NON FIRST NORMAL FORM RELATIONS TO REPRESENT HIERARCHICALLY ORGANIZED DATA

by Serge ABITEBOUL(+) and Nicole BIDOIT(+)(*)

(+) Institut National de Recherche en Informatique et Automatique, 78153 Le Chesnay, FRANCE.

(*) Laboratoire de Recherche en Informatique, Université de Paris-Sud, Orsay, FRANCE.

INTRODUCTION

Several investigators have stressed that the first normal form (1NF) condition [Co] is not convenient for handling a variety of database applications [M,K,JS,SP]. The first purpose of this paper is to present a database model, namely, the Verso model, where data is organized in non 1NF relations. The values for some attributes in a Verso instance are atomic whereas the values for other attributes are simpler Verso instances. As we shall see, this recursive definition of the data structure induces a hierarchical organization of the data. Several models have tried to capture the notion of hierarchical data organization [IMS,HY]. The advantage of our approach is that, by using relation as underlying structure, we are able to preserve some of the positive features of the relational model, for instance, a simple algebraic query language.

As mentioned earlier, the first major theme of this paper is to formally present the data structures and operations in Verso. In a Verso schema, some dependencies (very similar to Delobel's Generalized Hierarchical Dependencies [D]) are implicitely specified. Therefore, some semantic connections among the attributes are implied by the choice of a Verso schema. Furthermore, the operations that we propose on Verso instances take advantage of these semantic connections. In particular, some queries which would typically require joins in the pure relational model can be expressed by a selection in the Verso model removing the need for the user to specify access paths.

The second major theme of the paper is the investigation of some key issues raised by this data organization. In particular, data restructuring is studied via the notions of schema equivalence and dominance. Necessary and sufficient conditions of equivalence and dominance are exhibited based on some elementary schema transformations. Also, a natural connection between Verso instances and relational database instances satisfying the Universal Relation Scheme Assumption [FMU,MW] is investigated.

In [SP,JS] , a non 1NF extension to the relational model (NF2) is also proposed. In an NF2 relation, a non atomic value is a set of atomic values (which is a very restricted case of the non atomic values in Verso). Besides the operations of the relational algebra, they propose to use two new operations, namely nest and unnest, which allow to transform a 1NF relation into an NF2 one, and conversely. As we shall see, the operations in the Verso model generalize all the operations of the relational algebra. Furthermore, the nest and unnest operations can be seen as very primitive subcases of the data restructuring mechanisms exhibited here.

1. PRELIMINARIES

In the following, we assume that the reader is familiar with the relational model. In this section, we briefly review some well-known concepts and present the notation used throughout the paper.

We assume the existence of an infinite set U of attributes, and for each A in U, of a set of values called the domain of A and denoted dom(A). A relational schema is a finite set of attributes. Let V be a relational schema. A tuple v over V is a mapping from V into $\bigcup_{A \text{ in } V} dom(A)$ such that v(A) is in dom(A) for each A. A (first normal form) relation over V is a finite set of tuples over V. The set of tuples over V is denoted tup(V), and the set of relations rel(V). The relational operations of union, intersection, difference, join, projection and selection are respectively denoted $\cup$, $\cap$, $-$, $*$, π and select_C (where C is an elementary condition of the form $A < a$, $A \leq a$, $A = a$, $A \geq a$, $A > a$ for some A in U and some a in dom(A)).

A relational database schema is a finite set of relational schemas. A relational (database) instance r of some relational database schema R is

a mapping r from R such that, for each X in R, r(X) is in rel(X). A relational instance satisfies the Universal Relation Schema Assumption (URSA) if $r(X) \supseteq \pi_X(r(Y))$ for each X, Y in R and $X \subseteq Y$.

In the paper, we also consider finite strings of attributes. Let $A_1 \ldots A_n$ be a string of attributes. An ordered tuple x over $A_1 \ldots A_n$ is an element of the cartesian product $dom(A_1) \times \ldots dom(A_n)$. The set of ordered tuples over some string X is denoted Otup(X). For each string X of attributes, the set {A in X} is denoted set(X). For each ordered tuple x over X, the corresponding tuple over set(X) is denoted map(x).

In general, A,B,... denote attributes, a,b,... values, V,W,X,Y... relational schemas (or finite strings of attributes), v,w,x,y... (ordered) tuples, R,S,... relational database schemas and r,s,... relational database instances. We also use the classical convention of writing XY for the union of two sets X and Y of attributes or for the concatenation of two strings X and Y of attributes.

2. THE VERSO MODEL

In this section, we present the data structure and operations of the Verso model using the auxiliary concept of format.

Let us consider first an exemple. A department consists of a set of COURSEs, the BOOKs for each course, the STUDENTs in the course and their GRADEs. We can represent an instance of a department like in Figure 2.1. Intuitively, the department can be considered as a relation over three attributes, say COURSE, A_1 and A_2. The values in dom(COURSE) are atomic whereas the values in $dom(A_1)$ and $dom(A_2)$ are simpler Verso instances. Let us make two remarks. The first one is that, in the example, there is no BOOK required for the physics COURSE. (Thus, null values can be represented in a Verso instance). The second remark is that an implicit connection is assumed between the attributes STUDENT and BOOK through the attribute COURSE. In other word, a "join" is forced between COURSE STUDENT and COURSE BOOK.

In order to formalize the notion of Verso instance, we need the auxiliary concept of format. Intuitively, a format specifies the underlying structure of a Verso instance.

Definition : A format is recursively defined by :

(i) let X be a finite string of attributes with no repeated attribute, then X is a (flat) format over the set X of attributes, and

(ii) let X be a finite string of attributes with no repeated attribute, and $f_1,\ldots,f_n$ some formats over $Y_1,\ldots,Y_n$, resp., such that the sets $X,Y_1 \ldots Y_n$, are pairwaise disjoint, then the string $X(f_1) * \ldots (f_n)*$ is a format over the set $XY_1 \ldots Y_n$.

For instance, f=COURSE STUDENT GRADE is a flat format over {COURSE,STUDENT,GRADE} and g=COURSE(STUDENT(GRADE)*)*(BOOK)* is a format over {COURSE,STUDENT,GRADE,BOOK}.

In the following, we shall use a directed tree representation for formats. The representation of the format g is given in Figure 2.1.

We now define the Verso instances :

Definition : Let f be a format. The set of all (Verso) instances over f, denoted inst(f), is recursively defined by :

(i) if $f \equiv X$ then inst(f) is a finite subset of Otup(X), and

(ii) if $f \equiv X(f_1)*\ldots(f_n)*$ then I is in inst(f) iff

a) I is a finite subset of $Otup(X) \times inst(f_1) \times \ldots \times inst(f_n)$, and

b) if $\langle u,I_1,\ldots I_n\rangle$ and $\langle u',I'_1,\ldots I'_n\rangle$ are in I for some $u,u',I_1,I'_1,\ldots I'_n$ then $u \neq u'$ or $\langle u,I_1,\ldots I_n\rangle = \langle u',I'_1,\ldots I'_n\rangle$.

Intuitively, the (a) condition states that I is atomic on the attributes in X and not atomic on the "attributes" $f_1,\ldots f_n$. The (b) condition forces X to be a key. It is clear that the mathematical notation for Verso instance is cumbersome and not really readable. Therefore, in the following, instances will be presented using the "bucket" technique of [PS] . (See Figure 2.1).

In the relational model a database schema consists of several schemas. Similarly, we have :

Definition : A Verso database schema S is a finite set of formats. A Verso database instance s of the schema S is a mapping from S in $\bigcup_{f \text{ in } S}$ inst(f) such that s(f) is an instance over f for each f in S.

We now introduce four binary operations (fusion, difference, join, and cartesian product) and four unary ones (projection, selection, restriction and renaming) on Verso instances. These operations are natural extensions of the classical relational operations. We start by presenting fusion and difference of instances over identical formats. We shall then extend these two operations to instances over not identical but "compatible" formats.

The first operation, namely fusion, allows to "add" the information contents of two instances.

Definition : Let f be a format and I, J instances over f. Then, the fusion of I and J, denoted $I \oplus J$, is the instance over f recursively defined by

i) if $f \equiv X$, $I \oplus J = I \cup J$

ii) if $f \equiv X(f_1)* \ldots (f_n)*$, $n > 0$, then

$$I \oplus J = \bigcup \left\{ u(I_1 \oplus J_1)\ldots(I_n \oplus J_n) \;\middle|\; \begin{matrix} u\, I_1 \ldots I_n \text{ in } I \\ u\, J_1 \ldots J_n \text{ in } J \end{matrix} \right\}$$

$$\left\{u\ I_1...I_n \ / \begin{array}{l} u\ I_1...I_n \text{ in I and} \\ \text{for all } J_1...J_n,\ u\ J_1...J_n \notin J \end{array}\right\}$$

$$\cup \left\{u\ J_1...J_n \ / \begin{array}{l} u\ J_1...J_n \text{ in J and} \\ \text{for all } I_1...I_n,\ u\ I_1...I_n \notin I \end{array}\right\}$$

An example of fusion is given in Figure 2.2.

The second operation, namely difference, allows to "substract" the information contained in an instance from the information contained in another one.

Definition : Let f be a format, and I, J instances over f. Then the difference of I and J, denoted $I \ominus J$, is an instance over f recursively defined by :

(1) if $f \equiv X$, $I \ominus J = I - J$ and ,

(2) if $f \equiv X(f_1)^* \ldots (f_n)^*$, $n > 0$, then

$$I \ominus J = \{u(I_1 \ominus J_1)...(I_n \ominus J_n) / u\ I_1...I_n \text{ in I, } u\ J_1...J_n \text{ in J and } I_i \ominus J_i \neq \emptyset \text{ for some } i\}$$

$$\cup \left\{u\ I_1...I_n \ / \begin{array}{l} u\ I_1...I_n \text{ in I and} \\ \text{for all } J_1...J_n,\ u\ J_1...J_n \notin J \end{array}\right\}$$

An example of difference is given in Figure 2.2. Note that the physics COURSE disappeared whereas the math COURSE is still in $I \ominus J$. This results from the condition "$I_i \ominus J_i \neq \emptyset$ for some i" which is true for math and not for physics.

As mentionned earlier, these two operations will be extended to deal with instances over different but "compatible" formats. To do that, we need the auxiliary concepts of format and instance extensions. Intuitively, a format g is an extension of a format f if the directed tree associated to g can be obtained from the directed tree associated to f by simple insertion of new subtrees.
Formally we have :

Definition : Let f be a format. Then an extension g of f is a format recursively obtained as follows :

(i) if $f \equiv X$, then $g \equiv X(g_1)^*...(g_m)^*$ for some $m \geq 0$

(ii) if $f \equiv X(f_1)^*...(f_n)^*$ then $g \equiv X(g_1)^*...(g_m)^*$ and there exists a subsequence $h_1...h_n$ of $g_1...g_m$ such that h_i is an extension of f_i for each i, $1 \leq i \leq n$.

Let f be a format and g an extension of f. Let I be an instance over f. Intuitively, the extension of I to g, denoted I^g, is obtained by "padding" at each level with empty instances. The following example illustrates these two concepts.

Example 2.1 The format g = COURSE(STUDENT GRADE*)*BOOK*(TIME ROOM)* is an extension of the format f = COURSE(STUDENT)*(BOOK)*. The directed trees, associated to f and g are shown in Figure 2.3. An instance over f and its extension over g are also given in Figure 2.3.

We are now able to formally define the notion of format compatibility.

Definition : Let f and g be two formats. Then f and g are compatible iff there exists a format h such that h is an extension of f and g.

Then, in order to "add" (respectively "substract") the information contained in an instance I over f and J over g, f and g compatible, it suffices to extend I and J to a common extension h of f and g, and then use the operations of fusion and difference.

The third binary operation, namely join, is defined directly on instances over compatible formats. It allows to "combine" the information contents of two instances.

Definition : Let f and g be two compatible formats and h an extension of both f and g. Let I, J be instances over f, g respectively. Then the join of I and J (according to h), denoted $I \circledast_h J$ is the instance over h recursively defined by :

(i) if $h \equiv X$ (so $f \equiv g \equiv X$) then $I \circledast_h J = I \quad J$ and,

(ii) if $h \equiv X(h_1)^*...(h_p)^*$, $f \equiv X(f_1)^*...(f_n)^*$, $g \equiv X(g_1)^*...(g_m)^*$ then

$$I \circledast_h J = \left\{uK_1...K_p \ / \ uI_1...I_n \text{ in I, } uJ_1...J_m \text{ in J and,} \right.$$
$$K_k = I_i \circledast_{h_k} J_j \text{ if } h_k \text{ is an extension of } f_i \text{ and } g_j$$
$$K_k = I_i \text{ if } h_k \text{ is an extension of } f_i \text{ only}$$
$$\left. K_k = J_j \text{ if } h_k \text{ is an extension of } g_j \text{ only}\right\}$$

To illustrate the previous definition, two instances over compatible formats are given in Figure 2.4, together with their join according to the format COURSE(STUDENT)*(BOOK)*. The last binary operation, namely cartesian product, is different from the preceeding ones in that its first operand is required to be an instance over a flat format.

Definition : Let $f \equiv X$ be a flat format and g a format over Y such that $X \cap Y = \emptyset$. Let I, J be instances over f, g respectively. The cartesian product of I and J, denoted $I \otimes J$, is the instance over X(g)* defined by :

$$I \otimes J = \{uJ \ / \ u \in I\}$$

Note that if f and g are both flat formats, and I, J instances over f and g, respectively, then $I \otimes J$ and $J \otimes I$ are different. So the cartesian product is not commutative.

We now turn to the unary operations. The first one, namely (Verso) projection, allows to "project out" some particular "subinstances". Formally :

Definition : Let g be a format and I an instance over g. Let f be a format such that g is an extension of f. Then the projection of I over f, denoted I [f], is an instance over f recursively defined by :

if $f \equiv X(f_1)^*...(f_n)^*$, $g \equiv X(g_1)^*...(g_m)^*$, and for each $1 \leq i_1 < i_2...< i_n \leq m$, g_{i_k} is an extension of f_k then

$$I[f] = \{uI_{i_1}[f_1]...I_{i_n}[f_n] \ / \ uI_1...I_n \text{ in } I\}$$

Intuitively, the result of the projection is simply obtained by removing all the "subinstances" corresponding to "subtrees" which are projected out.
An example of projection can be found in Figure 2.5.

The second unary operation is the (Verso) selection. Besides the relational like conditions of the form "Attribut comparator Value", the Verso selection allows some existential and non existential conditions on some "intermediary" results.

Definition : Let f be a format. Then a (Verso-)selection S over f is an expression recursively defined in the following way.

(1) if $f \equiv X$ then $S \equiv select_C$ where C is the conjonction of elementary conditions(1) on attributes in X.

(2) if $f \equiv X(f_1)^*...(f_n)^*$ then
$S \equiv$ select $\langle S_1...S_n \rangle_C$ where S_i is a Verso selection over f_i for each i, $1 \leq i \leq n$, and C is a conjoncton of

(a) elementary conditions on attributes in X
(b) conditions of the form $S_i \neq \emptyset$ or $S_i = \emptyset$ for some i.

The result of a selection is defined in the following way.

Definition : Let S $\equiv$ select $\langle S_1...S_n \rangle_C$ be a Verso-selection over the format $X(f_1)^*...(f_n)^*$. Let I be an instance over f. Then the result of the application of S to I, denoted S(I), is defined by

$$S(I) = \{uS_1(I_1)...S_n(I_n) \ / \ u \models c \text{ for each elementary condition c in C}, \ S_i(I_i) = \emptyset \text{ if } S_i = \emptyset \text{ in C}, \ S_i(I_i) \neq \emptyset \text{ if } S_i \neq \emptyset \text{ in C}\}$$

These definitions are illustrated in the following Example :

(1) An elementary condition on some attribute A is a condition of the form A=a, A≠a, A<a, A≤a, A>a of A≥a for some a in dome(A).

Example 2.3. Let f = COURSE(STUDENT(GRADE)*)*

Q_1 : Give the list of math students who got a grade larger than 10.
Q_2 : Give the courses in which some student is registered and didn't get any grade for this course.

The query Q_1 is answered by the selection S_1 :

$S_1 \equiv$ select $\langle S'_1 \rangle_{COURSE=math}$ with

$S'_1 \equiv$ select $\langle S''_1 \rangle_{S''_1 \neq \emptyset}$ where

$S''_1 \equiv select_{GRADE > 10}$.

The query Q_2 is answered by the selection S_2 :

$S_2 \equiv$ select $\langle S'_2 \rangle$ with

$S'_2 \equiv$ select $\langle S''_2 \rangle_{S''_2=\emptyset}$ where

S''_2 is the identity

Examples of applications of these two queries are given in Figure 2.5. Due to space limitation, we won't define the last two unary operations. However, an example of an application of each of them can be found in Figure 2.6.

A Verso query is obtained by combining the four binary operations (fusion, difference, join and cartesian product), the four unary ones already presented (projection, selection, restriction and renaming) plus another operation which will be presented in Section 4, namely restructuring. Together these operations will be shown to be "complete" in Section 5.

3. URSA INTERPRETATION OF THE VERSO MODEL

In this section, we exhibit a strong connection between format instances and relational database instances satisfying the universal relation schema assumption. We also give an "interpretation" of the Verso operations in terms of classical relational operations.

In order to do that, we need the notion of format skeleton. Intuitively, the format skeleton of a format f is the relational database schema which describes, in a non hierarchical way, the structure of instances over f.

Definition : Let f be a format. Then the format skeleton of f, denoted Skel(f), is the relational database schema recursively defined by :

(i) if $f \equiv X$ then Skel(f) = {set(X)} and,

(ii) if $f \equiv X(f_1)^*...(f_n)^*$ then
Skel(f) = {set(X)} $\cup$ {set(X)Y / Y in Skel(f_i), i in [1,n] }.

For example the format skeleton of COURSE(STUDENT)*(BOOK)* is the relational schema {COURSE,COURSE STUDENT,COURSE BOOK}. Using these

format skeletons, we are now able to "describe" a format instance by a relational database instance.

Definition : Let f be a format and I an instance over f. The instance skeleton of I, denoted skel(I), is the relational database instance over Skel(f) defined by :

(i) if $f \equiv X$ (so Skel(f) = {set(X)}) then
skel(I)(X) = {map(u) / u in I}

(ii) if $f \equiv X(f_1)^*...(f_n)^*$ (so Skel(f)={set(X)} ∪ {set(X)Y | Y in Skel(f_i), i in [1,n]})
then skel(I)(X) = {map(x) | $xI_1...I_n$ is in I}
and skel(I)(XY) = {map(x)*skel(I_i)(Y) | $uI_1...I_n$ is in I, $I_i \neq \emptyset$}
where Y is in Skel(f_i)

In the previous definition, since $X \cap Y = \emptyset$, the join is in fact a relational cartesian product. However, in the present paper, we use the symbol × to denote (ordered) cartesian product only. Figure 3.1 exhibits the instance skeleton of the instance of Figure 2.1.

We established a correspondance between formats and relational database schemas (Skel), and between instances over format and relational database instances (skel). It is clear that (1) not all relational database schemas correspond to some formats and (2) even if a relational database schema R corresponds to a format f, not all instances over R correspond to instances over f. This leads to the two following results.

Theorem 3.1 : [Ba2] Let R be a relational database schema. Then R is a format skeleton iff

(1) R is closed under intersection, and
(2) for each X in R, $X \cap R$ is totally ordered by inclusion.

Theorem 3.2 : Let f be a format and R=Skel(f). Let r be an instance over R. Then the following two assertions are equivalent :

(1) r=skel(I) for some I over f, and
(2) r satisfies the URSA.

By the previous theorem, skel is a mapping from instances over f onto relational database instances over Skel(f) satisfying the URSA. Therefore, it would be interesting to characterize the Verso-operations on instances in terms of relational operations on relational database instances. Indeed, for binary operations such a characterization is given in Theorem 3.3.

Theorem 3.3. Let f and g be two compatible formats and h an extension of both f and g. Let I and J be instances over f and g respectively. Let r = skel(I^h) and s = skel(J^h). Then

(1) ∀ X in Skel(h), skel(I ⊕ J)(X) = r(X) ∪ s(X) ,

(2) ∀ X in Skel(h),
skel(IΘJ)(X)=r(X)-s(X) $\cup_{X \subseteq Y,\ Y \text{ in Skel}(h)} \pi_X[r(Y)-s(Y)]$

and,

(3) ∀ X in Skel(f) ∩ Skel(g),
skel(I ⊛ J)(X) = r(X) ∩ s(X)

∀ X in Skel(f) - Skel(g),
skel(I ⊛ J)(X) = $r(X)*_{Y \subseteq X,\ Y \text{ in Skel}(g)}$ s(Y)

∀ X in Skel(g)-Skel(f),
skel(I ⊛ J)(X) = $s(x)*_{Y \subseteq X,\ Y \text{ in Skel}(f)}$ r(Y)

and,

∀ X ∈ Skel(h)-(Skel(f) ∪ Skel(g)),
skel(I ⊛ J)(X) = ∅

Note that in the previous theorem skel(I ⊕ J) = r ∪ s. It turns out that simple characterization of join and difference can also be obtained using set operations on relational schema instances.

Proposition 3.1. Let f and g be two compatible formats, h an extension of f and g. Let I and J be instances over f and g respectively. Let r = skel(I^h) and s = skel(J^h). Then

(1) skel(I ⊕ J) = r ∪ s
(2) skel(I ⊖ J) is the smallest [2] URSA-instance over Skel(h) containing r-s and,
(3) skel(I ⊛ J) is the largest USRA-instance over Skel(h) contained in the instance t over Skel(h) defined by :

(a) t(X) = r(X) ∩ s(X) if X in Skel(f) ∩ Skel(g)
(b) t(X) = r(X) if X in Skel(f) - Skel(g)
(c) t(X) = s(X) if X in Skel(g) - Skel(f)
(d) t(X) = ∅ otherwise

The previous two results give interpretations of some Verso operations (fusion, join, difference) in terms of relational operations. Although not done here, the other operations can also be interpreted using relational operations on the corresponding relational database instance.

4. DATA RESTRUCTURING

In order to define the last unary Verso-operation, namely restructuring, we need to examine the semantics associated to a Verso instance. Thus, in the first part of this section,

(2) Let r and s be instances over the schema R $r \subseteq s$ iff $r(X) \subseteq s(X)$ for each X in R.

the focus is on the set of "facts" which can be deduced from a format instance. These sets of facts are then used to compare the representative power of formats. Finally, some elementary format transformations which allow data restructuring are presented.

In the following, the elementary unit of information, called a fact, is a tuple.

Two basic operations on sets of facts are considered. They are : the closure under projection and under join.

Definition : Let H be a set of facts. Then the closure of H under projection, denoted $\pi(H)$, is defined by :

$$\pi(H) = \{\pi_Y(x) \ / \ x \text{ in } H \cap TUP(X) \text{ for some } X \text{ and } Y \subseteq X\}$$

and, the closure of H under join, denoted *(H) is defined by :

$$*(H) = \{*_{x \text{ in } H'} \ x \ / \ H' \subseteq H\}.$$

Now, given a set of facts, it seems reasonable to deduce new facts by projection of known facts. The closure under join is already more arguable. For instance, if "toto" is taking "math" and "math" is taught by "Miss Jones", you don't want to conclude that "Miss Jones" is teaching "math" to "toto". The semantics that we are going to associate with format instances states that the "legal" joins are only the joins of tuples in the instance skeleton. More formally, we have :

Definition : Let I be an instance over the format f. Then the set of facts associated with I, denoted fact(I), is defined by :

$$fact(I) = \pi(*(\bigcup_{Z \text{ in } Skel(f)} skel(I)(Z))).$$

The previous definition is illustrated in Figure 4.1 where the set of facts associated with the instance I of Figure 2.1 is given.

The notion of set of facts associated to a format instance is used now to present the last unary operation, namely restructuring. Intuitively, this operation allows to modify the format of an instance without modifying its information content.

Formally we have :

Definition : Let f be a format. Then a restructuring is an expression of the form $format_f$. Let I be an instance. If there exists an instance J over f such that fact(I) = fact(J) then $format_f$ is defined for I and $format_f(I) = J$ (otherwise $format_f$ is not defined for I).

Now the problem arises : given two formats f and g, is it always possible to represent an instance over g by an instance over f ?

In order to answer this question and therefore present a way to compare the representative power of formats, we need the following notation.

Notation : Let f be a format. Then $SAT(f) = \{fact(I) \ / \ I \text{ is in } Inst(f)\}$.
This allows us to compare formats.

Definition : Let f and g be two formats : Then f is dominated by g, denoted $f \leq g$, iff

$$SAT(f) \subseteq SAT(g).$$

Also f and g are equivalent, denoted $f \equiv g$, iff $f \leq g$ and $g \leq f$. (i.e., $SAT(f) = SAT(g)$).

Intuitively, f is dominated by g iff each instance over f can be represented by an instance over g containing the same information. Two characterizations of format dominance are now presented. The first one (Lemma 4.1) is based on properties of the corresponding format skeletons. The second one (Theorem 4.1) is based on some elementary format transformations. We now present the first characterization of format dominance.

Lemma 4.1. : Let f and g be two formats. Then

$$f \leq g \text{ iff } Skel(f) \subseteq Skel(g).$$

Thus $f \equiv g$ iff $Skel(f) = Skel(g)$. □

In order to present the second characterization of format dominance, we exhibit three format transformations. These transformations are presented in their elementary versions and then generalized.

Definition :

a) Let $f \equiv X(f_1)*...(f_n)*$ and $g \equiv Y(g_1)*...(g_n)*$ then : g is obtained from f by elementary root permutation iff

(i) $f_i \equiv g_i$ for each i in [1,n] , and
(ii) set(X) = set(Y).

g is obtained from f by elementary branch permutation iff

(i) for each i in [1,n] , there exists j in [1,n] such that $g_j \equiv f_i$, and
(ii) $X \equiv Y$

b) Let $f \equiv XX_1(f_1)*...(f_n)*$ and $g \equiv X(X_1(f_1)*...(f_n)*)*$ then g is obtained from f by elementary compaction.

Now a root permutation on a format f is obtained by applying elementary root permutations to subformats(3) in f. Branch permutation and

(3) A subformat of a format f is a format corresponding to a subtree of the representation tree of f.

compaction are obtained from elementary branch permutation and elementary compaction in a similar manner. Figure 4.2 exhibits a sequence of these three transformations together with the extension defined in Section 3.

Now we have :

Theorem 4.2. Let f and g be two formats. Then $f \simeq g$ iff g can be obtained from f by a sequence of root and branch permutations. Also $f \leq g$ iff g can be obtained from f by a sequence of root and branch permutations, compactions and extensions. □

The proof follows easily from Lemma 4.1. Even if f is not dominated by g, some particular instances over f are representable by instances over g. That is because those particular instance satisfy some contraints on top of the contraints that are implied by the format f. We now define two kinds of dependencies which are going to capture these contraints.

Definition : Let S be a relational database schema and H be a set of facts. Then *S denotes the schema join dependancy (SJD) associated with S, and H satisfies *S, denoted $H \models *S$, iff[4]

$$H_{|\bigcup_{X \text{ in } S} X} = (*H_{|S})_{|\bigcup_{X \text{ in } S} X} .$$

Also, ∃ S denotes the existence schema dependency (ESD) associated with S and H satisfies ∃ S, denoted $H \models \exists S$, iff $H = \pi(H_{|S})$.

The next result uses the previous dependencies to characterize the sets of facts which are representable by a given format f.

Theorem 4.3 : Let f be a format and H be a set of facts. Then H can be represented by an instance over f iff

(i) $H \models *R$ for each $R \subseteq Skel(f)$

(ii) $H \models \exists S$ where $S=\{X \mid X=\bigcup_{Y \text{ in } R} Y,\ R \subseteq Skel(f)\}$

Note that the SJDs are very similar to Delobel's Generalized Hierarchical Dependencies. To conclude this section, we present a simple result on SED implication.

Proposition 4.1. Let R and S be two relational database schemas and H be a set of facts. Then the following two assertions are equivalent :

(i) $H \models \exists R$ and $H \models \exists S$,

(ii) $H \models \exists R \cap S$.

(4) For each set of facts H, and for each set X of attributes $H_{|X} = \{x/x \text{ in } H \cap Tup(X)\}$. For each schema S, $H_{|S} = \bigcup_{X \text{ in } S} H_{|X}$.

5. Expressive power of Verso operations

We already mentionned that Verso operations are "relationally complete". What we mean by that is : given a Verso format f and its corresponding database schema R = Skel(f), for every relational query q on R there exists a Verso query q' which yield the same result[5].

If Verso operations are complete, they also take advantage of the joins implicitely defined in the schema. To illustrate this remark, we now present a query which would typically require a join in the relational model but can be simply expressed by a selection in the Verso model.

Example 5.1.

Consider the format f=COURSE(STUDENT)*(EXAM-DAY)* Now consider the query : "What are the courses taken by toto which have an exam on November first ?". In the relational model, there would typically be two relations COURSE STUDENT and COURSE EXAM-DAY and the query would require a join operation. This query can be answered by the Verso selection :

$S \equiv \text{Select } \langle S_1, S_2 \rangle_{S_1 \neq \emptyset \wedge S_2 \neq \emptyset}$ where

$S_1 \equiv \text{select}_{\text{STUDENT=toto}}$, and

$S_2 \equiv \text{select}_{\text{EXAM-DAY=november 1st}}$.

Indeed, some very natural queries like "Give the list of courses with no known exam day" can be answered by a Verso selection whereas they would require the use of difference in a pure relational model.

To conclude this section, we propose a simple extension of the Verso selection which dramatically increases its power. Let us consider the following query on COURSE(STUDENT(GRADE)*)* : "Give the list of courses, students and grades such that toto got an A in the course and a student (not necessarily toto) got an F in the course". It should be noted that this query is complicated by the fact that they are several roles for the same attribute, namely STUDENT. Typically, such a query would require several joins in the classical relational model.

What we mean by such a query is in fact two selections on GRADE, say $S_1 \equiv \text{select}_{\text{GRADE=A}}$ and $S_2 \equiv \text{select}_{\text{GRADE=F}}$.

Now we need two selections on STUDENT(GRADE)* :

(5) To be precise, q' yield "almost" the same result since the result of q is a relation and the result of q' will be an instance over a flat format, i.e., a set of ordered tuples.

$S'_1 \equiv$ select $\langle S_1 \rangle_{STUDENT=toto \wedge S_1 \neq \emptyset}$

$S'_2 \equiv$ select $\langle S_2 \rangle_{S_2 \neq \emptyset}$

The first one filters toto if he got an A, and the second one any student who get an F. Now we can have

$S \equiv$ select $\langle S' \rangle_{S'_1 \neq \emptyset \wedge S'_2 \neq \emptyset}$ where

S' is the identy on STUDENT(GRADE)*.

It should be noted that this is not a selection as defined in Section 2 since two selections are realized on the subformat STUDENT(GRADE)*. Intuitively, these two selections S'_1 and S'_2 should be performed "in parallel". They are used exclusively as conditions.

In the complete paper, this "super" selection is formally defined. The following result is proved there :

Theorem 5.1 : Let f be a format, R=Skel(f) the corrresponding database schema. Let q be a selection-projection-join query on R such that every projection in q is a projection on some union of attribute sets in R. Then these exists a Verso query q' consisting of a (Verso) super selection followed by a (Verso) projection, and such that q' is equivalent to q.

Conclusion

The Verso model is a database model based on some particular (not first normal form) relations allowing incomplete information representation and implicit specification of meaningful joins. Furthermore, a simple algebraic query language can be defined for the Verso model. The Verso model is a formalization of some of the concepts used in the Verso database machine [Ba1]. The data structure in the Verso machine is a Verso instance fully sorted according to the format and stored sequentially. This storage organization allows fast access and processing [BRS]. Indeed, all the operations presented in the paper (including the "super" selection) but the format modification can be realized by the specialized filter of the Verso machine. Typically, some queries which would require joins in the relational model can be realized by the filter if the joins are implicit in the Verso format. However, joins which are not implicitely in the format remain quite costly since they involve format modifications.

References

[Ba1] F. Bancilhon & al., "Verso : A Relational Back End Data Base Machine", Proc. Inter. Worshop on Database Machines, San Diego, 1982.

[Ba2] F. Bancilhon & al., "Les V-relations : Definitions, Modifications, Interrogation", Tech. Notes VERSO 1, 1982.

[BRS] F. Bancilhon, P. Richard, M. Scholl, "On Line Processing of Compacted Relations", Proc. Inter. Conf. on VLDB, Mexico, 1982.

[C] E.F. Codd, "A Relational Model of Data for Large Shared Data Banks", CACM 13, N° 6, 1970.

[D] C. Delobel, "Normalization and Hierarchical Dependencies in the Relational Data Model", ACM Trans. on Database Systems, 3, 1978.

[FMU] R. Fagin, A. Mendelzon, J. Ullman, "A Simplified Universal Relation Assumption and its Properties", Trans. on Database Systems, 7, 1982.

[HY] R. Hull, C.K. Yap, "The Format Model : A Theory of Database Organization", Proc. of Symp. on Princ. of Database Systems, 1982.

[IMS] Information Management System/360, Version 2, General Information Manual, IBM form #GH20-0765.

[JS] G. Jaeshke, H.J. Scheck, "Remarks on the Algebra of Non First Normal Form Relations", Proc. ACM SIGACT-SIGMOD, Los Angeles, 1982.

[K] I. Kobayashi, "An Overview of the Database Management Technology", Techn. Report TRCS-4-1, Sanno College, Kanagawa 259-11, Japan, 1980.

[Mai] D. Maier, "The Theory of Relational Databases", Computer Science Press.

[MW] D. Maier, D. Warren, "Specifying Connections for a Universal Relation Scheme Database", Proc. SIGMOD, 1982.

[Mak] A. Makinouchi, "A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model", Proc. Inter. Conf. on VLDB, Tokio, 1977.

[PS] P. Pauthe, Seznec, "An Editor for Verso Relations" D.E.A. Report, Orsay University, 1983.

[SP] H-J Scheck, P. Pistor, "Data Structures for an Integrated Data Base Management and Information Retrieval System", Proc. Inter. Conf. on VLDB, Mexico, 1982.

[U] J.D. Ullman, "Principles of Database Systems", Computer Science Press.

[V] A. Verroust, "Characterization of Well-Behaved Database Schemetas and their Update Semantics", Proc. Inter. Conf. on VLDB, Florence, 1983.

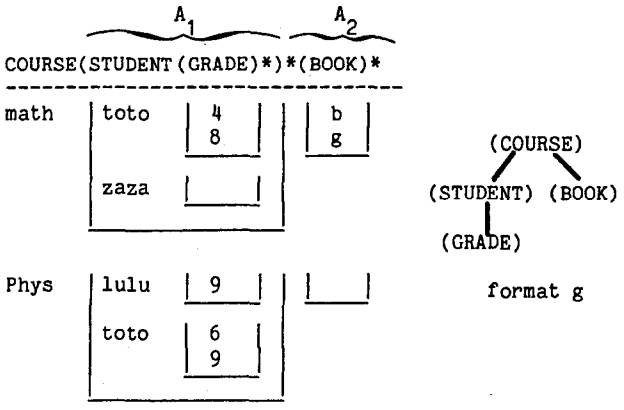
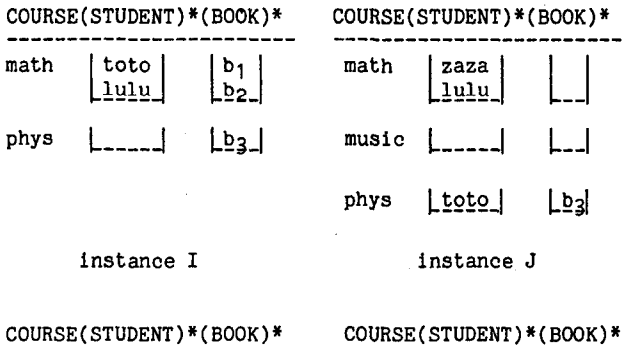


Figure 2.1



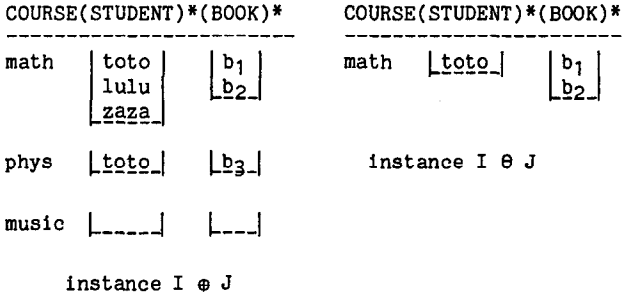


Figure 2.2.

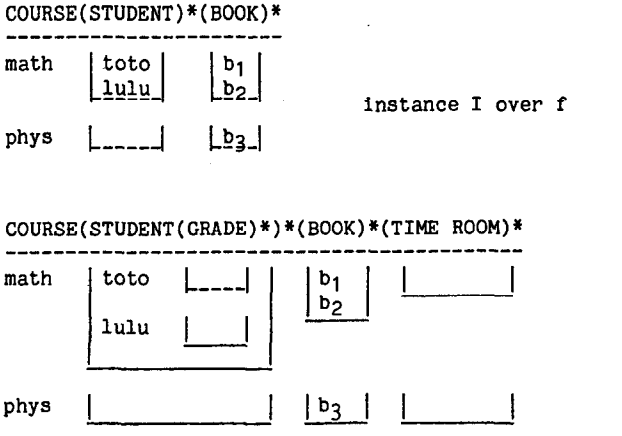
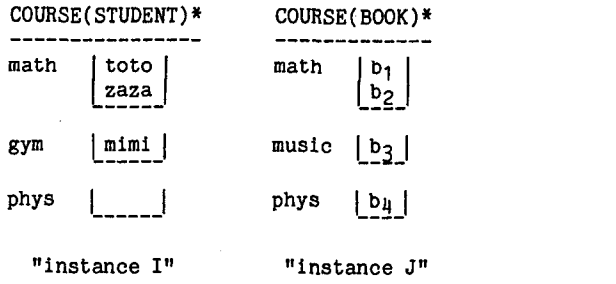


Figure 2.3



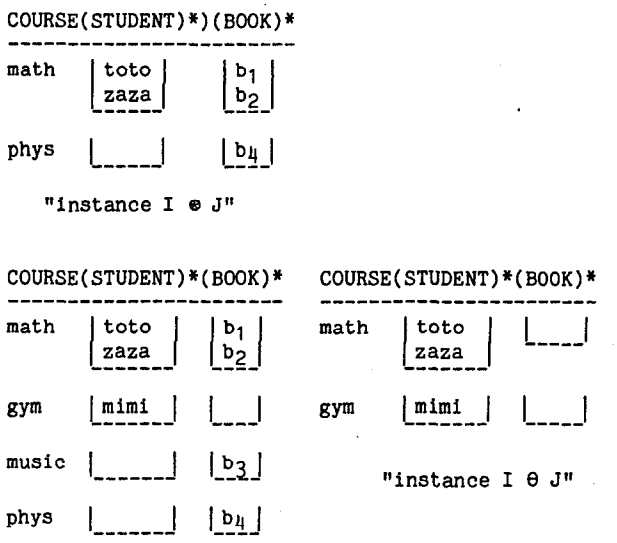


Figure 2.4

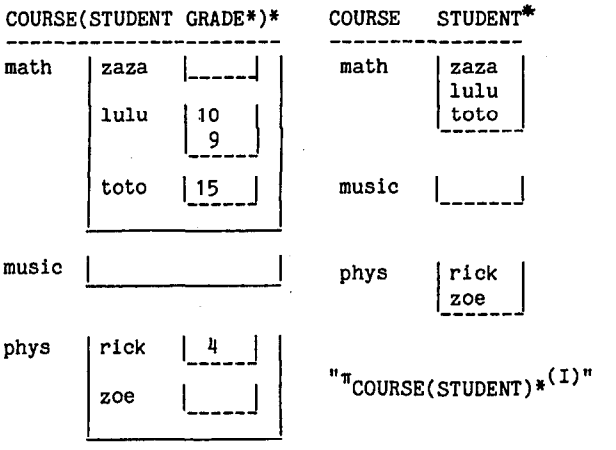
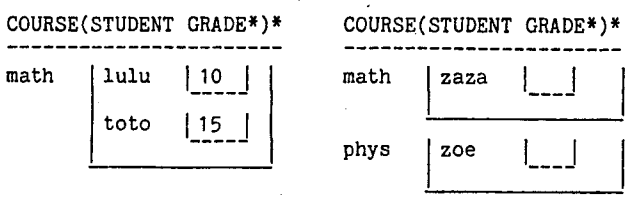
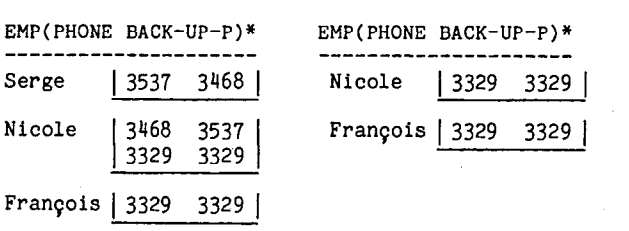


Figure 2.5 (part I)



"$S_1(I)$" "$S_2(I)$"

Figure 2.5 (part II)



"instance J" "$\text{restrict}_{\text{PHONE=BACK-UP-P}}(J)$"

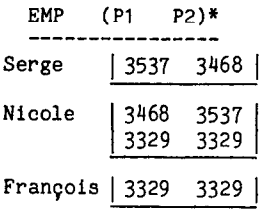


Figure 2.6

"$\text{rename}_{\text{PHONE} \to \text{P1}, \text{BACK-UP-P} \to \text{P2}}(J)$"

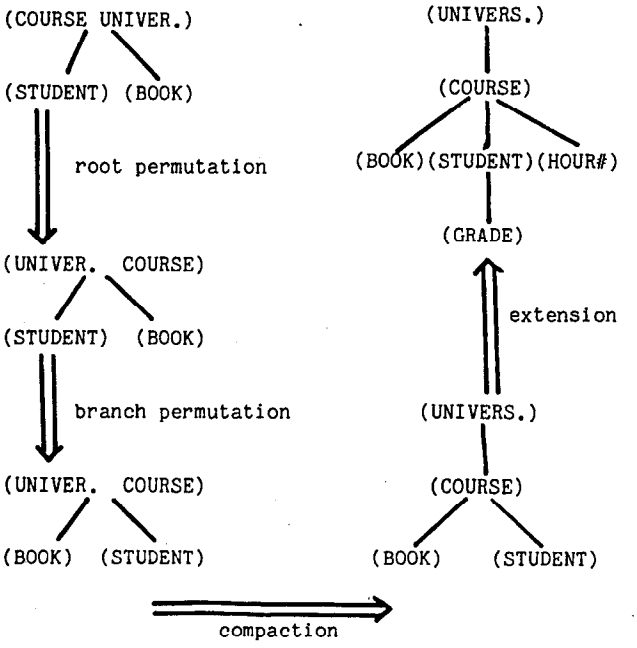
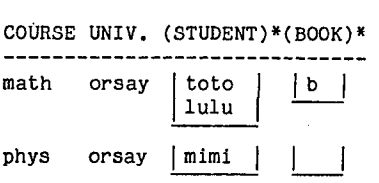


Figure 4.2 (Part I)



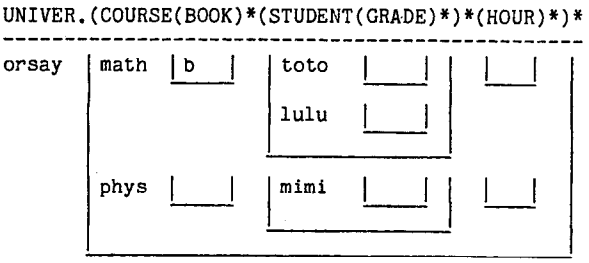


Figure 4.2

COUR.	COUR. STUD	COUR. STUD GRADE	COUR. BOOK
math	math toto	math toto 4	math b
phys	math zaza	math toto 8	math g
	phys lulu	phys lulu 9	
	phys toto	phys toto 6	
		phys toto 9	

Figure 3.1

<math,toto,4,b>, <math,toto,8,b>,

<math,toto,4,g>, <math,toto,8,g>,

<math,toto,4>,...,<math,4>,...,<8,g>,

<phys,lulu,9>, <phys,toto,6>, <phys,toto,9>,

<phys,lulu>,...,<toto,9>,

Fact(I)

Figure 4.1