COMPUTERS AND BIOMEDICAL RESEARCH 24, 476–498 (1991)

Management of Complex Immunogenetics Information Using an Enhanced Relational Model

T. BARSALOU,[1] W. SUJANSKY,[2] L. A. HERZENBERG,[3] AND G. WIEDERHOLD[4]

[1]IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10538, [2]Section on Medical Informatics, Stanford University, [3]Department of Genetics, Stanford University School of Medicine [4]Departments of Computer Science and Medicine, Stanford University

Received May 22, 1990

Flow cytometry has become a technique of paramount importance in the armamentarium of the scientist in such domains as immunogenetics. In the PENGUIN project, we are currently developing the architecture for an expert database system to facilitate the design of flow-cytometry experiments. This paper describes the core of this architecture—a methodology for managing complex biomedical information in an extended relational framework. More specifically, we exploit a semantic data model to enhance relational databases with structuring and manipulation tools that take more domain information into account and provide the user with an appropriate level of abstraction. We present specific applications of the structural model to database schema management, data retrieval and browsing, and integrity maintenance. © 1991 Academic Press, Inc.

1. INTRODUCTION

In the late 1960s, a group of biomedical researchers at Stanford University set out to develop a means of analyzing and isolating viable cells that have different cell-surface phenotypes.The idea was that those phenotypes could be differentiated by using various combinations of fluorescence-tagged monoclonal antibodies. This work eventually led to a flow-cytometry sorting method named *fluorescence-activated cell sorting* (FACS) (*9*). Since then, the method has been improved continuously, and numerous biological and clinical applications have been developed and employed. By now, an estimated several thousand FACS analyzers and sorters are in use; nearly all biological research institutions and large numbers of clinical research laboratories throughout the world run FACS instruments. The FACS technique is heavily used in the fields of immunology and molecular and cell biology, including clinical immunology.

Because of the format and rate of generation of raw FACS data, computer support has been an integral part of FACS analysis since the initial engineering stage (*16*); various groups developed software that could handle data from

the early one- and two-parameter machines. The major impediment to more widespread use of the technology, however, involves the requirement for greater skills in reagent selection, protocol definition, machine operation, and data analysis than are typically available today in basic and clinical research settings. Developing an *expert flow-cytometry workstation* that would provide a new level of automatic analysis and control for FACS and would greatly facilitate FACS use by reducing the need for on-site human expertise should therefore significantly improve the potential for using this versatile methodology in biomedical research and clinical practice. The development of the flow-cytometry workstation encompasses the design of different software modules for assisting in the design of experiment protocols, creating new experiment protocols and editing the existing ones, controlling and operating the instrument, analyzing and displaying FACS-experiment results, managing and retrieving FACS data, and reporting to other programs, such as to statistical packages.

Our research group is now building the expert flow-cytometry workstation. In the PENGUIN project, we focus particularly on the area of computer-assisted planning of FACS experiments (*4*). Because decision support in this domain requires the integration of diverse data and knowledge sources, we are investigating in PENGUIN the combined use of database, artificial intelligence, and object-oriented techniques to design a general architecture for *expert database systems*. In this paper, we describe the core of this architecture—a methodology to manage immunogenetics and FACS-staining information using an extended relational model.

In Section 2, we present an overview of the PENGUIN project. In Section 3, we introduce the semantic model that we use to extend the relational model. In Section 4, we discuss the problems associated with defining a complex biomedical database and we describe briefly the schema resulting from the design process. In Section 5, we demonstrate the application of the semantic model for managing the relational schema, querying the FACS database, and browsing and updating information. In Section 6, we address the issue of consistency checking and integrity maintenance. In Section 7, we present specific examples of the system's use, before concluding in Section 8.

2. AN OBJECT-ORIENTED ARCHITECTURE FOR EXPERT DATABASE SYSTEMS

Because the design of FACS experimental protocols is both data- and knowledge-intensive, and because human expertise in the domain is scarce, providing assistance in this process will be helpful both in achieving good FACS performance and in structuring and formalizing the biomedical knowledge involved in flow cytometry. Decision support in this domain requires the combined use of data- and knowledge-based methods.

2.1. The PENGUIN Project

Combining database and expert-system technologies into *expert database systems* (BDSs) is now an active area of research (*10, 15*). EDSs can be seen as *mediators*—"software modules that exploit encoded knowledge about some

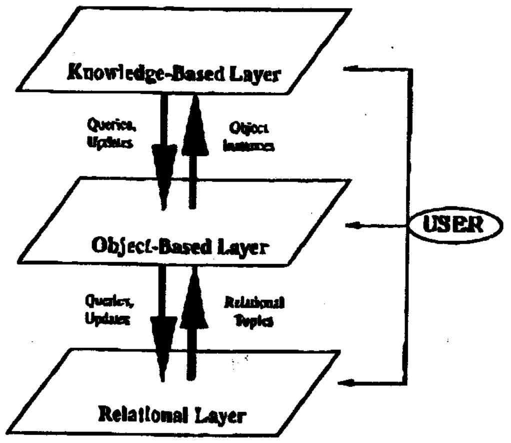


FIG. 1. A three-layer framework for EDSs. The object layer mediates between the relational layer and the knowledge-based layer, combining relational tuples into object instances. Similarly, it translates high-level requests for information into precise relational queries.

sets or subsets of data to create information" (*6, 19*). From our viewpoint of assisting scientists in developing FACS protocols, an EDS should

- Enhance relational database management systems (DBMSs) with structuring and manipulation tools that provide the user with an appropriate level of abstraction
- Allow expert systems to access and handle efficiently information stored in databases and to take advantage of database techniques for dealing with persistency.

In the PENGUIN project, we are investigating the hypothesis that an object-oriented approach can serve as a unifying framework for developing such EDSs.

The object-oriented paradigm has emerged as a pervasive and useful concept in many areas of computer science. Objects undoubtedly offer the appropriate level of abstraction to represent complex, real-world entities that are manipulated by EDSs. Storing information in the form of complex objects, however, can seriously inhibit sharing and flexibility, since persistent objects bind application-dependent knowledge to the data (*18*). A desirable compromise is to define an *object-based layer* on top of a relational DBMS (*7*). This approach calls not for storing objects explicitly in the database, but rather for generating temporary object instances by binding data from base relations to predefined object templates. Those instances can later be manipulated in various ways by an intelligent agent.

We therefore introduce a three-layer, domain-independent architecture for EDSs, where the object layer mediates between a database layer and a knowledge-based layer. Such an architecture is shown in Fig. 1.

A layered architecture with coupling of distinct components explicitly pre-

cludes physical integration. We require, however, *conceptual* integration, in that the layers, even though they are separate entities, should present a consistent interface to the users. Conceptual integration means that we can interact with the system at any given level, without incurring the overhead of the layers above that one, as shown in Fig. 1. Note also that, since the architecture is distributed and heterogeneous in nature, the knowledge layer does not actually fall within PENGUIN's boundaries. In fact, any application can exploit the object layer through a standard communication protocol. Hence, our emphasis in PENGUIN is on the two lower layers.

At the core of PENGUIN is a semantic data model—*the structural model*. Although PENGUIN exploits the structural model's knowledge of the database in both the relational and the object layer, we focus here on the use of the structural model to enhance relational-database operations. Extensive descriptions of the object layer can be found in (*1, 2, 6, 7, 20*).

2.2. Development Environment

PENGUIN's architecture fits well into the client–server model, where multiple workstations access a centralized, remote database server through a variety of tailored interfaces. The database server is a microVax-II computer from Digital Equipment Corporation (DEC). The database package that we use is Rdb/VMS, DEC's relational database product. On the server, COMMON LISP programs handle communication with the clients and access the database through an Rdb-to-Lisp programmatic interface.

The workstations are Apple Macintosh II personal computers. PENGUIN integrates two different environments on these workstations. A *standalone application* implements the entire object layer and various functionalities of the relational layer. A *hypertext authoring environment* provides data-retrieval and browsing tools for the relational layer—tools that adhere to a direct-manipulation style of interface (*22*). The standalone program is written in Object Pascal and totals about 25,000 lines of code. The hypertext tool is HyperCard (*13*); its programming language, HyperTalk, has been extended with external commands and functions to provide additional capabilities such as interprocess communication.

3. The Structural Model

The *structural model* is a "semantic data model" that augments the relational model by representing the knowledge about the constraints and dependencies among the domain relations in the database (*21*). (See (*8*) for a detailed discussion of semantic data models.)

3.1. Concepts and Symbols

The primitives of the model are *relations*, as determined by the normalization process, and *connections* to describe those relationships between the relations (*12*). Formally, a connection is specified by the connection type, a pair of source

JAN 29 '99 10:50AM U.C.D. HEALTH P.3/13

and destination relations, and their shared connecting attributes. Three types of connections are defined that correspond to precise integrity constraints, define the permissible cardinality of the relationships, and encode the relationships' semantics:

Ownership connection. A single tuple in the owner relation owns many lower-level tuples of the same type in the owned relation, which implies a one-to-many cardinality. This connection embodies the concept of dependency, where owned tuples are specifically related to and dependent on a single owner tuple. Aggregation hierarchies with "has some" type links thus can be implemented using ownership connections. Syntactic and integrity rules for the ownership connection are: (1) the key of the owned relation is the concatenation of the key of the owner relation and (at least) one other attribute; (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 ———•.

Reference connection. Multiple tuples in the same referencing relation refer to the same descriptive tuple of a different referenced relation; this case therefore implies a many-to-one cardinality. The connection of two related concepts, one being more general than the other, corresponds to an abstraction process. Rules for the reference connections are: (1) the key of the referenced relation matches the referencing attributes of the source relation; (2) a destination (referenced) tuple must exist when a referencing tuple is inserted; (3) deletion of a referenced tuple in the destination relation necessitates corrective action on the matching tuples of the source relation. The graphical symbol of the reference connection is ———→.

Subset connection. This connection has a partial one-to-one mapping from one general relation to another more specialized one; hence, a tuple in the general relation is linked to at most one tuple in the subset relation. Categorical hierarchies with "is a" type links can be represented using the subset formalism. The rules for the subset connection are: (1) the key of the subset relation matches the key of the general relation—that is, only the nonkey attributes differ; (2) a general tuple must exist when a subset tuple is inserted; (3) deletion of a general tuple implies deletion of the corresponding subset tuples. The graphical symbol of the subset connection is ———⊃.

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 (*17*). Most important, the connections provide the structural knowledge that is needed to overcome many limitations of the relational model, as we shall demonstrate below.

3.2. Representation in a Relational Database

The structural model (SM) for a database is itself stored in relational form and is accessible only through the DBMS. Encoding the SM in this way ensures persistency of the metadata and allows a uniform access method to both the

extensional database and the metadatabase. A relational representation also offers good portability, since it can be reimplemented easily in other relational environments. In contrast, a host-language file encoding the structural model might vary in format across different hardware platforms and operating systems.

Our relational metadatabase consists of two relations—CONNECTED_RELATIONS and CONNECT_ATTRIBUTES. CONNECTED_RELATIONS encodes pairs of relations connected in the structural model; CONNECT_ATTRIBUTES encodes the shared attributes defining each connection. The metadatabase has the following schema:

CONNECTED_RELATIONS	CONNECT_ATTRIBUTES
Connection_Id (Key Field)	Connection_Id (Key Field)
Source_Relation	Source_Relation_Attribute (Key Field)
Destination_Relation	Destination_Relation_Attribute
Connection_Type	
Cardinality_Constraint	

The Connection_Type field of CONNECTED_RELATIONS contains one of six SM connection types (that is, one of the three types specified in Section 3.1 or the inverse of one of the types). The Cardinality_Constraint field contains connection-specific information regarding the allowed cardinalities of corresponding source and destination tuples. Note that there is a one-to-many relationship between CONNECTED_RELATIONS and CONNECT_ATTRIBUTES; for each Source_Relation/Destination_Relation pair (tuple) in CONNECTED_RELATIONS, there may be one or more tuples in CONNECT_ATTRIBUTES. This relationship is consistent with our concept of the structural model, in which relations are logically connected through the values of one or more pairs of corresponding attributes.

For the structural model to be used efficiently, we must import it from the relational format into a host-language data structure in main memory. This operation is done by a host-language routine that accesses the DBMS using an embedded query language. The relational encoding of the structural model is converted into an adjacency-list representation. If we think of the structural model as a graph, with relations as the vertices and SM connections as the edges, the internal representation of the SM is then a modified adjacency list for that graph.

4. The Design of a Complex Biomedical Database

Using the SM, we designed a relational-database schema to encode many of the data and much of the knowledge relevant to the FACS domain. Designing the schema was an iterative and time-consuming process due to the richness of domain knowledge. Principles of the SM facilitated the design process by defining the types of relations supported by the model and by prescribing methods to encode certain semantic relationships.

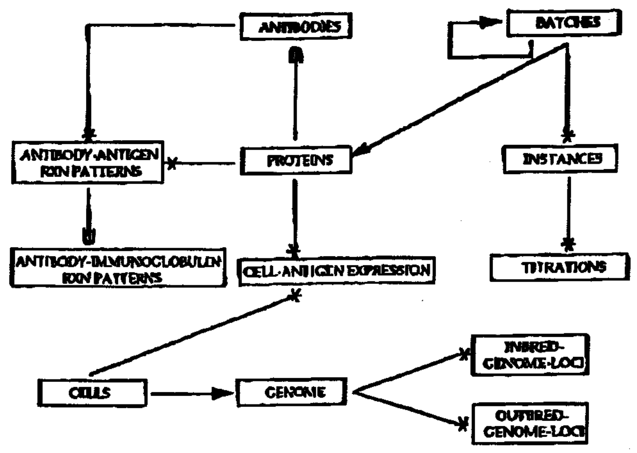


FIG. 2. Structural model for a subset of the immunogenetics database. The corresponding schema appears in Appendix A.

The database schema represents a complex semantic structure containing 34 relations, 150 attributes, and 36 SM connections. A partial listing of the schema appears in Appendix A and the structural model for this subset of the schema is shown in Fig. 2. The Boyce–Codd normal form (BCNF) (*11*) was used to help maintain data integrity and to improve storage efficiency. Four distinct areas of FACS-related information are encoded:

1. *Static genetic knowledge*, including the molecular composition of protein structures, the alleles present at particular genome loci, and the expression of cell surface markers
2. *Dynamic biochemical knowledge*, including reaction patterns between antibodies and other proteins, and the effects of antibodies on target cells
3. *Cytology-related information*, including a taxonomy of cell types and the cell composition of tissues and
4. *Laboratory-specific inventory data*, including the characteristics and availability of specific FACS reagents.

The complex FACS domain yielded several interesting structures in the database schema. For example, the taxonomy of cell types and the cell composition of tissues are encoded as a Boolean type lattice stored in a database relation. Disjoint, intersecting, and equivalent cell categories can thus be represented. Although the relational representation of this complex data structure facilitates the incremental addition of new cell types and new tissue-composition data, novel querying techniques will be required to extract the relationships implicit in such a structure. Another interesting case is the BATCHES relation. Reagent solutions in BATCHES may be derived from parent solutions in the same

relation; they are therefore represented as recursive entities through the definition of a self-referencing connection on BATCHES.

5. Database Operations Using the Structural Model

Although objects can represent more complex (and hence more useful) abstractions than tuples and relations, we believe that supporting separate relational access to data is still of significant value to many users and applications. We therefore use the structural model to develop a relational layer with enhanced capabilities, as compared to standard relational DBMSs. (Additional descriptions of the relational layer can be found in (*3*, *5*).)

5.1. An Object-Oriented Schema Editor

In relational DBMSs, the application domain is encoded in a schema. Unfortunately, users face numerous difficulties when they try to comprehend the schema—that is, when they try to map the database model to their own mental model of the application domain. Schemas are generally complex, and have numerous relations that do not necessarily correspond to domain entities (because of the requirements of normalization theory). Moreover, although the relational model has a regular and homogeneous structure, it blurs the semantic distinction between entities and relationships; relations are used to represent both types of construct. Finally, at a more mundane level, the user must remember and figure out many things—name and significance of relations and attributes, format and unit of the attributes—before she can use the database.

To alleviate those problems, PENGUIN provides an object-oriented graphical editor for the definition and manipulation of structural models for relational databases. Through this editor, the user employs visual tools to explore the application domain and to browse through the database schema.

The database editor strictly adheres to the requirements of the Macintosh human-interface guidelines. With standardized gestures for pointing, clicking, and dragging, the user can define relations and those relations' attributes, link those relations with connections (represented by arrows with different shades of gray), and assign names to them. The user can view multiple structural models in separate windows. PENGUIN supports Color QuickDraw as well as shading, so that the user can apply colors and shades to the model for maximum readability. The user has also the ability to scroll through the entire structural graph at a glance, whatever that graph's size. With a special gesture, she can invoke a condensed pop-up view of the entire graph; scrolling within the pop-up view will modify the portion of the structural model displayed within the active window. Finally, PENGUIN at all times maintains an alphabetized list of existing relations in a separate "list-mode" window; by selecting a relation from that list, the user can bring onto the screen in the editor window the portion of the structural model containing that relation.

PENGUIN exploits the precise syntactic and semantic rules of the structural model to check the validity of any specific database model and to take corrective action if some of those rules are violated. For instance, a cycle made entirely of ownership connections (which is not valid) will be detected; the system will then require that one of the connections be deleted to break that cycle. Furthermore, since PENGUIN uses a distributed model of information processing, it is crucial to keep the local and remote schemas (on the workstation and on the database server, respectively) consistent. The user can request at any time that a consistency check be performed; if the two schemas do not match (for instance, a relation is missing in the workstation's schema), the inconsistencies have to be resolved (for instance, a missing local relation's definition can be imported from the server to the workstation). This mechanism therefore permits complete definition of a new database on the server from the workstation-based structural model, and, symmetrically, definition of the structural model from the central database schema, although the definition is only partial in this direction (the designer still has to specify the connections among relations, since the central DBMS has no notion of the structural model). Fig. 3 summarizes the functionalities of the schema editor for the FACS database.

5.2. Hypertext-Based Information Retrieval and Manipulation

The expert flow-cytometry workstation demands intuitive human interfaces that maximize the flow of information between machine and user. Accordingly, we have applied hypertext authoring tools to the design of high-grade user-interface environments for PENGUIN's relational layer. The HyperCard interfaces exploit simultaneously and in synergy the browsing and direct-manipulation features of hypertext, the analytical-querying and concurrent-access features of relational DBMSs, and the structural model defined with the object-oriented editor.

PENGUIN imposes only four related constraints on the design of HyperCard stacks: Each HyperCard stack must correspond to a particular database, each background of a stack must correspond to a database relation, each field of a background must correspond to an attribute of a relation, and each card that shares a background must correspond to a record of a relation.

PENGUIN defines a hybrid interface system for retrieving information, combining the browsing capabilities of HyperCard for facile, underconstrained exploration and the analytical-querying capabilities of DBMSs for selective, multi-criteria search. PENGUIN's search system consists therefore of a declarative query language for downloading data from the server to the HyperCard stack, and of a combination of that same query language and of navigational access through dynamic links for searching locally in the stack.

5.2.1. Retrieval from the Database

Users retrieve records from the database by manipulating a visual query language. The basic idea is the following: For each background in the stack, PENGUIN provides a corresponding search card, where all database-retrieval

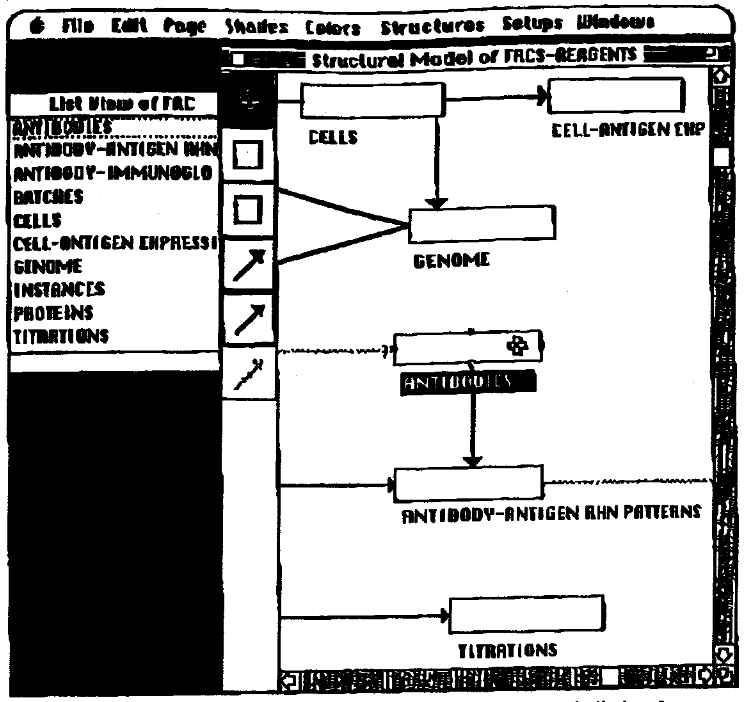
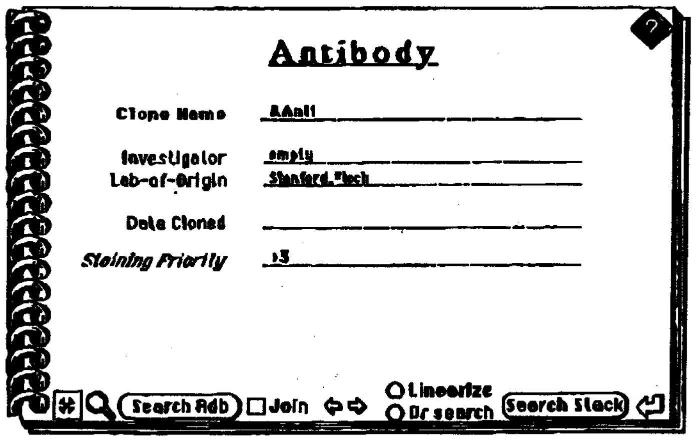


FIG. 3. A graphical representation of the database schema. The schema is displayed as a network of relations and structural connections in the right window; both relations and connections are mouse-sensitive objects, so the user can readily obtain more information about the domain by selecting any one object. From top to bottom, the palette in the right window contains the tools for selecting an object, for defining a new relation, for defining a new self-referencing relation (corresponding to a recursive schema), and for connecting two existing relations with an ownership, a reference, or a subset connection. The left window lists the relations in alphabetical order.

operations are specified and performed. Fig. 4 presents such a search card for the ANTIBODY background of Fig. 5, which in turn corresponds to a projection on the ANTIBODIES database relation, which is described in Appendix A. (Note that a background name in PENGUIN and the corresponding relation's name do not have to be identical.)

The interaction style is a mixture of form fill-in and direct manipulation. The user first enters search conditions in the form of declarative statements for any number of the search card's fields. Using simple point-and-click interactions, she can then directly manipulate the query (for example, changing the order of the Boolean connectives among the search conditions) before executing it. The visual query language accommodates single-relation queries as well as joins



Clone Name &Anti
Investigator empty
Lab-of-Origin Stanford,*tech
Date Cloned
Staining Priority >3

(a)

```
SELECT  Clone-Name,Investigator,Lab-of-Origin,Date-Cloned,Staining-Priority
FROM    ANTIBODIES e
WHERE   e.Clone-Name  STARTS_WITH "Anti" AND
        e.Investigator IS NULL AND e.Staining-Priority > 3 AND
        (e.Lab-of-Origin = "Stanford" OR e.Lab-of-Origin CONTAINS "tech")
```

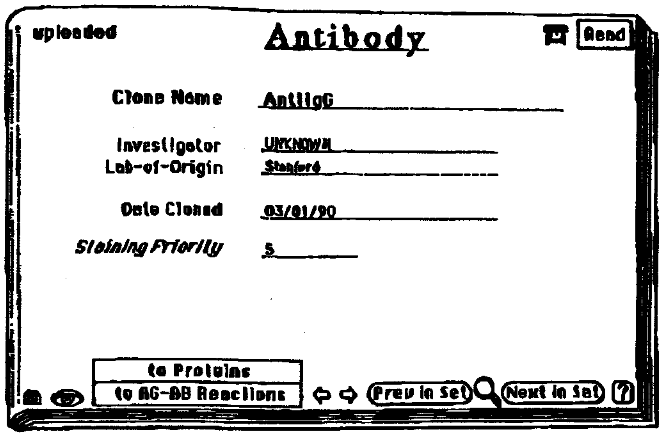
(b)

FIG. 4. Using the search card for ANTIBODY to query the database. (a) The search card contains the five fields defined for the ANTIBODY background. The Search Stack button allows the user to perform a search locally on the stack instead of remotely on the database (Search Rdb button). The Join button, when selected, defines queries over multiple relations. The magnifying-glass button gives access to a query log of the current session. Finally, the cross-like button terminates the session with the server. To specify a query, the user enters a search clause for each attribute by clicking on the corresponding field and typing the clause, which can contain literals and operators ("&" for starting with, "*" for contains, "empty" for is null, and ">" for greater than). Multiple clauses for the same field are separated by a comma. (b) The corresponding SQL query. Note that the default logical arrangement of the ANDs and ORs connectors can be changed easily through the use of the Linearize and Or search buttons.

over multiple relations, Boolean operations, and a variety of pattern-matching operators—thus offering a wide range of search options to query the database.

Let us demonstrate a single-relation query. Figure 4 illustrates a query against the ANTIBODY background. Once the query has been formulated declaratively, PENGUIN translates it into the correct database expressions, which are then sent to the database server for execution. After execution, the server transmits back to PENGUIN the database tuples that satisfy this query. For each tuple of the relation being queried, PENGUIN finally creates a new card in the corresponding background. Figure 5 displays such a card that presents one of the tuples satisfying the query of Fig. 4a.

Queries involving search on multiple relations simultaneously—join quer-



Clone Name AntiIgG
Investigator UNKNOWN
Lab-of-Origin Stanford
Date Cloned 03/01/90
Staining Priority 5

FIG. 5. The ANTIBODY background for a projection on the ANTIBODIES relation and its navigational capabilities. The Read button at the upper-right corner of the card allows the user to toggle between read and write modes. The eye-shaped button moves the user to the search card for this background, where she can specify all search operations. The newly created antibody card is part of the answer set of the query of Fig. 4. The Prev in Set and Next in Set buttons allow the user to look at the other cards of the answer set. ANTIBODY is connected to PROTEINS, and AG-AB REACTIONS, as shown by the corresponding buttons. By selecting To AG-AB Reactions, the user can directly retrieve all AG-AB REACTIONS cards, such that Clone-Name = "AntiIgG," without specifying any query expression.

ies—are also handled in a similar direct-manipulation style. As a result, our hypertext interface can facilitate the design of FACS staining protocols. For example, the relation CELL-ANTIGEN EXPRESSION can be queried to determine the surface markers on a target cell of interest. A subsequent query to ANTIBODY-ANTIGEN RXN PATTERNS lists the antibodies that avidly conjugate with that surface marker. A join query on INSTANCES and TITRATIONS returns the fluorochromes that bind to the selected antibody and indicates the locations and quantities of the solutions containing those combinations. In this fashion, information valuable in the design of staining protocols can be obtained from the database directly through investigators' queries or indirectly via a knowledge-based planning system. Section 7 provides more detailed examples of the system's use.

5.2.2. *Browsing through the Stack*

PENGUIN supports various ways of navigating through the HyperCard stack, once data have been transferred from the database. PENGUIN combines HyperCard's navigational access with the pattern-matching retrieval capabilities of a DBMS. We have extended HyperCard with a custom-designed search engine. As a result, the same query language is used consistently to retrieve information from the database and to search the HyperCard stack. No matter what the target of a search (the database or the stack) is, the result of that search defines a set of cards (newly created if the target was the database; already present if the target was the stack). PENGUIN then enables the user to look at the result set easily, as shown in Fig. 5, by using two buttons to move back and forth in that set.

The semantics of the structural model plays a critical role in supporting navigation through the stack. The structural connections establish dynamic links between cards of various backgrounds. By definition, two backgrounds related by a structural connection share some common fields; for example, ANTIBODY and AG-AB REACTIONS share the Clone-Name field. A dynamic link between cards of those backgrounds therefore corresponds to common data values for the shared fields. PENGUIN's browsing strategy follows from this characteristic.

As illustrated in Fig. 5, a background contains a button for every other background to which it is connected. In the context of a specific card, the user can now find relevant cards in another related background simply by clicking on the appropriate button; PENGUIN determines the fields involved in the connection, builds the query using the card's data values for those fields, and retrieves all corresponding cards in the second background. Note that more than one card will be found only if the connection being followed has a one-to-many cardinality. In that case, the same two-button mechanism as that used for scanning a query's answer set is applied to loop through the set.

5.3. Update Operations

PENGUIN supports all update operations—insert, delete, and modify—in addition to the retrieval capabilities. The user can perform update operations only by switching from read mode to write mode. A different set of buttons for performing updates subsequently appears on each data card.

Insertions and deletions occur in batch mode and require distinct transactions. New cards can be added to the stack at any time; those new tuples are not immediately inserted in the database. Instead, the user has to request an insertion transaction to add to the database all the tuples created since the last insertion operation. Cards created between two insertion transactions can therefore be modified or even deleted without consequence to the central database. Likewise, a deletion operation consists of two steps: (1) specification of a selection expression, using the same query language as that employed for data retrieval, and (2) request to delete all the database tuples that satisfy the selection expression. Unlike insertions and deletions, however, modifications occur on one tuple at a time during a standard retrieval transaction, where each modified tuple has been loaded from the database in the course of that session.

During every update operation, PENGUIN provides concurrency control

over the centralized database; multiple workstations can thus access the same database through the HyperCard interface and can protect the consistency of their respective transactions by setting and releasing locks on relations.

6. Consistency Checking

One rationale for the structural model is that it declaratively defines semantic integrity constraints over a database in BCNF. These constraints encode existence dependencies among tuples in different relations and ensure that the database remains semantically consistent over its active lifetime. Semantic inconsistencies result in data instances that are not useful or are incorrect.

Each SM connection encodes a characteristic set of integrity constraints (see Section 3.1). For example, the reference connection requires that a referenced tuple exist when a referencing tuple is inserted, and that the deletion of a referenced tuple be accompanied by corrective action on any corresponding tuple of the referencing relation (*14*). Violation of this constraint may result in tuples that reference nonexistent data instances, that is, a reference that provides no information.

6.1. Batch-Mode Versus Interactive-Mode Checking

Integrity constraints can be monitored and enforced automatically by consistency-checking programs that compare database states against constraints implied by the structural model. Such programs may operate in interactive or batch mode. If consistency checking occurs at update time (interactive mode), the user is notified that a requested transaction will cause an integrity violation. The user then is given the option of aborting the transaction or correcting the errant condition—for example, he can insert a missing tuple. If consistency checking occurs after multiple updates (batch mode), the existing database state is examined and all integrity violations are reported to the database administrator for subsequent correction.

Although batch-mode consistency checking avoids the extra computational cost at update time that is incurred with interactive-mode checking, it has the disadvantage of allowing inconsistent database states between batch runs. Inconsistent states may simply return incomplete data in response to queries. More seriously, they may lead to subsequent database states that appear consistent, but contain incorrect information. Existing inconsistent states may also be propagated across the database between batch-mode runs, requiring a recursive traversal of the SM graph to uncover all integrity violations. In contrast, interactive-mode consistency checking can be local to the relation being updated and is therefore more efficient. In our prototype implementation, we are running batch-mode consistency checking; however, we plan to upgrade to interactive consistency checking. Given our analysis of the database and of its usage (see Section 6.2), this should not introduce an intolerable delay during transactions.

We illustrate the importance of maintaining semantic integrity constraints as

$$\begin{tikzcd}
\text{OUTBRED-GENOME-LOCI} & \text{GENOME} \arrow[l, "*"'] & \text{CELLS} \arrow[l]
\end{tikzcd}$$

FIG. 6. A sample SM representation to illustrate inconsistent database states.

prescribed by the structural model. Our example uses a subset of the immunogenetics database that we have defined. The GENOME relation contains general attributes that are pertinent to all genome entities. The OUTBRED-GENOME-LOCI relation describes the identity and source of each allele at a heterozygous outbred genome locus. The CELLS relation references the source genome of specific cell identifiers. The structural model for this subset of the database is represented in Fig. 6.

In the absence of interactive consistency checking, a database state might arise where tuples in the relation OUTBRED-GENOME-LOCI exist without corresponding tuples in the owning relation GENOME. This state is a violation of the ownership-connection integrity constraint. The previously owned tuples are now attributes without an entity and, therefore, are semantically meaningless. Furthermore, the subsequent insertion of a GENOME tuple with a key that coincidentally matches that of the orphaned OUTBRED-GENOME-LOCI tuples results in semantically incorrect attributes for the inserted genome. At this point, the database is syntactically correct with respect to the SM and no consistency violations can be detected, but semantically incorrect information exists. The potential for generating undetectable errors in the database in this manner underscores the importance of *interactive-mode* consistency checking.

6.2. *Performance of Consistency-Checking Operations*

Consistency-checking at update time introduces an additional computational cost to each transaction. The amount of this cost depends on several factors:

1. The *connectedness of the structural model* for the database, that is, how many other relations, on average, must be checked for semantic consistency with the updated relation
2. The average *cardinality of the extensions* of the database, that is, how many tuples of the connected relations must be checked for existence dependencies
3. The *extent of indexing on the keys* of connected relations, that is, the cost of searching connected relations for corresponding tuples
4. The *frequency of update transactions* on the database, that is, how often consistency checking will be required
5. The general *performance characteristic of the DBMS* and of the hardware platform; that is how quickly the system can execute a given operation.

TABLE 1

AVERAGE TIME (IN SECONDS) REQUIRED TO VERIFY THE EXISTENCE OF A CORRESPONDING TUPLE IN A CONNECTED RELATION

	Number of join attributes							
	1		2		3		4	
Tuples in connected relation	Indexed	Not indexed	Indexed	Not indexed	Indexed	Not indexed	Indexed	Not indexed
100	0.17	0.30	0.25	0.39	0.32	0.45	0.40	0.52
1000	0.17	1.42	0.25	1.83	0.32	1.92	0.40	2.03
10000	0.17	13.96	0.25	15.85	0.32	NA	0.40	NA

Clearly, the larger, more connected, and less indexed a database is, the greater the performance degradation of consistency checking at update time will be. If update transactions are frequent, interactive consistency checking may increase the total operational load of the system significantly. The benefits of consistency checking with respect to database integrity must then be weighed against the operational costs of consistency checking with respect to users' needs.

We anticipate a minor performance degradation on implementation of interactive consistency checking for the immunogenetics database. With the exception of laboratory-specific data, the database encodes relatively static knowledge in the FACS staining domain; therefore, updates will be infrequent following the initial data entry. Moreover, although the SM is richly connected, the cardinalities of the database extension will be relatively small, and most key fields will be indexed.

We performed benchmark testing of our prototype consistency-checking module to determine the time required to search a connected relation for the existence of tuple. (The hardware platform and relational database that we used are described in Section 2.2.) The consistency-checking algorithms we implemented access the database via query statements embedded in a host language. To investigate the effects of various database parameters on the performance cost of consistency checking, we varied the cardinalities of database relations, the number of join attributes connecting relations, and the extent of indexing on the join attributes. The results, listed in Table 1, indicate that indexing of join attributes is critical in maintaining acceptable performance when interactive-mode consistency checking is performed.

7. PENGUIN IN IMMUNOGENETICS RESEARCH

Several sample queries follow that demonstrate the usefulness of PENGUIN's database and its hypertext interface to immunogenetics laboratory researchers. The relations and attributes appearing in the queries are part of the sample database scheme in Appendix A.

1. Suppose an investigator has a sample of B cells and wishes to sort the CD5+ cells from the CD5− ones. The required query returns a set of surface determinants specific to the CD5+ portion of the B-cell population. Although this is a straightforward concept, it requires a fairly complex query including several nested subqueries, as shown by the corresponding SQL expression:

```
SELECT Protein-Name
FROM   Cell-Antigen-Expression X, Cells C
WHERE  (X.Cell-Id# = C.Cell-Id#)
       AND (Cell-Type = "B-Cell")
       AND (X.Cell-ID# IN (SELECT Cell-Id#
                           FROM   Cell-Antigen-Expression
                           WHERE  (Protein-Name = "CD5"))
       AND (Protein-Name NOT IN
                          (SELECT Protein-Name
                           FROM   Cell-Antigen-Expression X, Cells C
                           WHERE  (X.Cell-Id# = C.Cell-Id#)
                                  AND (Cell-Type = "B-Cell")
                                  AND (X.Cell-Id# NOT IN
                                       (SELECT Cell-Id#
                                        FROM   Cell-Antigen-
                                               Expression
                                        WHERE  (Protein-Name
                                               = "CD5")))))
```

From an end-user perspective, this query is quite cumbersome, so we define it instead as a relational view on the database. Such views can then be handled by PENGUIN's HyperCard interface.

2. Suppose that the surface antigen MAC-1 is among the retrieved proteins specific to the target population (CD5+ B cells). The investigator now wants to select an antibody reacting with the MAC-1 surface determinant that she can use for sorting the CD5+ B cells. The following query returns a set of antibodies that bind to MAC-1 with high affinity and are available in a form conjugated to the fluorescent dye "fluorescein":

```
SELECT Clone-Name, Batch#, Instance#
FROM   Antibody-Antigen-Rxn-Patterns A, Instances I
WHERE  (A.Protein-Name = "MAC-1")
       AND (A.Rxn-Strength = "HIGH")
       AND (I.Fluor-Label = "fluorescein")
       AND (A.Clone-Name = I.Protein-Name)
```

3. Finally, the investigator needs a list of the currently available titrations for the selected antibody-dye combination. Suppose that, from Query 2, she chooses the combination represented by Protein-Name = "antiMAC-1" and Fluor-Label = "fluorescein." In addition, she is only interested in titrations that were derived from antibody-dye instances prepared after January 1. The

following query returns this information, sorted by Batch#, Instance#, and Titration#:

```
SELECT F.Batch#, F.Instance#, Titration#, Date-Titrated, Default-Amt-to-
       Use
FROM   FACS-Titrations F, Instances I
WHERE  (I.Protein-Name = "antiMAC-1")
       AND (I.Fluor-Label = "fluorescein")
       AND (I.Date-Created > "Jan-1-1991")
       AND (I.Protein-Name = F.Protein-Name)
       AND (I.Batch# = F.Batch#)
       AND (I.Instance# = F.Instance#)
       AND (I.Titration# = F.Titration#)
ORDER BY F.Batch#, F.Instance#, Titration#
```

All three queries can be issued directly via the DBMS query interpreter. Alternatively, the investigator can use PENGUIN's hypertext interface, which generates the query expressions automatically. Query 1 would then be issued from a search card similar to the one shown in Fig. 4; on the other hand, Queries 2 and 3, which are join queries, would be specified using the join editor, as described in (5).

We used a test database to benchmark those three queries. The schema corresponds to that described in Appendix A, and the extensions of the relations involved are as follows:

PROTEINS: 200 tuples. 25% of the proteins are antibodies; these same antibodies also appear in reagent preparations (INSTANCES and TITRATIONS).

ANTIBODY-ANTIGEN-RXN-PATTERNS: 150 tuples. Each of the 50 antibodies binds to an average of three antigens, each with varying affinity.

CELLS: 200 tuples. 25% are B-cells; 20% of these express the antigen CD5 (i.e., 5% of all cells). The same 20% express the antigen MAC-1 (again 5% of all cells).

CELL-ANTIGEN-EXPRESSION: 670 tuples. Each cell expresses an average of three "random" proteins. In addition, 50 cell types express IgM, 10 express CD5, and 10 express MAC-1.

INSTANCES: 1000 tuples. This number is based on 50 reagent proteins, two batches per protein, and 10 fluorochrome conjugations per batch.

TITRATIONS: 5000 tuples. There are five titrations per instance.

In PENGUIN's client–server environment (described in Section 2.2), the cost of executing a query can be broken down into five components: (1) translation from the HyperCard representation to the database query language, (2) transmission of the query to the server, (3) query execution by the DBMS, (4) transmission of query results back to the client, and (5) result processing in HyperCard. Our experience indicates that the query-execution and result-processing times usually exceed the overhead associated with the translation and transmission steps (Components 1, 2, and 4). Performance figures for Component 3—execution of the sample queries on the database server—appear below:

	Query 1	Query 2	Query 3
Elapsed time	02:10.96	00:02.87	00:02.66
CPU time	02:06.56	00:00.73	00:00.82

Not surprisingly, the complexity of Query 1 results in a substantial load on the database server; Queries 2 and 3, on the other hand, perform well, even though both involve multiple relations.

Finally, once the data satisfying a request has been transmitted to a client workstation, it is processed in HyperCard (Component 5). This formatting step takes on the order of 1.5 to 2 sec per record in the answer set on a basic Macintosh II running HyperCard 1.1. In the context of ad hoc probing and browsing operations such as the ones presented here, we believe that PENGUIN's enhanced capabilities outweigh the performance penalty that HyperCard imposes. In addition, more recent hardware and software environments together with optimization of the result-processing code should significantly decrease the run time of this component, which, at the moment, represents the main operational bottleneck.

8. Discussion and Conclusion

The PENGUIN system is driven by the semantics of the structural model. The structural model adds to the relational model the concept of connections between pairs of relations. The three connection types (ownership, reference, and subset) describe semantic relationships among entities, embody formal structural constraints, and specify consistency rules in the face of update operations on the database. The versatility of the structural model has proved essential in our work, as we have exploited the semantics of the connections in various ways:

• *Database design*. The three types of connection have the expressive power to model the structure of any database and to represent complex relationships. The structural model for the flow-cytometry database illustrates nontrivial relationships among immunologic entities. In addition, the structural model supports the integration of multiple user views into a consistent, comprehensive schema, which, in turn, can serve multiple objectives.

• *Schema exploration*. Because the relational model is machine-oriented rather than user-oriented, schemas for relational databases tend to be opaque and difficult to understand. This is particularly true for large databases that can comprise hundreds of relations. By expressing the schema in a more easily understandable, graphic format, the structural model alleviates this problem. Through the use of an object-oriented structural-model editor, the user can

now employ direct-manipulation tools to define, manipulate, and explore the database easily.

• *Data retrieval and browsing*. Relational query languages, such as SQL, offer sophisticated ad hoc retrieval capabilities. Yet, they are complex to use and demand a detailed knowledge of the database schema. The semantics of the structural model can facilitate the processes of formulating a query and of manipulating the body of data returned by this query. We presented PENGUIN's hypertext interface that incorporates such features as a join editor, which can build a join query from just the relations involved (that is, it derives the specification of the join attributes from the structural model), and a browser, which establishes dynamic links along the structural connections to allow facile navigation in the data set.

• *Integrity maintenance*. Relational DBMSs have integrity-maintenance capabilities that are usually limited to the definition of local restrictions on the domain of attributes. The structural model, on the other hand, has the sufficient descriptive knowledge of a database to support dynamic enforcement of global database consistency. When applied to a specific schema, the domain-independent rules associated with each type of connection define complex relationships spanning different parts of the database. As a result, an update transaction on a single relation can trigger a number of corrective actions to guarantee the integrity of the database as a whole. Alternatively, the transaction can be simply rejected.

• *Object generation*. As we discussed briefly in Section 2, the structural model provides the foundational semantics for generating and manipulating dynamically objects in the object layer.

Note, finally, that, because of its simplicity, the structural model can be stored in a relational database. We therefore have a coherent representational framework for both extensional information (the raw data) and intensional information (the relational schema extended with the structural model).

APPENDIX A: Immunogenetics Database Schema

A relevant subset of the immunogenetics database schema that we designed is described in this appendix. Each relation assumes the form:

RELATION-NAME (Description of relation contents)
Key Attributes:>
Dependent Attributes

For a listing of the complete database schema, please contact the authors.

PROTEINS (All protein structures appearing in database)
Protein-Name:>
Protein-Species, Protein-Class, Supergene-Family,
Bib-Reference, Molecular-Wt, Isoelectric-Pt., Engineered?.

ANTIBODIES (Information regarding proteins that are antibodies)
Clone-Name:>

Immunogen, Parent, Cell-Type, Protocol, Hybridization-Process, Fusion-Partner, Drug-Labeling, Date-Cloned, Lab-of-Origin, Investigator, Maintenance-Chars, Staining-Priority.

ANTIBODY-ANTIGEN-RXN-PATTERNS (Description of reaction between antibody and antigen proteins)

Clone-Name, Protein-Name:>

Rxn-Strength, Affinity, Comments, References.

ANTIBODY-IMMUNOGLOBULIN-RXN-PATTERNS (Description of reaction between antibody and immunoblobulin)

Antibody-Clone-Name, Target-Clone-Name:>

Rxn-Chain, Rxn-Type, References.

BATCHES (Stock solution of an antibody)

Protein-Name, Batch#:>

Procedure, Parent-Batch#, Produced-By, Managed-By, Date-Created, Location, Total-Amt-Produced, Amt-Remaining, Target-Amt, Mg-per-Ml-Flag.

INSTANCES (Stock solution of antibody conjugated with fluorochrome)

Protein-Name, Batch#, Instance#:>

Fluor-Label, Label/Protein-Ratio, Brightness, Fluor/Molecule, High-Background?, High-Background-Source, Managed-By, Date-Created, Location, Total-Amt-Produced, Amt-Remaining, Target-Amt, Mg-per-Ml-Flag, Stock-Conc.

TITRATIONS (Antibody-fluorochrome titrated for specific experiment)

Protein-Name, Batch#, Instance#, Titration#:>

Date-Titrated, Titrated-By, Default-Amt-to-Use, Default-Vol-to-Use, Target-Cell-Type, Amt/Cell-to-Use.

GENOME (General description of all genomes in database)

Genome-Id#:>

Species-Name, Genome-Name, Genome-Type, Source, Obtained-By, Transgene-Present?.

INBRED-GENOME-LOCI (Homozygous allele at each inbred genome locus)

Genome-Id#:>

Allele.

OUTBRED-GENOME-LOCI (Maternal or paternal source for each outbred genome locus)

Genome-Id#, Locus:>

Allele-1, Allele-2, Allele-1-Source, Allele-2-Source.

CELLS (Reference relation for cell Id#s)

Cell-Id#:>

Cell-Type, Genome-Id#.

CELL-ANTIGEN-EXPRESSION (Specifies cytoplasmic or surface expression of antigen)

Cell-Id#, Protein-Name:>

Expression-Type.

Acknowledgments

The research described in this article was done while the first author was at Stanford University. We thank Wayne Moore for his help in designing the immunogenetics database schema. The implementation of PENGUIN would not have been possible without the assistance of R. Martin Chavez. We also thank Lyn Dupré for her editorial comments. This work was supported by the National Library of Medicine under Grant R01 LM04836, and by the National Institute of Health under Grant HD01287. Additional computer facilities were provided by the SUMEX-AIM resource under NIH Grant RR-00785.

References

1. Barsalou, T. "View Objects for Relational Databases." Ph.D. thesis, Medical Information Sciences Program, Stanford University, Stanford, CA, 1990.
2. Barsalou T. An object-based architecture for biomedical expert database systems. *In* "Proceedings of the Twelfth Symposium on Computer Applications in Medical Care" (R. A. Greenes, Ed.), pp. 572–578. Inst. Elec. Electron. Eng., Washington, DC, 1988. Reprinted in *Comput. Methods Prog. Biomed.* 30, Nos. 2/3, 157 (1989).
3. Barsalou, T., Chavez, R. M., and Wiederhold, G. Hypertext interfaces for decision-support systems: A case study. *In* "Proceedings of MEDINFO '89" (B. Barber, D. Cao, D. Qin, and G. Wagner, Eds.), pp. 126–130. IMIA, IFIP, Singapore, 1989.
4. Barsalou, T., Moore, W. A., Herzenberg, L. A., and Wiederhold, G. A database system to facilitate the design of FACS experiment protocols (abstract). *Cytometry* 97 (1987).
5. Barsalou, T., and Wiederhold, G. A cooperative hypertext interface to relational databases. *In* "Proceedings of the Thirteenth Symposium on Computer Applications in Medical Care" (L. C. Kingsland III, Ed.), pp. 383–387. Inst. Elec. Electron. Eng., Washington, DC, 1989.
6. Barsalou, T., and Wiederhold, G. Knowledge-directed mediation between application objects and base data. *In* "Data and Knowledge Base Integration" (S. M. Deen and G. P. Thomas, Eds.), Chap. 4. Pitman, London, 1990.
7. Barsalou, T., and Wiederhold, G. Complex objects for relational databases. *Comput. Aided Design* 22, No. 8, 458 (1990).
8. Bic, L., and Gilbert, J. P. Learning from artificial intelligence: New trends in database technology. *IEEE Comput.* 19, No. 3, 44 (1986).
9. Bonner, W. A., Hulett, H. R., Sweet, R. G., and Herzenberg, L. A. Fluorescence-activated cell sorting. *Rev. Sci. Instrum.* 43, 404 (1972).
10. Brodie, M. L., and Mylopoulos, J. Knowledge bases vs. databases. *In* "On Knowledge Base Management Systems" (M. L. Brodie and J. Mylopoulos, Eds.), pp. 83–86. Springer-Verlag, New York, 1986.
11. Codd, E. F. A relational model for large shared data banks. *Commun. ACM* 13, No. 6, 377 (1970).
12. El Masri, R., and Wiederhold, G. Data model integration using the structural model. *In* "Proceedings of the International Conference on Management of Data," pp. 191–202. ACM-SIGMOD, Boston, MA, 1979.
13. Goodman, D. "The complete HyperCard handbook." Bantam, New York, 1987.
14. Keller, A. M. "Updating relational databases through views." Ph.D. thesis, Department of Computer Science, Stanford University, Stanford, CA, 1985.
15. Kershberg, L. (Ed.). "Proceedings of the First International Workshop on Expert Database Systems." Benjamin Cummings, Redwood City, CA, 1986.
16. Parks, D. R., Lanier, L. L., and Herzenberg, L. A. Flow cytometry and fluorescence-activated cell sorting. *In* "The Handbook of Experimental Immunology" (C. C. Blackwell,

D. M. Weir, L. A. Herzenberg, and L. A. Herzenberg, Eds.), Chap. 29. Blackwell Sci., Edinburgh, 1985.

17. WIEDERHOLD, G. "Database Design." Computer Science Series. McGraw-Hill, New York, 1983.

18. WIEDERHOLD, G. Views, objects and databases. *IEEE Comput.* 19, No. 12, 37 (1986).

19. WIEDERHOLD, G. The architecture of future information systems. *In* "Proceedings of the International Symposium on Database Systems for Advanced Applications." KISS, IPSJ, Seoul, 1989.

20. WIEDERHOLD, G., BARSALOU, T., AND CHAUDHURI, S. "Object Management in a Relational Framework." Technical Report STAN-CS-89-1245, Department of Computer Science, Stanford University, 1989.

21. WIEDERHOLD, G., AND EL MASRI, R. The structural model for database design. *In* "Entity-Relationship Approach To System Analysis and Design," pp. 237–257. North-Holland, Amsterdam, 1980.

22. WOLF, C. G., AND RHYNE, J. R. "A Taxonomic Approach to Understanding Direct Manipulation." Technical Report RC 13104, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1987.

COMPUTERS AND BIOMEDICAL RESEARCH 24, 499–508 (1991)

Editorial: The Role of a Clinically Based Computer Department of Instruction in a School of Medicine

WILLIAM S. YAMAMOTO*

Department of Computer Medicine, George Washington University School of Medicine, Washington, DC 20037

Received August 6, 1990

The evolution of activities and educational directions of a department of instruction in medical computer technology in a school of medicine are reviewed. During the 18 years covered, the society at large has undergone marked change in availability and use of computation in every aspect of medical care. It is argued that a department of instruction should be clinical and develop revenue sources based on patient care, perform technical services for the institution with a decentralized structure, and perform both health services and scientific research. Distinction should be drawn between utilization of computing in medical specialties, library function, and instruction in computer science. The last is the proper arena for the academic content of instruction and is best labelled as the philosophical basis of medical knowledge, in particular, its epistemology. Contemporary pressures for teaching introductory computer skills are probably temporary. © 1991 Academic Press, Inc.

INTRODUCTION

The report of the American Association of Medical Colleges (AAMC) on the General and Professional Education of Physicians (GPEP) (*1*) recommends, among other things, the establishment of an academic focus for computer science in the medical school. A compendium of computer activities in medical education in 1985 lists such activities in various medical schools but omits this institution (*2*). At George Washington University we are, however, completing the second decade of an experiment to foster and establish instruction and care support in information and engineering technology relevant to medical care. A review of the history (1973–1989) of this effort may be useful to those interested or involved in responding to the AAMC analysis. A catechism may serve as the appropriate vehicle for discussion of the educational role of the Department of Computer Medicine at George Washington University and consequent perspectives on the role of medical information sciences in medical education and medical practice.

* With grateful thanks for assistance from Victor Fernandez, MD.