USE OF DATA REPRESENTATION MAPPING IN AUTOMATIC
GENERATION OF DATA BASE ACCESS PROCEDURES

Koichi Furukawa

Eléctrotechnical Laboratory
Ibaraki, Japan

ABSTRACT

Formal definition of data structure mappings are given and
used to transform relational algebraic formula into data
base access procedures which search a formally defined data
structure. Especially, a hierarcical data structure is
introduced to represent a set of relations which are
inherently hierarchical. We show that a retrieval procedure
using data accesses along a two level hierarchy can be
obtained from an original formula on flat relations by
applying equivalence transformation rules and data
representation mappings.

1. INTRODUCTION

During the latest decade, data base theories and data
structure theories have been developed independently. In
data base area, much research has been concentrated on
relational data base theories. As far as query processing is
concerned, many logical query languages as well as deductive
query evaluation systems have been proposed. Query optimizations and processing based on relational algebra have also been investigated by many researchers.

On the other hand, theoretical researches on data structure have been developed in the field of program correctness studies [Hoare 72]. Recently, [Hansson 79] developed a framework to formally define data structures and prove correctness of programs manipulating them.

In this paper, we will combine the above two theoretical approaches and put them in a framework of data abstraction theory. Namely, we regard relations as objects of an abstract data type and relational algebra as its associated operations. Furthermore, we formally define data structures and represent relations by them. Our final goal is to transform relational algebraic query formulas into data base access procedures which search the defined data structures. Especially, we consider to represent a set of relations which are inherently hierarchical by a hierarchical data structure. We show that a retrieval procedure using data accesses along a two level hierarchy can be obtained from an original formula on flat relations by applying equivalence transformation rules and data representation mappings.

2. RELATION AND RELATIONAL ALGEBRA

First, we define the relational data base formally.
Definition. Let $D_1, \ldots, D_n$ be $n$ domains (not necessarily distinct) of elements and $A_1, \ldots, A_n$ be $n$ identifiers which are associated with $D_1, \ldots, D_n$ respectively. A relation $R$ on the set of identifiers $\{A_1, \ldots, A_n\}$ is a subset of the cartesian product $D_1 \times \ldots \times D_n$. $A_i$ is called an attribute of the relation $R$. The relation $R$ on the attribute set $\{A_1, \ldots, A_n\}$ is denoted by $R(A_1, \ldots, A_n)$.

We use $A, B, \ldots$ for a single attribute and $X, Y, \ldots$ for a set of attributes (including the null set).

Definition. A relation type $R(A_1, \ldots, A_n)$ on the attribute set $\{A_1, \ldots, A_n\}$ is a set of all relations on the set $\{A_1, \ldots, A_n\}$.

A relation $R(A_1, \ldots, A_n)$ is an element of the relation type $R(A_1, \ldots, A_n)$. A conceptual schema which determines the logical data structure is defined by a set of relation types. In this paper, we regard hierarchical data organizations to be matters of internal schema, which determine the physical data structure.

Figure 2-1 shows an example of a relational data base where the conceptual schema is the set, $\{\text{EMP(emp, loc), ROOM(room#, sect)}\}$.

Next, we define the relational algebraic operations ([Codd 72], [Reiter 78]).

Definition. Let $R(A, X)$ be a relation. Then, the projection $\Pi_A$ of $R$ with respect to $A$ is a relation given by the equation
\[ \prod_A (R) = \{ x : (\exists z) <z, x> \in R \} \]

where \( x \) represents the sequence of values corresponding to the attribute set \( X \) and \( z \) the value corresponding to a chosen attribute \( A \).

**Definition.** Let \( R(X) \) be a relation, and \( y \) be a variable which corresponds to \( X \). Let \( f \) be a logical function \( X \rightarrow \{T, F\} \). Then, the selection of \( R \) with respect to \( f(y) \), denoted by \( \mathcal{R}_{f(x)}(R) \), is a set of \( x \) values given by the equation

\[ \mathcal{R}_{f(x)}(R) = \{ z : \langle z, x \rangle \in R \land f(x) = T \} \].

The selection operation \( \mathcal{R}_{f(x)}(R) \) is an associative retrieval of the relation \( R \) under the condition that \( f(y) = T \).

**Definition.** Let \( R_1(A, X) \) and \( R_2(B, Y) \) be relations, and \( f(A, B) \) a logical function. Then, the join of \( R_1(A, X) \) and \( R_2(B, Y) \) with respect to \( f(A, B) \) is denoted by \( R_1(A, X) \star_{f(A,B)} R_2(B, Y) \), and is given by

\[ R_1(A, X) \star_{f(A,B)} R_2(B, Y) \]

\[ = \{ \langle a, b, v \rangle : \langle a, x \rangle \in R_1 \land \langle b, v \rangle \in R_2 \land f(a, b) = T \} \].

If the join condition \( f(A, B) \) is given by an equality formula such as \([\text{loc} = \text{room}]\), then \( a = b \) always holds in the above set expression and the answer becomes redundant. In this case, we remove one of the two attributes from the result. This modified join is called the natural join.
3. QUERY TRANSFORMATION INTO HIERARCHICAL DATA ORGANIZATION

Figure 3-1 displays the logical structure of the data base given in Figure 2-1. This diagram is a Backman diagram and uses arrows to indicate one-to-many mappings (Backman [1969]). One-to-many mappings correspond to hierarchical relations and the arrows represent the access paths in hierarchical data bases; namely, subordinate data are retrieved by specifying the superordinate data and the access path between them.

Figure 3-2 shows an example of a concrete tree structure which has the logical structure given in Figure 3-1. Traversal from the root node in Figure 3-2 corresponds to retrieval along access path "a" in Figure 3-1. Given a piece of section information, we select an appropriate arc and follow it to reach the set of room#'s which belong to the section.

Let us assume that each set of room#'s forms a relation distinctively. We denote a set of room#’s under the selector "sect = ai" as ROOM.ai(room#). This relation can be defined by a selection followed by a projection as follows:

\[ \text{ROOM.ai(room#)} = \prod_{\text{sect} \in \Phi} (\text{ROOM(room#,sect)}). \quad (3.1) \]

If we regard ai to be a variable, then the above formula defines any relation under any selector.
Next, we define a hierarchical data organization for the EMP relation similar to that which was defined for the ROOM relation. Let us denote EMP.bj(emp) to be a relation which comprises the employees who are located at bj. Then, the formula corresponding to (3.1) is given by:

\[ EMP.bj(emp) = \prod_{loc \in \text{loc=bj}} (EMP(emp,loc)). \] (3.2)

Let us now consider the example query "Get all employees who belong to INFO section":

\[ Q = \prod_{loc \in \text{loc=INFO}} \prod_{sect \in \text{sect=INFO}} (EMP(emp,loc) \times (ROOM(room#,sect))). \] (3.3)

It has been shown in [Furukawa 79] that Q is equivalently transformed to:

\[ Q = \bigcup_{bj \in ROOM.INFO(room#)} EMP.bj(emp). \] (3.4)

The evaluation of formula (3.4) proceeds as follows: first, compute ROOM.INFO(room#) according to the definition (3.1); then retrieve EMP.bj(emp) for all bj in ROOM.INFO(room#); finally collect them together. This procedure is exactly the structural retrieval along access path comprising "a" and then "b" in Figure 4-2. However, this access method is somewhat different from that of the hierarchical data bases. In order to retrieve all of the employees who are working in room bj, the above method relies upon the use of associativity rather than a pointer to access EMP.bj
relation. Although associative retrieval can be realized very efficiently with hashing techniques and/or specialized hardware, it is still less efficient than the pointer access method. We will show further transformation of expression (3.4) based on the data structure mapping to realize the exact access method of the hierarchical model.

4. FORMAL DEFINITION OF DATA STRUCTURES AND PRIMITIVE FUNCTIONS

In this section, we formally define a hierarchical structure and its related primitive functions to manipulate the structure. In defining a data structure, we adopt a framework developed by [Hansson 79].

First, we define a list consisting of arbitrary components.

Axiom 1
\[ \forall w \{ \text{list}(w) \leftrightarrow w = \text{NIL} \lor \exists x, y \in \text{list}(y) \} \]

Then, we define a selection list (slist) to represent a set of subrelations such as \{ EMP.r : r \in \text{room}\}.

Axiom 2
\[ \forall w \{ \text{slist}(w) \leftrightarrow w = \text{NIL} \lor \left( \exists x, t_1 \in \text{EsEtl}[w = (s, x), t_1 \land \forall s' (s', x') \in t_1 \rightarrow s' = s) \land \text{list}(x) \land \text{slist}(t_1) \right) \} \],

where \( \in \) is a membership predicate for slist and will be
defined later.

This axiom defines a list of pairs whose first components differs from each other. We introduce the following useful lemmas concerning to the slist.

**Lemma 2.1** \( \text{slist}(\text{NIL}). \)

**Lemma 2.2**

\[ \forall s \forall x \forall w \{ \text{slist}(\langle s, x \rangle, w) \leftrightarrow \forall s' \forall x' [\langle s', x' \rangle \leftarrow w \rightarrow s' \text{\&} x \text{\&} \text{slist}(w)] \}. \]

Next, we define a membership predicate as follows:

**Axiom 3.**

\[ \forall e \forall w \{ e \in w \leftrightarrow \text{slist}(w) \text{\&} E\text{hd}\text{tl}[\text{hd}=e \lor e \in \text{tl}] \}. \]

Like Axiom 2, we also introduce the following lemmas:

**Lemma 3.1** \( \forall e \{ e \in \text{NIL} \}. \)

**Lemma 3.2** \( \forall e \forall w \{ e \in e.w \leftrightarrow \text{slist}(w) \}. \)

**Lemma 3.3**

\[ \forall e \forall \text{hd.tl} \{ e \in \text{hd.tl} \leftrightarrow \text{slist}(\text{hd.tl}) \text{\&} e \in \text{tl} \}. \]

We are interested in representing hierarchy using slists. The following axiom defines a hierarchical data structure called multi-level slist structure or simply
mls-structure in terms of slist.

Axiom 4.

\[ Vw[\text{mls-structure}(w)] \]
\[ \iff w = \text{NIL} \lor \text{slist}(w) \land VsVx[\langle s, x \rangle \in w \rightarrow \text{slist}(x)]. \]

Finally, we define a double list (dlist) whose components are all lists.

Axiom 5

\[ Vw[\text{dlist}(w)] \]
\[ \iff w = \text{NIL} \lor \text{EhdEt1}[w = \text{hd}.tl \land \text{list}(\text{hd}) \land \text{dlist}(\text{tl})]. \]

Now, we introduce primitive predicates and their corresponding functions for manipulating slist and/or dlists.

(1) \text{assoc}(s, w, x)

\text{assoc}(s, w, x) corresponds to a LISP ASSOC function and is equal to "x = \text{ASSOC}(s, w)". A formal definition of \text{assoc} is given as follows:

Axiom 6

1. Vs[assoc(s, NIL, NIL)].
2. VsVxVw[assoc(s, s, x .w, x) \iff \text{slist}(s, x .w)].
3. VsVs'VxVx'Vw[assoc(s, s', x' .w, x) \iff \text{slist}(s', x' .w) \land s' \approx s' \land \text{assoc}(s, w, x)].
assoc(s,w,x) is similar to \( \langle s,x \rangle \in w \). They differ only when \( w \) is an empty list, where assoc(s,NIL,x) is true for \( x=\text{NIL} \), but \( \langle s,x \rangle \in w \) becomes false. Relations between assoc and \( \in \) are stated by the following two theorems.

**Theorem 1** \( \forall s\forall x\forall w\{\langle s,x \rangle \in w \rightarrow \text{assoc}(s,w,x)\} \).

**Theorem 2** \( \forall s\forall x\forall w\{\lnot \langle s,x \rangle \in w \rightarrow \text{assoc}(s,w,\text{NIL})\} \).

(2) project-sublist(\( w,w' \))

project-sublist(\( w,w' \)) corresponds to a projection \( \Pi \) of a relational algebra. It projects out first components of each pair of \( w \) and obtains \( w' \) consisting of all of the second components of the pairs. Formally, it is stated as follows:

**Axiom 7**

1. project-sublist(\( \text{NIL},\text{NIL} \)).
2. \( \forall s\forall x\forall w\forall w'\{\text{project-sublist}(s,x \cdot w,x \cdot w') \leftarrow \text{slist}(s,x \cdot w) \& \text{project-sublist}(w,w')\} \).

We denote a function corresponding to project-sublist(\( w,w' \)) as \( \text{PRSUBL}(w) \).

(3) project-selector(\( w,w' \))

project-selector(\( w,w' \)) is similar to project-sublist(\( w,w' \)). The roles of first components and the second components are interchanged.
Axiom 8

1. project-selector(NIL,NIL).
2. VsVxWwBw'[project-selector(s,x,w,s.w')]
   \[\leftrightarrow \text{ slist}(s,x\,\cdot\,w) \land \text{project-selector}(w,w')].

A corresponding function is denoted by PRSEL(w).

(4) dunion(w,w')

\[\text{dunion}(w,w')\] is a predicate which manipulates dlists
and makes a list \(w'\) consisting of all elements of all
components of \(w\).

Axiom 9

1. dunion(NIL,NIL).
2. VxVwWw'[dunion(x,w,APPEND(x,w'))]
   \[\leftrightarrow \text{ dlist}(x,w) \land \text{dunion}(w,w')].

A corresponding function is denoted by DUNION(w).

5. DATA STRUCTURE MAPPING AND FURTHER QUERY TRANSFORMATION

In order to represent ROOM.s and EMP.r using an
mls-structure, we need to associate each room\# in ROOM.s
relations with EMP.r. We introduce a pseudo relation called
HROOM.s(room\#, Emp) where Emp is a pointer to an EMP.r
relation. The original ROOM.s(room\#) relation is obtained
from HROOM.s(room\#, Emp) by

11
ROOM.s(room#) = \prod_{\text{Emp}} (HROOM.s(room#, Emp)) \tag{5.1}

We organize HROOM.s and EMP.r in a ROOM-FILE as shown in Figure 5-1, where ROOM-FILE satisfies the following assertion:

Assertion 1 \ mls-structure(ROOM-FILE).

The mappings from ROOM-FILE to HROOM.s and EMP.r are given as follows:

Mapping 1 \ For any section s,

\[
\begin{align*}
&\text{if } \text{Ex}[<s,x> \in \text{ROOM-FILE}] \\
&\text{then } \text{Repr(ASSOC(s,ROOM-FILE))} = HROOM.s \\
&\text{else } HROOM.s = \emptyset.
\end{align*}
\]

From Assertion 1 and Mapping 1, it is shown that any non-empty HROOM.s is represented by a selection list. Namely, the representation function \text{Repr} maps a selection list to a pseudo relation.

Each EMP.r is then represented by a second component of some element of the selection list corresponding to an HROOM.s such that r belongs to s.

Mapping 2 \ For any room# r,

1. \left\{ \begin{align*}
&\text{Ww} [\text{EsEx}[<s,w> \in \text{ROOM-FILE} \& <r,x> \in w] \\
&\quad \rightarrow \text{Repr(ASSOC(r,w))} = \text{EMP.r}]
\end{align*} \right.

2. \neg \text{EwEsEx}[<s,w> \in \text{ROOM-FILE} \& <r,x> \in w] \rightarrow \text{EMP.r} = \emptyset.
From Mapping 1 and 2, it is shown that the following equations holds:

**Mapping 1'**  For any section s, HROOM.s is obtained from

\[ HROOM.s = \text{Repr}(\text{ASSOC}(s, \text{ROOM-FILE})). \]

**Mapping 2'**  For any room# r, EMP.r is obtained from

\[ EMP.r = \bigcup_{w \in \text{PRSUBL}(\text{ROOM-FILE})} \begin{cases} \text{Repr}(\text{ASSOC}(r,w)) & \text{if } r \in \text{PRSEL}(w) \\ \emptyset & \text{else} \end{cases}. \]

We are now able to construct a mapping from ROOM-FILE to each ROOM.s:

**Mapping 1''**  For any section s, ROOM.s is obtained from

\[ ROOM.s = \text{Repr}(\text{PRSEL}(\text{ASSOC}(s, \text{ROOM-FILE}))). \]

Now, let us consider how to transform the query (3.4) into a program which searches the ROOM-FILE mls-structure.

By applying Mapping 1'' and 2' to (3.4), we obtain

\[ Q = \bigcup_{r \in \text{Repr}(\text{PRSEL}(\text{ASSOC}(\text{INFO}, \text{ROOM-FILE})))} \bigcup_{w \in \text{PRSUBL}(\text{ROOM-FILE})} \begin{cases} \text{Repr}(\text{ASSOC}(r,w)) & \text{if } r \in \text{PRSEL}(w) \\ \emptyset & \text{else} \end{cases}. \]  \tag{5.2}

Since the functional dependency room# → section holds, r belongs to only one section (in this case, INFO), and "r \in PRSEL(w)" becomes false for any other sections. Therefore, (5.2) is reduced to
\[ Q = \bigcup_{r \in \text{Repr(\text{PRSEL(ASSOC(INFO,ROOM-FILE))})}} (\text{Repr(\text{ASSOC}(r,\text{ASSOC(INFO,ROOM-FILE)})))} \]

(5.3)

Generally, a reduction theorem from (5.2) to (5.3) is stated as follows:

**Theorem 3** Let \( M \) be an mls-structure, \( S \) be a constant corresponding to a first level selector of \( M \) and \( f \) be an arbitrary function. Then,

\[ \bigcup_{r \in \text{PRSEL(\text{ASSOC}(S,M))}} \big( \bigcup_{w \in \text{PRSUBL}(M)} (\text{if } r \in \text{PRSEL}(w) \text{ then } f(r,w) \text{ else } \emptyset) \big) \]

\[ = \bigcup_{r \in \text{PRSEL(\text{ASSOC}(S,M))}} f(r,\text{ASSOC}(S,M)). \]

Since the range of \( r \) covers the entire set of all first components of pairs in \text{HROOM.INFO}, we do not need to search \text{HROOM.INFO} list for each \( r \), but simply union all second elements of pairs in \text{HROOM.INFO}. The following theorem suggests us to use a DUNION function to compute the above union.

**Theorem 4** Let \( W \) be an slist. Then,

\[ \bigcup_{r \in \text{PRSEL}(W)} (\text{Repr(\text{ASSOC}(r,W)))} \]

\[ = \text{Repr(DUNION(PRSUBL(W)))}. \]

By applying Theorem 4 to (5.3), we obtain the following simple program to compute \( Q \):

\[ Q = \text{Repr(DUNION(PRSUBL(\text{ASSOC(INFO,ROOM-FILE)))))} \]

(5.4)
Axiom 6, 7 and 9 define ASSOC, PRSUBL and DUNION respectively. These axioms can be considered to be PROLOG program segments and can be executed directly. It is also possible to generate LISP programs by adding a suitable set of transformation rules and applying the same transformation methods as above.

6. CONCLUDING REMARKS

In this paper, we have represented a set of subrelations by a formally defined list structure and generated a data retrieval procedure which manipulates the structure given a higher level relational algebraic query formula.

The idea that we represent relations instead of tuples or values in tuples is closely related to the data abstraction concept. We do not care the detailed structure of each relation, which is another lower level problem to be solved separately.

We also have shown that during the transformation of relational algebraic formulas into data structure manipulation procedures, simplifications due to the structure are performed. We think this simplification process is very important in data abstraction theory.

[Yonezawa 80] has attacked a similar problem to ours in the frame of first order logic. Yonezawa has achieved a very
neet method to generate recursive procedures for recursively defined queries.

Further consideration is needed to see whether our methods can be applied to the recursive case or not. We also plan to apply our methods to the network case.

ACKNOWLEDGEMENT

The author is grateful to Drs. Osamu Ishii and Akio Tojo for the opportunity to make the present study and to the staffs of the Information Systems Section and especially to Dr. Akinori Yonezawa of Tokyo Institute of Technology for their stimulus and helpful discussions.

REFERENCE


EMP(emp, loc)

SMITH 101
JONES 102
BROWN 101
HENRY 301
NELSON 102
MURPHY 201
LONG 202
MORGAN 201
LEE 202

ROOM(room#, sect)

101 INFO
102 ARCH
201 LANG
202 INFO
301 ARCH

Figure 2-1. An Example Relational Data Base.

sect
  ↓ a
room#
  ↓ b
emp

Figure 3-1. A Backman diagram for EMP, ROOM relational data base.
Figure 3-2. A structural organization for the ROOM and EMP relations.

Figure 5-1. Organization of HROOM.s and EMP.r in the ROOM-FILE mls-structure.