-85-330, Apr.1985.

[Ashok 84] Ashok,V., McKnight,W., and Ramanathan,J.: "Integrated Environment for Information Management in VLSI Design"; IEEE Data Engineering Conf. 1, Los Angeles, Apr.1984.

[Bancilhon 85] Bancilhon,F., Kim,W., and Korth,H.F.: "A Model of CAD Transactions"; VLDB 11, Aug.1985, Stockholm.

[Batory 80] Batory,D.S. and Kim,W. "Modeling Concepts for VLSI CAD Objects"; *ACM TODS*, vol.10 no.5, Sep.1985, pp.322–346.

[Beech 83]] Beech,D. and Feldman,J.S.: "The Integrated Datamodel: A Database Perspective"; Proc. VLDB 9, Florence, 1983.

[Borning 79] Borning,A.: THINGLAB, *A Constraint Simulation Laboratory*; Rep.No. CS-79-746, Stanford Univ., 1979.

[Brown 83] Brown,Harold,D., Tong,C., and Foyster,G.: "Environment for Circuit Design"; *IEEE Computer*, Vol.16,No.12, Dec.1983.

[Carter 86] Carter,Harold W.: "Computer-Aided Design of Integrated Circuits"; *IEEE Computer*, vol.19 no.4, Apr.1986, pp.19-36.

[Dayal 82] Dayal,U. and Bernstein,P.A.: "On the Correct Translation of Update Operations On the Relational View"; *ACM TODS*, vol.7 no.3, Sep.1983.

[DittrichD 86] Dittrich,K. and Dayal,U. (Eds.): "Proceedings 1986 International Workshop on Object-Oriented Database Systems"; IEEE CS Order no.734, Sep.1986

[Fischer 83] Fischer,G. and Boecker,H.D.; "The Nature of Design Processes and how a Computer System can Support Them"; in Degano and Sandewall(eds): *Integrated Interactive Computing Systems*, North-Holland 1983.

[Gray 80] Gray, Mike: "Databases for Computer-Aided Design"; in *New Applications of Databases*, Gardarin and Gelenbe(eds), Academic Press 1984.

[Hardwick 80] Hardwick,M.: "The Texas Tech Abstraction Management System"; Tech.Rep., Texas Tech Univ., Lubbock TX, Received Mar.1985.

[Hartzband 85] Hartzband,D.J. and Maryanski,F.J.: "Enhancing Knowledge Representation in Engineering Databases"; IEEE Computer, Vol.18 no.9, Sep.1985, pp.39-48.

[Hewitt 77] Hewitt,C.: "Viewing Control Structures as Patterns of Passing Messages"; *Art.Intell., vol.8, pp.323-364., 1977.*

[Holland 85] Holland,L., Korn,G., Matson,J., and Wolfe,P.: "Engineering Support System Software"; IEEE Micro, Vol.5 no.5, Oct.1985, pp.17-21.

[IDA 86] Institute for Defense Analysis: *Engineering Information System Specifications, 2 vols.*; Report prepared for the VHSIC program, Wright-Patterson AFB, OH, June 1986.

[Katz 83] Katz, Randy H.: "Managing the Chip Design Database"; IEEE Computer, Dec.1983, pp.16-36.

[Katz 84] Katz,R.H. and Lehman,T.J.: "Database Support for Versions and Alternatives for Large Files"; IEEE TSE, vol.SE10 no.2, Mar.1984, pp.191-200.

[Keller 86]
Arthur M. Keller: "Choosing a View Update Translator by Dialog at View Definition Time"; *IEEE Computer*, January 1986.

[Ketabchi 86] Ketabchi,M. and Berzins,V.: "Component Aggregation: A Mechanism for Organizing Efficient Engineering Databases"; IEEE CS Data Engineering Conf.2, Los Angeles, Feb.1986.

[Liskov 77] Liskov,B., Snyder,A., Atkinson,R., and Schaffert,C.: "Abstraction Mechanisms in CLU"; CACM, vol.20 no.8, Aug.1977, pp.564-576.

[Lochovsky 85] Lochovsky,F. (Ed.): *Object-oriented Systems*, special issue of the IEEE CS Database Engineering Bulletin, December 1985; reprinted in Database Engineering 1985, IEEE CS order no. 758, 1986.

[Lorie 83] Lorie,R.(ed): "Engineering Design Applications"; ACM-SIGMOD 83 vol.2, Proc. Database Week, San Jose, May.1983.

[Mermet 85] Mermet,J.: "CASCADE: vers un systeme integre de CAO de VLSI"; INRIA Bull., no.102, Jun.1985, pp.16–29.

[Milton 83] Milton,Jack and Wiederhold,Gio: "Network and Relational Systems in VLSI Design: Contradictory Performance?"; IEEE CS DataBase Engineering, Vol.1, 1983, pp.167–170.

[Minoura 82] Toshimi Minoura and Gio Wiederhold: "Resilient Extended True-Copy Token Scheme for a Distributed Database System"; *IEEE Transactions on Software Engineering*, vol.SE-8 no.3, May 1982, pp.173–188.

[Moreau 85] Moreau, J-P. and Vuilllemin,J.: "Le Projet SYCOMORE"; INRIA Bull., no.102, Jun.1985, pp.2–10.

[Mukherjee 86] Mukherjee,A.: *Introduction to nMOS and CMOS VLSI Systems Design*; Prentice-Hall, 1986.

[Narfelt 85] Narfelt, K-H. and Schefstrom, D.: "Extending the Scope of a Program Library" ; Ada in Use, Barnes and Fisher(eds), Ada Letters, Vol.V no.2, Sep.-Oct.1985, pp.25–40.

[Neuman 82] Neuman,T. and Hornung,C.: "Consistency and Transactions in CAD Database"; VLDB 8, McLeod and Villasenor(eds), Mexico City, 1982, pp.181–188.

[Roth 84] Roth,M.A., Korth,H.F., and Silberschatz,A.: "Theory of Non-First-Normal Form Relational Databases"; Tech.Rep.84-38, Dep.of CS, Univ.of Texas, Austin, Dec.1984.

[Roussopoulos 86] Roussopoulos,N. and Kang,H.: "Preliminary Design of ADMS ±: A Workstation-Mainframe Integrated Architecture for Database Management Systems"; VLDB 12, Kyoto 1986, pp.355-362.

[Schkolnick 80] Schkolnick,M. and Sorensen,P.: "Denormalization: A Performance Oriented Database Design Technique"; AICA '80, Bologna Italy, Oct.1980.

[SmSm 77] Smith, John and Smith, Diane C.P.: "Database Abstractions: Aggregation and Generalization"; *ACM TODS*; vol.2 no.2. June 1977, pp.105–133.

[Spooner 86] Spooner,D.L., Milicia,M.A., and Faatz,D.B.: "Modeling Mechanical CAD Data with Data Abstraction and Object-Oriented Techniques"; IEEE CS Data Engineering Conf.2, Los Angeles, Feb.1986.

[Stonebraker 83] Stonebraker,M., Rubenstein,B., and Guttman,A.: "Application of Abstract Data Types and Abstract Indices to CAD Data Bases"; ACM-SIGMOD 83, Database Week, San Jose CA, May.1983, pp.107–113.

[Stonebraker 85] Stonebraker,M. and Rowe,L.: "The Design of POSTGRES"; Tech.Report UC Berkeley, Nov.1985.

[Stroustrup 86] Stroustrup, B.: *The C++ Programming Language*; Addison-Wesley, 1986.

[Tichy 82] Tichy,Walter F.: "Design Implementation and Evaluation of a Revision Control System"; Proc.6th Int. Conf. Software Eng., Tokyo, Sept.1982.

[Udagawa 84] Udagawa,Y. and Mizoguchi,T.: "An Advanced Database System: ADAM — Towards Integrated Management of Engineering Data"; IEEE Data Engineering Conf. 1, Los Angeles, Apr.1984.

[Wilkins 85] Wilkins,M.W., Berlin,R., Payne,T., and Wiederhold,G: "Relational and Entity-Relationship Model Databases and Specialized Design Files in VLSI Design"; ACM-DA 22, 1985, pp.410–516.

[Zara 85] Zara,R.V. and Henke,D.R.: "Building A Layered Database For Design Automation"; ACM-DA 22, 1985, pp. 645-651.

Object Oriented Databases: Motivations and Approaches

Surajit Chaudhuri
Stanford University

1 Introduction

Object-oriented database systems is currently an active area of research. This development has been motivated largely by the success of object-oriented programming systems in modeling applications. There is considerable interest in object-oriented databases for developing Engineering Information Systems (EIS). EIS includes CAD, CAM and CASE systems.

An object-oriented database should have a data model that captures the style of object-oriented programming and an implementation that meets the performance requirements of the applications. Research in object-oriented databases has resulted in the development of many exciting ideas. However, differences in proposed data models often make it hard to compare and evaluate the contributions made to this field. This paper defines a framework to discuss and compare some of the important ideas in the area of object-oriented databases.

The rest of the paper is organized as follows. Section 2 introduces our framework which is used to identify prominent research directions in object-oriented databases. The following sections are devoted to a survey of the past work and a discussion of future research in the context of this framework. We summarize our study and mention a few related issues in the conclusion.

2 Framework

In this section, we propose a framework to discuss past developments and research issues in object-oriented databases. The framework is based on those concepts of object-oriented databases that distinguish it from the relational systems. Based on our survey of recent work in this area, we conclude that the key aspects of object-oriented databases are the following:

- Abstraction mechanisms in the data model
- Schema management to maintain semantic relationships among types and to aid in schema evolution
- Performance considerations in data access and storage

The rest of this section introduces these three concepts in some detail.

Abstraction Mechanisms: Most programming languages use data structures that are more complex than the tuples of the relations. Below, we provide some examples of the common data structures:

Example 1 (Nested record structure): The *Person* relation has two attributes *name* and *address*. The latter is a nested attribute consisting of several attributes:

$$Person(name, address(street, apartment, city, state))$$

An instance of this relation is as follows:

$$Person(bob, (ventura, 1, PA, CA))$$

Example 2 (Set valued attribute): The *Person* relation in this example has two attributes *name* and *employed_by*. A person may be employed by more than one organizations and therefore the latter is set-valued:

$$Person(name, employed\_by\{employer\})$$

An instance of this relation is as follows:

$$Person(bob, \{IBM, HP, Xerox\})$$

Example 3 (Sequence-valued attribute): Assume that the *employed_by* attribute denotes the chronological job history of the person. Then, the schema may be denoted as follows:

$$Person(name, employed\_by < employer >)$$

Example 4 (Recursive structure): Assume that we want to represent the manager of every person. A manager is a person and therefore one could represent the manager by mentioning his name in the *manager* attribute. Instead, if we represent the manager by an identity (conceptually similar to pointer to the record of the manager), then we need no more rely on the assumptions that names be unique:

$$Person(name, manager[Person])$$

An instance of this relation is as follows:

$$Person(bob, I567)$$

$I567$ is a permanent identity of bob's manager.

We have illustrated three aspects of data structuring (nested attributes, set-valued attributes and identity) which may be found in object-oriented models. A model supports one or more of the abstraction facilities mentioned in the examples above. Many algebras have been proposed for the richer structures discussed in the examples. The necessity for exten ling the relational algebra should be obvious. For example, the traditional selection operation of relational algebra needs to be redefined for set-valued attributes. For a set-valued attribute, a selection constraint may be interpreted as existential (i.e., true if at least one member of the set satisfies it) or universal (i.e., true if all members of the set satisfy it). We will refer to such abstraction, derived from the use of the complex structures, as *structural abstraction* .

Although the structural abstraction provides operations over the data structures, is suffers from the drawback that the algebra does not take into account the semantics of the domains over which the complex structures are defined. Procedures are needed to capture the additional semantics. This suggests an approach to abstraction based on supporting procedures. In such an abstraction mechanism, procedures characterize one or more types whose representations are hidden (*encapsulation*). We term this kind of abstraction as *procedural abstraction*. The challenge is to propose an appropriate computational model and to find means to optimize the procedures.

Schema Management: EIS as well as other complex applications need to have a large number of interrelated types. The referential integrity is the only relationship that is supported by the current relational systems. We need to represent additional dependencies among types to capture the semantics of the applications. This area of research is related to past work in semantic data modeling [Hul 87]. Another challenge in a system with large number of types is that of providing a good environment for querying and incrementally updating the schema. This aspect of schema management is related to *inheritance* techniques in object-oriented programming languages [Bob 85].

Performance of access and storage: The access pattern for EIS applications does not quite match the traditional set-oriented retrieval offered by the relational storage. These applications are characterized by having first an associative access phase; followed by intensive operations on a few objects [Zan 86]. The associative phase needs to be supported by appropriate access paths over storage structures. Although traditional relational storage systems can be used, some of the implementations have devised new object storage systems. Operations on the selected set of objects can be carried out in main memory. Therefore, efficient binding of objects to main memory data structures must be considered. For example, depending on the application requirements, part of an object, an object, or an object and all its related objects may need to be materialized. The ability to *specify* a binding pattern is useful.

We now discuss the main issues and the past developments in the issues of abstraction, schema management, access and storage structures in object-oriented databases.

3 Abstraction in Databases

Object Identity: A database represents relationships among objects. In the relational model, it is not possible to distinguish between two objects that have the same set of properties. However, we are accustomed to objects in the programming languages (e.g., data structures in memory) that have the same properties (i.e, same content) but who may be distinguished (e.g., by virtue of the addresses). We introduce the concept of *object-identity (o-id)* to represent such objects. An o-id provides a uniform designation of an entity. It is immutable and persistent. Unlike keys in relational databases, the object-identity is independent of the properties of the object. Thus, two distinct objects may coexist which have identical properties. The only predicate defined over object-identities is equality. We will study how the concept of object-identity has been incorporated in the models that provide structural abstraction. In the context of procedural abstraction, an interesting interaction between object-identity and the recursive views will be mentioned.

Structural Abstraction: A structural abstraction facility provides the specification of a structure, and an algebra (or a calculus) to manipulate the structure. The set of data models that we will mention here may broadly be divided into two groups. The first set of models that we discuss do not support the notion of object-identity. However, these nested and set valued attributes may be represented in these models.

Models without Identity: In this section, we discuss the data models that extend the relational structures with nested and set valued attributes. An example of such an extension to the relational model is the nested normal form [Rot 84]. In this model, the structure corresponds to set-valued nested attributes:

Example 5: Every person has name, and a set of cars, each of which has the name of the manufacturing company and the year of production:

$$Person(name, cars\{(company, year)\}$$

An example tuple will be as follows:

$$Person(bob, \{(SAAB, 78), (Honda, 86), (Merc, 87)\})$$

The relational algebra is extended by the operators *nest* and *unnest*. Nest converts a flat schema such as $Parent(father, child, age)$ to a nested one: $Parent(father, children\{(child, age)\}$ and unnest flattens a nested relation into the first-normal form. The only selection predicate between any two nested attributes is the *equality* of values. The *set membership* predicate is used for checking presence of a tuple in a nested relation. The nesting operation was studied also in the structural model [Wie 79]. A similar algebra has been proposed by Zicari [Zic 87] for office documents. In the latter model, instead of *set*, *sequence* is used as a constructor. Zicari illustrates some algebraic operations that are useful in practice.

The algebras defined over such complex structures extend the relational algebra. Such algebras are able to capture the representation discussed in the examples 1,2 and 3. Additional predicates (such as membership) are needed to relate set-valued attributes. Bancilhon [Ban 87] presents an algebra which defines a partial-order relation over the set of complex objects. According to that definition, the set of complex objects forms a lattice with respect to this partial order. Two binary operations *union* and *intersection* are defined using the partial order over the complex objects.

The *restructuring* operations are also introduced to transform a nested structure into a flat form or vice versa. The meaningful structure transformations depend on the semantics of the applications. *Path expressions* (e.g., person.child.age) are often used to denote the relation corresponding to a nested attribute. Finally, one must redefine semantics of the traditional relational operations.

Models with Identity: From the point of view of structural abstraction, o-ids provide the functionality of the pointer data structure in programming languages. A simple example of a model with identity is RM/T [1][Cod 79]. It is a model with o-ids but has only relational structures. However, example 4 may be captured by RM/T. Neither nested nor set-valued attributes are present in RM/T. In the LDM [2]model [Kup 85], an attribute could be set-valued. Thus, a set of o-ids (instead of values as in the models without identity) denote the property of an attribute.

DMDM [3] [Mai 85] may be seen as a generalization of both LDM and the nested relational systems. DMDM allows nested attributes in addition to the set-valued attributes. An object may have multiple types in DMDM. The query language permits expressing constraints such as two objects (or sub-objects) being the same. Thus, DMDM is an amalgam of the nested relational form the notion of o-id.

Both LDM and DMDM lack the *set-constructor* which is necessary to implement the *nest* operator. The absence of *nest* restricts the ability to restructure the complex objects.

Procedural Abstraction: In order to express the behavior, we manipulate the data structure which represents the information. The question is how powerful such a manipulation language needs to be? The algebra provides basic manipulative capabilities and can be used for defining *views*. The views enable abstract structures to be defined in terms of the extensional structures.

All the models for structural abstraction that have been defined in the last section as well as the relational model provide a view mechanism. For example, the view-definition facility in DMDM is used to define new types as well as to define new attributes for an existing type. The latter is equivalent to the definition of an attribute in a POSTGRES relation using the generic Quel facility.

Recently, recursive query languages have been proposed for relational databases. Most of these extensions are based on least fixpoint semantics. Use of recursive rules in a data model that supports object-identity further complicates the termination. In such a model, the semantics of a rule may be interpreted as *generation* of an object with properties as in the head of the rule when existing objects in the database match the rule body. Under such an interpretation, for a recursive rule, the computation may not terminate. Such consideration forced DMDM to restrict its view mechanism to be non-recursive. Bancilhon [Ban 87] uses a notion of a *maximal object* as the semantics of the rule. In his model, recursion does not necessarily create termination problems because no two objects may be equal in value (i.e., o-ids are not supported).

Since there will always be applications that need more manipulative power, it has often been argued that one should use the full power of a programming language for manipulation. In other words, the database and the programming language must be integrated. One could then use the data structuring facilities of the programming language and define suitable abstract data types for encapsulation. From the point of view of performance, one could hope for a better management of the store in such an integrated environment [Zdo 87]. However, satisfactory optimization of a query written in a programming language becomes very hard. The following important issues must be examined for any proposal for procedural abstraction:

- Support for associative access

[1]Relational Model/ Tasmania
[2]Logical Data Model
[3]David Maier's Data Model

- Optimization
- Computational Model

We will now illustrate these issues with some examples of the schemes that have been proposed for procedural abstraction:

1. IRIS [Lyn 86] uses functional programming as the computational model. The query is a functional expression instead of a relational expression. Although optimization of IRIS has not been discussed in the literature, the restricted control mechanism of the functional language provides opportunities to utilize rewrite rules. OSQL [Lyn1 88] provides a language for associative access.

2. The POSTGRES model [Sto 86] allows Quel and C-procedures as the value of a relational attribute in a tuple The query interface remains relational as the procedural abstractions are hidden as attributes of the relations. POSTGRES also allows Prolog style rules to be used as the computational model. The optimization techniques are limited to selective materialization and relational query optimizations [Jhi 88].

3. Bancilhon [Ban1 86] proposes a relational structure for an object and uses logic programming (Typed Prolog) as the computational model to define the procedural abstraction. This language is used to define methods. The query interface is relational and datalog optimizations may be used.

4. OPAL [Mai 85] has Smalltalk as the computational model. A *path syntax* is proposed for associative access and optimization.

Procedural abstraction is extremely attractive from the point of view of application programmers. However, the performance problem makes support of such abstraction costly. Complete encapsulation suffers from the difficulty of specifying and maintaining indexes [Mai 86]. The indexes on procedures correspond to materialized views. Incremental maintenance of these views is an intrinsically hard problem.

Directions in Capturing Abstraction: Databases deal with large volumes of data. Therefore, optimization of queries is extremely important. Optimization of a query language demands that the number of features in the language is few and that their interaction be well-studied [Mai 87]. The importance of associative access indicates that an algebra based on structural abstraction will be useful in itself and as a substrate for building procedural abstractions. We have mentioned some of the algebras that have been proposed for complex objects. Although these may be compared formally from the point of view of expressivity and computational properties, a closer look at the target applications is necessary to ensure that an algebra is useful. We could construct canonical queries and compare how they may be modeled by the different object formalisms. The canonical queries could be drawn from an appropriate domain such as EIS.

4 Schema Management

One of the advantages of relational databases is the ability to query the schema. The schema is stored in relational form and could be accessed by the programs. Dynamic SQL [Dat 84] provides an opportunity to use the schema and to phrase queries dynamically. As we incorporate complex data structures and aim to model applications such as EIS, there will be an increasing emphasis on maintaining the relationships among types in the schema and using such information for querying and updating the database dynamically. There are two main issues that are being addressed in the research on schema management:

- Representation of relationships among Types
- Methodology for Type Definition

The problem of representation of relationships among types has been extensively studied in connection with semantic data modeling [Hul 87]. These semantic relationships can be distinguished from other constraints by virtue of having the following properties:

- Simple to detect and understand.
- Occur in many applications.
- Easily maintainable.
- Their logical properties often imply derived relationships.
- Useful for query optimization and cooperative response.

Examples of popular relationships are ***set-of***, ***part-of*** (component of a tuple), ***is-a*** and ***cardinality constraints*** on the association between types. An example of a semantic model is the ***structural model*** [ElMas 79]. This model allows reference and is-a relationships to be represented. Most of the object-oriented data models use a subset of the semantic relationships that were previously studied. The ***is-a***[4] relationship is by far the most widely used.

The structural model [ElMas 80] has recently been utilized to *generate* object-descriptions from relational databases [Bar 88]. Such an approach provides an interpretation of the complex objects in terms of the semantics of the relational databases. This aspect has been explained in the rest of the papers of this volume.

The issue of providing a methodology for type definition has been a recent area of research in object oriented databases. The aim is to have better manageability of the schema information so that schema evolution is easy to do and understand. The technique to do so has been borrowed from object-oriented languages and is called *inheritance*. Inheritance allows functionality of an abstract data type (ADT) to be augmented by functionality of another ADT [Weg 87].

Indeed, there is little or no agreement among the different notions of inheritance beyond the very general definition given above. In order to define an inheritance scheme, we must define what functionality can be borrowed, from whom can the functionality be borrowed and specify a scheme

[4]A Type *a is-a b* only if the set of instances of *a* is contained in the set of instances of *b*. *b* is called the ***supertype*** of *a*.

to resolve or prevent conflicts that may arise because of multiple inheritance. We will briefly touch on each of these issues.

The *functionality* that may be borrowed depends on the data model. Some of the possible choices are:

- Include or refine a data structure from another type (type hierarchy).
- Include or refine the *properties* of (i.e., constraints on) the procedural abstractions associated with another type.
- Include or refine the *implementation* of a procedure from other procedures (This is the most troublesome one).
- Obtain *default values* (in the presence of incomplete information) from another type

Next, we discuss restrictions on the sources from which an abstract data type might borrow its functionality. Most object oriented databases constrain the *dependency-graph* [5] to a hierarchy or an acyclic graph. Typically, types inherit only from their supertypes. The semantics of the is-a relationship imply that the subtype *must* inherit its functionality from its supertype. In such a case, the refinement involved in inheritance must be constrained to maintain the semantics of the is-a relationship [Car 85]. For example, an attribute in a subtype could only restrict the type specified for that attribute in its supertype.

The concept of including and refining the functionality also raises the issue of *conflicts* of inheritance. The problems that often arise in inheritance have to do with name-conflicts (semantic overloading). The techniques for resolution of such conflicts have not been very satisfactory. No sufficiently powerful and semantically sound model of inheritance is available yet. ORION allows resolution of conflicts based on a set of invariant conditions and a set of resolution rules [Ban 87]. The dependency-graph in ORION is restricted to a lattice. Most of the object-oriented programming languages use a *precedence order* among supertypes for conflict resolution. They also provide techniques for method (i.e., procedure) specialization and combination [Bob 85].

The inheritance of logical properties in the presence of is-a relationships has also been exploited computationally in some systems; e.g., KRYPTON [Bra 83], LOGIN [Has 84]. The idea is to recognize that the is-a relationship results in a simple logical subtheory and this characteristic may be utilized for evaluating logic queries. KRYPTON was developed for answering first-order queries and LOGIN is a system for evaluating logic programs.

Schema management also influences the design of physical structures. The problem of minimizing the database reorganization or updates in the event of a schema evolution is a hard one. Another issue is the design of access structures that are useful in the presence of semantic relationships. ORION uses a specialized indexing scheme for access to is-a hierarchies [Kim 87].

Directions in Schema Management: The technique of inheritance and its associated problems have been studied widely in the context of programming languages and artificial intelligence [Bob 85] [Tou 86]. The application of inheritance techniques to database systems must take into account the fact that the object-oriented databases will have large number of types and their management is likely to be distributed. Therefore, we must identify simple principles for inheritance

[5]An arc *ab* exists in this graph if type *a* borrows functionality from type *b*

such that they are easily comprehensible and may be maintained over large systems. We need to clearly distinguish the technique of inheritance and semantic relationships. Some preliminary work in this direction has been reported in [Cha 88]. Much work needs to be done to support the inheritance of procedure implementations.

The issue of minimizing database reorganization in the event of a schema evolution remains an open research issue. This aspect is closely tied to the access structures and such considerations are important in databases for EIS applications.

5 Performance of Access and Storage

The strategy for efficient retrieval and storage depends on the conceptualization and the expected access pattern of the objects. As discussed in the previous sections, an object type can be defined either by structural or procedural abstraction.

If an object type is defined by procedural abstraction using a general purpose computational model (e.g., C or Smalltalk), the retrieval of objects of that type can be improved by techniques such as caching, precomputation and prefetching [Row 86]. The prefetching relation among object types provides a hint to the system to bring in data of a related type when one type is accessed. This information is specified in the schema. Precomputation is important if an object definition involves several joins. In the absence of the precomputed result, on-the-fly materialization of the object will be too slow. However, there are performance issues even when we decide to precompute a set of objects. We must have means for identifying conditions under which the precomputed set should be invalidated. POSTGRES [Sto 86] uses *trigger locks* to check such conditions. These techniques are used in POSTGRES and are also applicable even when no procedural abstractions are used.

We will now restrict ourselves to structural abstraction only. The key question is whether the complex object should be clustered with its subobjects or whether the it should be stored in *normalized* form. The advantage of the former method is that accessing the entire complex object is more efficient. However, when the objects grow larger, the clustering algorithms are likely to perform poorly [Wie 86]. The normalized form is based on the idea that the set of attributes may be decomposed into flat relations by introducing additional surrogates. Such a representation is preferable when access requests to the object are unpredictable and limited to only a few attributes of the complex object at one time. These two schemes were compared and the normalized scheme appears more attractive overall [Val 86]. However, it has been argued that it may be effective to maintain two access structures. One of these may be value based for operations such as join and nest and the other may be structure-based for performing projection and unnest [Des 88]. The implementation of nested normal form in ANDA [Des 88] consists of a global multi-level index for value-based access to tuples and subtuples. In addition, a hash table is provided for structural access A problem in both these schemes is that the indexes are based on tuple-ids. This makes it hard to perform range queries over attributes.

More complications arise if the access to objects is not constrained to be through classes (or types). If associative access is allowed on any heterogeneous set of objects (as in OPAL [Mail 86]), validating and maintaining indexes on these storage structures become difficult. Maier [Mail 86] discusses in detail the language and systems issues that arise in having an associative access in the context of the Smalltalk model of computation.

The access pattern for complex objects is likely to involve an associative access followed by intensive operations on a few objects. Since main memory is becoming cheaper, the object-oriented databases can take advantage of this technological development. Most of the object systems materialize and cache objects in main memory for efficiency [Mai 86] [Row 86] [Des 88]. We believe that the past work in main memory database algorithms [Amm 85] will be useful.

Another issue is the level of abstraction provided to the higher layers of the system software by the storage manager. The traditional database systems allow only predefined persistent data types. Persistent programming languages provide mechanisms to make arbitrary data objects persistent [Akt 83]. However, the applications written in both these environments must manage the two level store. Thatte [Tha 86] suggested use of a *recoverable virtual store* to provide a uniform storage abstraction. However, the issue of clustering and efficient access structures over such a model has not been discussed in his proposal. The Camelot file-management system provides support for transactions [Spe 88].

We are likely to see a more distributed model of computation and therefore the emphasis on main memory binding is likely to become more important. A language to describe binding and access properties of objects will be a useful tool. While storage and retrieval of complex data objects is important, we should appreciate that we also need support for domains such as space and time. Many of the EIS applications deal with models of space and time. Therefore, spatial access methods [Ore 86] are important for object-oriented databases. Finally, the problem of database design for an object data model remains to be explored.

6 Conclusion

In this paper, we discussed the key driving forces behind research in object-oriented databases. The research in abstraction mechanisms led to the development of object models based on structural and procedural abstraction. The latter paradigm provides an integrated programming and data language for application development (*tight coupling*). The models based on structural abstractions must provide a host interface for application development (*loose coupling*). However, the impedance mismatch between the data and the programming language is expected to be less than the current programming environments with a relational data language.

We observed that the structural abstraction mechanisms that have been developed are similar. It is quite hard to evaluate them in the absence of any yardsticks based on the applications. The procedural abstractions seem ideal for an application designer. However, this kind of abstraction suffers from the problem of optimizing access. We therefore feel that an associative language based on structural abstraction will be valuable also as a basis for procedural abstraction.

The research in schema management is concerned with the study of the semantic relationships and the use of inheritance as a technique for defining types incrementally. We pointed out that the database perspective of inheritance demands that our inheritance schemes be such that the type management of large systems is not unduly complex. We expect more research along this direction.

The problem of efficient access and storage varies depending on the conceptualization of objects. The procedural abstractions present difficulty in indexing. We studied schemes for indexing and storing objects defined using structural abstraction. Storing objects in normalized form is attractive for schema evolution and when accesses to subobjects are required.

A comparative evaluation of various ideas in object oriented databases has been made difficult by the lack of strong ties to applications. Recently, there has been an initiative by US Air Force VHSIC data management task force for development of a model and prototype of engineering information system [Wie 88] and a part of such an application model may be used for evaluating the concepts in object-oriented databases.

We should point out that there are many other dimensions of research that we have left out in our discussion. Modeling CAD transactions requires support for long-lived transaction. The design of a query language is another important issue. While a declarative language based on complex objects may be useful as the internal description, it is important that any user query language be icon-based. A linear language will be too clumsy for dealing with complex structures.

The area of expert database systems [Ker 86] and extensible systems [Sto 86], [Cdv 87], [Haa 88] are related to object oriented databases. The expert database systems are concerned with how (and what) domain knowledge may be utilized for database management and how data management may be integrated with expert system shells. The research in object oriented and expert databases share the common concern for language integration. The extensible systems must also address the issue of utilizing domain knowledge.

This paper focused on the key research directions that have been prominent in research on object-oriented databases. We defined a simple framework to discuss the past developments in this area. A previous survey by Bancilhon [Ban 88] points to the promising research issues. We believe that our study integrates some of the issues raised in that paper in a framework based on the need for abstraction, schema management and performance of access and storage. Zdonik [Zdo 87] provides an overview of research issues in a tight coupling of programming and database systems.

The next paper in this volume describes an object model that illustrates the key concept of *defining* objects based on the semantics of relational databases, augmented with the structural model. The structural abstraction is a nested relation and the operations are limited to selections over such a structure. The model uses procedural abstraction for display and update of the objects. A Hypertext based graphic language acts as the query interface. The model is used for immunologic research applications.

7 Acknowledgments

I am grateful to Prof. Gio Wiederhold for encouraging me to do this survey. My discussions with Waqar Hasan, Prof. Jeff Ullman, Peter Rathmann, Inder Pal Singh and Prof. Gio Wiederhold were useful. I am indebted to Pandurang Nayak, Waqar Hasan and Inder Pal Singh for their comments on the draft. This work was performed as part of the KBMS project, supported by DARPA Contract No. N00039-84-C-02111.

References

[Amm 85] Ammann A. et.al., "Design of a Memory Resident DBMS"; IEEE COMPCON Proceedings, Feb 1985.

[Akt 83] Atkinson M.P. et.al.; "An approach to Persistent Programming"; The Computer Journal, vol 26, no. 4, pp 360-365, Dec 1983.

[Ban 86] Bancilhon F., Khosafian S.; "A Calculus for Complex Objects"; PODS, 1986.

[Banl 86] Bancilhon F.; "A Logic Programming/Object-Oriented Cocktail"; SIGMOD Record, Vol. 15, No. 3, Sep 1986.

[Ban 87] Bannerjee J. et.al.; "Semantics and Implementation of Schema Evolution in Object-oriented Databases"; ACM SIGMOD 1987.

[Ban 88] Bancilhon F., "Object-Oriented database Systems"; PODS, 1988.

[Bar 88] Barsalou T., Wiederhold G., "An Object-Based Architecture for Expert Database Systems"; 1988 (This volume).

[Bob 85] Bobrow D., Stefik M.; "Object-Oriented Programming: Themes and Variations"; AI Magazine, 1985.

[Bra 83] Brachman R.J.; "Krypton: A Functional Approach to Knowledge Representation"; IEEE Computer, Oct 1983.

[Car 85] Cardelli L.; "A Semantics of Multiple Inheritance"; ATT, 1985.

[Cdv 87] Carey M.J. et.al.; "A Data Model and Query Language for EXODUS"; CSTR-734, Dec 1987.

[Cha 88] Chaudhuri S., Hasan W.; "Relationship among Types and Prototypes"; Working paper, 1988.

[Cod 79] Codd E.F.; "Extending the Database Relational Model to Capture More Meaning"; ACM TODS, Vol 4, No 4, Dec 1979.

[Dat 84] Date C.J.; "A Guide to DB2"; Addison Wesley, 1984.

[Des 88] Deshpande A., Van Gucht D.; "An Implementation for Nested Relational Databases"; VLDB 1988.

[ElMas 79] El Masri R., Wiederhold G., "Data Model Integration using Structural Model"; SIGMOD 1979.

[ElMas 80] El Masri R., Wiederhold G., "Properties of relationships and their representation"; AFIPS Conference Proceedings, 1980.

[Has 84] Hasan Ait Kaci; "A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures"; Ph.D thesis, University of Pennsylvania, 1984.

[Haa 88] Haas L.M. et.al.; "An Extensible Processor for an Extended Relational Query Language"; IBM RJ 6182, Apr. 1988.

[Hul 87] Hull R., King R.; "Semantic Database Modeling: Survey, Applications, and Research Issues"; CRI-87-20, Univ. of Colorado, 1987.

[Jhi 88] Jhingran A.; "A Performance Study of Query Optimization Algorithms on a Database System Supporting Procedural Objects"; VLDB 1988, Los Angeles.

[Ker 86] Kerschberg L. (Ed); "Expert Database Systems", 1986.

[Kim 87] Kim W. et.al.; "Indexing Techniques For Object-Oriented Databases"; TR-87-14, MCC 1987.

[Kup 85] Kuper G.M.; "The Logical Data Model: A New Approach To Database Logic"; Ph.D thesis, Stanford University, 1985.

[Lyn 86] Lyngbaek P. et.al.; "Design and Implementation of the Iris Object manager"; STL-86-17, HP Labs, 1986.

[Lynl 88] Lyngbaek P. et.al.; "IRIS/OSQL 3.0 reference Manual"; HPL-DTD-88-3.

[Mai 85] Maier D.; "DMDM"; Draft, 1985.

[Mai 86] Maier D. et.al.; "Development of an Object-Oriented DBMS"; OOPSLA 1986.

[Mail 86] Maier D., Stein J.; "Indexing in an Object-oriented DBMS"; Proceedings of International Workshop on Object-oriented Database Systems; 1986.

[Mai 87] Maier D., "Why Database Languages Are a Bad Idea"; Position Paper, 1987.

[Ore 86] Orenstein J. A.; "Spatial Query Processing in an Object-oriented Database System"; SIGMOD 1986.

[Rot 84] Roth M.A. et.al.; "Theory of Non-First-Normal-Form Relational Databases"; TR-84-36, Dec 1984.

[Row 86] Rowe L.A.; "A Shared Object Hierarchy"; Proceedings of the International Workshop on Object-oriented Database Systems, 1986.

[Spe 88] Spector A.Z. et.al.; "CAMELOT: A Flexible, Distributed Transaction Processing System"; IEEE COMPCON 1988, Feb-Mar 1988.

[Sto 86] Stonebraker M., Rowe L.A.; "The design of POSTGRES"; ACM SIGMOD, 1986.

[Tha 86] Thatte S.M.; "Persistent Memory: A Storage Architecture for Object-Oriented Database Systems"; Proceedings of the INternational Workshop on Object-oriented Database Systems, 1986.

[Tou 86] Touretzky D.S, "The Mathematics of Inheritance Systems"; Addison Wesley, 1987.

[Val 86] Valduriez P. et.al.; "Implementation Techniques of Complex Objects"; VLDB 1986.

[Weg 87] Wegner P.; "Dimensions of Object-Based Language Design"; OOPSLA 1987.

[Wie 79] Wiederhold G., El Masri R.; "A Structural Model for Database Systems; Proceedings of International Conference on Entity-Relationship approach, Dec. 1979, pp. 247-267.

[Wie 86] Wiederhold G.; "Views, Objects and Databases"; IEEE Computer, Dec 1986.

[Wie 88] Wiederhold G., Winslett M.; "Modeling an Engineering Information System"; Working paper, 1988.

[Zic 87] Zicari R. et.al.; "An Algebra for Structured Office Documents"; IBM RJ-5559, Mar 1987.

[Zan 86] Zanialo C. et.al.; "Object Oriented Database Systems and Knowledge Systems"; in Expert Database Systems: Larry Kerschberg (Ed), 1986.

[Zdo 87] Zdonik S., Bloom T.; "Issues in the Design of Object-Oriented Programming Languages"; OOPSLA 1987.

[Zdol 85] Zdonik S., Wegner P.; "Towards Object oriented database environments"; Brown University TR, 1985.

An Object-Based Architecture for Biomedical Expert Database Systems

Thierry Barsalou
Stanford University

Abstract

Objects play a major role in both database and artificial intelligence research. In this paper, we present a novel architecture for expert database systems that introduces an object-based interface between relational databases and expert systems. We exploit a semantic model of the database structure to map relations automatically into object templates, where each template can be a complex combination of join and projection operations. Moreover, we arrange the templates into object networks that represent different views of the same database. Separate processes instantiate those templates using data from the base relations, cache the resulting instances in main memory, navigate through a given network's objects, and update the database according to changes made at the object layer. In the context of an immunologic-research application, we demonstrate the capabilities of a prototype implementation of the architecture. The resulting model provides enhanced tools for database structuring and manipulation. In addition, this architecture supports efficient bidirectional communication between database and expert systems through the shared object layer.

Introduction

Databases and expert systems share a common goal—generating useful information for action—but accomplish their tasks separately, using different principles. Databases manipulate large bodies of data with relatively little (implicit) knowledge, whereas expert systems generally manipulate small bodies of data using large, knowledge bases. It is clear, however, that future information systems will require both the problem-solving capabilities of expert systems (ESs) and the data-handling capabilities of database management systems (DBMSs) [15,20]. Indeed, combining database and expert system technologies into *expert database systems* is an emerging research area. The concept of an expert database system (EDS), however, connotes various definitions and architectures. We shall use here the notion of "a system for developing applications requiring knowledge-directed processing of shared information" [24]. From our perspective of developing advanced biomedical information systems, this definition conveys two precise scenarios: (1) enhancing DBMSs with structuring and manipulation tools that take more semantics into account; (2) allowing ESs to access and to handle efficiently information stored in database(s).

The object-oriented paradigm has gained much attention in recent years for the development of languages, programming environments, and applications. In the database field, object-oriented DBMSs have emerged. For example, the GemStone system [18] is the result of turning the Smalltalk-80 language [12] into a full DBMS. However, augmenting a single-user, memory-based language with features such as concurrency control and data security typically requires a significant programming effort. The concept of an entity is also widely used by database design tools. In the field of artificial intelligence, frames are a well-known knowledge representation scheme [19]. Although frames were conceived separately from the object paradigm, the two are in fact consistent with each other. Our research hypothesis is that the object-oriented approach can also serve as a unifying scheme for developing EDSs. This paper presents a novel architecture, called **PENGUIN**, for EDSs. We introduce an object-based interface on top of a relational database, which can improve database transactions and serve as an efficient scheme for coupling database and expert systems. We demonstrate that the object-oriented and relational models can be combined into a single system that integrates the performance and functionalities of both paradigms. One consideration is central to this motivation. In the perspective of information retrieval, most computer-aided design/computer-aided manufacturing (CAD/CAM), engineering and scientific information systems must provide simultaneously:

- Suitable abstractions of real-world objects together with the set of operators necessary to manipulate instances of those objects. These operations on objects are quite simple (e.g., fetching a single object, following one connection) but they probably represent the majority of the transactions. They should then be carried out efficiently and easily.
- Persistent storage of shared information and arbitrary access to data using the expressiveness of declarative query languages. Although not of everyday use, the ability to compose joins of arbitrary complexity has always been an important feature of relational systems and should be preserved in the next generation of DBMSs.

In Section 2, we summarize past relevant work and interdisciplinary research in the fields of database and artificial intelligence. In Section 3, we present **PENGUIN**'s architecture and motivate our design decisions, based on a comparison of the relational model and the object-oriented paradigm. In Section 4, we describe the immunologic application used as a vehicle for development and testing of **PENGUIN**. In Section 5, we demonstrate the functionalities of the interface. In Section 6, we discuss the benefits of the system with regard to our two scenarios for EDSs, before concluding in Section 7.

The ES/DBMS Connection

Database management systems evolved in response to well-perceived, acute needs for efficient storage, maintenance, and retrieval of increasingly large amounts of data. Those needs led to the concept of a data model. A data model consists of (1) a collection of entities, (2) a collection of operators, and (3) a collection of integrity rules [10]. However, traditional data models (relational, network, and hierarchical) are machine-oriented (that is, geared toward effective computational representation) rather than user oriented. The main contribution of AI technology to database research is a new generation of data models, called semantic data models, which incorporate more semantics, and provide higher level data modeling capabilities [4]. The *entity-relationship* (ER) model [7] defines two basic components, entity sets and relationship sets, with both sets having properties. The ER model is primarily a database design tool, where the user specifies a pictorial representation of the various entities and relationships. The *hierarchical semantic* model [25] extends the relational model with the key concepts of aggregation and generalization. The *semantic data* model [14] defines a database as a collection of base and nonbase classes. Classes are composed of entities of various types (e.g., concrete objects, events, abstractions). *Taxis* [22] uses the semantic network formalism to design interactive information systems. The resulting semantic net is then translated into an extended relational format. POSTGRES [28] proposes to extend the relational model with an abstract data-type facility and procedures as a data type. This machinery allows to design an object hierarchy which is explicitly stored in the relational database [23]. Strategies for speeding up operations include object caching and prefetching. All these models achieve the same important goal, albeit by different means: They allow the user to think and interact with the database in terms of meaning rather than in lower-level terms of representation.

In contrast, artificial intelligence, and more specifically, expert system research has focused on developing rich knowledge representation languages and symbolic reasoning mechanisms, achieving impressive results in limited domains. However, ESs clearly suffer from a scaling problem when they move into real-world, large scale domains. In such cases, one of the major challenges is a communication problem: To be useful, ESs (particularly in medicine) need to access large bodies of facts that are already stored in existing databases. A key research issue, then, is the development of interfaces that provide communication between the two systems [30]. There are two strategies for coupling an ES with a DBMS. *Loose coupling* corresponds to a "static" use of the communication channel where the inference phase and the data-retrieval phase have to be completely separate; the ES can only access and process those facts that are currently loaded in a virtual window before asking the DBMS for a new window of data. *Tight coupling* corresponds to a more sophisticated and flexible "dynamic" use of the interface; interactions between the ES and the DBMS can take place at any moment. In this case, the database represents just an extension of the ES's knowledge base. Beside the coupling strategy, another important issue is the format for passing data back and forth between the DBMS and the ES. Many existing schemes [8,13] support only a too restricted, direct one-to-one mapping of one relation into one object. ESs often need to reconstruct entities from base data scattered over several relations (and maybe several databases as well). Other schemes [21] do provide many-to-one mapping capabilities but limit both the complexity of the generated objects and the update operations on those objects.

PENGUIN's Architecture

Brodie *et al.* observe that "there is a far reaching intersection between both knowledge representation languages and data models, and that they should be treated under a unifying approach" [6]. PENGUIN explores the idea that the object-oriented paradigm can be this unifying approach for expert database systems.

General Principles

PENGUIN introduces an object-based interface on top of a relational database. Wiederhold has drawn an analogy between the object and database view concepts [35]. The basic challenge discussed in this paper was to develop an interface that could carry out simultaneously two complementary tasks:

1. A database query retrieves all the information pertaining to the object instances of interest;
2. A binding process links the data to the object template by following some predefined mapping knowledge.

Note that denormalization through join queries is not enough for that matter. Normalization is a necessary step in the design of relational databases [33]. Using functional dependencies among attributes, it leads to the well-defined Boyce-Codd normal form of the relational schema [3]. Unfortunately, normalization requires that information relevant to one object be distributed over several relations. Therefore, large join operations (explicit or through definition of views) are needed to bring back together the various pieces of data, that is to denormalize. However, the data will not be arranged as expected from an object viewpoint. Take for example the two relations EMPLOYEE and SUPERVISION to capture all the supervisees for all employees. Retrieving all the supervisees of John produces much redundant information:

Supervisor	*Department*	*Supervisee*	*Years*
John	Marketing	Mary	8
John	Marketing	William	3
John	...	...	...

Clearly, denormalization through multiple joins is awkward and unsatisfactory. An intelligent binding process requires some additional knowledge to maintain the relationships between the real-world entities and the database normlaized relations. Such composition knowledge can eliminate redundancies and support satisfactory rearrangement of the object's attributes.

Using this analogy between views and objects, we define three components of the object layer:

1. The **object generator** maps relations into object templates where each template can be a complex combination of join (combining two relations through shared attributes) and projection (restricting the set of attributes of a relation) operations on the base relations. In addition, an ***object network*** groups together related templates, thereby identifying different object views of the same database. The whole process is knowledge-driven, using the semantics of the database structure. We define the ***object schema*** as the set of object networks constructed over a given database. Like the data schema for a relational database, the object schema represents the domain-specific information needed to gain access to PENGUIN's objects; this information enables us to combine well-organized, regular tabular structures—the relations—into complex, heterogeneous entities—the objects.
2. The **object instantiator** provides nonprocedural access to the actual object instances. First, a declarative query (e.g., `Select instances of template` x `where attribute` $y < 0.5$) specifies the template of interest. Combining the database-access function (stored in the

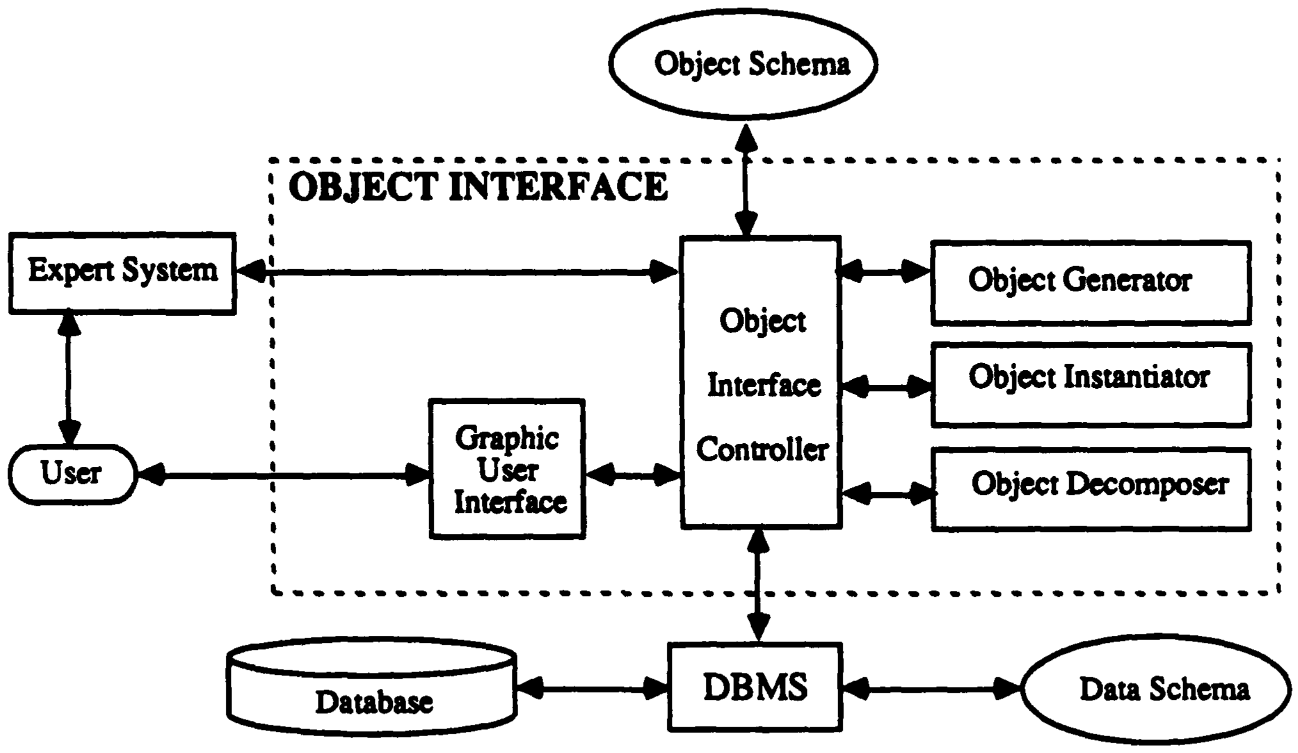


Figure 1: PENGUIN's architecture.

template), and the specific selection criteria (e.g., $y < 0.5$), PENGUIN automatically generates the relational query and transmits it to the DBMS, which in turn transmits back the set of matching relational tuples. In addition to performing the database-access function, the object template specifies the structure and linkage of the data elements within the object. This information is necessary for the tuples to be correctly assembled into the desired instances. Those instances are then made available to the expert system directly, or to the user through a graphic interface.

3. The **object decomposer** implements the inverse function; that is, it maps the object instances back to the base relations. This component is invoked when changes to some object instances (e.g., deletion of an instance, update of some attributes) need to be made persistent at the database level. An object instance is generated by collapsing (potentially) many tuples from several relations. By the same token, one update operation on an object may result in a number of update operations that need to be performed on the base relations.

Figure 1 summarizes PENGUIN's architecture and illustrates the interactions among the relational database, the expert system, the object layer, and the user—interactions orchestrated by a controller unit within the object interface.

We now contrast the relational data model with the object-oriented paradigm and motivate the decisions made when designing PENGUIN.

The Relational Data Model

Since Codd [9] introduced the idea of organizing data in tables, the relational model has found widespread acceptance in all areas of data management. Indeed, this model offers key advantages:

- Simplicity is the main attraction. The relational model provides only one conceptual data structure—the relation—which is easy to understand and is intuitive to most people.
- The algebra for manipulating relations has a sound grounding in set theory. Moreover, there is a formal equivalence between procedural relational algebra and declarative relational calculus, which allows the development of powerful nonprocedural high-level query languages.
- The relational model has the valuable property of *closure under query* [29]; that is, the result of a query is a relation itself which in turn can be queried. This property enables the building of queries of arbitrary complexity.
- All relational DBMSs have the ability to share access to data and to maintain the database integrity. Current research is addressing the problem of distributed computing.
- Finally, these database systems use a variety of file and indexing structures that allow efficient handling of large numbers of data.

The relational model also has limitations:

- The limited set of data types is too constrained for engineering applications.
- Semantic information—that is, knowledge about the world being modeled by the database—is not incorporated in the relational model.
- Relational DBMSs rely on additional tools to provide both the user interface and the level of abstraction people need.

The Object-Oriented Paradigm

Object-oriented systems offer various advantages over systems based on traditional data models [26]:

- The object paradigm supports the distinction between the conceptual entities and instances of those entities. This approach enables *internal binding* [31]; that is, after retrieval, data are kept in memory in efficient data structures, connected to related information.
- Objects combine the properties of procedures and data: *State* is captured through the value of the instances' variables, and *behavior* is represented by the entity's methods that are triggered by message sending.
- The definition of class hierarchies allows inheritance of properties from general classes to more specific entities. Inheritance reduces redundancy and simplifies program maintenance by favoring a top-down approach.
- Because the object data model and the programming language are, in any instance, integrated, the latter can be used for both data retrieval and manipulation and application computations.

Evidently, there are also some drawbacks to this approach:

- Object-oriented systems do not provide such indispensable features of DBMSs as file-management structures and concurrency control.
- Queries generally follow predefined links and tend to operate on individual objects. Thus, the processing of queries involving large and complex sets of data is not well supported.
- Provision of fully declarative query languages seems difficult to achieve in object-oriented databases.

Design Decisions

It is clear from these descriptions that each model could benefit from the other one's strengths and that the two could act in synergy. Combining the two paradigms thus seems a natural approach to explore in the perspective of developing expert database systems.

Despite the current trend toward object-oriented DBMSs [18], **PENGUIN** keeps a relational database system as its underlying data repository. Indeed we believe that the relational model should be extended rather than replaced. The relational model has become a de facto standard and thus some degree of upward compatibility should be kept between the relational format and any next-generation model. A related matter is the issue of homogeneity [27]: Applications developed in the near future are likely to use and integrate data from different sources. Such a process will be greatly hampered if the various databases operate on different models.

We apply a **structural model** of the database to augment the relational model [32]. The structural model uses relations and *connections* to describe relationships among pairs of relations. Formally, a connection is specified by the connection type, a pair of source and destination relations, and the attribute(s) shared by the two relations (which logically implement the connection). There are three types of connection:

1. The *ownership* connection (represented by $\longrightarrow\!\!*$) is a one-to-many relationship. Many tuples in the owned relation depend on one tuple in the owning relation. Thus, this connection embodies the concept of dependency that is extremely useful during update operations. For instance, deletion of a circuit also implies deletion of all its gates whereas a new gate can only be added to an existing circuit. One can then implement aggregation hierarchies with *"part-of"* type links using ownership connections.
2. The *reference* connection (represented by $\longrightarrow\!\!\!-$) is a many-to-one relationship, where multiple tuples in a relation refer to the same descriptive tuple of another relation. The connection of two related concepts, one being more general than the other, corresponds to an abstraction process. This bears immediate significance for summarization tasks where abstraction is a key idea; a referencing attribute can be used to extract an informative and discriminant feature from a set of related entities.
3. The *subset* connection (represented by $\longrightarrow\!\!\supset$) is a partial one-to-one mapping, such that a tuple in the general relation is linked to at most one tuple in a more specific relation. The general relation contains all common data. Subrelations are designed to hold data particular to the subgroups. The subset formalism can then represent categorical hierarchies with generalization of subclasses to larger classes.

These connections capture the knowledge about constraints and dependencies among relations in the database [36]. In contrast with other semantic data models, this relatively simple and compact model can be implemented easily on top of any standard relational DBMS. Most important, the structural model not only provides the semantics sufficient for the design of the object-based interface, but also can enhance traditional relational transactions in several ways. For instance, structural knowledge adds the concepts of aggregation, categorization, and generalization to the relational model [25]. It also allows us to define *dynamic* integrity constraints that state how the database consistency is to be maintained when an update process requires associated changes [2].

PENGUIN's object template generator uses the structural model together with the traditional data schema to define the *object schema*. In turn, the object schema drives the two processes of object instantiation and object decomposition.

A related decision regarding this architecture was not to store the objects explicitly in the relational database. Several factors motivated this choice:

- Different users require different *views* of the information included in an object. Update anomalies and problems of redundancy would arise if the objects corresponding to the different views were to be stored as such in the database.
- In early stages of any project development, changes to the set of classes and to the inheritance network are made frequently. If the objects were explicitly stored in the database, the database schema would have to be changed accordingly.
- Because we expect the description of the objects—the object templates—to be relatively small when compared to the large number of data in the database, those templates can be kept in main memory.

For all these reasons, our proposal is not to store the objects directly in persistent form but rather to generate their descriptions, which are later instantiated as needed. We think this is a more flexible approach.

A Flow Cytometry Application

Flow cytometry, also known as Fluorescence Activated Cell Sorting (FACS), is emerging as a major source of information for biomedical research and clinical practice [5]. The FACS instrument allows biomedical researchers to analyze and isolate viable cells that have different cell-surface phenotypes marked by fluorescence-tagged monoclonal antibodies. Immunologic research makes heavy use of powerful flow cytometry.

However, designing good FACS experiment protocols (that is, finding the best combination of fluorochrome-antibody pairs aimed at a particular phenotype) is complex and requires skilled investigators, who are in scarce supply. Work is under way to develop a system, the FACS Workstation, that will assist researchers in designing protocols for FACS immunologic experiments [1]. This system requires combining database and expert-system technologies to fulfill both the data- and knowledge-intensive tasks of planning a FACS protocol. The FACS Workstation system can thus serve as the testbed for the implementation of the PENGUIN architecture. Figure 2 shows the structural model of the database that contains all the base information about the monoclonal-antibody reagents, their target antigens, and the various cell populations.

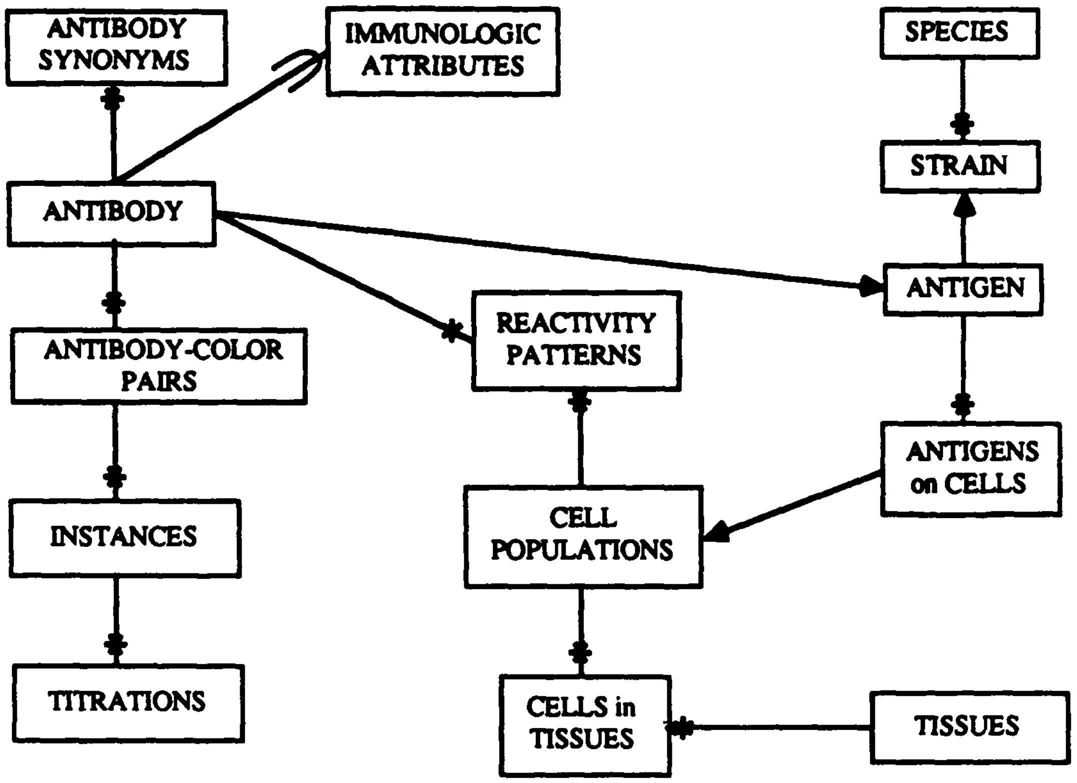


Figure 2: The structural model for the FACS reagent database.

Object-Interface Operations

We now use the FACS reagent database described above to demonstrate the functionalities of PENGUIN's prototype implementation.

Object Generation

The object-generation module defines new object templates using semantic information provided by the structural model. Each template is organized around (or "anchored on") one *pivot relation* in the underlying database, so that every object instance can be uniquely identified by its values for the pivot-relation key attribute(s). Most existing ES-DB coupling schemes support only a direct one-to-one mapping of one relation into one object (or frame); however, database normalization requires that information concerning a real-world complex entity be distributed over several relations [33]. We address this problem by allowing both projection and join operations to be used for defining new templates. As a result, we have a many-to-one mapping of relations into objects.

Obviously, not every relation in the database schema has to be included in a given new template. On the other hand, the relations that may be relevant to one object are always linked to the pivot relation through simple combinations of the three types of structural connections; (this is the idea of locality). Those connections indeed correspond to the join operations that need to be performed to bring back an object together. We have thus defined rules to determine, once the pivot relation has been specified, which other relations are valid candidates for inclusion in the template. For instance, any relation connected to the pivot relation by a chain of ownership connections (of arbitrary length) is a candidate. Using those rules, an algorithm we designed can extract a *candidate graph* from the structural model, which is itself a directed acyclic graph, and convert this candidate graph into a *candidate tree*, where the root is the pivot relation and all other nodes are valid candidate relations.

The generation process is driven by information from the structural model, which allows automatic generation of much of the internal structure of an object template. Thus, during a session, little user interaction is required. The user specifies only the pivot relation, as well as any number of secondary relations and attributes of these relations, using a graphical representation of the database structural model and of the candidate tree, and intuitive "point-and-click" interactions. Based on this input, PENGUIN will automatically (1) derive the database access function (in the form of an SQL-like join expression) and the linkage of the various data elements within the object (that is, the hierarchical structure of the selected relations, which is derived from the candidate tree), (2) include compulsory attributes (e.g., the attributes that implement the joins on the selected relations), and (3) insert the newly created template into an object network by connecting it to other object templates already defined in the network (thereby updating the object schema). It is worth stressing that those template connections are abstracted from the underlying database structural connections; therefore, concepts such as aggregation and generalization (added by the structural model) are explicitly carried over to the object layer and can be used to provide inheritance of attributes among templates (for instance, in a categorical hierarchy implemented with subset connections) or to enhance the information-retrieval phase (as shown below).

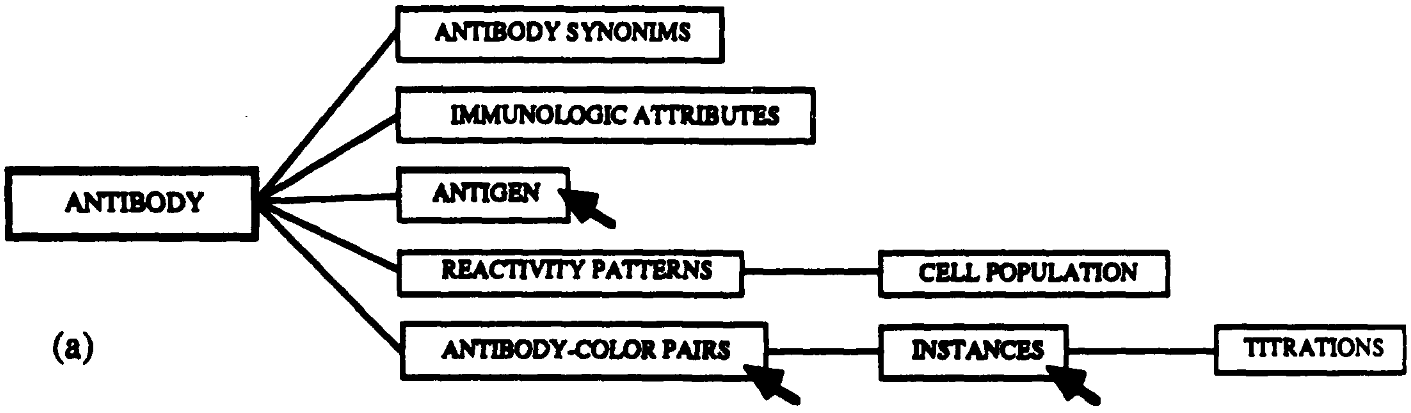
Figure 3 illustrates the object-generation process. Using the FACS reagent database structural model of Figure 2, the user creates a template anchored on the ANTIBODY relation. From this pivot relation and from the structural model, PENGUIN derives the candidate tree that is specific to the new template. Using this tree, the user chooses to include relations ANTIGEN, ANTIBODY-COLOR PAIRS, and INSTANCES (Figure 3a). PENGUIN now generates the data access function and the default structure of the various data elements for the template (Figure 3b, where relations are in boldface and attributes are in italics) before inserting it in an object network by defining a reference connection with the ANTIGEN OBJECT template (Figure 3c). The resulting template is a complex unit, incorporating many attributes from various relations.

Browsing and Navigation

The object-instantiation protocol performs all the operations of information retrieval and manipulation that are necessary to instantiate and display any object template. Because of the motivations outlined earlier, we emphasize simple operations like fetching an object instance and following a connection to related instances. The structural model and the object networks are central to the realization of these tasks because the the former contains the semantic needed for proper binding of information and the latter represents specific users' views of the database.

Template Instantiation

Our goal was to provide *nonprocedural* access to data in a similar fashion to the declarative approach offered by relational query languages. Since an object template contains both the access function to retrieve all the base tuples and the structural information to merge those tuples into

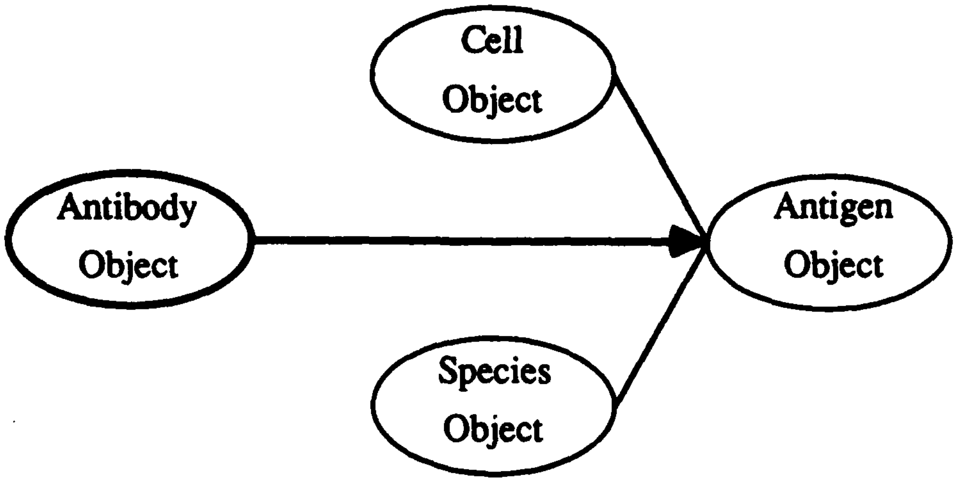


(a)

```
( ANTIBODY OBJECT
    (Hierarchical-Structure
        ( ANTIBODY  clone-name  fine-specificity  brightness  )
        ( ANTIGEN  antigen-name  antigen-species  strain)
        ( ANTIBODY-COLOR PAIRS   color-name  f-to-p-ratio
            (INSTANCES  batch-number  date-creation  )))

    (Data-Access-Function
        (antibody.antigen-name = antigen.antigen-name and
         antibody.antigen-species = antigen.antigen-species and
         antibody.clone-name = antibody-color-pairs.clone-name and
         antibody.clone-name = instance.clone-name)))
```

(b)



(c)

Figure 3: The object-generation process. (a) The candidate tree derived from the structural model; (b) partial internal structure of the new template; (c) insertion of the new template in an object network.

complex, heterogeneous entities, one needs only to specify a selection clause to retrieve the desired set of instances. Specification of the selection clause can be done directly by an expert system. Alternatively, **PENGUIN**'s graphic interface can present to the user the template's tree of relations and their attributes. Simple, mouse-oriented interactions then guide the selection process. Based on this information and on the database-access function, **PENGUIN** dynamically builds a relational query that is executed by the DBMS. Note that the set of relational tuples returned by the DBMS may contain more than one tuple for any given instance. Therefore, the following algorithm ensures that each tuple in the answer set can be traced back to the proper instance. The answer set is assembled into a new instance as follows:

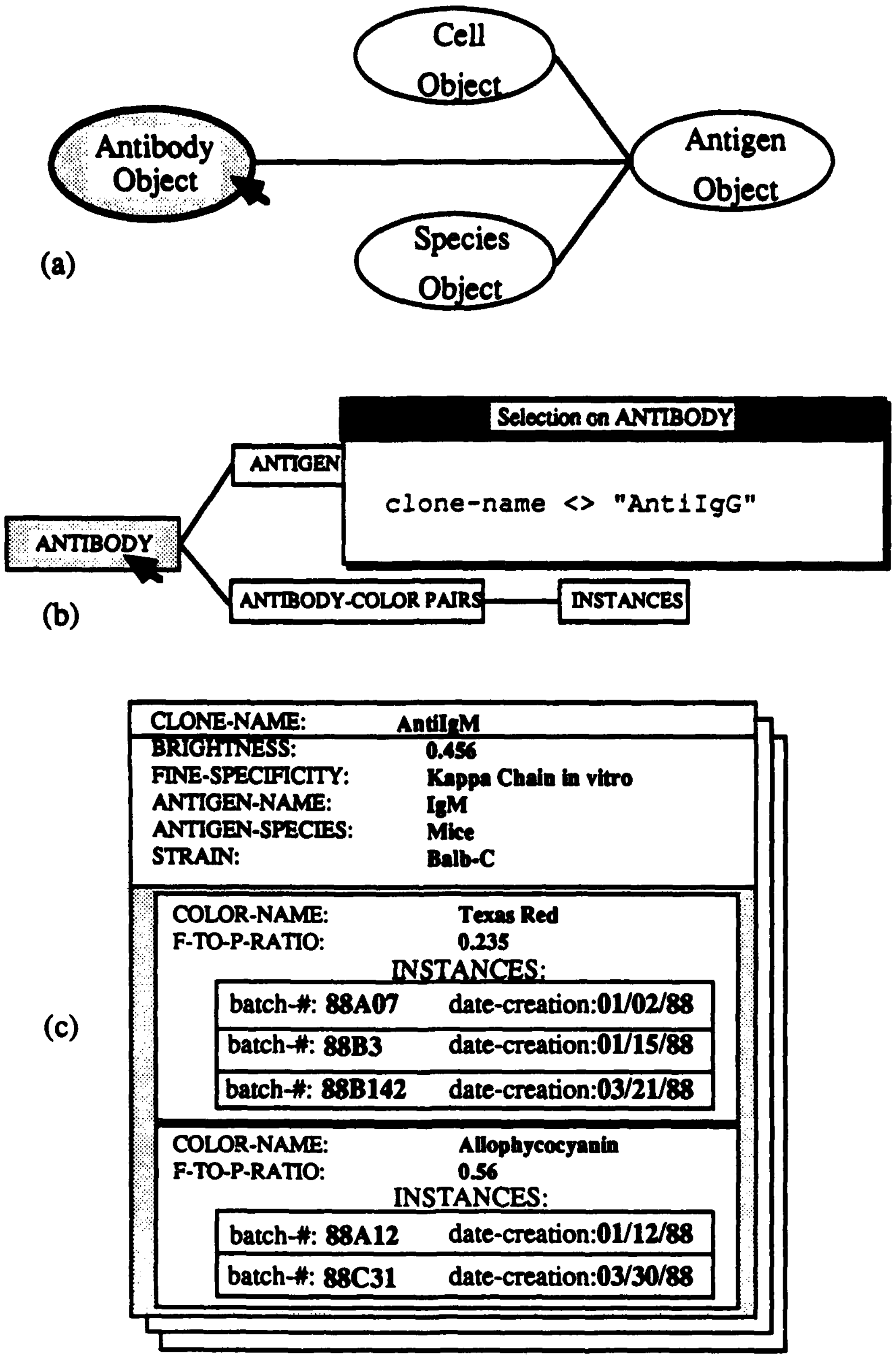
- The object attributes are grouped by their original relations; key and non-key parts are identified within each group.
- The type of connection between the pivot relation and the groups' original relation determines an expected cardinality for each of these attribute groups. Subset, reference, inverse ownership, and inverse subset connections entail a cardinality of 1. Conversely, ownership and inverse reference links correspond to an n cardinality with $n \geq 1$.
- Attributes of groups with cardinality 1 are directly associated with their tuple counterpart.
- For each tuple in the answer set and for each cardinality-n group, the values of the group's attributes constitute one distinct element, uniquely identified by its key part, which is then added as such to the object.

This algorithm guarantees correct arrangement of the data elements for each newly created instance within the object structure; it also supports the generation of *multi-valued* attributes that are, by definition, absent from the relational model but often critical to symbolic reasoning. Retrieving a set of instances for a given template can be accomodated as well. An additional step is added to the algorithm, so that each tuple in the answer set can be traced back to the proper instance based on the key attributes of the primary relation. All the instances that satisfy the selection clause are then attached to the template.

The expert system can now directly manipulate those instances. Alternatively, the graphic interface can display them in a separate window, using the template's tree of relations as the default presentation format. However, additional formats for different presentations can be stored with any object template. The user can then switch her perspective on the information contained in the instances (e.g., collapsing and expanding nested subunits).

As an example, consider the ANTIBODY OBJECT template created in Figure 3. To generate instances of this object, the user first selects the correct node in the object network (Figure 4a); she then uses the template's tree of relations to specify the selection clause—here only one attribute is involved: `clone-name <> "AntiIgG"` (Figure 4b). **PENGUIN** now assembles the set of resulting tuples into several ANTIBODY OBJECT instances, which are displayed using a hierarchical format (Figure 4c).

By contrast, merely obtaining the set of tuples in a standard DBMS (not to mention assembling them into meaningful entities) would require the user to type the following SQL query:



```
clone-name <> "AntiIgG"
```

CLONE-NAME:	AntiIgM
BRIGHTNESS:	0.456
FINE-SPECIFICITY:	Kappa Chain in vitro
ANTIGEN-NAME:	IgM
ANTIGEN-SPECIES:	Mice
STRAIN:	Balb-C

COLOR-NAME:	Texas Red
F-TO-P-RATIO:	0.235

INSTANCES:

batch-#: 88A07	date-creation:01/02/88
batch-#: 88B3	date-creation:01/15/88
batch-#: 88B142	date-creation:03/21/88

COLOR-NAME:	Allophycocyanin
F-TO-P-RATIO:	0.56

INSTANCES:

batch-#: 88A12	date-creation:01/12/88
batch-#: 88C31	date-creation:03/30/88

Figure 4: An instance of the ANTIBODY OBJECT template. (a) Choosing a template in an object network; (b) defining the selection clause; (c) displaying the set of new instances.

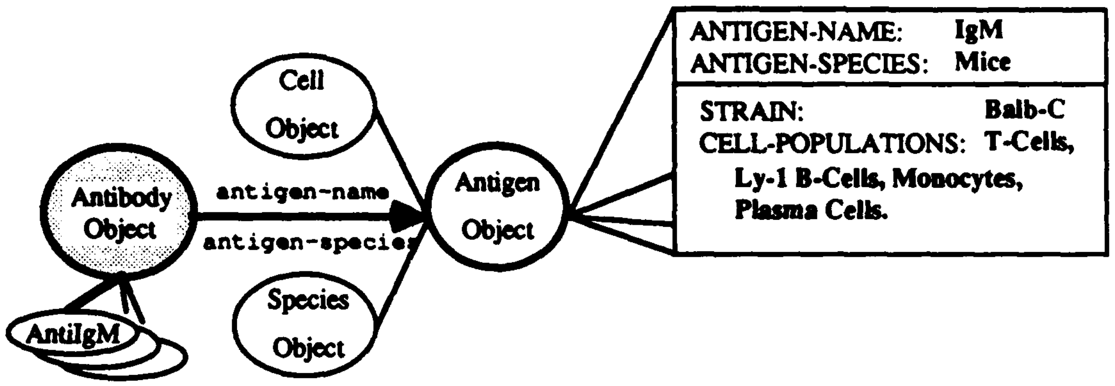


Figure 5: An ANTIGEN OBJECT instance created by navigating from the ANTIBODY OBJECT template.

```
SELECT  a.clone_name, a.brightness, a.fine_specificity,
        b.antigen_name, b.antigen_species, b.strain
        c.color_name, c.f_p_ratio, d.batch_number, d.date_creation
  FROM  antibody a, antigen b, antibody_color_pairs c, instances d
 WHERE  a.clone_name <> "AntiIgG"
        and a.clone_name = c.clone_name
        and a.clone_name = d.clone_name
        and a.antigen_name = b.antigen_name
        and a.antigen_species = b.antigen_species
```

Finally, a further abstraction process can generate a *summary* of the new instances using the semantics carried by the reference connection [34]. For example, a set of ANTIBODY OBJECT instances will be usefully summarized through the **antigen-name** and **antigen-species** attributes, which implement the reference connection between the ANTIBODY and ANTIGEN relations.

Caching of Existing Instances

Most medical decision-support systems typically make many references to the same data during a consultation. Clearly, satisfactory performance cannot be achieved if each subsequen access to a fact requires a database retrieval from secondary storage. Our approach is to reduce disk access by caching existing instances in main memory and by keeping a log file of previous queries formulated in the current session. Since objects are a compact structure in which to store data, we can expect a large number of instances to fit into memory. Binding is implemented through the object network structure. Each invocation of the object instantiator produces instances of a given template in a specific object network, which are uniquely identified by surrogates [17]. Those surrogates are appended to the template's INSTANCES slot, thus binding the new instances to the correct node in the correct object network.

Moreover, for templates that contain only a few instances, or ones that we know in advance will be heavily used, we can implement prebinding where retrieval of the data and generation of the object instances are done at loading time.

Navigation through an Object Network

The links connecting objects within a network provide the means of offering a flexible navigation scheme. The method of ***instantiation in context*** employs inheritance of values in the network to minimize user interaction when fetching instances that are related to one another. For this purpose, an instance of a particular object O_1 is set as the current instance I_C; another object, O_2, among all the templates directly connected to O_1 can now be chosen. The semantics of the link $O_1 \sim O_2$ (that may include several structural connections) then defines the set of I_C's attribute-value pairs that are inherited. This set now constitutes the selection clause for the related object O_2. Let A_{O_C} be the set of attributes of O_C and K_{O_S} be the set of key attributes of O_S. There are then two situations:

1. $K_{O_S} \subset A_{O_C}$ when, for instance, $O_C \longrightarrow\!\!\ni O_S$. Again, there are two cases:

   (a) K_{O_S} corresponds to a cardinality-1 group in O_C. The values for K_{O_S} are inherited from O_C and O_S is instantiated without any user interaction.

   (b) K_{O_S} corresponds to a cardinality-n group in O_C. One has then to choose the element of interest in the group before instantiation can proceed.

2. $K_{O_S} - A_{O_C} \neq \emptyset$ when, for instance, $O_C \longrightarrow\!\!* \; other \longrightarrow\!\!* \; O_S$. Here, the attributes from $K_{O_S} \cap A_{O_C}$ are inherited from O_C and the user is asked to provide the remaining ones to complete the selection clause.

As an example of such transactions, let us assume that I_C is the ANTIBODY OBJECT instance displayed in Figure 4. Among all the neighbors of the ANTIBODY OBJECT template (now O_1) in the network of Figure 3(c), the user chooses ANTIGEN OBJECT (now O_2). Since O_1 and O_2 are linked by a reference connection, PENGUIN retrieves a single instance of O_2, using the values of I_C's attributes **antigen-name** and **antigen-species**, and displays it as shown in Figure 5.

This method constitutes a user-friendly, time-saving solution to the problem of traversing an object network in a given data environment where context instantiations can be done iteratively, moving from node to node, and using data from prior instances.

Object Decomposition

An indispensable feature of our architecture is that it permits any update operation on the object instances (that is, any sequence of insertion, deletion, and modification transactions). Of course, one expects those changes to be made persistent, and hence to be moved from the object representation to the base relations of the database. However, the object templates are defined using join operations on the database relations. We are then facing the well-known problem of updating relational databases through views involving multiple relations [11]. Updating through views is inherently ambiguous, as a change made to the view can translate into different modifications to the underlying database.

Keller [16] has shown that, using the structural semantics of the database, one can enumerate such ambiguities, and that one can choose a specific translator at view-definition time. Because of the analogy between relational views and PENGUIN's objects, Keller's algorithm applies in our case. Therefore, when creating a new object template, a simple dialogue, the content of which depends on the structure of the object, allows the user to select one of the semantically valid translators. The chosen translator is then stored as part of the template definition. As an illustration, part of such a dialogue to select an update translator for the ANTIBODY OBJECT template of Figure 3 follows:

```
Is replacement of instances allowed? Yes
Can clone-name be changed? No
Can a new tuple be created in the antigen relation? Yes
Can antigen-name or antigen-species be changed for an existing antigen? No
```

Later on, when an object instance is modified, the object decomposer will use this information to resolve any ambiguity completely and to update the database correctly.

Benefits

By adding an object-based interface to a relational DBMS, our architecture integrates many of the benefits that are found separately in either of the two models today.

The interface provides a clear separation between storage and working representations. Through PENGUIN's graphic interface, the user is able to interact at a higher level of abstraction than that of the information explicitly stored in the database. Indeed, a single object, usually represented by many base tuples over various relations, is now retrieved declaratively without having to write complex SQL statements, and is handled as though it were a single unit. Moreover, the same real-world entity can have different descriptions for different purposes, ranging from a high-level, black-box template to a low-level, detailed one.

Providing **multiple views** of the same database information is a necessity for biomedical applications. Our approach, which relies on the definition of object networks, significantly enhances the view mechanism found in relational DBMSs. As previously noted, a template network is made of all the related objects to which a group of users has access. Different views then correspond to separate template networks, although a given object can be in as many networks as are defined for the specific application. This provides a compact representation, since an object is stored only once but can be shared by many views. Figure 6 illustrates this view concept where templates of various object networks map to the same base relations.

This architecture provides **sharing of information** contained in the database. Any application-specific structure is imposed at the interface level, not at the database level. Further, the basic distinction between an object and its instances should support a multiple-granularity approach to concurrency control, where locking could include any one of a single instance, an object and its instances, or a set of objects.

Growth of the system and **flexibility** are easily accomodated. The database administrator can augment the underlying database with new attributes or relations without affecting work on existing templates and object networks. Conversely, the addition, modification or reorganization of objects does not require that the database schema be modified accordingly.

Through the caching mechanism, we avoid the significant overhead of making a database request each time a data item is needed. Because the real- and virtual-memory capabilities of today's workstations are constantly expanding, and because the templates are a compact representation scheme, this technique should be adequate for many biomedical applications. We then provide **high performance** for browsing and navigation operations.

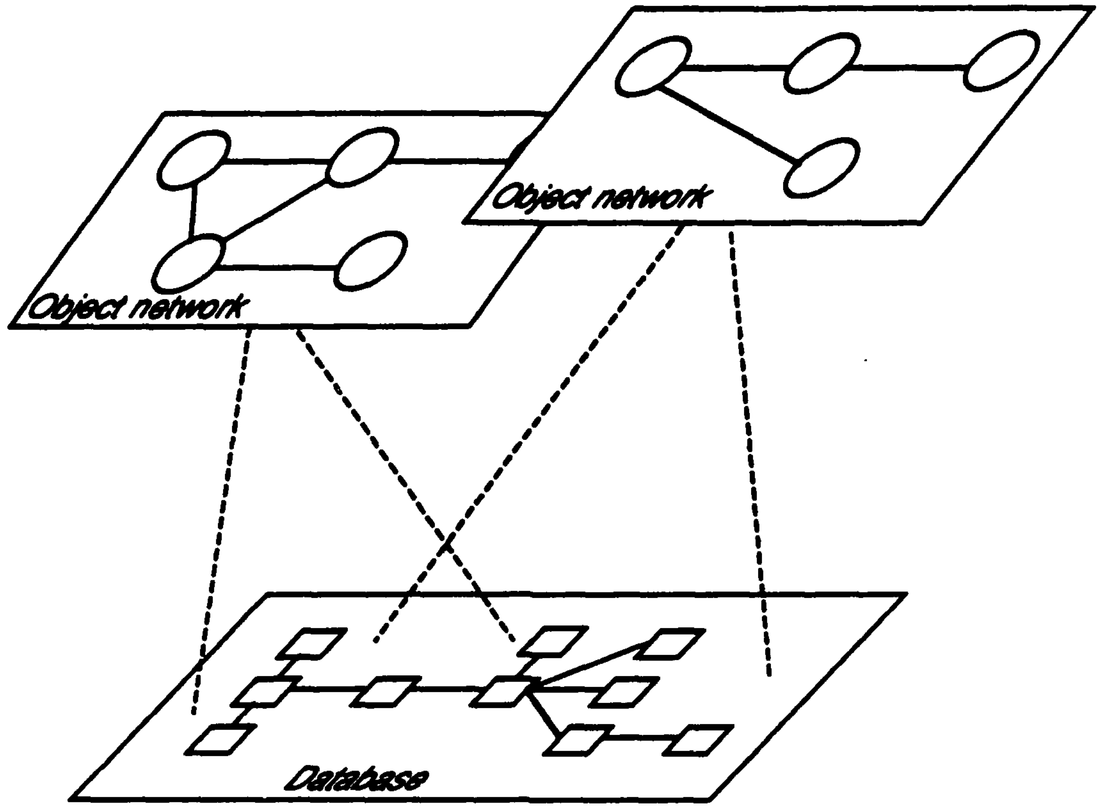


Figure 6: Database views and object networks.

In addition to improving database transactions, the object layer represents an **efficient tight coupling mechanism** between database and knowledge base systems. Through PENGUIN's mapping scheme, an expert system can request at any time access to arbitrarily complex objects; upon receipt of such requests, the object layer dynamically generates the corresponding SQL queries, assembles the relational tuples into object instances, and transfers them in the expected format to the expert system. Similarly, any update operation on objects can be translated into a series of database update transactions. The object layer thus acts as an "intelligent" data server for the expert system. Moreover, our notion of an *object schema* achieves domain-independence. We keep the domain-specific information (namely, the set of objects for a given application) separate from the mapping, instantiation, and decomposition protocols, much in the same way expert system shells separate the knowledge base from the inference engine.

Conclusion

The concept of objects has been successfully, albeit separately, applied to both the database and the expert system field. However, we believe it also represents a powerful unifying paradigm in the design of expert database systems. We have introduced a novel, domain-independent architecture for EDSs that is suitable for many biomedical applications. Using a semantic data model called the structural model, PENGUIN combines the object-oriented and the relational model to create an object-based interface between relational databases and expert systems. We have defined three components for this interface. The object generator creates object templates that are complex units incorporating many attributes from various base relations. The object instantiator generates object instances from base data and provides flexible browsing tools. The object decomposer translates object-update operations into relational-update transactions to maintain consistency between the storage representation (the relations) and the working representation (the objects). Results of PENGUIN's prototype implementation indicate that an object-based architecture provides (1) efficient access to and manipulation of complex units of biomedical information while preserving the advantages associated with persistent storage of data in relational format, and (2) a domain-independent, efficient communication channel between relational database systems and expert systems.

Acknowledgments. Thanks are due to Benoit Mulsant, Marty Chavez, and Edward Herskovits for reviewing this paper, and to Lyn Dupre for helpful editorial comments. This work was supported by DARPA contract N39-84-C-0211 for Knowledge Based Management Systems and by Digital Equipment Corporation under the Quantum project. Additional computer facilities were provided by the SUMEX-AIM resource under NIH grant RR-00785.

References

[1] T. Barsalou, W.A. Moore, L.A. Herzenberg, and G. Wiederhold. A database system to facilitate the design of FACS experiment protocols (abstract). *Cytometry*, page 97, August 1987.

[2] T. Barsalou and G. Wiederhold. Applying a semantic model to an immunology database. In W.W. Stead, editor, *Proceedings of the Eleventh Symposium on Computer Applications in Medical Care*, pages 871–877, Washington, D.C., November 1987. IEEE Computer Society Press.

[3] P.A. Bernstein and N. Goodman. What does Boyce-Codd normal form do? In *Proceedings of the 6th Conference on Very Large Data Bases*, pages 245–268, Montreal, Canada, October 1980.

[4] L. Bic and J.P. Gilbert. Learning from artificial intelligence: New trends in database technology. *IEEE Computer*, 19(3):44–54, 1986.

[5] W.A. Bonner, H.R. Hulett, R.G. Sweet, and L.A. Herzenberg. Fluorescence activated cell sorting. *Rev. Sci. Instru.*, 43:404–409, 1972.

[6] M.L. Brodie and J. Mylopoulos. Knowledge bases vs databases. In M.L. Brodie and J. Mylopoulos, editors, *On knowledge base management systems*, pages 83–86. Springer-Verlag, New York, NY, 1986.

[7] P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Systems*, 1(1):9–36, 1976.

[8] E.C. Chow. Representing databases in frames. In *Proceedings of AAAI-87*, pages 405–410, Seattle, WA, 1987. American Association for Artificial Intelligence.

[9] E.F. Codd. A relational model for large shared data banks. *Communications of the ACM*. 13(6):377-387, 1970.

[10] C.J. Date. *An Introduction to Database Systems*. Addison Wesley, Reading, MA, 1983.

[11] A.L. Furtado and M.A. Casanova. Updating relational views. In W.Kim, D.S. Reiner, and D.S. Batory, editors, *Query processing in database systems*. Springer-Verlag, New York, NY, 1985.

[12] A. Goldberg and D. Robson. *Smalltalk-80: the language and its implementation*. Addison-Wesley, 1983.

[13] J. Gomsi and M. Desanti. BOOPS and the relational database. *AI Expert*, 2(10):60–66, 1987.

[14] M. Hammer and D.J. McLeod. Database description with SDM: A semantic data model. *ACM Trans. Database Systems*, 6(3):351–386, 1982.

[15] D.J. Hartzband and F.J. Maryanski. Enhancing knowledge representation in engineering databases. *IEEE Computer*, 18(9):39–48, 1985.

[16] A.M. Keller. The role of semantics in translating view updates. *IEEE Computer*, 19(1):63–73, January 1986.

[17] S.N. Khoshafian and G.P. Copeland. Object identity. In N. Meyrowitz, editor, *Proceedings of OOPSLA 86*, pages 406–416, New York, NY, 1986. ACM.

[18] D. Maier, J. Stein, A. Otis, and A. Purdy. Development of an object-oriented DBMS. In N. Meyrowitz, editor, *Proceedings of OOPSLA 86*, pages 472–482, New York, NY, 1986. ACM.

[19] M. Minsky. A framework for representing knowledge. In P. Winston, editor, *The psychology of computer vision*, pages 211-277. McGraw-Hill, 1975.

[20] M. Missikoff and G.Wiederhold. Towards a unified approach for expert and database systems. In L. Kerschberg, editor, *Proceedings of the First International Workshop on Expert Database Systems*, pages 383–401. Benjamin Cumming Publishing Co., Inc., 1986.

[21] J. Moad. Building a bridge to expert systems. *Datamation*, (1):17-19, 1987.

[22] J. Mylopoulos, P.A. Bernstein, and H.K.T. Wong. A language facility for designing database-intensive applications. *ACM Trans. Database Systems*, 5(2):185–207, 1980.

[23] L. Rowe. A shared object hierarchy. In *Proceedings of the International Workshop on Object-oriented Database Systems*, pages 160–170, Asilomar, CA, September 1986. IEEE.

[24] J.M. Smith. Expert database systems: A database perspective. In L. Kerschberg, editor, *Proceedings of the First International Workshop on Expert Database Systems*, pages 3–17. Benjamin Cumming Publishing Co., Inc., 1986.

[25] J.M. Smith and D.C.P. Smith. Database abstraction: Aggregation and generalization. *ACM Trans. Database Systems*, 2(2):105–133, 1977.

[26] M. Stefik and D.G. Bobrow. Object-oriented programming: Themes and variations. *AI Magazine*, 6(4):40–62, 1986.

[27] M. Stonebraker. Object management in POSTGRES using procedures. In *Proceedings of the International Workshop on Object-oriented Database Systems*, pages 66–72, Asilomar, CA, September 1986. IEEE.

[28] M. Stonebraker and L. Rowe. The design of POSTGRES. In *Proceedings of the International Conference on Management of Data*, Washington, D.C., May 1986. ACM-SIGMOD.

[29] J.D. Ullman. Database theory: Past and future. In *Proceedings of the 6th Symposium on Principles of Database Systems*, pages 1–10, San Diego, CA, March 1987. ACM SIGACT-SIGMOD-SIGART.

[30] Y. Vassiliou. Knowledge-based and database systems: Enhancements, coupling or integration? In M.L. Brodie and J. Mylopoulos, editors, *On knowledge base management systems*, pages 87–93. Springer-Verlag, New York, NY, 1986.

[31] G. Wiederhold. Binding in information processing. Technical Report STAN-CS-81-851, Stanford University, 1981.

[32] G. Wiederhold. *Databases for health care*. Lecture Notes in Medical Informatics. Springer-Verlag, New York, NY, 1981.

[33] G. Wiederhold. *Database Design*. Computer Science Series. McGraw Hill, New York, NY, 1983. 2d edition.

[34] G. Wiederhold. Knowledge bases. chapter 1, pages 110–122. Proceedings of the International Symposium on Fifth Generation and Super Computers, 1984.

[35] G. Wiederhold. Views, objects and databases. *Computer*, 19(12):37–44, December 1986.

[36] G. Wiederhold and R. ElMasri. The structural model for database design. In *Entity-relationship Approach to System Analysis and Design*, pages 237–257. North-Holland, 1980.

Applying a Semantic Model to an Immunology Database

Thierry Barsalou, Gio Wiederhold
Stanford University

Abstract

Work is underway to develop a research database that will assist investigators in designing protocols for flow-cytometry immunologic experiments. Flexibility, consistency, cooperation and transparency have been identified as highly desirable goals. The relational model, used for implementing the database, has however some limitations. We are exploiting a structural model of the database to develop application-independent capabilities that address those relational weaknesses. Examples of applying such semantic information in the field of integrity maintenance and data abstraction through generation of objects are presented.

Overview

Flow Cytometry and its Biomedical Applications

In the late 1960s, a group of biomedical researchers at Stanford University set out to develop means to analyze and isolate viable cells that have different cell-surface phenotypes detected by fluorescence-tagged antibodies. This eventually led to a flow cytometry sorting method named *fluorescence activated cell sorting* with the acronym FACS [1]. This method has been improved continually and numerous biological and clinical applications have been developed and employed. By now, an estimated several thousand FACS analyzers and sorters are in use in nearly all biological research institutions and in large numbers of clinical research laboratories throughout the world.

The FACS technique is heavily used in the fields of immunology and cell biology, including clinical immunology. In essence, cell samples are stained with fluorochrome-conjugated antibodies that recognize cell-surface determinants on subpopulations. FACS can then be used to measure frequencies of subpopulations in the sample and to sort individual subpopulations for functional analysis. The development of "hybridomas" that produce unlimited quantities of highly specific monoclonal antibodies greatly improved this methodology. Cell staining with fluorochrome conjugates of these monoclonal antibodies permits investigators to obtain absolutely reproducible results anywhere and at any time using well-calibrated and reproducibly operated FACS instrumentation.

FACS has four key advantages over other technologies dealing with the properties of small particles: (1) the ability to take quantitative data on individual cells, (2) the ability to obtain correlated data on multiple cell parameters, (3) the small samples needed for analysis (or sorting), and, most important, (4) the ability to sort (viably, if desired) one or all cells in a sample displaying a particular combination of quantitative parametric measurements.

Information Needs

In the past, most immunofluorescence studies were limited to the use of a single fluorochrome. Now, because an increasing number of suitable fluorochromes has become available and because new FACS instruments are fitted frequently with more than one light source, it has become possible to design and perform multicolor studies. This has greatly increased the scope of FACS experiments for examining small subpopulations of clinically relevant cells.

Many different types of information and knowledge need to be combined however when the complex protocols required for these multiparameter experiments are prepared. The FACS users in an immunology research setting typically draw from a series of more than 50 monoclonal antibody reagents, each of which may be available in two or more forms as conjugates with different fluorochromes. In a multiparameter experiment, which may contain 50 to 100 samples, cells in each sample usually are stained with two or three separate reagents, each recognizing a different cell-surface determinant and each coupled to a different fluorochrome. In producing an experimental protocol, investigators must juggle reagents taking a variety of factors into account. For example,

- Certain fluorochromes are detected more efficiently by the FACS instrumentation than are others
- Certain fluorochrome-emission spectra overlap more than others do
- Certain determinants are present in greater abundance on the cell surface than others are
- Different reagents can have the same antigen specificity
- Certains reagents interact with other reagents
- Certain reagents are in short supply

Thus, during the protocol-design process, various questions need to be answered. What are the cell populations to be sorted? What target antigens are likely to produce the best differentiation among these populations? How does the expression of these surface markers vary with the tissue of origin or the ongoing treatment? What is the best combination of antibodies and fluorochromes against those antigens? Finally, a more basic but equally important problem is how to cope with the extraordinary fast evolution and growth in immunologic knowledge.

A Database for FACS Reagents

Because the design of experimental protocols is both data- and knowledge-intensive and because human expertise in the domain is scarce, providing assistance in this process is important for achieving wider dissemination of FACS, especially in clinical research settings. A first step in this direction is the development of a reagent database that will meet the basic information needs of FACS investigators.

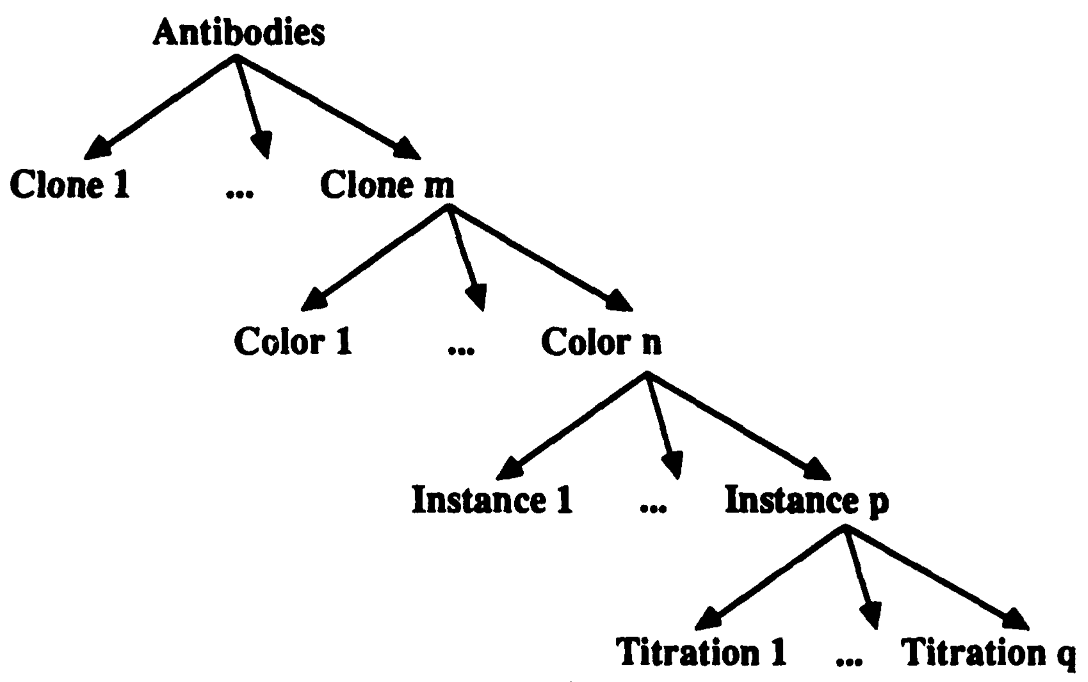


Figure 1: The antibody hierarchy.

Domain specifications have allowed us to identify the major database entities, and to define the data attributes. The database is organized around two main components, the antibody and the antigen. The considerations that guided the definition of the attributes follow:

Antibody component. Information about antibodies includes the description of the antibody itself (name, clone of origin, immunologic attributes, antigen specificity), data about the attached fluorochromes and information on the preparations. The last is necessary because different preparations can be made for the same antibody-fluorochrome pair and because a preparation has changing titration over time. As shown in Figure 1, the antibodies can be represented in a hierarchical structure: several fluorochromes can be used with a given antibody; similarly, different preparations uniquely identified by a batch number are usely made for each antibody fluorochrome pair; finally, each preparation has several titration instances attached to it.

Antigen component. Different elements are considered when the attributes for an antigen are defined. The species provides a first level of discrimination. The race (or strain, for mice) is an important factor for cell-surface antigen expression. The same antigen can be present in different organs as well as in different cell populations where it is expressed differently. Finally, a cell population is present in different proportions in different organs. Besides information about the antigens themselves, the data about cells and organs are equally important when designing experimental protocols and are defined as two additional entities.

Based on this analysis and on interviews with our experts, we have elicited the general querying capabilities of the database. Examples of those queries are:

- Retrieve all matching antibodies for a given target antigen
- Display the list of antigens classified either by organ or by cell population

- Retrieve all reagents that exist in combination with a certain fluorochrome
- Find the antibodies matching a specific combination of fluorochrome and antigen specificity
- Retrieve all antigens for a specific combination of species and cell population; for each of these antigens, list the matching antibody instances with their preparation information

Design Issues

Database Interface Desiderata

When designing a database application to be used by non computer science people, it is highly desirable to define general principles ensuring the system's persistence:

- **Flexibility**. A database is by definition an abstraction of the real world. Because the real world, especially in biomedical research, is bound to evolve rapidly, the database should be able to change accordingly. This implies a database model that can accomodate extensions or modifications of the existing entities and, to some degree, independence of the application from both the software and hardware of the underlying database management system.
- **Consistency**. Enforcing integrity of the data can take two different forms of interest to the user. *Static* integrity constraints specify the conditions that the database should obey. Typically, these constraints restrict the value of an attribute to some range or define arithmetic relationships among different fields. More subtle are the *dynamic* integrity constraints, which state how the database consistency is to be maintained when an update transaction requires associated changes. For example, an antibody panel is useful to the investigator only when discussed in terms of the antigen that the antibody panel is designed to detect. Therefore, an antigen cannot be deleted from the database unless all the antibodies matching this specific antigen are erased at the same time.
- **Cooperation**. Cooperation is an important principle for interactive applications that are designed to convey information between the user and the database [3]. Successful cooperation bridges the gap between the arbitrary database representation of the world and the user's set of information needs and domain-specific knowledge. Cooperative engagement can take place at several points during an interactive session:
  - Data update can be improved if all interdependent entities representing a more general concept are inserted or deleted together.
  - Traditional query languages are difficult to learn and awkward to use for casual users. These languages typically do nothing to prevent syntactic and semantic errors during query formulation. Active assistance of the system by guiding step by step the building of the query can alleviate this important source of users' frustration.
  - Other sources of rigidity stemming from the impossibility of handling minor changes or of reusing previous queries can be avoided if the system has elliptical querying capabilities of the form *"What if lymphocytes is replaced by B-lymphocytes in the previous query?"* and provides log files of all queries in the current session.

- Finally, post-query cooperation through summarization of overwhelming information or aggregation of data into more meaningful abstract concepts can make query answers more informative and thus more useful. Such cooperation can be most adequately supported by object-oriented programming languages. These languages represent entities as object instances— that is, collections of data elements and methods (procedures operating on the instance). Main features of object-oriented systems include hierarchies of objects, inheritance of attribute values from general instances down to more specific ones in the hierarchy and communication between objects through message passing. These provide the grounds for working at a higher level of abstraction than traditional applications allow.

- Transparency. As noted, data manipulation languages (DML) often are opaque to the user. This can lead to misinterpretation of complex queries results. Developing a transparent data manipulation syntax may reduce the query language expressiveness but also will benefit casual users by avoiding ambiguities and misunderstandings.

Extending the Relational Model

The relational database model [2] was selected for this application. This model provides only one data structure, the relation, which is easy to understand and is intuitive to most people. Moreover, the relational calculus provides a mathematically complete declarative language to manipulate data. The inherent simplicity and flexibility of this approach also were considered essential in a research environment where it is almost impossible to predict a priori the ultimate format for the information.

However, the relational model *per se* cannot deal with many of the desiderata stated in the previous section. We now review these limitations in light of the goals description:

- Static integrity constraints are included in most relational DMLs but this is not the case for dynamic ones. Although the principles for enforcing these constraints are quite simple and can be encoded in a few rules, knowledge of the structure of the database also is required. Such structural knowledge, which includes how the entities are related to one another as well as the semantics of these relationships, is completely absent from the relational model.

- Cooperative engagement is a relatively new research issue. The relational model and other prevailing models (hierarchical, network) are static models that do not accommodate active involvement of the database system by, for instance, data abstraction through object generation. Yet, most applications are oriented toward such operations on individual objects rather than on data tuples. We would like to combine the simplicity and efficiency of the relational model and the attractive capabilities of the object-oriented paradigm. However, much domain-specific knowledge and a set of application-independent procedures are needed to provide effective cooperation.

- Although relational calculus is a powerful language, it cannot be seen as the biomedical researchers' sole means of communication with the database. An additional interface that translate English-like statements into the appropriate relational calculus expression is needed to provide transparency and ease of use. Such an interface is not commonly available in relational database systems.

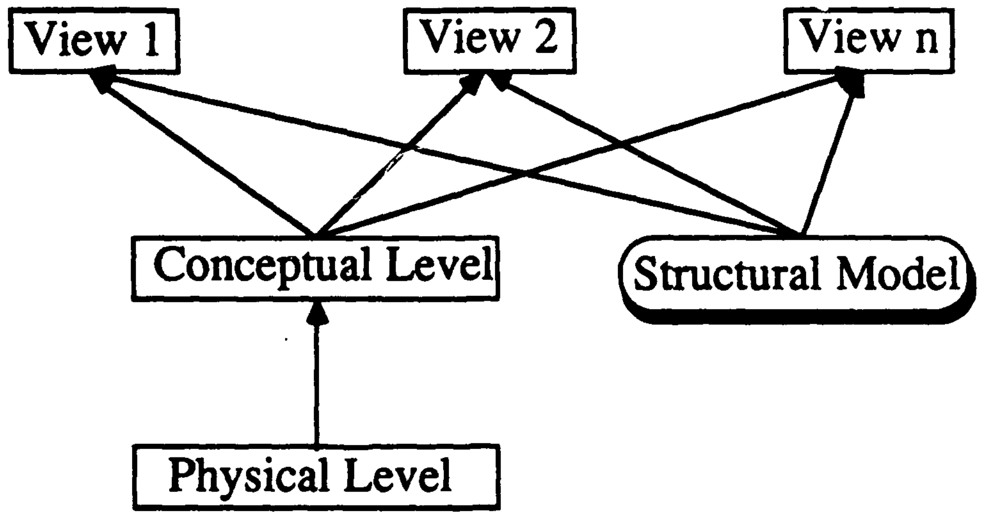


Figure 2: Combining the structural model and the conceptual level of abstraction.

The problems of the relational model share a common feature; namely, the absence of any semantic information about the database. Database theory has always recognized the existence of three levels of abstraction [4]: (1) the physical level, which describes how the data are physically stored, (2) the conceptual level, which describes which data are stored, and (3) the view level, which describes only those parts of the database appropriate for the information needs of a class of users (many views can coexist for the same database). The primary function of database applications is to create a map for the user from the view level to the conceptual level. The **structural model**, proposed by El Masri and Wiederhold [8], augments the relational model by capturing the knowledge about constraints and dependencies among relations in the database. It can provide a means of addressing several limitations of the relational model. As illustrated in Figure 2, we propose juxtaposing the conceptual level and the structural model, and using both of them to generate a more accurate and meaningful map. Thus, we will allow the user to interact at a higher level than what is actually stored in the database.

Modeling the Structural Knowledge

Normalization of the Database Schema

When researchers design a database, some problems can arise. Among the undesirable properties a bad design may have are unwarranted redundancy of data, potential inconsistency, loss of data and inability to represent certain information. To avoid these problems, one can normalize the initial model by using functional dependencies among attributes (see [5] for a complete discussion of normalization). This approach leads to a normal form of the relational schema by decomposing entities into relations and by separating the attributes of each relation into a ruling part, or key, that uniquely identifies a dependent part.

For the sake of brevity, we will use only the antibody entity as supporting material for the examples throughout the remainder of this paper. As required by normalization, the antibody concept has to be split into five distinct relations, the ruling parts of which and information about dependent parts, are listed below:

ANTIBODIES
Antibody Name :)
Antigen Specificity, Antibody Data.

IMMUNOLOGIC ATTRIBUTES
Antibody Name :)
Immunologic Characteristics.

ANTIBODY-COLOR PAIRS
Antibody Name, Color Name :)
Data Specific to Pairs.

INSTANCES
Antibody Name, Color Name, Batch Number :)
Data Specific to Preparations.

TITRATIONS
Antibody Name, Color Name, Batch Number, Date of Titration :)
Data Specific to Titrations.

Primitives of the Structural Model

The primitives of the structural model are *relations* as determined by the normalization and *connections* to describe relationships between the relations. Formally, a connection is specified by the connection type and a pair of source and destination relations. Three types of connections are defined that correspond to precise integrity constraints, define the permissible cardinality of the relationships and encode the relationships' semantics:

1. **Ownership connection.** A single tuple owns many lower-level tuples of the same type, which implies a one-to-many cardinality. The connection embodies the concept of dependency, where owned tuples are specifically related to a single owner tuple. Hierarchies with "*... has some ...*" type links thus can be implemented using ownership connections. Syntactic and integrity rules for the ownership connection are as follows: (1) the ruling part of the owned relation is the concatenation of the ruling part of the owner relation and one (at least) attribute to distinguish individuals in the owned sets; (2) a new tuple can be inserted into the owned relation only if there is a matching tuple in the owner relation; (3) deletion of an owner tuple implies deletion of the owned tuples.
The graphical symbol of the ownership connection is $\longrightarrow\!\!*$.
2. **Reference connection.** Multiple tuples in the same relation refer to the same descriptive tuple of a different relation; this situation corresponds to a many-to-one cardinality. The connection of two related concepts, one being more general than the other, corresponds to an

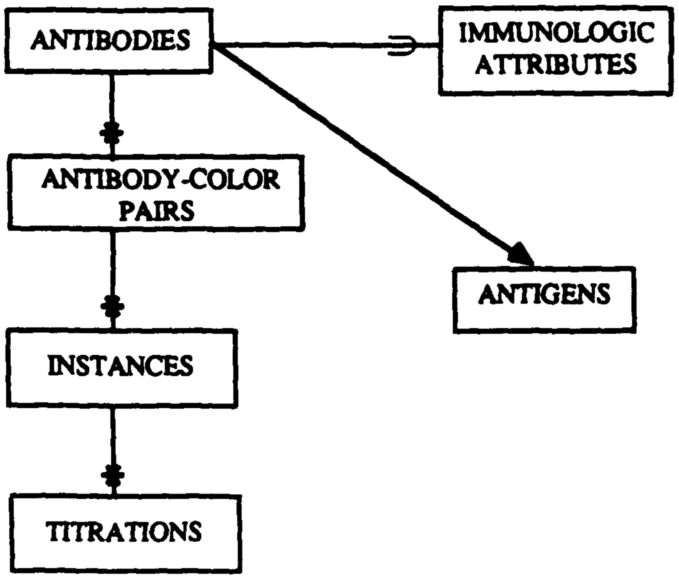


Figure 3: Partial structural model of the reagent database.

abstraction process. As we will see, this bears immediate significance for summarization tasks where abstraction is the key idea. Rules for the reference connection are as follows: (1) the ruling part of a referenced relation matches the referencing attributes of the primary relation; (2) a destination (referenced) tuple must exist when a referencing tuple is inserted; (3) if source (referencing) tuples exist, deletion of a destination (referenced) tuple is prohibited.
The graphical symbol of the reference connection is $\longrightarrow$.

3. **Subset connection.** This connection has a partial one-to-one mapping from one relation to another, so a tuple in the superset relation is linked to at most one tuple in the subset relation. Categorical hierarchies with "... *is a* ..." type links also can be represented using the subset formalism. The rules for the subset connection areas follows: (1) the ruling part of a subrelation matches the ruling part of its connected general relation—i.e., only the dependent parts differ; (2) a general tuple must exist when a subset tuple is inserted; (3) deletion of the general tuple implies deletion of the corresponding subset tuple.
The graphical symbol of the subset connection is $\longrightarrow\!\!\ni$.

The structural connections permit modeling of all relational schema, although complex relationships such as many-to-many connections may require combinations of the three types. Most important, the connections provide the structural knowledge that is needed to overcome the limitations of the relational model.

The Database Structural Model

Figure 3 shows the final model of the part of the database that deals with the antibody entity. It illustrates the connections among the base relations. Those relations and the connections together constitute the formal description of the database.

The ANTIBODIES relation is connected with the IMMUNOLOGIC ATTRIBUTES relation through a subset connection, because each tuple in the former relates to at most one tuple in the latter (which holds immunologic characteristics of the antibody tuple). The specificity of an antibody refers directly to an antigen instance, which explains the reference connection between the ANTIBODIES and ANTIGENS relations. Finally, the hierarchical structure presented in Figure 1 translates into a chain of ownership connections: ANTIBODIES owns ANTIBODY-COLOR PAIRS, which owns INSTANCES, which owns TITRATIONS.

Functional Design

We now apply the structural model in two precise contexts and show how this knowledge, albeit limited in depth, can be exploited fruitfully.

Integrity Maintenance Using Dynamic Constraints

As we have shown, the three structural connections are associated with precise integrity maintenance rules that we use for enforcing database consistency in a dynamic fashion. The goal here is to react to an integrity violation attempt by finding means to correct the inconsistency and not simply by rejecting the whole transaction. During each database update, algorithms can then check the consistency of a tuple insertion or deletion against the model's structural knowledge. We will present here only the case of the ownership connection; similar algorithms do exist for the subset and the reference connections.

Insertion in an Owned Relation. The ownership structural constraint specifies the concept of dependency. It is an error if a tuple inserted in an owned relation does not depend on an existing tuple in the owner relation, and such error should be handled by either inserting a tuple in the owner relation or aborting the entire transaction. However, if those two relations are part of a dependency hierarchy implemented by a chain of ownership connections, even this is not sufficient. Such error checking has to be propagated up through the hierarchy. Formally, a chain of ownership connections of length n is defined by:
A set of n relations R_1 to R_n such that if RP_m is the ruling part of R_m,

$$RP_m \subset RP_{m+1}, \text{ for } m = 1 \text{ to } n-1. \qquad (1)$$

The following algorithm handles the general case of inserting a tuple at any point in an ownership chain of length n:

Let t be the tuple to be inserted in R_m, with $2 \leq m \leq n$, and the ruling part RP_m be the set of attributes $\{ a_1 \ldots a_i \}$.

1. From (1), $RP_{m-1} = \{ a_1 \ldots a_j \}$ with $j < i$
2. Determine whether there is a tuple t' in R_{m-1} such that $t'.a_k = t.a_k$, for $k = 1$ to j
3. If t' is not found, there is a potential violation, so t' needs to be inserted in R_{m-1}
   (a) The values for the ruling part attributes of t' are assigned by $t'.a_k = t.a_k$, for $k = 1$ to j

```
Inserting a tuple into the INSTANCES relation:

BATCH-NUMBER? "1235-86A"
CLONE-NAME? "AntiIgD"
COLOR-NAME? "Fluorescein"
DEFAULT-AMOUNT? 0.02
AVAILABLE-AMOUNT? 25.0
DATE-CREATION? 03/01/87

Warning: There is no ANTIBODY-COLOR PAIRS tuple owning this INSTANCES tuple!
Insert it? Y
F-P-RATIO? 0.2

Warning: There is no ANTIBODIES tuple owning this ANTIBODY-COLOR PAIRS tuple!
Insert it? Y
ANTIGEN-NAME? "IgD"
ANTIGEN-SPECIE? "Human"
BRIGHTNESS? 2

ANTIBODIES tuple inserted!
ANTIBODY-COLOR PAIRS tuple inserted!
INSTANCES tuple inserted!
```

Figure 4: Recursive insertion following an ownership chain upward.

(b) The values for the dependent part attributes of t' are given by the user

(c) If $m > 2$, make a recursive call to 1. with $t = t'$ and $m = m - 1$, to check that t' is owned by a tuple in $R_m - 2$

(d) Else $m = 2$, t' is inserted in R_1

4. Tuple t' exists, consistency is ensured and t is inserted in R_m

This algorithm detects the violation of an ownership constraint and performs recursive insertions of tuples up through the hierarchy until an owning tuple is found or the top level is reached. Because the system knows about the chain structure, the ruling part of all subsequent tuples to be inserted at higher levels is determined automatically from the original tuple. Thus, not only is the consistency of the database maintained, but also the user saves time, because entering redundant information is avoided; only the dependent part attributes need to be specified.

Figure 4 illustrates this algorithm using the hierarchy ANTIBODIES $\longrightarrow$ ANTIBODY-COLOR PAIRS $\longrightarrow$ INSTANCES. A tuple with ruling part { *"AntiIgD"*, *"Fluorescein"*, *"1235-86A"* } is inserted in the relation INSTANCES. However, no tuple with ruling part { *"AntiIgD"*, *"Fluorescein"* } is found in the owning relation ANTIBODY-COLOR

PAIRS so a new tuple is inserted once the dependent part attributes have been specified by the user. Similarly, an ANTIBODIES tuple with ruling part { *"AntiIgD"* } is added to the database, because the new tuple in ANTIBODY-COLOR PAIRS does not have an owner in the ANTIBODIES relation.

Deletion in an Owner Relation. When inserting a tuple in a chain of ownership connections, we were moving up in the hierarchy to enforce consistency. Deletion takes the opposite approach: It is perfectly legal to delete an owned tuple (like the { *"AntiIgD"*, *"Fluorescein"*, *"1235-86"* } tuple we just inserted), but, on the other hand, deleting an owner tuple also implies erasing all its owned tuples, the existence of which depends on the owner tuple's existence. We now have to go down in the hierarchy in a breadth-first search manner. The algorithm to delete a tuple in an ownership chain of length n is then as follows:

Let t be the tuple to be deleted in R_m, with $1 \leq m < n$, and the ruling part RP_m be the set of attributes $\{ a_1 \ldots a_i \}$.

```
1. For $j = m + 1$ to $n$
    (a) Find all tuples $t'$ in $R_j$ such that $t'.a_k = t.a_k$, for $k = 1$ to $i$
    (b) Delete those tuples $t'$ in $R_j$
2. Delete tuple $t$ in $R_m$
```

As an example, if an ANTIBODIES tuple with ruling part { *"AntiIgG"* } is deleted, the chain ANTIBODIES ⟶ ANTIBODY-COLOR PAIRS ⟶ INSTANCES ⟶ TITRATIONS is traversed down. All tuples in each of the three relations ANTIBODY-COLOR PAIRS, INSTANCES and TITRATIONS with the attribute 'antibody name' equal to *"AntiIgG"* are deleted at the same time.

It is important to note that these two algorithms, as well as those for the reference and subset connections, are application-independent and adaptive to changes in the database structure. They are easily transportable to different application domains, because all the necessary domain-dependent information is kept in the structural model, which is a separate entity.

Objects and Relations

Scientists, especially in the FACS domain, deal with abstract entities and objects rather than with simple tuples. Structural knowledge of the database is again used to generate and manipulate objects at different levels of granularity.

Summarization. The typical response to a database query is to list the set of items from the database that satisfy the conditions presented in the query. However, this list can be excessively long, and thus inappropriate, for a conversational system. In such cases, a more adequate response is to give a description of the set, rather than merely listing its elements. This means abstracting an object of coarse granularity from this data set by finding attributes that have summarization power.

The reference connection is precisely aimed at supporting abstraction processes, because tuples in a referencing relation are related to a more general concept in a referenced relation [6]. By looking through the reference attributes of a data set, we can combine tuples that have a common reference: this allows us to provide the class description, reducing the amount of information presented on the screen.

For instance, if a query response consists of a list of antibodies, we can use the reference connection between the ANTIBODIES and ANTIGENS relations to summarize and present the data in a more informative way. Such a response would be:

```
     There are 45 antibodies satisfying the query.
       Summarization on antigen specificity gives:

             1:  22 antibodies against IgM,
             2:  18 antibodies against IgD,
             3:  3 antibodies against Ly-1,
             4:  2 antibodies against B220.
           Which ones do you want to see?  ...
```

This approach is similar to the interaction you would expect to occur with a human being; it is a reasonably informative communication process, not an exchange of lengthy lists of raw data. Summarization is thus a first step toward cooperative database systems.

Object Instances. Objects are complex entities that cannot be stored explicitly in a relational database. Indeed, normalization requires that information concerning an object be distributed over several relations. A good example of an object is the antibody, which of course corresponds to the ANTIBODIES relation but also has some information stored in the IMMUNOLOGIC ATTRIBUTES, ANTIGENS and ANTIBODY-COLOR PAIRS relations. Thus, multiple tuples from different relations need to be brought together by a join operation to reconstitute a meaningful object such as an antibody.

In our view, however, listing those tuples that do correspond to the object of interest is not enough. Consider as an example a scientist looking for information about an "AntiIgG" antibody's specificity and available fluorochromes. The typical database answer would be:

Antibody Name	Antigen Name	Antigen Species	Fluorochrome
AntiIgG	IgG	Human	Fluorescein
AntiIgG	IgG	Human	Texas Red
AntiIgG	IgG	Human	Allophycocyanin
AntiIgG	IgG	...	...

Although all the information the scientist requested is there, this view of the antibody is still extremely akward and clearly is not satisfactory. We need a way to avoid redundancies and to rearrange attributes in a meaningful way.

To solve this problem, we propose to use the structural model and a new module, the **view-object generator** [7]. Three components are necessary in this architecture:

1. The set of relations of the database application that serve as a permanent repository for all the data needed to generate object instances

2. A set of object templates that is the metadescription of the object types, similar to the schema defined for a database. These templates, generated semiautomatically using the database structural information, specify the structure and mappings of the data elements within an object
3. A view-object generator, which is responsible for producing object instances. It performs the following steps:
   (a) Specifies the primary relation, thus identifying the object template of interest
   (b) Builds an appropriate query to extract from the multiple base relations the tuples that correspond to the object instance
   (c) Combines the data and the predefined object template to produce the desired object instance

Non procedural access to object instances would be provided by this architecture. The user only needs to specify the object prototype and the selection clause to obtain the desired instance. The object-oriented approach offers then a better, more compact user interface to relational database applications.

Inserting Concepts. Besides integrity maintenance, the semantics associated with the structural connections can be used in another way during insertion transactions. The insertion rules can actually be reversed to offer cooperative insertion of related data elements:

- If a tuple is inserted in an owner relation, it may be desirable to insert immediately the tuple's dependent instances in the owned relations (e.g., all the antibody-color pairs owned by a newly inserted antibody). The ruling parts of these items are partially inferred from the initial tuple ruling part.
- If a tuple is inserted in a referenced relation, it may be desirable to insert immediately the referring instances in the referencing relation (e.g., all the antibodies referring to a given new antigen).

Conclusion

We have presented the initial results of a research program aimed at addressing the fundamental limitations of the relational database model in a biomedical research context. The structural model captures semantic information about entities and connections; this information is then used in a variety of ways. The structural model plays an important role in relieving many burdens of traditional database applications. Flexibility, consistency and cooperation are much improved by new knowledge-oriented capabilities, such as dynamic constraints and manipulation of objects. Although much work is still needed to provide a fully workable implementation, we believe this approach shows great promise of providing more intelligent database application systems. Moreover, the generation of objects from the base data can be an efficient communication means when an interface between database and expert systems is needed.

Acknowledgments. Many ideas that are being developed and applied in this research have been initially proposed by the Knowledge Base Management Systems group at Stanford University. Computer facilities were provided by the SUMEX-AIM resource under NIH grant RR-00785 and by the Knowledge Base Management Systems under DARPA contract N39-84-C-0211. We thank Mark Frisse, Isabelle de Zegher-Geets and Lyn Dupre for their useful comments.

References

[1] W.A. Bonner, H.R. Hulett, R.G. Sweet, and L.A. Herzenberg. Fluorescence activated cell sorting. *Rev. Sci. Instru.*, 43:404–409, 1972.

[2] E.F. Codd. A relational model for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[3] S.J. Kaplan. *Cooperative responses from a portable natural language database query system.* PhD thesis, Computer Science Department, University of Pennsylvania, Philadelphia, PA, 1979.

[4] H.F. Korth and A. Silberschatz. *Database System Concepts.* Computer Science Series. McGraw Hill, New York, NY, 1986.

[5] G. Wiederhold. *Database Design.* Computer Science Series. McGraw Hill, New York, NY, 1983. 2d edition.

[6] G. Wiederhold. Knowledge bases. chapter 1, pages 110–122. Proceedings of the International Symposium on Fifth Generation and Super Computers, 1984.

[7] G. Wiederhold. Views, objects and databases. *Computer*, 19(12):37–44, December 1986.

[8] G. Wiederhold and R. ElMasri. The structural model for database design. In *Entity-relationship Approach to System Analysis and Design*, pages 237–257. North-Holland, 1980.

Connections

Gio Wiederhold

This material is an extract from the forthcoming book *Design Concepts and Implementation of Databases.* It expands the definitions of connections among relations as presented in *Database Design* [Wiederhold 83]. It defines the concept of *connections*, which together with *relational* concepts makes up the structural model used in this research.

INTRODUCTION

When systems are complex we deal with them through simplified abstractions. The abstraction used for databases is a *database model.* Models permit formalization of selected, important aspects of databases. Focusing our attention on a few aspects helps us deal with the complexity of real-world entities and their relationships.

Knowledge about relationships helps in the creation of new information. Relationships permit entities to be combined in new and imaginative ways. A transaction program selects and combines data about entities according to relationships defined in the database model. Relationships among the data form a *structure* which establishes the meaning of the data. Relationships are independent of the processing programs which exploit them. The objective of this chapter is to develop concepts for recognizing and describing this structure, so that guidance can be provided for the design of databases.

Our Model For readers with previous exposure to database models we note that the model used here, the *structural model*, is an extension of the *relational model.* The extensions permit more semantics to be expressed, while retaining the strengths of the relational model. As Codd points out in his RMT paper [Codd 79] the structural model captures semantics that are similar to his directions in extending the relational model, although the objectives were different.

The structural model can also be viewed as a formalization of the *entity-relationship* (ER) and *entity-category-relationship* (ECR) models.

We can use the structural model to guide relational as well as hierarchical or network implementations. Since the structural model focuses on the conceptual structure of the data, rather than on the representation within a DBMS, it is also valid when the database is stored in main memory or managed by the user on traditional files.

Our Example We use throughout examples from the world of movies. The relations we refer to are shown in Figure 1.

Hitch_movies RELATION: – US feature films by Alfred Hitchcock
film :⟩ title, director, studio, year, prc;

Pre50_movies RELATION: RELATION: – early US feature films by Hitchcock
film :⟩ title, director, studio, year, prc;

Actors RELATION: – Past and current actors
stage_name :⟩ birth_name, first_name, gender, born, died, typecast, origin;

Famous_actors RELATION: – Stars
stage_name :⟩ *birth_name** *first_name, gender, born, died, typecast, origin*, fee;

Some_awards RELATION: – Awards of actors in postwar bw Hitchcock films
stage_name, award :⟩ last, category, for ;

Casts RELATION: films, actor :⟩ role_type, fee_received; – Actors in specific movies

Producers RELATION: – Executives in the movie industry
name :⟩ birth_name, born, died, origin, studio;

Directors RELATION: – Managers for Actors and editors
ref_name :⟩ birth_name, born, died, origin;

Studios RELATION:
name :⟩ company, country, location, first_yr last_yr;

US_studios RELATION:
name :⟩ *company, location, first_yr last_yr*;

* Attributes in *italics* are inherited.

Figure 1 Sample Relations.

1 COMPONENTS OF THE STRUCTURAL MODEL

To design a database design we must model the components which will represent the information to be stored about entities and their relationships, The model must allow the manipulation of the conceptual building blocks for the database, since designing requires synthesis and analysis of many alternatives for representation. The effects of such manipulations must be predictable, so that the model must be formal.

The abstractions must be able to represent those database semantics which are needeed by the designer to develop specifications adequate for definition of the structure of the database. That structure defines the intension, the extension is filled in after implementation. Only when the structure is well understood can a suitable file design be chosen.

The model we use for the design task is the *structural model*. It is the tool used to bridge the gap between potential users, for whom ER concepts are comfortable, and implementors, who need much more specific information.

1-1 The Structural Model

The structural model is based on a well-known formal model, the relational model, which defines *relations*, sets of data, and the operations needed to manipulate them. The structural model augments the relational model with a formal definition of relationships between relations, called *connections*. Connections permit specifications of structural constraints between relations, so that the integrity of the database can be maintained.

The structurally relevant semantics of the database is encoded into the connections. We distinguish four types of connections:

Reference, to refer to entities which further describe a property. An example of a reference among the movie relations is `Producer` ≻—— `Works_for`—— `Studios`.

Ownership, to decribe properties of an entity in more detail. An example in our domain is `Famous_actors` —— `Got`——* `Some_awards`.

Subset, to link general classes to their subclasses. In our example we have a general class `Actors` —— `Is`——∋ `Leading_actors`.

Identity, to define copies and derivations of data. An example is counting stored objects, perhaps by some other property, and placing the results into a new relation: `COUNT(Movies BY producer)` —— `Becomes`——= `Producer.productivity`.

These connections will be defined in in Sec. 2.

We stated informally earlier that the meaning of entities is largely defined by their participation in relationships. We can now state that the semantics of relations are largely determined by their participation in connections. We can use their connections to classify relations into seven types, which have distinct structural features, as summarized in Sec. 3.

The similarity of the terms *relations* and *relationships* is unfortunate, but so well established that trying to change it would create further confusion now. It is well to recall that these terms have different origins, and that they are used at different levels of abstractions, as shown in Table 1.

Table 1 Entity-relationship (ER) and relational/structural model (SM) terminology.

Abstract layers		Implementation layers	
ER model	Structural model	Software	Hardware
Entity	Relation	File	Disk
Object	Tuple	Record	Block
Relationship	Connection	Pointer field	Address field
Rel.shp instance	Link	Pointer	Address
Property	Attribute	Data field	Word or Byte

The mappings between these layers are not one-to-one. Modeling of an entity may require multiple relations, and relations may require multiple files for their storage, or may be combined.

1-3 Connections as a Model of Relationships

Connections formalize relationships among entities. They always connect two relations, and the destination of a connection is always the ruling part of a relation.

For example, we can now define a connection between Hitch_movies and US_studios. Just as a relation has an extension consisting of a set of tuples, a connection will have a set of connection instances or *links*. A link exists when two tuples, each in relations for which a connection is defined, satisfy the rules for the connection. The most critical constraint is that the attribute values along the connection must match. Figure 2 shows the connection Made_in and links for selected Hitch_Movies made prior to 1950. We restrict here the Movies relation shown to the film code and studio, and the Studios to name and city.

$\boxed{\text{Pre50\_movies}}$ —— Made_in >—— $\boxed{\text{US\_Studios}}$

tuples		links	tuples	
Pre50 RELATION:		Made_in CONNECTION:	US_studios RELATION:	
film :)	studio;	studio >—— name;	name	:) city;
H32	Selznick	Selznick – Selznick		
H33	U.A.	U.A. – U.A.	U.A.	Hollywood
H34	RKO	RKO – RKO	MGM	Culver City
H35	RKO	RKO – RKO	Universal	N. Hollywood
H36	Universal	Universal – Universal	Warners	Burbank
H37	Universal	Universal – Universal	Columbia	Burbank
H38	Fox	Fox – Fox	Paramount	Los Angeles
H41	Selznick	Selznick – Selznick	RKO	Hollywood
H42	RKO	RKO – RKO	Fox	Century City
H43	Selznick	Selznick – Selznick	Selznick	Hollywood
H45	Transatlantic	Warners – Warners		
H48	Warners			

Figure 2 Some links for a connection.

Not all tuples are linked. Some studios (MGM, Columbia) were not used by Hitchcock during that time and one studio was across the Atlantic Ocean.

Missing links can lead to incomplete information. If we wanted to know all cities in which Hitch_movies were made "London" would not appear. Because in most data-processing situations we care about completeness we define constraints with our connections. Section 2 defines the constraints which are attached to our connection types. For the Made_in connection a reference connection is appropriate. It requires existence of the referent, here a Studios.name. Then we must either omit "H45 (Rope)" or enter "Transatlantic" into the relation US_studios. The latter change modifies the semantics of the relation, so we better rename it to just Studios. Figure 2 also shows the elements needed to specify a connection,

Connection_name CONNECTION:

Source_relation.attributes $\xrightarrow{?}$ Destination_relation.attributes [Extra-constraints]; 1

The symbol marked "?" denotes the type of the connection.

2 CONNECTIONS

Entities are the first ER super type formalized in the structural model. We focus here on the second super type to be formalized: relationships. We used relations to formalize entities, for relationships we use *connections*. We showed an example of a connection in Fig. 2 and announced that connections imply constraints.

Motivation for Connection Constraints We expect attributes in a tuple to have valid values. That expectation includes cases where the values are instances in other, *foreign*, relations. Connection constraints define these expectations.

The constraint appropriate to the `Made_in` connection between `Movies` and `Studios` is given by a *reference connection*. The reference connection provides a constraint to assure that no `Studio` information will be deleted that is referenced by `film` tuples in the `Movies` relation. For instance, we cannot delete the information about the `Studio` named `Fox` while we have records for the `Movie` "`H38 :) Lifeboat`". This constraint enforces the integrity of the entire object identified by `H38`.

With this motivation we proceed to further detail about connections.

Relating Entities by Attributes Connections are instantiated via attribute values that match. One requirement is then that they belong to the same domain. So that the comparison can work in practice the domains must also be represented by the same type.

A neccessary condition for a connection is that the domains match. But this is not a sufficient condition; not all attributes having matching domains should be connected. The essence of a connection is that there is a well specified relationship, which is expected to remain valid over time. That condition is similar to that leading to relations: we group into a relation all attributes that we expect to be dependent on the same ruling part.

$$\begin{tikzcd}
\texttt{Movies} \arrow[r, -*] & \texttt{Producers} \\
\texttt{Movies} \arrow[r, -*] & \texttt{Studios}
\end{tikzcd}$$

Figure 3 Related information.

GENERAL SYNTAX AND SEMANTICS OF CONNECTIONS A connection formalizes a relationship that exists between two relations. The connections may derive from relationships between the entities of the original Entity-Relationship model or may have been created while entities where transformed to relations in BCNF. Whenever an entity is decomposed into multiple relations connections are created to retain the original information. It does not matter here how the connection was established. If one starts originally with a model which consists of BCNF relations the connections can be specied directly on those relations and their attributes.

A connection defines a semantic relationship among two relations. Links exist when values for the attributes specified by the connection match. In Fig. 2 the `Made_in` connection specified the attributes `Movie.studio` and `Studios.name`. Each match leads to a link instance.

It is immaterial to the model if the link defined by some connection is implemented within the database. The link exists conceptually as soon as attributes for the connection match.

The connection links do not contain any new information:

A link can be computed from the definition of a connection and the actual tuples of the connected relations.

The format of a connection was given in Eq. 1. It is composed of five elements:

1 The Connection_name used for identification and disambiguation when relations are connected in multiple ways

2 The type defines the logical constraints. We recognize four types, as shown in Table 2.

3 The source relation and attribute(s)

4 The destination relation and attribute(s) match in number and domain the source attributes. They are also always the ruling part.

5 Any additional quantitative constraints. Those will be presented when needed.

Table 2 Connection types in the structural model.

Connection Type	Symbol	Direction
Ownership	—*	from single owner to multiple owned tuples
Reference	≻—	from multiple primary to single foreign tuples
Subset	—∋	from single general to single subset tuples
Identity	—■	from single source to single derived tuples

We now review these four connection types.

2-1 Ownership Connections

An *ownership connection* provides links for multiple included dependent attributes. The general form is

Name CONNECTION: Owner.ruling_part —* Member.ruling_attributes; 2

Figure 4 shows multiple `Roles_played` by some `actors` of the `Casting` relation. Any new entry in `Roles_played` needs a corresponding `actor`. If an `actor` is removed from `Casting` all corresponding `Roles_played` information for that `actor` is no longer relevant and should be removed as well.

The ownership connections defines structures to serve inclusion dependencies as defined in relational dependency theory. The operational semantics of owned tuples are exactly the same as those of dependent attributes: they go away when the owning tuple, which after all defines the object, disappears. Similarily, we can only insert owned tuples if we have an owner tuple.

Figure 4 shows multiple `Roles_played` subsequently by those `actors` of the `Casting` relation who were in `Hitchcock`'s last movie, `Family Plot`. Any new entry in `Roles_played` needs a corresponding `actor`. If an `actor` is removed from `Casting` all corresponding `Roles_played` information for that `actor` is no longer relevant and should be removed as well.

$$\begin{tikzcd}\boxed{\texttt{Casting}} \arrow[r, dash, "\texttt{Performs\_as}"] & \boxed{\texttt{Roles\_played}}\end{tikzcd}$$

Owner relation | Nest relation

tuples | links | tuples

Casting RELATION:		Roles_played RELATION:			
actor :)	...;	performer,	film :)	role,	year;
Karen Black	...	Karen Black	DCs2	haunted tenant	1976
		Karen Black	PHy6	pilot's wife	1978
		Karen Black	JRB35	inventor's wife	1979
		Karen Black	GKc4	fashionable	1981
		Karen Black	RA121	cynical Dean fan	1982
		Karen Black	HJg5	shrewd	1983
Bruce Dern	...	Bruce Dern	JFr19	terrorist	1975
		Bruce Dern	MRt6	beauty judge	1975
		Bruce Dern	HAs6	doper	1978
		Bruce Dern	WH12	get-away driver	1978
		Bruce Dern	JTr1	psychopath	1980
		Bruce Dern	SWu1	mad tattooist	1981
		Bruce Dern	JsM1	*	1982
William Devane	...	William Devane	JS19	Nazi?	1976
		William Devane	JS110	GI in England	1979
		William Devane	JDs1	homicidal alien	1979
		William Devane	JS111	promotor	1981
		William Devane	LyL1	craggy	1983
Barbara Harris	...	Barbara Harris	GNe5	unhappy mother	1977
		Barbara Harris	JS5	senator's wife	1979
Ed Lauter	...				
Cathleen Nesbitt	..	Cathleen Nesbitt	FZ20	old woman	1977

Figure 4 Ownership.

We refer to each group of owned tuples, as the six roles shown for Karen Black, as a *nest*. Some nests (Ed Lauter's) are empty. The owned relation (Roles_played) is the *nest relation*. The owning relation has one tuple for each owner object, the owned relation as many as needed for all the roles listed. Information pertaining two one set of objects is now distributed over two BCNF relations.

Cardinality of Ownership There maybe no, one, or many owned tuples in a nest. The cardinality of ownership is $1 \longrightarrow n$. Extra constraints can be applied to n. If Performs_as requires at least one tuple per nest, we must delete "Ed Lauter" or find a role for him. Such constraints complicate update transactions; insertions on both the owning and the nest relation must be executed as an *atomic transaction*.

Ownership connections can also model more complex situations. The Movies_info relation owns the Casts relation, and the tuples of the Casts relation are in turn connected to the Actors relation. Relations, as Casts, that represent relationships between independent entities are called *associations*. An association relation is neccessary to represent relationships which have an $n : m$ cardinality. Two connections will be needed as sketched in Fig. 5.

Two relations connect to the **Casts** relation:

```
  Movies_info  RELATION:   ___n
                           ___*Casts RELATION:films, actor :⟩ role;
  Famous_actors RELATION:  ___<
                              m
```

The **Casts RELATION** needs two attributes in its ruling part, one for each connection:

```
Performers CONNECTION:Movies_info.film——* Casts.films;
Played_in CONNECTION:Famous_actors.name ——< Casts.actor;
```

Figure 5 Association.

In our example `Casts` is a nest of of m actors in terms of `Famous_actors`, and of `Movies_info` in terms of n `films`. Each connection remains $1:m$ or $1:n$. The connections in an association need not be of *ownership* type, but could also be *reference* connections, as will be seen in Fig. 7.

STRUCTURE OF OWNERSHIP Ownership requires a match of the connecting attributes

$$\begin{array}{c} Owner.a_i = Owned.a_i \text{ for } i = 1,\ldots,j \\ k > j, \quad j = \| Owner.rp \|, \quad k = \| Owned.rp \| \end{array} \qquad 3$$

The owned relation will have one or more additional ruling part attributes $a_{j+1},\ldots$ to identify individual members r $0,\ldots,s$ of the owned sets. The size s of these owned sets ranges from $0 \rightarrow \#D(A_{j+1})\cdot\ldots$; $\#D$ is the cardinality of a domain.We can now state the rules for ownership connections:

Ownership rules

Semantics: Owned tuples represent sets of object attributes.

Structure: The ruling part of the nest relation is the catenation of the ruling part of the owner relation and an attribute to distinguish individual tuples in the nests.

Operation: A new tuple can be inserted into the nest relation only if there is a matching owner tuple in the owning relation. Deletion of an owner tuple implies deletion of its nest.

We see that with the definition of an ownership connection we also prescribe constraints, and rules how the constraint is to be maintained. These rules define the semantics of the model, and once implemented define the way of doing business.

2-2 Reference Connections

We encountered already relations that referenced other relations. The *reference connection* models this relationship type. The *primary relation* is the source relation of a reference connection the and *foreign relation* is the destination relation. The general form is

Name CONNECTION:Primary.attributes >—— Foreign.ruling_attributes; 4

The primary relation defines the object of interest, say some `Movies_info.film`, and an attribute, say `Movies.studio`. The foreign relation, `Studios`, provides additional information for the `studio` where the `film` was mode.

The foreign object is typically a higher level object or an abstraction. A single foreign tuple can hold information that otherwise would be held redundantly in many source tuples. Many `films` are made in the same `studio`, the address appears only once in `Studios` relation.

To assure that the referenced information is always available, the reference connection embodies the constraint that the foreign tuple must exist. Insertion in the primary relation is constrained, and deletion of a tuple in the foreign is not permitted while it is in use.

Multiple levels of references can be made, and multiple connections can be made to and from relations; however, circular connections will lead to update problems. Figure 6 shows connections for a valid network.

Given the relations as shown in Fig. 1 we can define

```
Shot_in CONNECTION: Movies.studio ⟩—— Studios.name;
Located_in CONNECTION: Studios.country ⟩—— Countries.country;
Resides_in CONNECTION: Casting.location ⟩—— Countries.country;
Is_from CONNECTION: Producer.origin ⟩—— Countries.country;
Who_is CONNECTION: Products.producer_name ⟩—— Producers.name;
Made_by CONNECTION: Movies.director ⟩—— Directors.ref_name;
```

Figure 6 Reference Connections.

In all these cases the reference connection will assure that destination tuples are available to further describe the source objects. The foreign relations are maintained distinctly but cooperatively with the primary relations.

The existence of a tuple in a foreign relation implies only weakly that there exists a tuple with the corresponding value in the primary relation. If there is no reference then the tuple in the foreign relation may be deleted. For instance, in the `Studios` foreign relation we have listed "`Columbia`", but this studio was not used by Hitchcock. and could be omitted from a database which focuses on Hitchcock. A foreign relation can be the source of other connections, but these will not constrain deletion.

The database model forms a network of connections of all types. As indicated earlier, *associations* can also be formed with reference connections.

Given the three relations:

```
Movies_info RELATION: film :⟩ title, year, director, studio, ...;
Producers   RELATION: name :⟩ birth_name, born, died, origin;
Products    RELATION: films, producer_name :⟩ position;
```

and the connection `Who_is` from Fig. 6

```
Who_is CONNECTION: Producer.name ——⟨ Products.producer;
```

we can connect `Products` also to `Movies_info`, making it into an association relation by adding

```
Produced_by CONNECTION: Movies_info.film ——* Products.film;
```

We can only keep `Products` tuples for those `films` which appear in `Movies_info`.

Figure 7 *Association with a reference connection.*

STRUCTURE OF REFERENCES A tuple of the foreign entity relation provides a more detailed description for an object in the primary relation. Any attribute of a relation can reference the ruling part of a foreign relation, if its domain matches. Large, unconstrained domains do not lend themselves for referencing. For instance, *real numbers* are quite unsuitable in practice.

The destination of a reference connection is always the ruling part of the foreign relation. Most foreign relations have ruling parts composed of only one attribute; we are used to reference abstractions by simple names. We expect to find tuples in a foreign relation to match any value assumed by the primary attribute; absence of a foreign tuple constitutes an error. A foreign relation hence defines the domain for the primary attribute.

Cardinality of Reference Connections We expect to have many fewer foreign objects than source objects. One entity or abstraction will be named by many tuples. The cardinality is hence $n \succ\!\!-\!\!-\, 1$, just the opposite of the ownership connection. That reversal made us change the direction of the reference arrow in Fig. 7.

We can now define the structural rules.

Reference Rules

Semantics: References indicate further information about the objects, typically descriptions of abstract classes.

Structure: The ruling part of a foreign relation matches the primary attribute of the primary or referencing relation.

Operation: Tuples in foreign relations may not be removed while any reference exists. The removal of primary tuples from the primary relation does not imply removal of the corresponding foreign tuple.

If the rules defined by a reference connection are not obeyed, vital information may not be available during processing, causing transactions to fail.

Self-reference It is possible to define a reference connection that has the same relation as source and destination. The primary attribute will differ from the foreign attribute. Figure 8 shows such an extension.

We wish to show costars of `film actors` in our model. The `Casts` relation is appropriate since it has the proper ruling parts. We add an attribute `costar` and a reference connection.

```
Casts RELATION:actor, film :) part, role, award, costar;
Starred_with CONNECTION: Casts.(costar, film) >—— Casts.(actor, film);
```

Only some `actors` will have a value in `costar`. Others will have a `null` entry. We deal with the domain definition for that case by defining *artificial* entries in domains and tuples, or by defining subsets, as shown in Sec. 2-3.

This example also shows a case where the reference connection requires two attributes.

Figure 8 Self-referencing connection.

This construct introduces a cycle on the connection level, but should not cycle on the instance level. We will not permit an `actor` to use the `Starred_with` connection to refer to him- or her-self as `costar`, no matter what their personal opinions are.

2-3 Subset Connections

We often deal with entities that are quite similar, but for which different information is to be collected. In our example we have many kinds of `People` that work in the movies, but the information we should keep on them varies. In Fig. 1 We had `Famous_actors`, `Producers`, and `Directors`. There is also some overlap between them, so similar attributes are redundant if we keep these overlapping subsets in wholly distinct relations. Defining a general relation `People` permits non-redundant modeling of shared attributes.

A *subset connection* connects a generalization to a subset. Such relations will have matching ruling parts, but will differ in dependent attributes. The general form is

Name CONNECTION: General.ruling_part——∋ Subset.ruling_part [Membership-constraint] ; 5

In Fig. 9 we show the *generalization*, `People`, and the subset connections to previously defined relations. To model this generalization we have to make changes in the way identifying names are handled for the various subsets. If we had all their `Social-Security-Numbers` (`SSN`) problems associated with names could have been avoided.

```
People RELATION:     -- general relation.
        last_name, first_name :⟩ sex, born, died, origin, address,
                is_actor, is_producer, is_director;

Is_actor CONNECTION:     -- connection to actors subset
        People.(last_name, first_name) ——∋
            Famous_actors.(birth_name, first_name);

Famous_actors RELATION:-- the real name is the ruling part
        birth_name, first_name :⟩ stage_name, agent, typecast, awards;

Is_producer CONNECTION:     -- connection to producers subset

        People (last_name, first_name) ——∋
            Producers.(last_name, first_name)
Producers RELATION:     -- name now split into two parts
        last_name, first_name :⟩ current_studio;

Is_director CONNECTION:     -- connection to directors subset
        People.(last_name, first_name) ——∋
            Director.(ref_name, first_name);

Directors RELATION:     -- reference name is unique
        ref_name :⟩ first_name, wwH, cur_fee, cur_perctg, awards;
```

Figure 9 Subset connections.

Subsets may have further subsets. We encountered already subsets of `Actors` in Fig. 2.

The use of subsets permits to specify a variety of dependent parts, all fc the same ruling part. Tuples in a subset are incomplete without the information provided in the general tuple. We have constraints that are similar to the ownership connection, but here classes are expanded, rather than attributes.

Subset rules

Semantics: Subsets identify specializations of the objects.

Structure: The ruling part of a subrelation matches the ruling part of its connected general relation. A general tuple may have no or one tuple in any connected subset relation.

Operation: Every subset tuple connects to one general tuple. Deletion of the general tuple forces deletion of the linked subtuple.

The *generalization* of subclasses to larger classes is essential to data-processing; the recognition of individual subclasses and the collection of detailed data is important to give depth to the database and is often needed in specific views. Subsets also have a role in distribution of data.

2-4 The Identity Connection

The *identity connection* recognizes replicated information. Replication of information occurs in databases when

1 derived results are stored in the database
2 copies of information are stored on distributed computers

The identity connection connects multiple attributes in the source relation to a tuple in the destination relation. The values should be identical. The ruling part attributes must always be included so that uniqueness of the destination tuples is assured. The general form is then

```
Name CONNECTION: Source.(ruling_part, dependent_attributes)——≡
                       Destination.(ruling_part, dependent_attributes)        6
                                   | constraint_relaxation;
```

The constraint_relaxation plays an important role. Without it the identity constraint is quite constricting.

The principle of relaxing the constraint is simply that *eventually*, i.e., after some time has passed, all destination tuples exist for all source tuples. The term "eventually" indicates that the constraint can be satisfied with *some delay*.

DELAYS IN ESTABLISHING IDENTITY The computation of derived data and the distribution of results require time; hence maintenance of an identity connection requires time. In order to build workable database systems, we must be able to specify what kind of time-delay is permissible, and for each kind specify the amount of delay acceptable. At the same time we must be able to guarantee that *sometime* the identity will be satisfied, so that then computations using the destination values will be predictable. An example of an identity connection, where a delay of a day is tolerable, is given in Fig. 10.

Note that having specifications does not solve the implementation problems. If, during execution, the identity connection is *not* set within the time delay specified, then all subsequent transactions must be held until the setting of the identity connection is completed. However, *without* specifications we can never determine if the systems is behaving acceptably.

If further changes in the source relation occur before earlier ones are completed, then setting of the earlier ones must be completed before the later ones are set. How these conditions are to be achieved is not of concern in the model. Most of the implementation problems arise in distributed systems, we present those in the textbook.

We recognize three types of temporal constraints and designate each by a symbol in the `relaxation` field of the identity connection. Examples for the three types of constraints are shown in Fig. 11.

! Prior to execution of a transaction named `!Trans_use` the destination tuples in the identity connection have to be set. If no transaction name is specified, then `!ANY` is implied, so that the identity connection must be set prior to *any* use of the destination tuples.

@ At a specific time, `@hh:mm:ss`, the identity connection must have been set. Any transaction which starts subsequently will obtain up-to-date destination tuples.

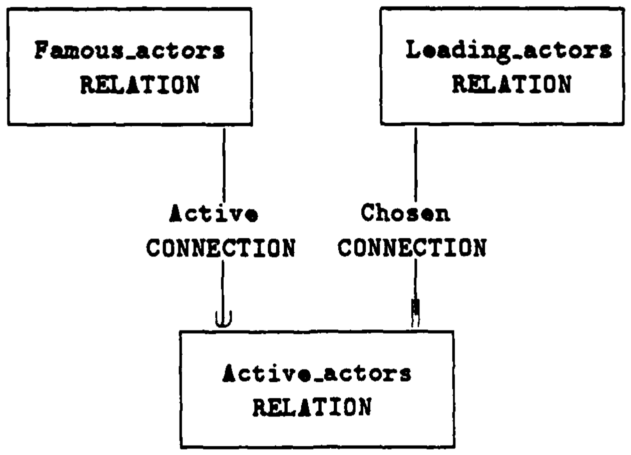
Δ After a specific interval, Δn `unit`, the identity connection must have been set. The `unit` may range from days to seconds. If the value of the interval $n = 0$, then the effect is identical to `!ANY`.

If no relaxation is specified, then the interpretation of the identity constraint is equivalent to `!ANY` as well.

Consider a subrelation of `Famous_actors` —— `active`——∋ `Active_actors`.

When, for a new movie, in the `Movie_making VIEW`, `Leading_actors` are chosen, those actor should also be placed into the relation `Active_actors` because we can expect that soon inquiries will be made about them.

But some delay may be needed to acquire all the data needed for the corresponding `Famous_actors` general relation. The model is graphically:



We allow two days for the update and specify:

```
Leading_actors.(name, born, role) —— Active——∗
                Active_actors.(stage_name, born, typelast)|Δ2d;
```

Time constraints tend to be relaxed when identity connections relate distinct views.

Figure 10 Use of an identity connection to link views.

Many financial transactions require that input data are up-to-date:

```
    Studio.employee.(name,hours_worked)——≡
              Payroll.(name,hours_to_be_payed) | !pay_transaction ;
```

Before schedules can be displayed in the morning all committments must be accounted for:

```
    Actors_schedules.(stage_name, title, character) ——≡
              Studio_display.(name,project,as) | @8:00:00;
```

Changes to the schedule for today should be rapidly displayed, five minutes is the limit:

```
    Change_list.(stage_name, title, character) | date = today ——≡
              Studio_display.(name,project,as) | Δ5m;
```

Here a simple computation, selecting only changes applicable for today, is specified as well.

Figure 11 Three types of delay specifications.

Fig. 11 also shows an example of a simple computation. In general, the replicated information at the destination need not be a literal copy of the data as the source. Especially if the destination is a central node in a network only selected and aggregated information should be transmitted from the sources. The source nodes can keep much more detailed information themselves, as shown in Fig. 12.

Cinemas transmit weekly the gross receipts to the corporate office:

```
Cinema_receipts.(id, SUM(receipts | day = today-7 ..  today-1))——≡
                  Total_receipts.(theater, amount) | @Monday 8:00:00;
```

Figure 12 Delayed aggregation.

Replacing Programmming with Declarations The identity connections we show here specify simple computations. Networks of such connections can describe complex computations, which are now scheduled and managed wholly by the system, rather than by explicit transaction programs.

Identity rules

Semantics: Identities define derivations of data.

Structure: Derived tuples match the source relation. The attributes of a identity relation match the specified attributes and expressions of its connected source relation. Multiple copies require multiple connection definitions.

Operation: Eventually corresponding identity tuples must be created and updated. Deletion of the source tuple forces eventual deletion of the derived tuple. The permissible delays are specified; timestamps will be needed for validation.

We are starting here a new approach to *declarative programming*.

2-5 Naming of Connections

It can be useful to attach names to connections which identify their *role.* There are typically two candidate role names for a connection, since we use distinct terms for each direction that can be traversed. Since connections have a defined semantic direction we choose in the model the name appropriate for that direction. For instance Made_in connects Movies.studio $\succ\!\!-\!\!-$ Studios.name.

The simpler rolename Made goes from Studios to Movies and may be used when querying the database.

Role names are not limited to connections among relations, but may also be used for simple attributes - which have a 1:1 dependency on their ruling parts.

Role names are useful in enhancing the semantics of the database model, and are easily related to the queries that the database will respond to.

Figure 13 shows some examples of role names.

Connection name	source = destination		type
	role name →	← role name	
Made_by	Movies.director=Directors.ref_name		ref
	Made_by	Made	
Shot_in	Movies.studio=Studios.name		ref
	Shot_in	Shot	
Located_in	Studio.country=Countries.country		ref
	Located_in	Has	
Cast	Movies.film=Casts.films		own
	Cast_is	Performed	
Played_in	Famous_actors.name=Casts.actor		own
	Played_in	Has_actor	
Star_cast	Movies.film=Stars.films		own
	Star_cast_is	Stars	
Starred_in	Famous_actors.name=Stars.actor		own
	Starred_in	Has_star	
Prod_By	Movies.film=Products.films		own
	Produced_By	Produced	
Works_in	Casting.location=Countries.country		ref
	Works_in	Has_available	

Figure 13 Role names.

The final entry shows that there is a close relationship between an attribute name within a relation and the role name we assign to a connection.

We notice that more than one relationship can exist between entities. In our Fig. 13 we distinguish the Cast and the Stars of Famous_actors—— Movies. To support that structure we need distinct association relations and corresponding connections to identify the members of the two distinct relationships.

3 RELATION TYPES AS A DUAL OF CONNECTIONS

A structural model consists of relations and connections. The relations are constrained by their participation in connections. We can hence also define *relation types*, each characterized by their connections. Sometimes semantics are more obvious when they are attached to relations, since the relations deal with well defined object instances.

We can define seven relation types:

Entity relations
Foreign entity relations
Nest relations
Associative relations
Lexicons
Subrelations
Derived relations

The two concepts of connection types and relation types are duals of each other; a relation type defines the connections it participates in and a connection type defines the relation types it connects. Both concepts are useful, however, in dealing with models and their implementation, and both are found instantiated in database systems. Constraints between relations are more easily described using connections. Data elements and their semantics are better described using relations. A data-processing transaction is defined on both.

Use of graphic symbols is extremely useful in presenting models for review to the participants in a database design process. The design process is described in *Database design* [Wiederhold 83]. It justifies the defintion of distinct view models, using these primitives, and a subsequent integration. When the view models are brought together conflicts can be recognized and resolved. Changes to the view models will be reflected eventually in *database submodels*, the external specifications given to the user groups.

The interaction with the user groups is through diagrams of relations, connections, and a data dictionary which lists the details. Tables or lists do not effectively illustrate the dependencies and hence the semantics of the view and database models.

Table 3 reviews the sematics of the relation types described, using the rules defined with the connections.

The construction of each of these relation types obeys certain rules so that we can also describe these types using a Backus-Naur notation. Since such a description is essentially syntactic, some semantic constraints are added to the rules in quotes.

It is possible to create from these basic semantic types other forms of relations which satisfy special conditions. A synthesis of published material indicates that these five types, plus the concepts of subrelations, cover the semantic possibilities controlling the structure of databases in an economic and conceptually convenient manner. Combinations of ownership and reference connections, for instance, lead to four types of $n \cdot m$ relationships among two relations. Each of the four cases has distinctive semantics.

If we ha·e constructed and normalized a view model in an economic fashion, that is, have kept no redundant attributes, then only attributes which implement reference, ownership, or subset linkages and explicit derivations will be duplicated. Further minimization of redundancy can be achieved by minimizing the number of relations. The extent to which this is desirable depends on the further use of the view model. If the view model is to become a basis for extended applications using the database it may by transformed to a *database submodel.*

Table 3 Connection semantics applied to relation types.

A *A primary entity relation:*

1 Not referenced within the view model.
2 The ruling part defines the entity.
3 The existence of tuples is determined externally, i.e., by the user.

B *A foreign entity relation:*

1 Referenced from within the view model.
2 The ruling part defines the entity and establishes domains for primary attributes. A ruling part typically has a single attribute.
3 Existence of tuples is determined externally, but deletion is constrained by existing references.

C *A nest relation:*

1 Each tuple must have an owner tuple within the view model.
2 The ruling part defines one specific owner and the tuple within the owned set.
3 An owned nest can have zero or more tuples.

D *An associative relation of order n:*

1 Each tuple has n owners within the view model.
2 The ruling part defines each of n specific owners only.
3 One combination of owners can have 0 or 1 tuple in the association.

E *A lexicon:*

1 Referenced within the view model.
2 Either part can be the ruling part.
3 The existence of tuples is determined externally but deletion is constrained by existing references.
4 Its existence is transparent to the model.

F *Subrelations:*

1 Referenced from any general relation.
2 The ruling part matches the ruling part of the general relation.
3 The dependent part contains attributes which do not apply to non-matching tuples within the general relation.
4 Insertion requires existence of a matching tuple in the general relation.

G *Derived relations:*

1 Generated from any source relation or equivalent expression.
2 The ruling part matches the ruling part of the source relation.
3 The dependent part is the same and eventually contains the same values.
4 Insertion requires existence of a matching tuple in the source relation.

4 SUMMARY

We have defined the concepts of relation types and connection types for the structural model. With each type we associate

1 Semantics
2 Structure
3 Constraints
4 Graphic symbols The entire structure can be syntactically defined as shown in Table 4.

Table 4 Backus-Naur description of relation types.

DEFINITIONS:	
⟨value attribute⟩	::= *"a collection of data elements of one domain"*
⟨reference attribute⟩	::= *"a collection of references to one relation"*
⟨value attributes⟩	::=⟨value attribute⟩ | ⟨value attributes⟩,⟨value attribute⟩
⟨attributes⟩	::=⟨value attributes⟩ | ⟨reference attribute⟩
⟨value key⟩	::=⟨value attributes⟩ *"unique objects"*
⟨reference key⟩	::=⟨reference attribute⟩ *"unique objects"*
⟨primary entity key⟩	::=⟨value key⟩ *"not referenced"*
⟨foreign entity key⟩	::=⟨reference attribute⟩ *"foreign"*
⟨nest key⟩	::=⟨attributes⟩ *"unique within each nest"*
⟨dependent part⟩	::= null | ⟨attributes⟩
RULING PARTS:	
⟨entity ruling part⟩	::=⟨primary entity key⟩
⟨foreign ruling par⟩	::=⟨foreign entity key⟩
⟨qualification⟩	::=⟨general ruling part⟩ *"in owner relation"*
⟨nest ruling part⟩	::=⟨qualification⟩,⟨nest key⟩
⟨assoc. ruling part⟩	::=⟨entity ruling part⟩,⟨entity ruling part⟩ | ⟨assoc. ruling part⟩,⟨entity ruling part⟩
⟨general ruling part⟩	::=⟨entity ruling part⟩ | ⟨foreign ruling part⟩ | ⟨nest ruling part⟩ | ⟨assoc. ruling part⟩
RELATIONS:	
⟨entity⟩	::=⟨entity ruling part⟩ :⟩ ⟨dependent part⟩
⟨foreign entity⟩	::=⟨foreign ruling part⟩ :⟩ ⟨dependent part⟩
⟨nest⟩	::=⟨nest ruling part⟩ :⟩ ⟨dependent part⟩
⟨association⟩	::=⟨assoc. ruling part⟩ :⟩ ⟨dependent part⟩
⟨lexicon⟩	::=⟨key⟩⟨:⟩ ⟨key⟩
⟨subrelation⟩	::=⟨general ruling part⟩ :⟩ ⟨dependent part⟩

The material in this chapter includes results of researchers and practitioners. While the more practical aspects of current work have been selected, many of the approaches in the literature have not been widely verified through application in the implementation of large systems. The structural model, and its relatives and predecessors, has been used in a variety of situations, but any application has to be combined with *common sense* and *insight*.

References on Objects in Databases

This list consists of BIBLIO.DBD entries on 'Object's up to Jul1988, filtered based on Surajit Chaudhuri's recommendations, Oct1988. (c) Gio Wiederhold

Abiteboul,S. and Hull,R.B.: @ INRIA, USC "Restructuring of Semantic Database Objects and Office Forms"; ICDT 86, Rome, Sep.1986.

Abrial,Jean-Raymond: @ IMAG, Grenoble "Data Semantics"; Klimbie74 and Koffeman (IFIP TC-2), Mar.1974.

Data Base model and concepts: binary relationships.
It seems amazing that a lot of concepts in modelling which are only now being implemented (or even discussed) were in this paper back in 1973. A must for anyone interested in data models. Only pages 1-17 are relevant to database modeling. The rest of the paper deals manily with specifying procedures in databases. To quote the author ... this paper belongs to the 'structured programming' area more than to the 'Database' literature... . The first part brings out the need for independence of querying from the mechanism used to get the answer - semantic data independence. Notions of binary relations and objects and access functions. The idea of constraints associated with operations on objects seems a lot like the ideas of Abstract Data Types. — Maier

Afsarmanesh,H., Knapp,D., McLeod,D., and Parker,A.: "An Extensible Object-Oriented Approach to Databases for VLSI/CAD"; Univ.of Southern Cal., CSD, TR-85-330, Apr.1985.

Afsarmanesh,Hamidah: "3DIS: An Extensible Object-Oriented Information Model"; USC CSD, TR-85-21, Oct.1985.

Ahlsen,M., Bjornerstedt,A., and Hulten,C.: @ CS Dept., Chalmers Univ. of Tech. and Univ. of Goteborg, Goteborg "OPAL: An Object-Based System for Application Development"; IEEE DB Eng.Bull., Vol.8 No.4, Dec.1985. Also; Syslab Report No.38, Oct.1985.

Albano,A., Giannotti,F., Orsini,R. and Pedreschi,D.: @ Universita di Pisa, Italy "Data Types and Objects in Conceptual Modeling"; EDSW, Vol.2, Oct.1984.

Andrews,T. and Harris,C.: "Combining Language and Database Advances in an Object-Oriented Development Environment"; OOPSLA 87, 1987.

A brief overview of VBASE. The schema definition language is block structured support for one-to-one, one-to-many, and many-to-many relationships. Provides an inverse mechanism, the ability to cluster objects and support for triggers and multiple inheritance.

Andrews.T.: @ Ontologic "Vbase Integrated Object System: Functional Specification"; Ontologic, Nov.1986.

Andrews.T.: @ Ontologic "Vbase Integrated Object System: Technical Overview"; Ontologic, Mar.1987.

Atzeni,P., Ausiello,G., Batini,C. and Moscarini,M.: @ Universita' di Roma, Istituto di Automatica "Conceptual Relations Among Data Base Schemata"; Rpt. 80-32, Dec.1980.

Such conceptual relations compare the conceptual content of schemata according to the following criteria: — ability to represent objects and relations among objects of the real world — ability to 'filter' incorrect representations of the real world — ability to represent transformations of the real world — ability to 'filter' incorrect transactions which represent transformations of the real world. In section 2, two different types of conceptual inclusion and equivalence between schemata are defined in our approach in terms of a query language Q and an integrity constraints language IC.

Ballou,N., Chou,H.T., Garza,J.F., Kim,W., Petrie,C., Russinoff,D., Steiner,D., and Woelk, D.: @ MCC, Austin,Texas "Coupling on Expert System Shell with an Object Oriented Database System"; rcvd. Nov.1987.
Features hypothetical transactions, schema changes, truth maintenance, versions.

Bancilhon,F., Kim,W., and Korth,H.F.: "A Model of CAD Transactions"; VLDB 11, Aug.1985, Stockholm.
View Objects

Bancilhon,F. and Khoshafian,S.: @ MCC "A Calculus for Complex Objects"; ACM PODS, Cambridge MA, Mar.1986.

Banerjee,J., Kim,W., Kim,K-J., and Korth,H.F.: @ MCC "Semantics and Implementation of Schema Evolution in Object-Oriented Databases"; ACM-SIGMOD 87, May.1987.
single class hierarchy, multiple inheritance

Banerjee,J., Kim,W., and Kim,K-C.: "Queries in Object-Oriented Databases"; IEEE CS Data Engineering Conf. 4, Feb.1988, LosAngeles.

Baroody,A.J.,jr. and DeWitt,D.J.: @ Xerox Webster Research Center; Univ. Wisconsin-Madison "An Object-Oriented Approach to Database System Implementation"; ACM TODS, Vol.6 No.4, Dec.1981, pp.576–601.

Barsalou,T., and Wiederhold,G.: @ Stanford Univ. "An Object-Based Interface to a Relational Database System"; Medical Computer Science, Stanford Univ., 1986.

Barsalou,T. and Wiederhold,G.: "Applying a Semantic Model to an Immunology Database"; SCAMC 11, Stead(ed), IEEE CS Press, Nov.1987, pp.871–877.

Barsalou,Thierry: "An Object-based Architecture for Biomedical Expert Database Systems"; SCAMC 12, IEEE CS Press, Washington DC, November 1988.

Barsalou,T. and Wiederhold,G.: "Knowledge-based Mapping of Relations into Objects"; to be published in The International Journal of Artificial Intelligence in Engineering, Computational Mechanics Publ., UK, 1988.

Batory,D.S. and Kim,W.: "Modeling Concepts for VLSI CAD Objects"; presented at ACM-SIGMOD 1985. Also; ACM TODS, Vol.10 No.5, Sep.1985, pp.322–346.

Batory,D.S. and Kim,W.: "Support for Versions of VLSI CAD Objects"; TechRep TR-85-18, Univ. of Texas at Austin, Sep.1985.
Versions.

Beckstein,C., Goerz,G., and Tielemann,M.: @ Univ.of Erlangen-Nuernberg "FORK: a System for Object-and Rule-Oriented Programming"; Berichte des German Chapter of the ACM, vol.28, B.G.Teubner, Stuttgart 1987.
Portable extension of Flavors, with class-owned variables, types, structural connections, set variables, alternate ways to control inheritance, views (perspectives). Further work on truth-maintenance and temporal forms.

Beech,David: "Groundwork for an Object Database Model"; rcvd. Feb.1987, HP, Palo Alto. Also; to be published in proc., MIT press 1987.

Defines a formal model with data type constructors, sets, lists, and other goodies.—jeff

Beech,D. and Mahbod,B.: "Generalized Version Control in an Object-Oriented Database"; IEEE CS Data Engineering Conf. 4, Feb.1988, LosAngeles.

Bever,M. and Lorie,R.A.: "An Enhanced Referential Integrity Schema Supporting Complext Objects"; IBM Res. Report-RJ 5585, 1987.

Bobrow,D.G.: "COMMONLOOPS: Merging Common Lisp and Object-Oriented Programming"; Xerox PARC, ISL-85-8, Dec.1985.

Bouzegjoub,M. and Gardarin.G.: @ INRIA "The Design of an Expert System for Database Design"; 'New Applications of Databases' Gardarin and Gelenbe(eds), Academic 1984, pp.203–223.

SECSI uses simple French natural language (7 verbs) dialogue to obtain specs stored in frames. An inference engine creates relation schemas and constraints. From semantic network description of the application a relational schema is generated. Very simple Syntax of statements is used. Internal links are: INFO(object,categ), CONNEC(Obj1,obj2), INTG-CONST(contraint ...) and COMMENT(). There are also 4 classes of RULES: SIMPLIFICATION (for finding 1NF), DEPENDENCY, NORMALIZATION, OPTIMIZATION.

Brodie,M.L. and Ridjanovic,D.: @ CCA "Fundamental Concepts for Semantic Modelling of Objects"; CCA, report Oct.1984.

primitives and constructors used to design object and operational hierarchies.

Bronnenberg,W.J.H.J., Nijman,L., Odijk,E.A.M., and vanTwist,R.A.H.: @ Philips Res. Laboratories, Eindhoven, The Netherlands "DOOM: A Decentralized Object-Oriented Machine"; IEEE MICRO, Oct.1987, pp.52–69.

Brown,Alan W.: @ Comp.Lab, Univ.of Newcastle-upon-Tyne "A View Mechansism for An Integrated Project Support Environment"; Computing Laboratory, University of Newcastle-upon-Tyne, rcvd Jan.1987.

Proposes an object oriented mechanism for managing knowledge bases. An object is only accessible via a set of defined operators. Layered architecture, each layer is defined using the operators and datatypes defined in the next lower one. Each layer acts as a firewall. Special update operators of an object are defined in terms of the update operators from the next lower layer recursively. until we reach primitive data types. A view is then one of these layers (actually the adt levels are arranged in a tree structure.) Comment: essentially procedural semantics. There is always the possibility of throwing away too much in the course of creating a new level, and there is no possibility of recourse to more primitive, richer levels. PKR

Buneman,P., Davidson,S. and Watters,A.: "A semantics for complex objects and approximate queries"; ACM PODS, 1988.

Butler, Margaret: @ Univ. of California-Berkeley "Storage Reclamation in Object-Oriented Database Systems"; ACM-SIGMOD 87, May.1987.

Cameron,J.R.: "An Overview of JSD"; IEEE TSE, Vol.SE12 No.2, 1986, pp.222–240.

The Jackson Design Method begins with describing action on an object.

Carey,M.J., DeWitt,D.J., Richardson,J.E., and Shekita,E.J.: @ Univ. of Wisconsin-Madison, U.S.A. "Object and File Management in the EXODUS Extensible Database System"; VLDB 12, Aug.1986.

Carey,M.J., DeWitt,D.J., Vandenberg,S.L.: @ CSD, U. Wisconsin-Madison "A Data Model and Query Language for EXODUS"; CS Tech.Rep. No.734, Dec.1987.
EXCESS and EXTRA

Casais, Eduardo: @ Univ. de Geneve "An Object Oriented System Implementing KNOs"; ACM-COIS, Mar.1988, pp.284-290. dynamic properties argument inheritance, sim. Hypercard on SUNet.

Cockshot,W.P., Atkinson,M.P., Chisholm,K.J., Bailey,P.J., and Morrison,R.: @ Univ. of Edinburgh, UK "Persistent Object Management System"; SPE, Vol.14 No.1, Jan.1984, pp.49–71; ACM CR, 8408-0627.
Description of the software used to implement a persistent heap, includes an interprogram type checking, data integrity, disk transfer optimization, and name-space management

Codd,E.F.: "Extending the Database Relational Model to Capture More Meaning"; ACM TODS, Vol.4 No.4, Dec.1979, pp.397–434.
'Semantic models' for formatted databases, to capture in a more or less formal way more of the meaning of the data. Two major thrusts: relation and molecular semantics. Extensions to the relational model (RM/T). New rules for insertion, update, and deletion. with integrity constraints, as well as new algebraic cperators (Theta-select, outer join,....).

Cohen,Benjamin C.: @ David Sarnoff Res.Ctr., Princeton "Views and Objects in OB1: A PROLOG-based View-Object-Oriented Database"; TR. PRRL-88-TR-005, rcvd. Mar. 1988.
The view-object manager combines relational views and objects. The results are placed into a direct object manager. These view-objects remain defined through predicates, dont have identity, and are not directly updatetable, only base relations are to be updated, if possible, through the views.

Cox,B.: *Object-Oriented Programming, An evolutionary Approach*; Addison-Wesley, 1987.

Dadam,P. and Kuspert,K.: "Cooperative Object Buffer Management in the Advanced Information Management Prototype (AIM-P)"; VLDB 13, Brighton England, Sep.1987.

Dayal,U., Manola,F., Buchmann,A., Chakravarthy,U., Goldhirsch,D.,: Heiler,S., Orenstein,J., and Rosenthal,A. @ Computer Corporation of America "Simplifying Complex Objects: The PROBE Approach to Modelling and Querying Them"; Proc. German DB Conf., Darmstadt, Apr.1987. Also; CCA, Apr.1987.

deBy,Rolf A.: @ Univ.Twente, Netherlands "Integrating Structure and Behaviour of Object Classes in a Basic Semantic Data Model"; Memorandum INF-86-39, Univ.Twente, 1986.

Demolombe,R.: "STEL: an Extended Relational Model to Represent and Manipulate Structured Objects"; rcvd Mar.1985.
External schema transforms objects to relational terms. Nulls.

Derret,Nigel, Kent,W., and Lyngback,P.: "Some Aspects of Stored Operations in an Object-oriented Database"; H-P Laboratory draft report, rcvd Oct.1985. Also; IEEE DB Eng.Bull., Vol.8 No.4, Dec.1985.
description of ORIS DBMS prototype.

Diederich,J. and Milton,J.: "ODDESSY: An Object-Oriented Database Design System"; IEEE CS Data Engineering Conf. 3, LA, Feb.1987.

Dittrich,K. and Dayal,U.(Eds): *Proceedings 1986 International Workshop on Object-Oriented Database Systems*; IEEE CS Order No.734, Sep.1986.

Ecklund,E.F. and Price,D.M.: @ CS Dept., Oregon State Univ. "Multiple Version Management of Hypothetical Databases"; TR.No. CR-84-16, Jul.1984. Also; HICSS 18, 1985.

The extension of each database object is managed as a tree of one branch. All other branches are considered hypothetical. A branch grows when updated. Updates to the primary version must be serializable, but hypothetical versions are not subject to such a constraint. Conflicting updates can be accepted by creating new hypothetical versions.

Ege,A. and Ellis,C.: "Design and Implementation of GORDION, An Object Base Management System"; IEEE CS Data Engineering Conf. 3, LA, Feb.1987.

ElMasri,R. and Navathe,S.: @ Univ. of Houston and Univ. of Florida "Object Integration in Database Design"; IEEE CS Data Engineering Conf. 1, Los Angeles, Apr.1984.

Faloutsos,C., Sellis,T.K., and Roussopoulos,N.: @ Univ. of Maryland "Analysis of Object-Oriented Spatial Access Methods"; ACM-SIGMOD 87, May.1987.

Fishman,D.H., Beech,D., Cate,H.P., Chow,E.C., Connors,T., Davis,J.W.,: Derret,N., Hoch,C.G., Kent,W., Lyngbaek,P., Mahbod,B., Neimat,M.A., Ryan,T.A., and Shan,M.C. @ HP Labs. Palo Alto "Iris: An Object-Oriented DBMS"; STL-86-15 Dec.1986.

These papers describe Hewlett-Packard's project in implementing an object-oriented database system. The query language is an extension of SQL, which they immediately translate into relational algebra, so it is unclear how object-oriented it can be. However, they do support classes and subclasses, and definition of methods. —jdu

Gangopadhyay,D., Dayal,U., and Browne,J.C.: @ Univ.Texas at Austin and CCA "Semantics of Network Data Manipulation Languages: An Object-Oriented Approach"; VLDB 8, McLeod and Villasenor(eds), Mexico City, 1982, pp.357–369.

Gibbs,Simon J.: "An Object Oriented Office Data Model"; PhD Th., Univ.Toronto 1983, 276pp.

Hadzilacos,T. and Hadzilacos,V.: "Transaction Synchronisation in Object Bases"; ACM PODS, 1988.

Haerder,Theo: @ Univ.Kaiserlautern,FRG. "New Approaches to Object Processing in Engineering Database"; Abstract in Proc.Object Oriented Database Workshop, Asilomar, 1986, pp.217.

Hardwick,M. and Sinha,G.: "A Data Management System for Graphical Objects"; IEEE CS Data Engineering Conf. 2, Los Angeles, Feb.1986.

Hardwick,M. and Spooner,D.L.: "The ROSE data Manager: Using Object Technology to Support Interactive Engineering Applications"; rcvd. Jun.1987.

Harland,D.H. and Beloff,B.: @ Linn Products, Glasgow Scotland "OBJECT, A Persistent Object Store With an Integrated Garbage Collector"; ACM SIGPLAN Notices, Vol.22 No.4, Apr.1987, pp.70–79.

Heiler,S.: @ CCA "Object-Oriented Databases are Neccessary for Design Databases"; ACM DAC 24, Jun.1987.

Hilgard,E.R., Atkinson,R.L., and Atkinson,R.C.: *Introduction to Psychology (7th ed.)*;

Re persistent objects: On page 249 'Information in long-term memory is usually encoded in terms of its meaning, by either an abstract semantic code or a concrete imagery code. The more deeply or elaborately one encodes the meaning, the better the memory will be.'

'Memory for complex materials involves a constructive process. Construction may involve adding innferences to the material presented or fitting the material into stereotypes and schemata.'— Xiaolei

Hudson,S.E. and King,R.: @ Dept. of CS, Univ. of Arizona-Tuscon "Object Oriented Database Support for Software Environments"; ACM-SIGMOD 87, May.1987.

Kemper,A., Lockemann,P.C., and Wallrath,M.: @ Inst. for Informatik II, Univ. Karlsruhe, FRG "An Object-Oriented Database System for Engineering Applications"; ACM-SIGMOD 87, May.1987.

uses IBM Heidelberg DBMS

Ketabchi,M. and Berzins,V.: "Component Aggregation: A Mechanism for Organizing Efficient Engineering Databases"; IEEE CS Data Engineering Conf. 2, Los Angeles, Feb.1986. use of object definitions to control locality. Does not say how. Simplistic formulas.

Ketabchi,M.A. and Berzins,V.: @ Santa Clara Univ. "Mathematical Model of Composite Objects and Its Application for Organizing Engineering Databases"; IEEE TSE, Vol.14, No.1, Jan.1988, pp.71–83.

Assemblies having the same types of parts, i.e., equivalent objects form a Boolean algebra whose minterms represent atomic objects. Materalization of explosion views partitions the database into application-oriented clusters.

Khoshafian,S. and Copeland,G.: "Object Identity"; OOPSLA86, ACM SIGPLAN Vol.21 No.1, Nov.1986, pp.406–416.

Clear articulation of the distinction between 'object-oriented' and 'value-oriented'. But the semantic modelers have written about essentially the same distinction. That work usually contrasts 'object-oriented' vs. 'record-oriented'; and emphasizes the point that determining data relationships in record-oriented systems often involves equality between pairs of record entries (see the Hammer-McLeod SDM paper, Kent's work, the Hull-King survey). The object-oriented approach is the cornerstone of several semantic models, including Semantic Binary Data Model, FDM, SDM, and IFO. However, several other "semantic" data models do come dangerously close to a value-oriented philosophy, including Wiederhold's structural data model, Codd's RM/T, Su's SAM*. Although the Entity-Relationship model is object-oriented, in Chen's seminal paper he dwells heavily on its translation into the relational model, and thus gives it a very value-oriented tone. — Hull.

Khoshafian,S., Valduriez,P., and Copeland,G.: "Parallel Query Processing for Complex Objects"; IEEE CS Data Engineering Conf. 4, Feb.1988, LosAngeles.

Kim,W. et al: "Composite Object Support in an Object-Oriented Database System"; to appear in OOPSLA 2, Orlando FL, Oct.1987.

Kim,W, Chou,H-T., and Banerjee,J.: "Operations and Implementation of Complex Objects"; IEEE IEEE CS Data Engineering Conf. 3, LA, Feb.1987.

King,Roger: @ Univ.of Colorado "SEMBASE: A Semantic DBMS"; 'Expert Database Systems', Kerschberg(ed), B-C, 1985. Also; EDSW, Vol.1, Oct.1984, pp.151–171.

Uses graphic representations for objects on a SUN-workstation under UNIX.

King,R. and McLeod,D.: "Query A Database Design Methodology and Tool for Information Systems"; ACM TOOIS, Vol.3 No.1, Jan.1985, pp.2–21.

INSYDE prototype models objects and their flow.

Klein,J. and Reuter,A.: @ CS Dept., Univ. of Stuttgart, FRG "An Application Oriented Approach to View Updates"; ACM-COIS, Mar.1988, pp.243-249. object semantics explored in a workstation environment. par

Korth,Hank: @ Univ.Texas,Austin "Extending the Scope of Relational Languages"; IEEE Software, Vol.3 no.1, Jan.1986, pp.19–29.
include serving object-oriented and functional languages. Much intro. Operator relations specify processing steps.

LaFue,G.M.E.: @ Schlumberger-Doll Research "Basic Decisions about Linking an Expert System with a DBMS: A Case Study"; IEEE DB Eng.Bull., Vol.6 No.4, Dec.1983, pp.56–64. Also; IEEE Database Engineering, Vol.2, 1984, Kim, Ries, Lochovsky(eds), pp.238-246.
STROBE supports knowledge bases, containing related objects. Objects are of two types: classes and individuals. Individuals are the database extension and lowest level of the ISA hierarchy. Objects have slots, containing facets of the attributes: value(data/code), type(incl.reference), max, min, units,... . The unsuitability of interfacing STROBE with a relational DBMS as INGRES is discussed. STROBE operates on dipmeter data.

Lamersdorf,Winfried: *Semantische Reprasentation Komplexer Objektstrukturen: Modelle fur nichtkonventionelle Datenbankanwendungen*; Informatik-Fachberichte 100, 1986.
representation of the semantics of complex objects.

Larsen,R.P.: @ Rockwell Int'l "Rules-Based Object Clustering: A Data Structure for Symbolic VLSI Synthesis And Analysis"; ACM DAC 23, Jun.1986.

Law,K.H., Jouaneh,M.K., Spooner,D.L.: @ RPI (Troy NY) "Abstraction Database Concepts for Engineering Models"; Engineering with Computers, Vol.2, 1987, Springer Verlag, pp.79–94.
Constraints in engineering objects are manipulated during transformation. Focus is on generalization and abstraction hierarchies in solid CSG models.

Learmont,T., and Cattell,R.G.G.: @ Sun Microsystems "An Object-Oriented Interface to a Relational Database"; Sun Microsystems, Rcvd. Jul.1987.

Lerman,I-C. and Peter,P.: "Elaboration et Logiciel d'un Indice de Similarite Entre Objects d'un Type Quelconque: Application au Probleme de Consensus en Classification"; INRIA TR.434, Jul.1985.
rules about data types drive selection of statistical procedures.

Li,Q. and McLeod,D.: @ CS Dept., USC "Object Flavor Evolution in an Object-Oriented Database System"; ACM-COIS, Mar.1988, pp.265-275.
Changing semantics include evolution

Lieberman,Henry: @ MIT "Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems"; OOPSLA86, ACM SIGPLAN Vol.21 No.1, Nov.1986, pp.214–244.

Lipeck,U.W. and Neumann,K.: "Modeling and Manipulating Objects in Geoscientific Databases"; E-R 5, Dijon 1986, pp.105–124

Lobelle,Marc C.: @ Univ. Catholique de Louvain, Belgium "Integration of diskless workstation in UNIX United"; SPE, Vol.15 No.10, Oct.1985, pp.997–1010; ACM CR, Vol.27 No.5, May.1986.
tradeoff between a file (object) server or a disk server.

Lorie,Raymond and Plouffe,Wilfred: @ IBM Res. Lab., San Jose "Complex Objects and Their Use in Design Transactions"; IBM RJ 3706 (42922), Dec.1982.

Lyngbaek,P. and McLeod,D.: "Object Management in Distributed Database Systems"; ACM TOIS, Vol.2 No.2, Apr.1984, pp.96–122.

This paper notes: (1) It doesn't require that all the workstations in the network implement the same database model as long as they all provide the same network interface. (2) It recognizes that the given model lacks semantic expressiveness among other things.

Lyngbaek,P., Derret,N.P., Fishman,D.H., Kent,W., and Ryan,T.A.: @ HP Labs. Palo Alto "Design and Implementation of the Iris Object Manager"; TR.HP STL-86-17, Dec.1986.

Magel,K. and Shapiro,L.: @ North Dakota St. Univ. "JADE: An Object Oriented, Heterogeneous Model Management System" CSD, North Dakota St. Univ., Jul.1987.

Maier,D. and Ullman,J.D.: "Maximal Objects and the Semantics of Universal Relation Databases"; ACM TODS, Vol.8 No.1, Mar.1983, pp.1–14. Also; Stony Brook CS TR 80/016, 1980.

Modifies the universal relation concept when the underlying relational structure has 'cycles'. Maximal object is defined at schema definition time. The query processor takes the schema, object, and maximal object at input.

Maier,D., Otis,A., and Purdy,A.: "Object-Oriented Database Development at Servio Logic"; IEEE DB Eng.Bull., Vol.8 No.4, Dec.1985.

Maier,D.:: "DMDM (Dave Maiers Data Model)"; rcvd Apr.1985.

Object-based, with nesting and a frame-like structure, with binary relation storage

Maier,D., Stein,J., Otis,A., and Purdy,A.: @ Oregon Grad.Center. "Development of an Object-Oriented DBMS"; TR.CS/E-86-005.

Maier,David: @ Oregon Grad. Center "A Logic for Objects"; TR.CS/E-86-012, Nov.1986.

Maier,D., Nordquist,P., and Grossman,M.: "Displaying Database Objects"; EDS 1, Apr.1986

Maier,D. "Why Database Languages Are A Bad Idea"; rcvd. Mar.1988.

Use objects, also for commands, and manipulate them.

Manola,F. and Orenstein,J.A.: @ Computer Corp. of America "Toward a General Spatial Data Model for an Object-Oriented DBMS"; VLDB 12, Aug.1986.

McCoy,Kathleen F.: @ Univ. Delaware "The ROMPER System: Responding to Object-Related Misconceptions using Perspectives"; PhD thesis U.Penn, ACL Proc., 1986.

Two types of misconception: misclassification and misattribution of objects. The system responds to such misconceptions domain-independently and context-independently by a notion of object perspective, which acts to filter the user model, highlighting those aspects which are made important by previous dialogue, while supressing others. Output from ROMPER consists of rejection of detected user misconception followed by supportive information, in the form of formal specification. The information from response generator is not fed back to misconception detector.

McLeod,D. and Widjojo,S.: "Object Management and Sharing in Autonomous, Distributed Data-Knowledge Bases"; IEEE DB Eng.Bull., Vol.8 No.4, Dec.1985.

mentions problems of domain matching.

Meier,A. and Lorie,R.A.:: @ IBM San Jose "Implicit Hierarchical Joins for Complex Objects"; Rcvd summer 1983. Ownership connections, inclusion dependency.

Mermet,J.: @ IMAG "CASCADE: vers un systeme integre de CAO de VLSI"; INRIA Bull., No.102, Jun.1985, pp.16–29.

Proposal for an integrated design system. Considers object orientation or PROLOG, VAX. Notes that 200-1000 man-years are needed for a design.

Morris,K., Ullman,J.D., and VanGelder,A.: "Design Overview of the NAIL!System"; Proc. of ICLP 3, 1986. Also; Stanford Univ.CSD, rcvd Dec.1985, rev. May.1986, Stan-CS-86-1108.

The NAIL!System seems to be a much more powerful query language the ones commercially available today. It adds the power and dexterity of Prolog-like logic to standard query techniques. The NAIL! System exhibits a tendency to swing the database community from object-oriented query languages back to value-oriented query languages. There seemed to be ambiguity as to how to handle recursive rules. The paper gave some techniques but didn't prefer one over the other. Overall, the NAIL!System appears to be a superior attempt at strengthening conventional database query operations.— Avery

Motro,A., DAtri,A, and Tarantino,L.: @ Dept. of CS, USC "KIVIEW: An Object-oriented Browser"; Oct.1987.

Mylopoulos,J., Bernstein,A., and Wong: "A Language Facility for Designing Database-Intensive Applications"; report rcvd.1985.

Describes TAXIS, a language for the design of interactive information systems (IIS). Objects and actions are represented uniformly as frames.

Nestor,J.R., Wulf,W.A., and Lamb,D.A.: @ Tartan Labs and CMU "IDL - Interface Description Language; Formal Description"; CMU CSD, Draft rev.2.0, Jun.1982.

Data structure language for CAD, used for DIANA, USAF.

An object has a type, a location, and a value. Types are basic, node, private. Objects of type node are comprised of a set of attributes. Attributes are typed objects. Attributes may reference other objects.

Nguyen,G.T.: "Object Prototypes and Database Samples for Expert Database Systems"; EDS 1, Apr.1986.

objects derived from database, includes pessimistic validation of view updates

Nguyen,G.T., Rieu,D., U. Grenoble: "Semantics of CAD Objects for Generalized Databases"; ACM DAC 23, Jun.1986.

Nguyen,G. and Rieu,D.: "Expert Database Support for Consistent Dynamic Objects"; VLDB 13, Brighton England, Sep.1987.

pessimistic control, object quivalence established via heuristic rules, consistency should increase with updates.

Nierstrasz,O.M.: "Hybrid: A Unified Object-Oriented System"; IEEE DB Eng.Bull., Vol.8 No.4, Dec.1985.

Osborn,S.L., and Heaven,T.E.: @ Univ. of West. Ontario "The Design of a Relational Database System with Abstract Data Types for Domains"; ACM TODS Vol.11 No.3, Sep.1986, pp.357–373.

Operations on simple objects, operations on aggregates and 'transformations' can be defined on relations. It is possible to implement a transitive closure RAD uses the data dictionary. —Ong,J., Fogg,D., and Stonebraker,M.

Potter,W.D., Trueblood,R.P., Eastman,C.M. and Mathews,M.M.: "The Knowledge/Data Language: A Hyper-Semantic Data Model Specification Language"; Univ.of South Carolina, Dep.of CS, TR.86003, Nov.1986.

Also a query language. Based somewhat on DAPLEX and IRIS. Includes heuristics, uncertianty, constraints. Objects are eqiv to entities and functions to attributes (also computed or inferred).

Purdy,A., Maier,D., and Schuchardt,B: "Integrating an Object Server With Other Worlds"; ACM TOIS, Apr.1987. Also; TR.CS/E-86-013, Oregon Graduate Center, Dec.1986.

Gemstone.

Qian,X-L. and Wiederhold,G.: "Data Definition Facility of Critias"; ER 4, Chicago, Oct.1985, pp.46-55.

Describes a schema definition language CRITIAS. The language is intended to provide syntactic embodiment of a semantic data model. The basic modeling concepts of CRITIAS are entity types, attributes, and three types of relationships (subtype, association, and reference), to model the conceptual objects, their properties, and relationships among them. Language constructs provide for specifying semantic integrity constraints for these concepts. We also show that there is a direct mapping from CRITIAS constructs to relational schema so that the implementation of CRITIAS is straightforward.

Ramakrishnan,R. and Silberschatz,A.: "The Molecular-Relations Data Model"; rcvd Feb.1987.

Looks like 'molecules' are what Kuper and Vardi called ' r-values' in their 'logical data model.' The idea is that a molecule has scalar components and components that are pointers to (sets of) objects of a given type. —jeff

Ramamoorthy,C.V. and Sheu,P.C.: "Logic-Oriented Object Bases"; IEEE CS Data Engineering Conf. 3, LA, Feb.1987.

framework of databases that are constructed based on object model and augmented by mathematical logic.

Richardson,J.E., and Carey,M.J.: @ CS Dept., Univ. of Wisconsin "Programming Constructs for Database System Implementation in EXODUS"; ACM-SIGMOD 87, May.1987.

generators in E-language for persistant typed complex objects

Rogers,T.R., and Cattell,R.G.G.: @ Sun Microsystems "Object-Oriented Database User Interfaces"; Sun Microsystems, Rcvd.Jul.1987.

Rollinger,C-R., Studer,R., Uszkoreit,H., and Wachsmuth,I.: @ IBM Stuttgart FRG "Text-understanding in LILOG – Sorts and Reference Objects"; Systems-87 AI Kongress, Munchen FRG, Oct.1987.

Type network

Rosch,E., Mervis,C.B., Gray,W.D., Johnson,D.M., and Boyes-Braem,P.: "Basic Objects in Natural Categories"; Cognitive Psychology, Vol.8, 1976, pp.382–439.

Roth,M.A., Korth,H.F. and Silberschatz,A.: "Theory of Non-First-Normal-Form Relational Databases"; U. of Texas-Austin CS Report TR-84-36, Dec.1984.

For View-Objects.

Roussopoulos,N. and Kang,H.:: @ Univ. of Maryland "Preliminary Design of ADMS ±: A Workstation-Mainframe Integrated Architecture for Database Management Systems"; VLDB 12, Kyoto Aug.1986, pp.355-362.

Uses View concepts without update. Lazy update. Derived object locks.

Rowe,Lawrence A.: @ EECS, UCB "A Shared Object Hierarchy"; TR., UCB ERL, M86/85, Jun.1987 Also: Stonebraker 'The Postgres Papers'.

Schek,H-J. and Scholl,M.H.: "The Relational Model with Relation-Valued Attributes"; Information Systems, Vol.11,no.2, 1986, pp.137–147.
Nested relations.

Samaras,George: "An SQL Interface for Object-Oriented Engineering Databases"; PhD proposal for Spooner, rcvd. Jun.1988, RPI, Troy
extended SQL is to be translated into ROSE.

Schlageter,G., Unland,R., Wilkes,W., Ziechang,R., Maul,G., Nagl,M.: and Meyer,R. "OOPS - An Object Oriented Programming System with Integrated Data Management Facility"; IEEE IEEE CS Data Engineering Conf. 4, Feb.1988, LosAngeles.

Schrefl,M. and Neuhold,E.J.: "Object class definition by generalization using upward inheritance"; IEEE CS Data Engineering Conf. 4, Feb.1988, LosAngeles. Also; rcvd. May.1987.
Constructing objects by abstraction.

Sernadas,A., Sernadas,C., and Ehrich,H-D.: "Object-Oriented Specification of Databases: An Algebraic Approach"; VLDB 13, Brighton, UK, Sep.1987.

Sellis,T.K and Shapiro,L.: @ UCBerkeley and North Dakota State "Optimization of Extended Database Query Languages"; TR.UC Berkeley/Elec.Res.Lab 85-1, Jan.1985.
QUEL*, for multi-tuple etc object retrieval, optimization with a transitive closure operator.

Sellis,T., Roussopoulos,N., and Faloutsos,C.: "The R+ Tree: A Dynamic Index for Multi-Dimensional Objects"; VLDB 13, Brighton England, Sep.1987.
survey, avoid parent overlap of R-tree.

Shlaer,Sally and Mellor,Stephen: *Object-oriented Systems Analysis: Modeling the World in Data*; Yourdon Press, 1988, 144pp.
Picture book approach. Well done. The 'object-oriented' title was added later, and it really is a E-R and tuple based approach. No OO methods.

Spector,A.: "The TABS Project"; IEEE DB Eng.Bull.,Vol.8 No.2, Jun.1985, pp.19–25.
OS research project to investgate object-oriented transaction support

Spector,A.Z., Pausch,R.F., and Bruell,G.: @ CMU "CAMELOT: A Flexible, Distributed Transaction Processing System"; IEEE Compcon 88, San Francisco, Feb-Mar.1988, pp.432–437.
Camelot executes on a variety of uni- and multi- processors on top of the Unix-compatible Mach operating system. Automatic management of threads, nested transactions, flexible synchronization, long and short transactions, small and large data objects, non-blocking commit protocols, logging, multiple servers, multiple disks per node. Data type library supports B-trees, extensible hash tables, and dynamic storage allocation.

Spooner,D.L., Milicia,M.A., and Faatz,D.B: "Modeling Mechanical CAD Data with Data Abstraction and Object-Oriented Techniques"; IEEE CS Data Engineering Conf. 2, Los Angeles, Feb.1986.

Stonebraker,M. and Rowe,L.A.: "The Design of POSTGRES"; TR., UCB ERL, M86/85, Jun.1987, in 'The Postgres Papers'. Also; ACM SIGMOD'86, Washington DC, May.1986, pp.340–355. Also; Tech.Report UC Berkeley, rcvd Nov.1985.

recursions are introduced into the QUEL language and implemented.
Support complex objects, extendability, alerters and triggers, inferencing forwards and backwards, improved code for crash recovery, consider optical discs, workstations, tight-coupling. Use custom VLSI chips. Keep the relational model.

Stroustrup,B.: *The C++ Programming Language*; Addison-Wesley, 1986.
Includes storage of objects.

Studer,Rudi: "Modeling Time Aspects of Information Systems"; IEEE CS Data Engineering Conf. 2, Los Angeles, Feb.1986.
Analyses five object oriented semantic temporal database models in terms of their static structural facilities, their ability to handle dynamics of data, and their time modeling capabilities. A fairly good review —Downs.

Su,S.Y.W., Krishnamurthy,V., and Lam,H.: @ Univ.of Florida "An Object-Oriented Semantic Association Model (OSAM)"; rcvd. 1986.

Tanaka,K., Yoshikawa,M., and Ishihara,K.: "Schema Virtualization in Object-Oriented Databases"; IEEE CS Data Engineering Conf. 4, Feb.1988, LosAngeles.

Thatte,S.: "Persistent Memory: Storage Architecture for Object-oriented Databases"; Proc.Workshop on OODBS, Pacific Grove CA, ACM, Sep.1986.

Touati, Herve: @ CS Dept., Univ. of California-Berkeley "Is Ada an Object Oriented Programming Language?"; ACM-SIGPLAN NOTICES, Vol.22 No.5, May.1987. pp.23–26.

Trigg,Randy: "NoteCards: An Environment for Authoring and Idea Structuring"; Xerox PARC, 1985.
An idea structuring tool, but can also be used as a fairly general database system for loosely structured information. The basic object in NoteCards is an electronic note card containing an idea-sized unit of text, graphics, images, or whatever. Different kinds of note cards are defined in an inheritance hierarchy of note card types (e.g., text cards, sketch cards, query cards, etc.). Individual note cards can be connected to other note cards by arbitrarily typed links, forming networks of related cards. At present, link types are simply labels attached to each link. It is up to each user to utilize the link types to organize the note card network. NoteCards also includes a filing mechanism for building hierarchical structures using system-defined card and link types. There are also browser cards containing node-link diagrams (i.e., maps) of arbitrary pieces of the note card network and Sketch cards for organizing information in the form of drawings, text and links spatially. The functionality in NoteCards is accessible through a set of well-documented Lisp functions, allowing the user to create new types of note cards, develop programs that monitor or process the network, integrate other programs into the NoteCards environment.

Tsichritzis,D.: "Object Species"; IEEE DB Eng.Bull., Vol.8 No.4, Dec.1985..

Tsichritzis,D., Fiume,E., Gibbs,S., and Nierstrasz,O.: "KNOs: Knowledge Acquisition, Dissemination and Manipulation Objects"; ACM TOIS, Vol.5 No.1, Jan.1987, pp.96-112.

Ullman,J.D.: @ Stanford University "Database Theory - Past and Future"; ACM PODS, San Diego, Mar.1987.
'Object-oriented' vs. 'value-oriented' and other related matters.

Ullman, Jeffrey D.: *Principles of Database and Knowledge-based Systems*; Computer Sscience Press, 1988, 631pp.

models, datalog, ISBL, QUEL, QBE, SQL, DBTG as object-oriented, OPAL, physical structures for DBTG, hierarchies, relations, design theory, protection, UNIX with SQL/RT, transactions, distribution and locking.

Valduriez,P., Khoshafian,S., and Copeland,G.: @ MCC "Implementation techniques of Complex Objects"; VLDB 12, Aug.1986.

Vermeir,D. and Nijssen,G.M.: @ Univ.of Queensland, Brisbane "A Procedure to Define the Object Type (Structure) of a Conceptual Schema"; Inf.Sys., Vol.7, 1982, pp.329–336.

part of NIAM (Nijssen Information Analysis method) model.

Vianu,V.: "A Dynamic Framework for Object Projection Views"; ACM TODS, Vol.13 no.1, Mar.1988, pp.1–22.

view updates considering original FDs.

Wegner,Peter: "Language Paradigms for Programming in the Large"; Brown Univ. draft, rcvd Jul.1985.

view Comment: The object-oriented model in developing programming environment is of course the right direction. However, Peter is still sticking to the concept that operations are part of the object types, not as seperate action objects. This makes the operations much like procedural attachment and leads to an incomplete model. Although Peter tries to unify the object-oriented and the distributed programming paradigms, processes are only objects externally. The internal structure of processes are still not object-oriented. We can make the following correspondance between PL structure and object-oriented model: control structure ¡–¿ control transfer between objects (frames). And the control transfer between objects should be modeled by pre- and post- condition, triggers, assertions, etc. —Xiaolei

Wegner,Peter: "A Database Approach to Languages, Libraries and Environments"; Brown Univ. draft, rcvd Jul.1985.

Summary: It is interesting to notice that while we are trying to design database languages with the programming techniques, the programming people are exploring our semantic model to describe general programming systems. The main idea in these two papers is to describe programming systems using the object-oriented model, especially programming environment and distributed programming. Views are defined differently from views in database world. Views are objects too. Views are considered to be interfaces, e.g., the interface of a process object. An object can have multiple views and the viewer is another object, instead of the user as for a traditional database view. This generalizes the concept of generic process.

Wegner,Peter: "The Object-Oriented Classification Paradigm"; Tech.Rep, Brown Univ., rcvd Jun.1986.

what makes a programming language an 'Object-Oriented" PL (In a sense, Logic PL's are restricted forms of O-O PL's.)—jeff

Weiser,S.P.: "An Object-Oriented Protocol for Managing Data"; IEEE DB Eng.Bull., Vol.8 No.4, Dec.1985.

Wiederhold,Gio: "A Method for the Design of Multi-Objective Databases"; in Comp. in Eng. 1982, Vol.4, Rahavan,R. (ed), Proc.of the Annual Meeting of ASME, Jun.1982, pp.161-165.

Wiederhold,Gio: @ Stanford "Views, Objects, and Databases"; IEEE Computer, Vol.19 No.12, Dec.1986, pp.37–44; ACM CR 8708-0693. Also; Dittrich, Dayal, Buchmann (eds.) *Object-oriented Database Systems*, Springer Verlag, 1988.

Overview, does an appropriate job of whetting the appetite about an intriguing concept that appears to have significant potential. — Mayforth

Woelk,D., and Kim,W.: "Multimedia Information Management in an Object-Oriented Database System"; VLDB 13, Brighton England, Sep.1987.

Wolf,W.: @ ATT Bell Labs "An Object Oriented, Procedural Database For VLSI Chip Planning"; ACM DAC 23, Jun.1986.

Woo,C.C., Lochovsky,F.H.: "An Object-Based Approach to Modelling Office Work"; IEEE DB Eng.Bull., Vol.8 No.4, Dec.1985..

Yannakakis,M.: @ Bell Labs, Murray Hill NJ "Algorithms for Acyclic Database Schemes"; VLDB 7, Zaniolo and Delobel(eds), Sep.1981, pp.82–94.

Many real-world situations can be captured by a set of functional dependencies and a single join dependency of a particular form called acyclic [B.,]. The join dependency corresponds to a natural decomposition into meaningful objects (an acyclic database scheme). Our purpose in this paper is to describe efficient algorithms in this setting for various problems, such as computing projections, minimizing joins, inferring dependencies, and testing for dependency satisfaction.

Zdonik,S.B. and Wegner,P.: "A Database Approach to Languages, Libraries and Environments"; Brown University, Dep.of CS TR.CS-85-10, May.1985.

includes ENCORE specifications of the UNIX file system and of two views of an object.

Zdonik,S.: "Object Management Systems for Design Environments"; IEEE DB Eng. Bull., Vol.8 No.4, Dec.1985.