VIRTUAL INFORMATION

IN THE

INFOPLEX DATABASE COMPUTER

by

LAWRENCE ABRAM KRAKAUER

B.S.E., Princeton University
(1978)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1980

Signature of Author_____
Sloan School of Management
May 21, 1980

Certified by_____
Stuart E. Madnick
Thesis Supervisor

Accepted by_____
Michael S. Scott Morton
Chairman, Department Committee

VIRTUAL INFORMATION

IN THE

INFOPLEX DATABASE COMPUTER

by

LAWRENCE ABRAM KRAKAUER

Submitted to the Sloan School of Management
on May 21, 1980 in partial fulfillment of the
requirements for the Degree of Master of Science in
Management

ABSTRACT

The concept of virtualness is extended to information. Virtual information is, informally speaking, information that is derived from other information. The term virtual information was proposed in 1973 in a paper by Folinus, Madnick, and Schutzman{R7}. We extend that paper by defining virtual information in a precise manner. Database concepts related to virtual information (although typically not called virtual information in the literature) are reviewed. These concepts include data typing, data type abstractions, security, data independence, and multiple views of data. Finally, the function of virtual information in the INFOPLEX database computer is discussed{R15}.

Thesis Supervisor: Dr. Stuart E. Madnick

Title: Associate Professor of Management Science

Table of Contents

Introduction

The concept of virtualness plays a prominent role in the world of computers. For example, the VM/CMS operating system contains virtual memory, virtual machines, virtual disks, virtual card readers, virtual card punches, and virtual printers{R26}. In this paper, we extend the concept to information. The term virtual information was proposed in 1973 in a paper by Folinus, Madnick, and Schutzman{R7}. We extend that paper by defining virtual information in a precise manner. Database concepts related to virtual information (although typically not called virtual information in the literature) are reviewed. Finally, the function of virtual information in the INFOPLEX database computer is discussed{R15}.

In chapter 1, the concept of virtualness is described by example. Then a model of information is presented and virtual information is defined in terms of that model. An analogy is drawn between virtual information and the concepts of data type abstraction and data independence. Chapter 2 describes one application of virtual information in a non-DBMS context, and several applications in a DBMS context. Chapter 3 relates the ideas presented in the first and second chapters to the INFOPLEX data base computer.

Chapter 1. Virtual Information

The Concept of Virtualness

Bruce sells watches with leather and metal bands. A customer asks Bruce for a brand X watch with a metal band. Bruce has a brand X watch with a leather band, but he has a spare metal band in a drawer. What is Bruce's reply?

Mary-Louise is invited to John's party on the condition that she brings dessert. She has flour, sugar, eggs, vanilla, baking soda, salt, chocolate chips, and her mother's recipe for chocolate chip cookies. Can she attend the party?

David is also invited to John's party. He has no ingredients and no recipe (although his mother is a great cook), but he does have $10. The bakery is a two minute walk from David's front door. Can he attend the party?

In each of the above stories the answer to the posed question is effectively yes, conditional on there being sufficient time to complete the obvious transformation. We say that Bruce has a virtual brand X watch with a metal band. Mary-Louise and David have virtual cookies. For

example, as far as the customer is concerned, Bruce has a brand X watch with a metal strap. The customer is concerned only with results, not implementation.

The advantages of utilizing the concept of virtualness are obvious. For example, Bruce would have to keep a larger inventory if he kept "actual" watches only. David would be stuck with chocolate chip cookies if he had stocked actual cookies and if John had requested raisin cookies instead of just dessert. On the other hand, the major disadvantage is the speed at which each person can respond to a request.

What makes Mary-Louise's cookies virtual and when do they become edible? This question implies that cookies are more desireable in one form than another. We will refer to the more desireable, edible, concrete, useable form of the cookie as the materialization. The less tasty form of the cookie is the basis. Specifically, the cookie's basis comprises the ingredients. The recipe is the prescription for materializing the cookies from the ingredients. Of course, the ingredients themselves might be virtual, in which case they would be composed of even more basic ingredients. The most basic set of ingredients is referred to as the cookie's primary basis, everything else is

derived.

Cookies become edible when they are materialized, but what made them virtual in the first place? Suppose we kept an edible cookie in a drawer, and no longer kept ingredients around. The cookie is not in a usable form (not yet materialized) until the drawer is opened. However, there is a materialized cookie from the point of view of the ant in the drawer. The cookie outside the drawer is derived from the cookie inside the drawer, but the cookie outside the drawer is not a virtual cookie because the respective materializations are identical.

As another example, suppose that cookies are made from ingredients, but are temporarily stored in the drawer before being consumed. Let CI and CO denote cookies inside and outside of the drawer respectively. The ingredients, I, compose the primary basis for CO and CI, and the basis for CI. From previous examples, CO is not virtual with respect to its basis, CI, while CI is virtual with respect to its basis, I. It seems unreasonable, however, to say that CO is not virtual when its basis is. Therefore CO is virtual as well.

A Model of Information

We postulate the existence of bit strings called atomics. Atomics may be created, destroyed, and tested. An object t, denoted by O^t, is simply the name used to refer to an atomic. The association, or binding, of an object to an atomic gives a semantic meaning to the atomic and therefore contains information. For example, "Mike's age" is an object and 24 is an atomic. (We will use more descriptive representations than bit strings to describe atomics.) Binding "Mike's age" to 24 produces the information that Mike is 24. An atomic that is bound to an object t is called the materialization of O^t and is denoted by $\uparrow O^t$. The binding process is called the materialization process.

The database is the set of all objects. Suppose the database comprises q objects, $\{O^1,\ldots,O^q\}$. We say that $\uparrow O^i$ is independent of $\uparrow O^j$ iff for every set of materializations $\{\uparrow O^1,\ldots,\uparrow O^q\}$, no change in $\uparrow O^j$ results in a change in $\uparrow O^i$. If $\uparrow O^i$ is not independent of $\uparrow O^j$ then $\uparrow O^i$ is said to be derived from $\uparrow O^j$. If $\uparrow O^i$ is independent of all other materializations of objects in the database then $\uparrow O^i$ is said to be a basic object. The materialization of a basic object is the binding of the object to an atomic. The materialization of a derived object is the process of

mapping the materializations of other objects onto a new atomic and binding that atomic to the object. For example, consider the objects "Mike's birth year", "Mike's age", and "current year". The materialization of the object "Mike's birth year" is the process of mapping the materialization of Mike's age and the current year onto an atomic and binding that atomic to "Mike's birth year".

The materialization of a derived object is defined in terms of other objects, called the object's basis. The basis of O^t is denoted by B^t. More precisely, B^t is an ordered set of objects, $B^t=\{O^1,\ldots,O^p\}$ where $p\geq 0$. The size of the basis is $p=S^t$. The i^{th} member of B^t is called a component of O^t and is denoted by $r^{t,i}$. For example, "Mike's age" is the first component of "Mike's birth year". The materialization of B^t, $\uparrow B^t$, is defined to be $\{\uparrow O^1,\ldots,\uparrow O^p\}$. The materialization of two bases, $\uparrow B^u$ and $\uparrow B^v$, are identical iff $S^u=S^v$ and $\uparrow r^{u,i}=\uparrow r^{v,i}$ for i in $[1,S^u]$. The rule for materializing $\uparrow O^t$ from $\uparrow B^t$ is called the materialization mapping, M^t.

For O^i to be a component of O^t, we require that $\uparrow O^t$ be derived from $\uparrow O^i$. The rationale is that it doesn't make sense to define $\uparrow O^t$ in terms of objects whose materializations are completely unrelated to $\uparrow O^t$. Another requirement imposed on the components of O^t is that the

components must form a complete basis for O^t. A basis, B^t, is complete if the following is true. If O^t is materialized at two different instances to produce $\uparrow O^{t1}$ and O^{t2}, then $\uparrow O^{t1}=\uparrow O^{t2}$ iff $\uparrow B^{t1}=\uparrow B^{t2}$. Hence $\uparrow O^t$ is said to be completely determined by $\uparrow B^t$. A necessary condition for completeness is that all objects that $\uparrow O^t$ is derived from are either in $\uparrow B^t$ or are objects that $\uparrow B^t$ is derived from.

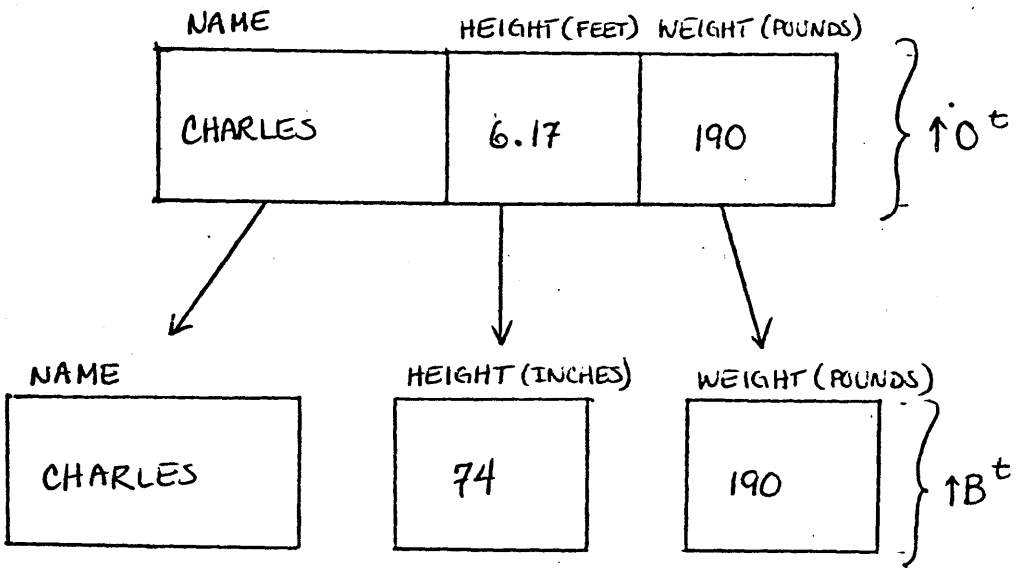
An object graph is used to show the relationships between objects. An object graph comprises nodes and directed edges. Each node, N^t, corresponds to an object, O^t. A directed edge, $E^{u,v}$, leads from N^u to N^v iff O^v is a component of O^u. When drawing object graphs, we will use the convention that the leftmost child of a node N^t corresponds to the first component of N^t and that components two through S^t follow from left to right. Therefore, B^t corresponds to the children of N^t. Cycles in the graph are not permitted. A leaf is a node with no edges leading from it. A node, N^t, is a leaf iff N^t is basic, otherwise N^t is derived. The primary basis of O^t is the set of all basic objects whose nodes are descendents of N^t in the graph.

We define a cell in terms of its materialization. The materialization of a cell is a substring of an atomic. If we can decompose $\uparrow O^t$ into a set of cells $\{C^{t,1},\ldots,C^{t,p}\}$, where $p=S^t$, such that each $\uparrow C^{t,i}$ is completely determined (in the sense used earlier) by $\uparrow r^{t,i}$, then we say that O^t is cellular. An example of a cellular object and a non-cellular object is shown in figure 1. The materialization of a cellular object, O^t, is defined to be identical to its basis, $\uparrow B^t$ ($\uparrow O^t=\uparrow B^t$), iff $\uparrow C^{t,i}=\uparrow r^{t,i}$ for each i in $[1,S^t]$. If an object, O^t, is not cellular then by definition, $\uparrow O^t\neq\uparrow B^t$.

Figure 2 shows an instance of two tables (relations), the corresponding object graph, and the materialization of each object. Objects are connected to their materializations by dashed lines. The tables are the PERSON and YEAR tables. Some nodes are labelled with example materializations, instead of names, to avoid the tedious process of naming each object. The (materialization of the) person's age is derived from (the materialization of) her birth year and the current year. If the current year were to change to 1981, then Joanne's age would materialize as 16. Note that all objects are cellular, except for '15' and '14'.

Figure 1

THE MATERIALIZATIONS OF A CELLULAR OBJECT AND ITS COMPONENTS



THE MATERIALIZATIONS OF A NON-CELLULAR OBJECT AND ITS COMPONENTS

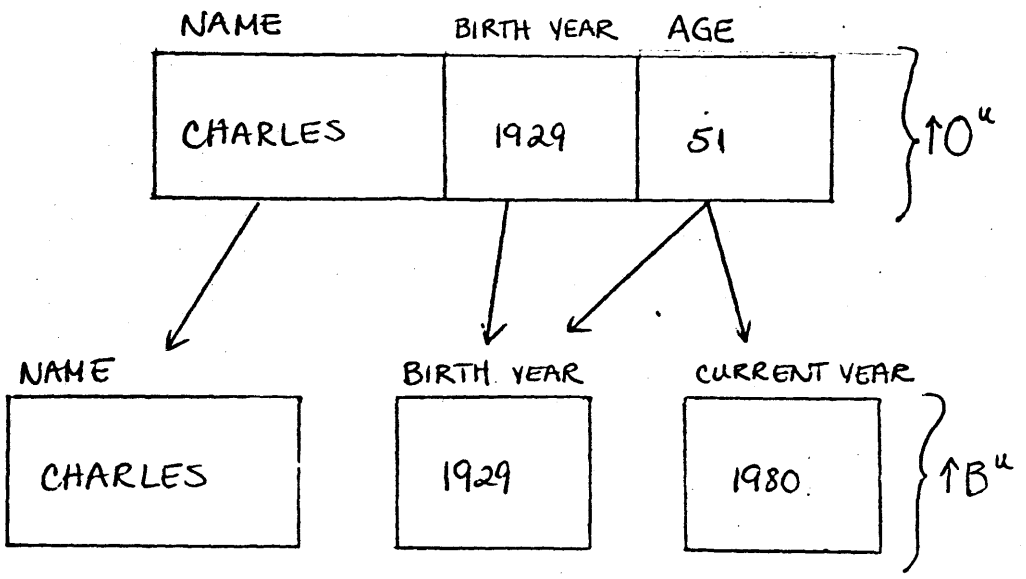


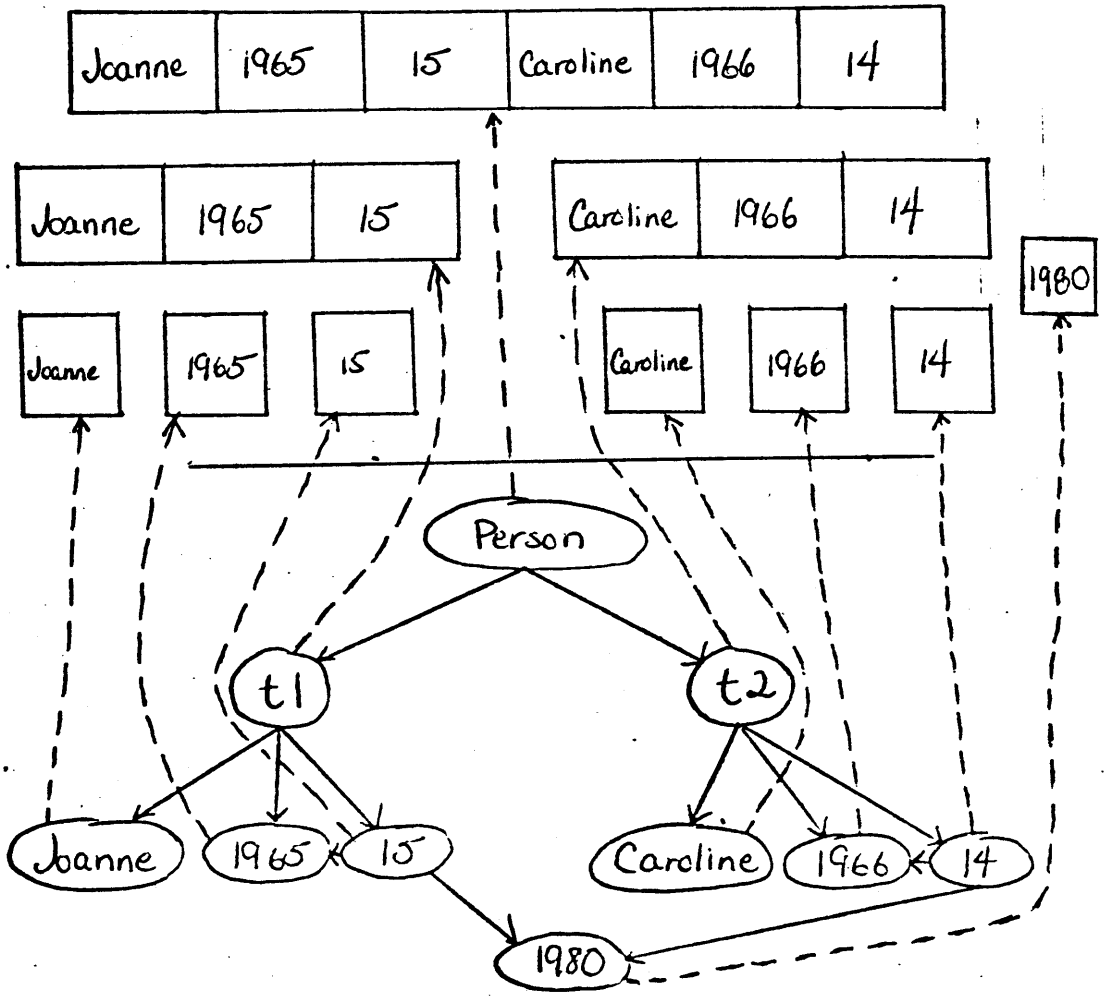
Figure 2

THE PERSON AND YEAR TABLES

	NAME	BIRTH YEAR	AGE
t1	Joanne	1965	15
t2	Caroline	1966	14

YEAR
1980

MATERIALIZATIONS OF OBJECTS



Object Graph

We are now in a position to define virtual information. Object O^t is virtual iff one of the following two statements is true.

(i) $\uparrow O^t \neq \uparrow B^t$

(ii) $\uparrow r^{t,i}$ is virtual for some i in $[1,S^t]$

Thus, O^t is virtual if its materialization is not identical to the materialization of its basis or if an object in its basis is virtual. If statement (i) is true then O^t is said to be virtual with respect to B^t. In figure 2, the objects '15' and '14' are virtual with respect to their bases. Objects t1, t2, and PERSON are virtual, but not with respect to their bases.

As a corollary to the definition of virtual information, we show that O^t is virtual if it is derived from a virtual object, O^u (although not necessarily completely described by O^u). If O^t is derived from O^u then a path must exist between N^t and N^u in the object graph. A path from N^t to N^u, $P^{t,u}$, is an ordered set of directed edges leading from N^t to N^u, where $P^{t,u}=\{E^{t,i1},\ldots,E^{ik,u}\}$. From the definition of virtual information, O^{ik} is virtual because O^u is virtual. Also, O^t is virtual if O^{i1} is virtual. Therefore, by induction, O^t must be virtual.

For another example of virtual information, consider the relations PAY and VPAY in figure 3. Each tuple (O^{t1},O^{t2},etc.) of PAY comprises a basic name object and a basic salary object. Suppose we want to study salary distributions within an organization. Since salary information is confidential, and executive salaries themselves are top secret, we define a new relation consisting only of the salaries under $50,000. In System R {R6} VPAY could be defined as follows.

```
DEFINE VIEW VPAY AS:
SELECT SALARY
FROM PAY
WHERE SALARY < 50000
```

The corresponding object graph and example materialization are also in figure 3. The System R statements define B^{VPAY} as O^{PAY} and M^{VPAY} in terms of the salary cell of $\lceil O^{PAY}$. The only virtual object in figure 3 is O^{VPAY}. The reason that O^{VPAY} is virtual is that, although it is cellular, $\lceil O^{VPAY} \neq \lceil O^{PAY}$. We can generalize this observation by noting that if the information embodied in the materialization of an object, $\lceil O^t$, is a subset of the information embodied in $\lceil B^t$, then O^t is a virtual object.

PAY

	NAME	SALARY
t1	Chris	12,000
t2	Marcia	9,211
t3	Frans	6,456
t4	Grover	11,592
t5	George	57,296

VPAY

SALARY
12,000
9,211
6,456
11,592

The Relations PAY and VPAY

Materializations of Objects
(not all shown)

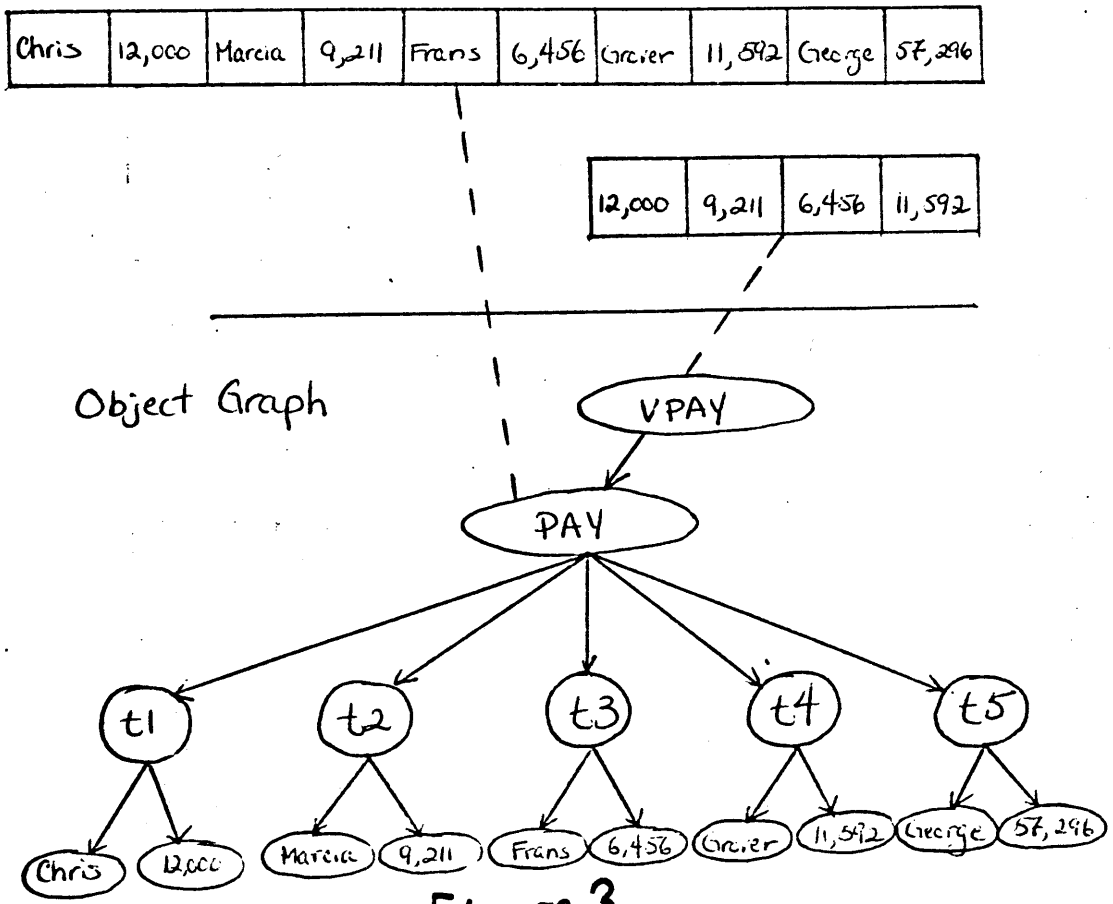


Figure 3

Virtual Information: A Functional Perspective

Virtual information enables construction of new objects from existing objects, to make the new objects appear as real as the existing objects. This is analagous to data type abstraction in programming languages{R18,R20}. Data type abstraction enables construction of new data types from existing data types, to make the new data types appear as real as the existing data types. We proceed to devlop this analogy further.

Liskov and Zilles define an abstract data type to be "a class of abstract objects which is completely characterized by the operations available on those objects"{R18}. In other words, to define an abstract data type one simply defines

(i) the operations that can manipulate the data type,

(ii) the internal representation of the data type, and

(iii) the implementation of each operation.

A programming language with an abstract data type capacity (such as CLU{R20} and, to a lesser extent, PASCAL{R25}) provides a set of primitive data types (for example, real, integer, boolean, etc.) to enable definition of new data types. Abstract data types can also be used to define new abstract data types. As an example, consider the definition of an integer stack{R18}. First, the relevant operations are enumerated: push, pop, looktop, erasetop, empty(the stack). Second, the internal representation is specified to be an array of integers with a pointer to the top of the stack. Third, each operation is defined in terms of manipulation of the stack and the pointer to the top of the stack.

Implementation details are very important when defining an abstract data type. However, when using the data type only the behavior -- the functionality -- of the data type is of importance. The programmer needs only to know how to manipulate the data type. His task is simplified by not being complicated by implementation details. The result should be that programs will be easier to write and maintain{R18}.

Virtual information is analagous to data type abstraction both in terms of implementation and in terms of benefits. The only difference is that there is no data type abstraction concept analagous to non-virtual information. A virtual object is completely characterized by operations on that object (as defined by the materialization mapping). We therefore take a functional perspective of virtual information. To investigate this perspective, it is useful to view information along a spectrum from pure data to pure algorithm{R7}. According to {R7}, "recorded facts which are independent of other information in the data base may be considered as pure data." In terms of our definition, basic objects are pure objects. An example of pure algorithm is the trigonometric sine function. In between pure data and pure algorithm lies derived information. A derived object is a combination of pure data and pure algorithm. A virtual object is a derived object with the additional quality that the materialization of the object's basis not be identical to the materialization of the object itself. Suppose employee salary information is represented by a table comprising tuples that comprise three fields: the employee name (EMPN), the employee's weekly salary in dollars (WSAL), and the employee's annual salary in dollars (ASAL). Although a functional dependency exists between $\lceil O^{WSAL}$ and $\lceil O^{ASAL}$, the materializations are independent. The table is

shown in figure 4. **Figure 4**

A Non - Virtual Table

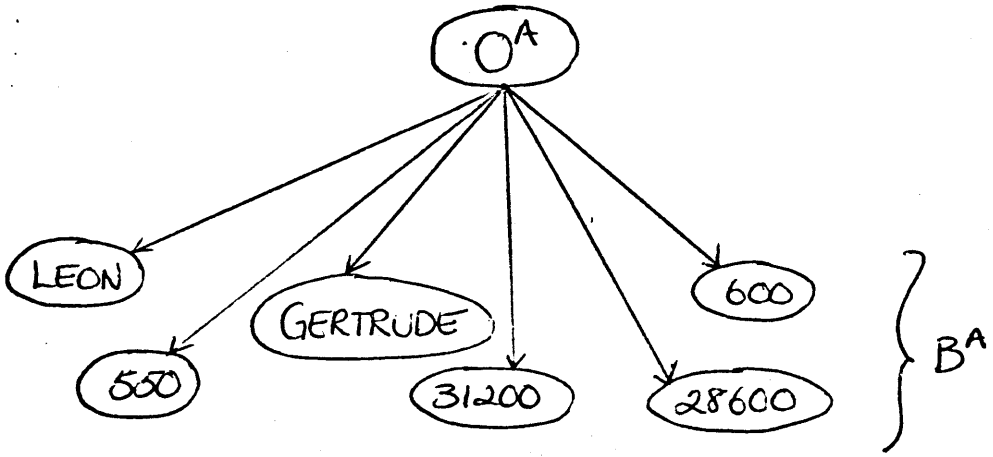
EMPN	WSAL	ASAL
LEON	600	31200
GERTRUDE	550	28600

$\uparrow O^A$

LEON	600	31200
GERTRUDE	550	28600

$\uparrow B^A$

Object Materialization



Object Graph

The functional dependency is undesirable because a change in $\uparrow O^{WSAL}$ will not automatically be reflected by a corresponding change in $\uparrow O^{ASAL}$. Figure 5 shows how the concept of virtual information can be applied to the tables in figure 4 to solve the dependency problem. In figure 5 the basis of table A' ($B^{A'}$) is different than B^{A} in figure 4, while $\uparrow O^{A'}$ is identical to $\uparrow O^{A}$. The pure data (WSAL) is combined with a pure algorithm (multiply by 52), via the materialization mapping, to give virtual information (ASAL). Note that the functionality of O^{A} is identical to $O^{A'}$; standard database operations such as insert, delete, update, and retrieve still apply assuming that $M^{A'}$ and $(M^{A'})^{-1}$ are suitably defined (for example, assuming that the update operation divides ASAL by 52 before updating WSAL). Of course, $M^{A'}$ may not always be invertible. Several example of this are given in {R16}.

Table A' in figure 5 is virtual with respect to table B, and table A' can be manipulated exactly as if $\uparrow O^{A'}$ were identical to $\uparrow O^{B}$. Once $O^{A'}$ is defined in terms of $M^{A'}$ and $B^{A'}$, $O^{A'}$ can be manipulated without regard to its implementation. The insulation of implementation and function is extremely important to database and database system design -- just as it is important to programming languages. In fact, this insulation is crucial to the support of data independence.

Figure 5

A Virtual Table

EMPN	WSAL	ASAL
LEON	600	31200
GERTRUDE	550	28600

$\uparrow O^{A'}$

EMPN	WSAL
LEON	600
GERTRUDE	550

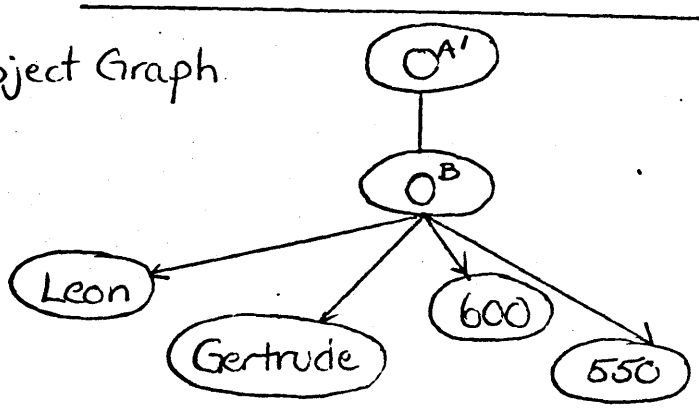
$\uparrow O^{B}$

LEON	600
GERTRUDE	550

$\uparrow B^{A'} = \uparrow B^{B}$

Object Materializations

Object Graph



$B^{A'} = B^{B}$

Data independence has been defined in various fashions, both precisely {R24} and imprecisely {R7,R15} in the literature. For our purposes it is sufficient to say that data independence exists if a change in $\uparrow B^t$ does not necessarily imply a change in $\uparrow O^t$, assuming that M^t is allowed to change. For example, data independence exists in the relations of figure 5 if a new field, EMP_AGE, can be added to $O^{A'}$ without changing $\uparrow O^B$ (note that $B^{A'}$ is contained in B^B). As another example, data independence exists at the applications programming level if changes to the physical storage structure do not force changes to the applications programs. We refer the reader to the literature for discussions of the many virtues of data independence {R13,R24}.

A virtual information capability facilitates data independence in the following way. Suppose the database comprises the O^A in figure 4. Noting that the ASAL can be derived from WASL, the database administrator decides to save storage space by eliminating the ASAL field. He creates O^B shown in figure 5. Without a virtual information capability, since $\uparrow B^A$ has changed to $\uparrow B^{A'}$, $\uparrow O^A$ would have to change to $\uparrow O^B$. Therefore, applications programs, for example, using $\uparrow O^A$ would have to be modified. However, with a virtual information capability, a new table A' (shown in figure 5) can be defined in terms of $\uparrow O^B$. We

must find an $M^{A'}$ such that $\uparrow O^{A}$ (in figure 4) is identical to $\uparrow O^{A'}$ (in figure 5). Indeed, they are identical in figures 4 and 5.

In practice, it is impossible to guarantee that we can find a suitable materialization mapping, because it is always possible to change a basis in such a way as to lose information. More importantly, however, a change in basis B^{t} to $B^{t'}$ and a corresponding change in M^{t} to $M^{t'}$, such that $\uparrow O^{t}$ is identical to $\uparrow O^{t'}$, may give us a non-invertible M^{t}. Data independence will exist for retrieval operations, but not for store operations. We say that an object O^{t} is storable at level k if M^{t}_{k-1} is invertible. This problem has been addressed in the computer literature {R16} and in implemented DBMS systems. In implemented systems there are two possible reasons for non-storable virtual information:

(i) The system doesn't support some forms of storable virtual information.

(ii) The materialization mapping is not invertible.

Chapter 2. Applications of the Virtual Information Concept

A Non-DBMS Application of the Virtual Information Concept

Data Reorganization/Translation

The process of data translation is one of changing the organization of the data in a database so that it can be processed on a new hardware/software system, or possibly more effectively on the same system {R1,R2}. Reasons for wanting to reorganize a database range from efficiency to necessity (e.g. new hardware/software). Figure 6 shows a simplification of the approach to the problem taken by members of the Data Translation Project at the University of Michigan {R1,R2,R3}. First, the source and target data are defined, with respect to structure, by a data definition language (DDL). Second, a mapping between the source and target is defined by a translation definition language (TDL). The Reader then reads the source database using the source DDL and translates the data into a common data form. The Translator uses the source data in the common form and the TDL mapping to produce target data in common form. Finally, the Writer uses the target data definition to convert the target data to the correct target format.

Figure 6

A Data Translation Scheme

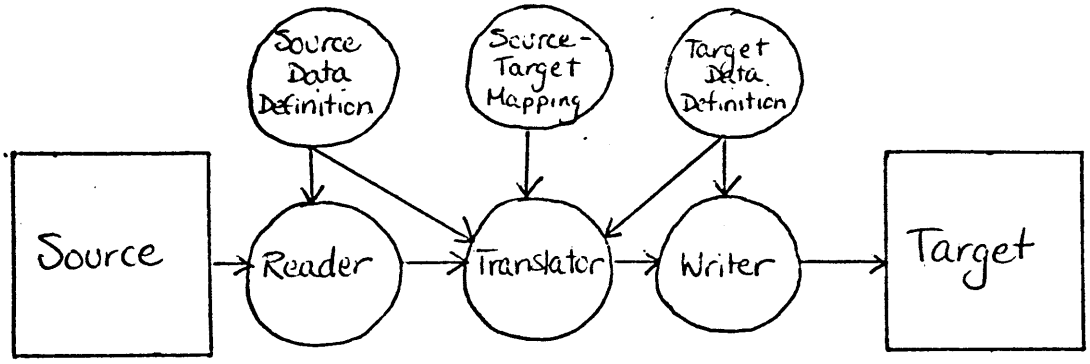
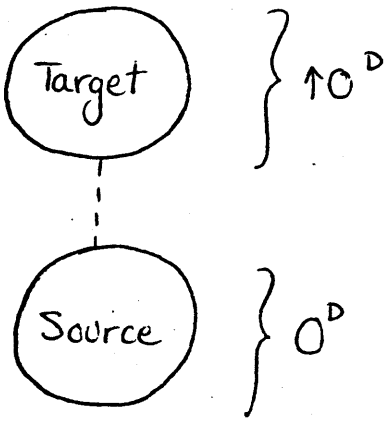


Figure 7

Data Translation as an Application of Virtual Information



The data translation problem is easily restated in terms of virtual information. Consider figure 7, where figure 6 is redrawn in the context of a DBMS, O^D. The problem is to find M^D such that $\uparrow O^D$ has the desired structure at the target level. Thus, we are looking for a mapping that will give us our desired virtual view of the source data.

DBMS Applications of the Virtual Information Concept

Multiple Views of Data

A view of an object is simply the materialization of the object. Objects O^t and O^u are said to be different views of O^v iff O^v is a descendent of both objects in the object representation graph. We discuss multiple views of data because much of the work on views -- especially in implemented DBMS systems -- has focussed on multiple views. Multiple views serve two major functions:

(i) The support of multiple data models, specifically the hierarchical, network and relational models.

(ii) The support of multiple users' views, each with its own requirements/authorization for a subset of the total database.

INFOPLEX, System R, the DBTG system, and the ANSI/X3/SPARC architecture all have been discussed in terms of multiple data model support {R15,R6,R13,and R5 respectively}. The difficulty in providing multiple data model support is that the database representation at the level below the models should be flexible enough to efficiently support not only the three standard models (hierarchical, network, and relational) but new models as well. Our purpose is to express the problem in terms of virtual information. Assume that O^D is the database implemented with a database model to be used to support hierarchical, network, and relational databases (O^h, O^n,and O^r respectively). In other words, the basis of O^h, O^n, and O^r must, in each case, be O^D. If the model were not virtual then $\uparrow O^h$, $\uparrow O^n$, $\uparrow O^r$, and $\uparrow O^D$ would all be identical (since they have a common basis). Without proof, we claim that if the materialization of different model views were always identical then little would be gained by supporting multiple models.

The support of multiple users' views is handled to some extent by most, if not all, DBMSs -- although the terminology differs. In IMS, the program communication block (PCB) specifies the mapping between "logical" and "physical" databases {R13}. In the DBTG system, the sub-schema definition specifies the mapping between the

"sub-schema" and the schema {R13}. In the System R, the view definition facility enables selection of a subset of one or more relations to be presented to the user as one view {R6}. The view support provided by IMS, DBTG, and System R is important for two reasons:

a. The user can view the part of the database he wants. This frees him from worrying about irrelevant fields, sets, tuples, etc. Since he only sees part of the database, other parts can be changed without affecting him -- thereby introducing data independence.

b. Views can be defined and permission to perform particular operations on data can be selectively granted to users. Thus views can be used for security purposes.

To see the connection between virtual information and the support of multiple users' views, we can look at a user's view as a view of a database with only a subset of the potential information. Obviously, the materialization of the user's view is not identical to the materialization of the user's basis, since the user's basis is identical to the basis of the database -- while the view is only a subset of the database.

Prevention of Errors

The prevention of errors comprises four areas: security, integrity, consistency, and reliability. These areas are defined as follows {R21}.

Security: prevent users from accessing and modifying data in unauthorized ways.

Integrity: prevent semantic errors made by users due to carelessness or lack of knowledge.

Consistency: prevent semantic errors due to interaction of multiple processes operating concurrently on shared data.

Reliability: prevent errors due to malfunctioning of hardware/software.

Virtual information is important to two of the above areas: security and integrity.

Security

Security is related to virtual information because it necessarily implies different information content available to different users. In fact, System R, DBTG, and IMS all provide security facilities in conjunction with their multiple view facilities. For example, in System R,

permission is granted to specified users to perform certain operations on specified views (in System R a user's overall database view is a combination of views of smaller objects). Another way that virtual information can be used to provide security is through encryption techniques. For example, a view of an object could be defined as a function of the view of the secure object and an encryption key. Without insertion of the correct key, the object would be garbled upon materialization.

Integrity

Virtual information supports integrity in two ways: data type conversion and elimination of redundancy. Data type conversion comprises three related concepts: type conversion (for example, real to integer), unit conversion (for example, kilograms to tons), and scaling (for example, thousands to millions). Unit conversion and scaling can be thought of as a subset of type conversion. Type conversion aids integrity two ways. First, it insures that comparison and other operations are done between like objects by performing meaningful conversions. Second, it rejects operations between unlike objects where no meaningful conversion can be made. A discussion of the relationship of data conversions to integrity can be found in {R21}. Conversion is a virtual information function in the

following sense. Assume that object O^t, whose materialization represents a real number, is the sole component of O^u. If O^u can be manipulated as an integer (that is, if conversion is defined between types integer and real) then $\uparrow O^u$ is a virtual view of O^t.

For an example of a data conversion facility, we consider the MIMS system {R23}. Using MIMS terminology, each field in a record can have an associated unit of measure (u/m). The user's view of that field is then always in that u/m. To enable u/m conversions, MIMS allows the user to create a file called the UM file. Each record in the UM file comprises a u/m code (feet, seconds, kg), a u/m type (distance, time, mass), a u/m definition, and a standard indicator. The u/m definition expression enables determination of relative magnitudes. The standard indicator specifies the internal units in which all measures of a given u/m type are to be stored. Figure 8 shows an example UM file. Thus, referring to figure 8, if a field is defined with a u/m of feet the internal representation (the materialization of its sole component) would be yards.

MIMS U/M File

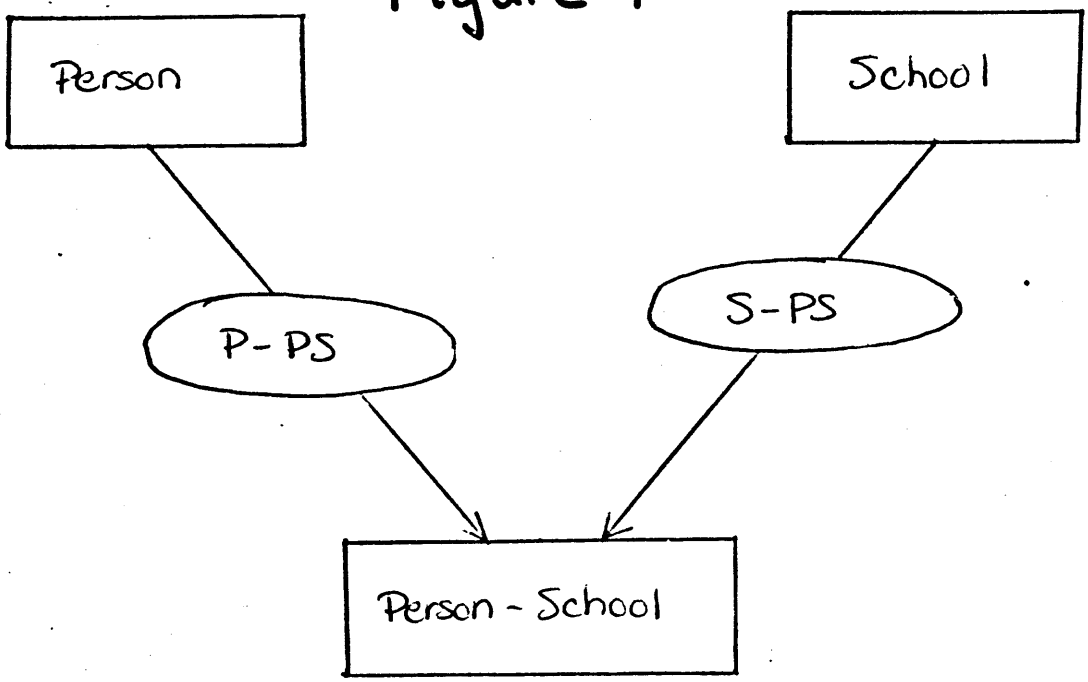
u/m code	u/m type	u/m definition	u/m standard
inch	distance	1	yes
feet	distance	12	no
yard	distance	36	no
sec	time	1	yes
min	time	60	no
hour	time	60*min	no
feet/sec	velocity	feet/sec	no
inch/sec	velocity	inch/sec	yes

Figure 8

A data conversion facility requires some sort of data typing facility. One proposal for such a facility is contained in {R22}. That proposal is couched in terms of domain definitions for relational attributes. The domain definition comprises a description of the objects in the domain, an ordering, and an action to be taken in case of attempted violation of the integrity of the domain. A better approach might be to use the concept of abstract data types (this is pointed out in {R22}), in order to hide representation details. These issues will be discussed later in reference to INFOPLEX.

Redundancy of data means that multiple updates (inserts, deletes) must be done when one object is to be updated (inserted, deleted). Database integrity is then susceptible to processes that either neglect to modify all occurences of the object or that are terminated (for example, due to a system crash) before all modifications can be completed. On the other hand, redundancy facilitates processes that only retrieve the redundant data since it tends to reduce inter-relation (record, set, etc.) references. For example, consider the network pictured in figure 9. Suppose we want to determine Kathy's school. We find her Person-School record via the P-PS set, since her school name (sname) is located in the Person-School record. We would have had to search in the P-PS set, were it not

Figure 9



Example Person Record

pname	age
Kathy	21

Example School Record

sname	location
Georgetown	Washington, D.C.

Example Person School Record

pname	sname	GPA
Kathy	Georgetown	3.95

for the redundancy in sname. To eliminate the redundancy while maintaining the benefits of redundancy, we could make pname and sname virtual within the Person-School record. In fact, DBTG provides this facility with its VIRTUAL SOURCE clause {R13}.

Data Encoding

Data encoding is typically used to reduce the storage space required by objects. For example, fixed length lines of the text might be compressed via the substitution of a special end-of-the-line character for trailing blanks. Data encoding clearly falls under the hat of virtual information, since the materialization of a decoded object would be different from the materialization of an encoded object.

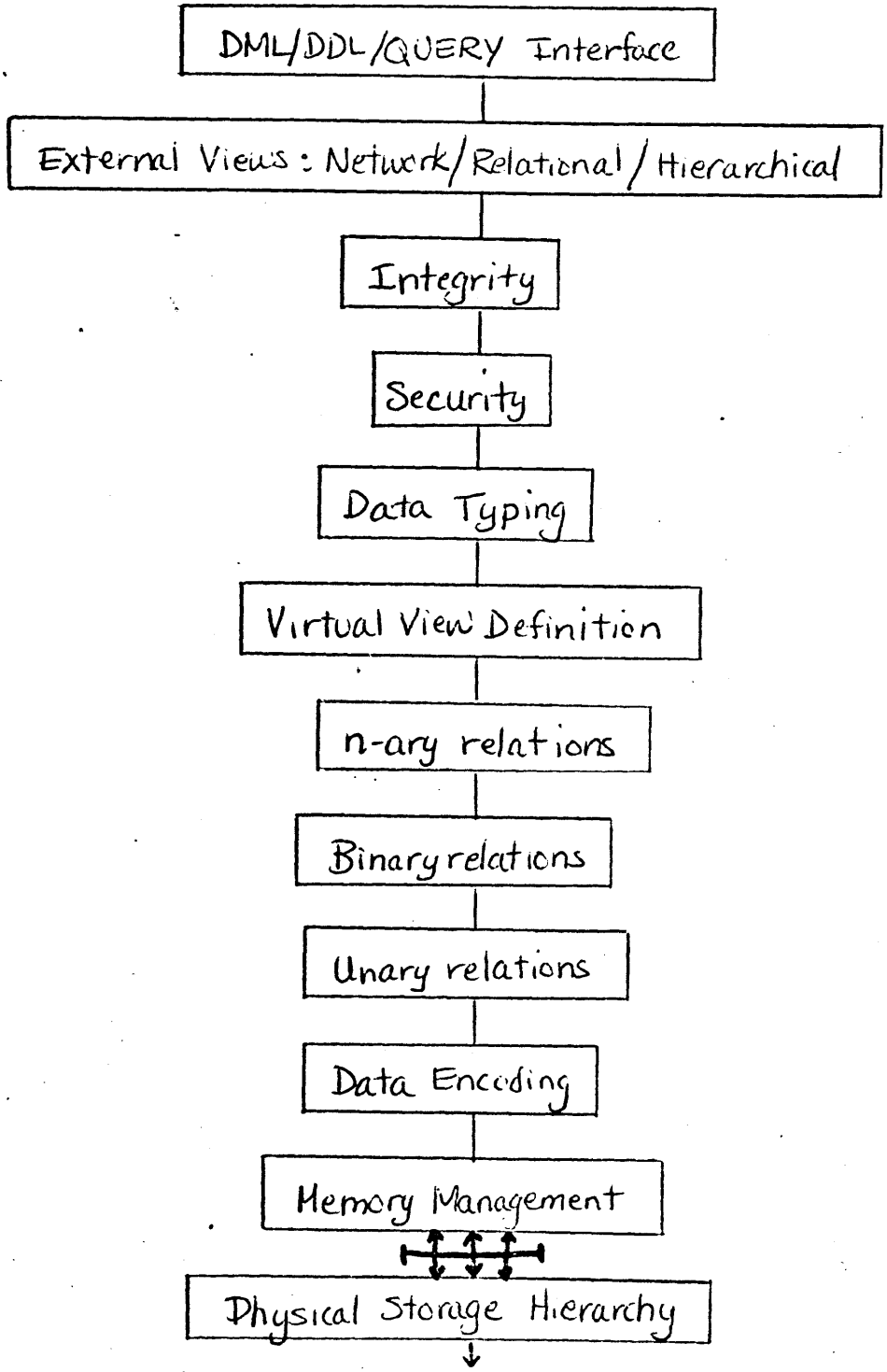
Chapter 3. Virtual Information Functions in INFOPLEX

INFOPLEX {R15} is a database computer, currently in the design phase. The motivation for INFOPLEX is the desire for and extremely high speed, high reliability, DBMS. INFOPLEX comprises a storage hierarchy for data storage and retrieval, and a functional hierarchy for performing information management functions. Our objective is to discuss the relevance of virtual information to INFOPLEX's functional hierarchy.

Three functions in the INFOPLEX computer are essentially virtual information functions. These are data typing (including conversion, unit of measure, scaling and encoding), virtual view definition (including virtual views of relations and attributes), and security. Although these functions are related, their placement in the INFOPLEX functional hierarchy cannot be at the same level. For example, data encoding must be done at a low level so that levels with data comparison functions can do the comparisons on decoded data. Our proposal for placement of virtual information functions in the INFOPLEX functional hierarchy is depicted in figure 10. Intuitively, the security, integrity, and data typing (except data encoding) levels belong above the view definition level because

Figure 10

VIRTUAL INFORMATION IN THE INFOPLEX FUNCTIONAL HIERARCHY



security, integrity, and data typing may be expressed in terms of virtual views (and objects). Placement will be discussed later in slightly greater detail. A complete discussion of the INFOPLEX functional hierarchy is beyond the scope of this paper. The reader is referred to {R15} and {R28}.

The Security Function

The security function, as stated previously, is to prevent users from accessing and modifying data in unauthorized ways. Although data access can be effectively controlled through encryption, a more general method is necessary to prevent data modification. We assume that the mechanism to assert the true identity of a user (the userid) already exists. We are then left with two major design issues:

(i) Location Strategies

1. Keep access permission information for objects in a user profile.
2. Keep access permission information for objects with the object itself.

(ii) Basis Strategies (not to be confused with an object's basis)

1. Control access on a views basis.
2. Control access on a query modification basis.

Logically, the user that creates an object should be solely responsible for granting access permission to other users. This is the strategy used in System R {R6}. If a user A wishes to grant permission to user's B and C, and location strategy 1 is used then A must access B's and C's user profiles--which may be located in a slow retrieval level of the INFOPLEX storage hierarchy (e.g. on tape). If location strategy 2 is used then only the object profile must be retrieved (which is likely to be readily accessible because the object is likely to have been recently accessed). Furthermore, location strategy 2 facilitates the granting of access to all users (System R uses a special keyword of PUBLIC for this purpose). Finally, the question "what users have access to this object" is easier to answer with location strategy 2. The only advantage of location strategy 1 is that access rights can be determined before an object is accessed. Overall, the better location strategy for INFOPLEX is 2.

Query modification is really a form of view definition in the sense that the modified query corresponds to a new view that only lasts for the duration of the query. The INGRES {R13} DBMS uses query modification as a basis for security. We prefer to seperate the view definitions functions from the security function, and therefore favor the use of view definitions as a basis for security (rather than something the security level does) in INFOPLEX. This is the approach taken, for example, by System R, IMS, and DBTG.

The Virtual View Definition Function (VVDF)

The VVD function is to enable the construction of new objects from objects already in the database. This means specifying, for object O^t, B^t and M^t. As was mentioned earlier, most existing DBMSs have some (perhaps limited) VVDF.

One of the most comprehensive VVDF facility is provided by System R {R6}. The System R user defines a view preceeding a query with the header "DEFINE VIEW" (see the example on page ?). The query itself is the view definition. A catalog holds all the view definitions. When an object is referred to, System R checks the catalog. If the name of the object is present in the catalog, the corresponding definition is substituted for the object

referenced.

A System R view definition of object O^t defines B^t simply by referencing other objects. The retrieval operation (i.e., the materialization mapping, M^t) is defined explicitly by the query itself. A view-defined object cannot be updated unless there is a one-to-one correspondence between the object and an object defined in a base relation (System R's terminology for basic objects).

We propose a similar view facility for INFOPLEX, located below the data typing and security levels. If it were above the data typing level, then the data typing level would not be able to do type conversion for objects defined by the VVD level. Security cannot be below VVD; if it were then the security function could not be handled via virtual views.

Suppose that a user wishes to define a hierarchical view of a collection of objects already defined at INFOPLEX's n-ary level. The user writes a query to define the virtual (hierarchical) object in terms of the existing n-ary objects. That query effectively defines all operations that can take place on the new object. We assume that the hierarchical external view level provides a mechanism for mapping hierarchical operations into n-ary

operations (although the mechanism would be non-trivial). Since the query explicitly defines the materialization mapping (for retrieval of atomics), but not $(M)^{-1}$ (for modification operations such as insert, delete, and update), the problem of inferring the latter exists. In general we can permit modification operations in a virtual n-ary relation (object) if the materializations of the attributes of the virtual relation are invertible functions of the materializations of the attributes of exactly one relation. We can delete or insert a tuple only if we meet the constraint that the attributes of the virtual relation include at least one candidate attribute from each of the relations in the basis of the virtual relation, and if we meet the invertibility requirement mentioned above. In addition, modification is possible when deletes and inserts are possible.

The Data Typing Function

Each object has its own materialization mapping that determines the behavior of the object by defining the operations that can manipulate the object. It is reasonable to expect that many objects would have similar materialization mappings. If the materialization mappings were identical, they could be catalogued, and each object could refer to the catalog for its materialization mapping.

We could refer to a catalog entry as an object type definition. Thus, instead of defining a new materialization mapping for each new object, we could associate each object with an object type. Types would include the usual types (real, integer, etc.) as well as complex types (array, relation, hierarchy, network, etc.). Such a data typing facility would require extensive type parameterization facilities to enable parameterization of, for example, range constraints, size constraints, and composite data types (e.g. array of integers).

In INFOPLEX data typing can be divided into two distinct levels, the data typing (DT) level and the data encoding (DE) level. The DE level is responsible for encoding techniques for minimizing storage use. Its inplementation would comprise a catalog of encoding/decoding techniques. Further details can be found in {R28}. The DT level is responsible for:

(i) insuring that operations between objects of differing data types are rejected or that one or both objects are properly converted

(ii) enforcing (i) with respect to different units of measure and scale factors

(iii) allowing user definition of the materialization of objects at the next higher level

(iv) providing whatever suport deemed necessary towards an abstract data typing facility.

The advantage of typing all objects is that the operation materialization mapping would be explicit and well-defined. The disadvantage is that it would be impossible to pre-specify all object types -- since the number of possible object types is infinite. A data type abstraction facility, similar to those used in programming languages, would have to be built{R19,R29}. The design of such a facility is non-trivial. No abstract data type facility for data bases has been developed in the literature. We prefer restricting INFOPLEX data typing to objects whose materializations have a relatively simple structure (e.g., real, integer, string, etc.). In addition, a limited data abstraction facility, along the lines of {R22}, would be quite useful.

To perform (i) to (iv) the DT level must parse all requests from higher levels in order to identify each of the objects references and each of the operations involved. Suppose the following request were issued:

```
SELECT EMP.NAME,EMP.VACATION_TIME
WHERE EMP.VACATION_TIME<EMP.SICK_TIME
```

The EMP relation and an example DT catalog is shown in figure 11. The DT level would convert the request to:

```
SELECT EMP.NAME,EMP.VACATION_TIME
WHERE REAL_GREATER_THAN(CONVERT_INT_TO_REAL(
    EMP.VACATION_TIME),EMP.VACATION_TIME)
```

When the response is returned by the lower levels, DT must modify the objects EMP.NAME and EMP.VACATION_TIME so that they are materialized in the prescribed format.

The above example illustrates two implementation considerations:

(i) Lower levels must be capable of handling the complex arithmetic and comparison operations generated by the DT level.

(ii) Virtual view definitions must be modified by the DT level before going to the virtual view definition (VVD) level. Note that if the VVD level were above the DT level then virtual objects could not be typed.

Figure 11

The EMP Relation:

NAME	VACATION_TIME	SICK_TIME
Stu	4	9.0
Molly	2	7.5
Florence	1	6.0
Matt	1	5.5
Susan	3	8.5

The DT Catalog:

Object	Units	Measure	Type	Materialization Format
Name	—	—	String	A10
Vacation_Time	Time	Weeks	Integer	9
Sick_Time	Time	Days	Real	9.9

Conclusion

Virtual information is a collection of many related database concepts. We have attempted to provide an information model and a corresponding notation with which virtual information concepts can be communicated. The major fault of the notation is that while the concept of retrieving information from the database in fairly well defined in terms of the materialization mapping, the concept of changing the information content of the database is poorly defined. A more desirable information model would completely characterize the behavior of objects in terms of the operations that can be performed on them.

The implementation of virtual information in INFOPLEX was discussed in terms of general issues. Further details could be given only if assumptions were made as to details of the interface between the virtual information levels and other levels. Hopefully, this paper will help in the determination of those details.

Bibliography

R1. Merten,A.G. and Fry,J.P. , "A Data Description Language Approach to File Translation", Data Translation Project, The University of Michigan, Ann Arbor, Michigan, April 1974

R2. Fry,J.P. and Jervis,D.W., "Towards a Formulation and Definition of Data Reorganization", Data Translation Project, The University of Michigan, Ann Arbor, Michigan, January 1974

R3. Fry,J.P., Smith,D.C.P., Taylor,R.W., Frank,R.L., Lum,V., Behymer,J.A., Schneiderman,B., "Stored-Data Description and Data Translation: A Model and Language", Information Systems, Vol. 2, pp.95-148, Pergamon Press, 1977

R4. Tsichritzis,D. and Klug,A., "The ANSI/X3/SPARC DBMS Framework", X3/SPARC Study Group on Database Systems (X3 Project 226)

R5. Klug,A. and Tsichritzis,D., "Multiple View Support Within the ANSI/SPARC Framework", Proceedings on Very Large Data Bases, 1977

R6. Astraham,M.M. et al, "System R Relational Approach to Database Management", ACM Transactions on Database Systems, Vol. 1, No. 2, June 1976

R7. Folinus,J.J., Madnick,S.E., Schutzman,H.B., "Virtual Information in Data-Base Systems", Information Systems

Group, Sloan School of Management, M.I.T., Cambridge, MA 02139

R8. Each,M.J., Coguen,N.F., Kaplan,M.M., "The ADAPT System: A Generalized Approach Towards Data Conversion", Proceedings on Very Large Data Bases, 1979, pp183-193

R9. Navathe,S.B., Schkolick,M., "View Representation in Logical Database Design", SIGMOD Proceedings 1978, pp144-156

R10. Navathe,S.B., Fry,J.P., "Restructuring for Large Databases: Three Levels of Abstraction", ACM Transactions on Database Systems, Vol. 1, No. 2, June 1976, pp138-158

R11. Brodie.M.L., Schmidt,J., "What Is The Use Of Abstract Data Types In Data Bases?", VLDB '78, pp140-141

R12. Smith,J.M., "A Normal Form For Abstract Syntax", Proceedings on Very Large Data Bases, 1978, pp156-162

R13. Date,C.J., "An Introduction to Database Systems", Addison-Wesley Publishing Co., Reading, Mass., 1975

R14. Cardenas,A., "Data Base Management Systems", Allyn and Bacon, Inc., Boston, Mass., 1979

R15. Madnick,S.E., "The Infoplex Database Computer: Concepts and Directions", Proceedings of the IEEE Computer Conference, pp168-176, February 26, 1979

R16. Codd,E.F., "Recent Investigations Into Relational

Data Base Systems", Proc. IFIP Congress 1974

R17. Liskov,B.H., Zilles,S.N., "Specification Techniques for Data Abstractions", SIGPLAN Notices, Vol. 10, No. 6, pp72-87, June 1975

R18. Liskov,B., Zilles,S., "Programming With Abstract Data Types", SIGPLAN Notices, Vol. 9, No. 4, pp50-59, April 1974

R19. Hammer,M.M., "Data Abstractions for Data Bases", SIGPLAN Notices, Vol. 8, No. 2, 1976, pp58-59

R20. Liskov,B., Snyder,A., Atkinson,R., Schaffert,C., "Abstraction Mechamisms in CLU", CACM, Vol. 8, pp564-576, August 1977

R21. Eswaran,K.P., Chamberlin,D.D., "Functional Specifications of a Subsystem for Data Base Integrity", Proc. International Conf. on Very Large Data Bases, Sept. 1975

R22. McLeod,D.J., "High Level Domain Definition in a Relational Data Base", Proc. ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure (March 1976)

R23. Mitrol, Inc., "MIMS Request Reference Manual", June 1975

R24. Stonebraker,M.R., "A Functional View of Data Independence", Proc. 1974 ACM SIGMOD Workshop on Data Description, Access and Control

R25. Jensen,K., Wirth,N., "Pascal: User Manual and

Report", Springer-Verlag, 1979

R26. I.B.M., CMS Users Guide, Second Edition, August 1977

R27. Chen,P.P.S., "The Entity-Relationship Model", ACM TODS, Vol. 1, No. 1, March 1976

R28. Hsu,M., "Architectural Specifications of the Functional Hierarchy of the Infoplex Database Machine", Unpublished Draft, Sloan School of Management, M.I.T., 1980

R29. "Data Abstractions For Databases", Lockemann,P.C. et al, ACM TODS, Vol. 4, No. 1, March 1979, pp60-75