Developing a Deductive Relational Database for Uniform Handling of Complex Queries

YOSIHISA UDAGAWA and SETSUO OHSUGA*

This paper discusses design and implementation of a database system based upon the predicate logic. Logic was chosen as the design principle since it provides a useful way to derive the facts derivable by using general axioms together with facts stored explicitly in a data base. Since first order logic is not broad enough to express some of the practical queries, we define a formal language for uniform handling of complex queries, called the multi-layer logic for relational databases. Roughly speaking, this logic is obtained by introducing a method of structuring types, i.e., set, into the first order many-sorted logic. As the result, this formal language has enough descriptive power to express practical queries which contain functions whose arguments are grouped by some other variables and/or which involve nested aggregation functions.

In this paper, we define the multi-layer logic for relational databases and discuss how to describe a query against relational databases with a formula in this logic. A query including virtual relations could not be reduced directly into retrieval procedures. To evaluate such a query, it is necessary to transform a query into one that contains no virtual relations. Query transformation algorithm is discussed. Implementation of SBDS-F3 is also given.

1. Introduction

This paper discusses design and implementation of a deductive relational database system for uniform handling of complex queries. The system is termed SBDS-F3 (Structure Based Deduction System-Fortran version 3). According to C. L. Chang [2], our research belongs to evaluational approach.

Query languages based on predicate logic have several advantages, e.g.

(1) to permit a user to request the data by its values;
(2) to provide a useful way to derive the facts derivable by using general axioms together with the facts stored explicitly in a database;
(3) to allow a database system to optimize execution of queries.

However, it is pointed out that this class of languages suffer from lack of expressive power. Query features for relational databases are generally categorized as follows:

(1) mapping: data values of a specified attribute associated with a known data value of other attribute;
(2) selection: entire record values associated with a specified key value;
(3) projection: data values of a specified attribute, domain and/or relation;
(4) restriction: data values that satisfy given conditions;
(5) universal quantification: data values that corresponding to all the values of a specified attribute or domain;
(6) arithmetic function: e.g. $+$, $-$, $*$, $\div$;
(7) aggregation function: e.g. average, minimum, maximum;
(8) group by: grouping of data values with a common domain value;
(9) composition: composition of (1) to (8).

In first order predicate logic, whether one-sorted or many-sorted, domains of variables are fixed during the interpretation of formulas. This means that only domain-element relations can be described in this logic. Thus, queries classified into (7), (8) and (9) are difficult to describe by languages based upon first order predicate logic. So far, various approaches have been proposed for extending the expressive power of query languages based on predicate logic. For example, E. F. Codd [1] uses built-in functions to manipulate functional operations. C. L. Chang [2] proposes to introduce numerical quantifiers, special symbols with the ad hoc meanings and procedural elements. S. Kunihuji [3] proposes the second-order m-sorted logic ($m \geq 0$), and formulates the systax and the semantics of his logic by model theoretic approach. But, as far as the authors are aware, they do not provide systematic solution for the problem.

In this paper, we define the multi-layer logic for relational databases, an extended many-sorted logic, and discuss implementation of a relational database system based on this logic. Roughly speaking, this logic is obtained by introducing a method of structuring types, i.e. set, into the first order manysorted logic. As the result, it has enough descriptive power to express practical queries which contain functions whose arguments are grouped by some other variables and/or which involve nested aggregation functions. This extension has some analogy to the introduction of methods of combining types in a general-purpose programming

*Institute of interdisciplinary research faculty of engineering, Tokyo University 4-6-1 Komaba, Meguro-ku, Tokyo 153, Japan.

language.

In Section 2, we define the multi-layer logic for relational databases and provide some examples. Section 3 discusses how to describe a query against relational databases with a formula in the multi-layer logic. Section 4 concerns with virtual relations and a query transformation (deduction) algorithm for a query involving them. In Section 5, we discuss the implementation of a database system SBDS-F3.

2. Definition of the Multi-Layer Logic for Relational Databases

2.1 Syntax

Definition 1. The alphabet of the multi-layer logic for relational databases is composed of the following:

(1) variable symbols: $x, y, \ldots, X, Y, \ldots$;
(2) function symbols: f, g, ...;
(3) predicate symbols: P, Q, ...;
(4) logical connectives and quantifiers: ~, &, V, $\rightarrow$, $\equiv$, $\exists$, $\forall$;
(5) punctuations: /, [,], ', (,), comma;
(6) truth symbols: T, F,
(7) domain symbols: I, S, ...;
(8) a set operator symbol: *.

In the multi-layer logic, as is the case of the many-sorted logic, it is assumed that there is a non-empty set I, whose elements are called base sorts. For each base sort i, there is a non-empty set D_i.

Definition 2. A set of sorts J(I) having a base sort set I is defined by a finite number of applications of the following two rules.

(1) If $i \in$ I, then $i \in$ J(I).
(2) If $i \in$ J(I), then $2^i - \{0\} \in$ J(I).

Definition 3. The sort of powerset *S of a domain S of a sort $j \in$ J(I) excluding the empty-set is $2^j - \{\phi\}$.

Example 1. If S={A, B}, then *S={{A}, {B}, {A, B}} and **S={{{A}}, {{B}}, {{A, B}}, {{A}, {B}}, {{A}, {A, B}}, {{B}, {A, B}}, {{A}, {B}, {A, B}}}. If S is a domain of a sort j, then *S and **S are domains of sorts $2^j - \{\phi\}$ and $2^{2^j-(\phi)} - \{\phi\}$.

Definition 4. Base elements of a domain are elements whose internal structures are ignored except for the purpose of identifying them.

Definition 5. Level of a base element is defined as 0, and level of a set of base elements as 1. If level of a set S is i, then level of a set *S is $i+1$.

Example 2. {{−3, 4}, {0, 8, 2}} is a set of level 2, and base elements are −3, 4, 0, 8, 2.

Definition 6. A term is defined recursively as follows:

(1) a constant of a sort $j \in$ J(I) is a term of a sort j;
(2) a variable defined over a domain of a sort $j \in$ J(I) is a term of a sort j;
(3) if $t_1, \ldots, t_n$ denote terms of sorts $j_1 \in$ J(I), ..., $j_n \in$ J(I) and f is a n-place function symbol from sorts $j_1, \ldots, j_n$ to a sort $j_f \in$ J(I), then $f(t_1, \ldots, t_n)$ is a term of a sort j_f, where $n \geq 1$.

Definition 7. If $x_0, x_1, \ldots, x_n$ ($n \geq 0$) are variables defined over S, *S, ..., $*_n \ldots *_1$S, respectively, then $(Q_n x_n/*_n \ldots *_1 S)(Q_{n-1} x_{n-1}/x_n) \ldots (Q_0 x_0/x_1)$ is a prefix in the multi-layer logic, where Q_i ($0 \leq i \leq n$) is either $\forall$ or $\exists$. When $n=0$, a prefix is $(Q_0 x_0/S)$, which is a prefix in the many-sorted logic.

Definition 8. A formula in the multi-layer logic is defined recursively as follows:

(1) if $t_1, \ldots, t_n$ are terms of sorts $j_1 \in$ J(I), ..., $j_n \in$ J(I) and P denotes a n-place predicate symbol, then $P(t_1, \ldots, t_n)$ is a (atomic) formula, where $n \geq 0$.
(2) if A and B are formulas, then so are (~A), (A&B), (A ∨ B), (A→B), (A≡B);
(3) if $P(x_1, \ldots, x_n)$ is a formula containing $x_1, \ldots, x_n$ as free variables and $x_1, \ldots, x_n$ are defined over domains $S_1, \ldots, S_n$ of sorts $j_1 \in$ J(I), ..., $j_n \in$ J(I), respectively, then $(Q_{i_1} x_{i_1}/S_{i_1}) \ldots (Q_{i_n} x_{i_n}/S_{i_n}) P(x_1, \ldots, x_n)$ is a formula, where $n \geq 0$ and $i_1, \ldots, i_n \in \{1, \ldots, n\}$.
(4) formulas are generated only by a finite number of applications of (1), (2) and (3).

Example 3. The formulas given below are formulas in the multilayer logic for relational databases.

$(\exists X/*N)(\forall x/X)(P(X, a/*M) \& Q(x))$;

$(\forall m/M)(\exists J/*N)(\forall j/J)(((\exists k/K)Q(b/I, f(J)) \&$
$P(m, j, k)) \rightarrow R(m, j))$,

where a/*M and b/I denote that the constants a and b are elements of the sets *M and I.

2.2 Semantics

Definition 9. A structure σ consists of:

(1) a non-empty set I of base sorts;
(2) domains S_j of sorts $j \in$ J(I);
(3) distinguished elements of domains S_j of sorts $j \in$ J(I);
(4) a set of n-place functions from a Cartesian product $X_{i=1}^n S_{j_i}$ to a domain S_f, where $j_i \in$ J(I) and $f \in$ J(I);
(5) a set of n-place predicates over a Cartesian product $X_{i=1}^n S_{j_i}$, where $j_i \in$ J(I).

Definition 10. An assignment φ from a formula G to the structure σ is a mapping that satisfies the following conditions:

(1) to each free variable in a formula G, we assign an element of a domain S_j of a sort $j \in$ J(I);
(2) to each domain symbol in a formula G, we as-

sign a domain S_j of a sort $j \in J(I)$;

(3) to each constant in a formula G, we assign some particular element of a domain S_j of a sort $j \in J(I)$;

(4) to each n-place function symbol f in a formula G, we assign a n-place function from a Cartesian product $X_{i=1}^{n} S_{j_i}$ to a doamin S_f, where $j_i \in J(I)$ and $f \in J(I)$ (if $n=0$, then we assign some particular element of S.);

(5) to each n-place predicate symbol P in a formula G, we assign a n-place predicate defined over a Cartesian product $X_{i=1}^{n} S_{j_i}$ (if $n=0$, then we assign either T or F.).

Definition 11. An assignment φ is similar mod $x_1, \ldots, x_n$ to φ' if the assignments is same except for the elements assigned to $x_1, \ldots, x_n$.

Definition 12. An interpretation of a formula G is an ordered pair (σ, φ), where σ is a structure for a given formula G and φ is an assignment from G to σ.

Definition 13. The valuation of a formula G over the interpretation (σ, φ), i.e. val(G, σ, φ), is recursively defined as follows.

(1) If G is a propositional letter, then val(G, σ, φ)=T iff φ(G)=T.

(2) If G is a n-place atom $P(t_1, \ldots, t_n)$, then val(G, σ, φ)=T iff $(\varphi(t_1), \ldots, \varphi(t_n)) \in \varphi(P(t_1, \ldots, t_n))$.

(3) Let A and B be formulas. If G is of form ˜A, (A&B), (A∨B), (A→B) and (A≡B), then val(G, σ, φ)=T iff the truth tables (same as classical logic) for φ(A) and φ(B) yield T.

(4) If G is of form $(\exists x/S)H(x)$, then val(G, σ, φ)=T iff there exists an assignment φ' which is similar mod x to φ and val(H, σ, φ')=T. That is, $(\exists x/S)H(x)$=T iff there is at least one element $e \in S$ such that H(e)=T.

(5) If G is of form $(\forall x/S)H(x)$, then val(G, σ, φ)=T iff for all assignments φ' which are similar mod x to φ and val(H, σ, φ')=T. That is, $(\forall x/S)H(x)$=T iff for all elements $e \in S$, H(e)=T.

(6) If val(G, σ, φ)≠T, then val(G, σ, φ)=F.

Definition 14. (Definition of predicate constants) EQ(X, Y): This predicate is defined over two sets, say X and Y, of level m ($m \geq 0$). It is assumed that base elements of both sets are ordered.
For $m=0$, EQ(X, Y)=T iff $X=Y$.
For $m>0$, EQ(X, Y)=T iff for pairs of $\langle x, y \rangle$, which are level $m-1$ sets of $\langle X, Y \rangle$, EQ(x, y)=T.
In the same manner NE(X, Y), GT(X, Y), etc. are defined.

Definition 15. LET(X, C): LET(X, C) is semantically equivalent to $(\exists X/{}^*D)$ EQ(X, C), where D is a set of level m ($m \geq 0$) and its sort includes the sort of C.

Example 4. The interpretation of the formula $(\exists X/{}^*N)(\forall x/X)$ (P(X, a/*M) & Q(x)) over the σ and φ below is as follows.

σ=({m, n},
$\{\{-1, 1, 2, 3\}_m, \{-2, -1, 0, 1, 2, 3\}_n\}$,
$\{\{-1, 1, 3\}_m\}$,
ϕ,
$\{\lambda s \lambda t.s \subseteq t, \lambda s.s>0\}$).

φ=(ϕ,
$\{M \to \{-1, 1, 2, 3\}_m, N \to \{-2, -1, 0, 1, 2, 3\}_n\}$,
$\{a \to \{-1, 1, 3\}_m\}$,
ϕ,
$\{P \to \lambda s \lambda t.s \subseteq t, Q \to \lambda s.s>0\}$).

val($(\exists X/{}^*N)(\forall x/X)$(P($X$, a/*M) & Q($x$)), σ, φ)

= val($(\exists X/{}^*\{-2, -1, 0, 1, 2, 3\}_n(\forall x/X)(\lambda X.X \subseteq \{-1, 1, 3\}_m$ & $\lambda x.x>0)$, σ, φ)

= val($\forall x/\{-2\}_n(\{-2\}_n \subseteq \{-1, 1, 3\}_m$ & x>0)
$\vee \forall x/\{-1\}_n(\{-1\}_n \subseteq \{-1, 1, 3\}_m$ & x>0)
$\vee \forall x/\{\ 0\}_n(\{\ 0\}_n \subseteq \{-1, 1, 3\}_m$ & x>0)
$\vee \forall x/\{\ 1\}_n(\{\ 1\}_n \subseteq \{-1, 1, 3\}_m$ & x>0)
⋮
$\vee \forall x/\{1, 3\}_n(\{1, 3\}_n \subseteq \{-1, 1, 3\}_m$ & $x>0$)
⋮
$\vee \forall x/\{-2, -1, 0, 1, 2, 3\}_n(\{-2, -1, 0, 1, 2, 3\}_n \subseteq \{-1, 1, 3\}_m$ & $x>0$), σ, φ)

=T.

Because the formulas $\forall x/\{1\}_n(\{1\}_n \subseteq \{-1, 1, 3\}_m$ & $x>0)$, $\forall x/\{1, 3\}_n(\{1, 3\}_n \subseteq \{-1, 1, 3\}_m$ & $x>0)$, etc. are true.

The multi-layer logic for relational databases is a kind of higher order logic. But only one-place prediactes are allowed to be variables. The relationship between this logic and the conventional higher order logic can be best illustrated by Fig. 1.

3. Describing Queries in the Multi-Layer Logic for Relational Databases

The application of the multi-layer logic to a relational database query language will be introduced by a series of example queries. The examples of this section are drawn from a database which describes lands. The database contains the following relations:

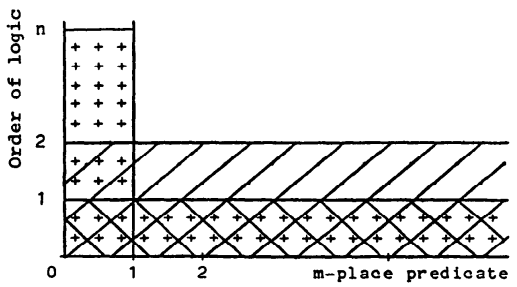

Fig. 1 The relationship between the multi-layer logic and the conventional logic. +++, \\\, /// indicate the multi-layer logic, 1-st order logic, 2-nd order logic, respectively.

LYP(LAND, YEAR, PRIC);
LU (LAND, USAG);
LDA(LAND, DIST, AREA).

The relation LYP has a row which gives the price for each land's identifier and the year acquired. The relation LU gives the usage of each land. The relation LDA gives, for each land, its area and the distance from the center of a city in which it is located. In the following formulation, a variable marked by the "^" symbol tells the system to output the value of it. Numerals following the "#" symbol indicate a number. Characters quoted by the symbol ' denote character constants.

A query against more than one relation with Boolean Conditions

Query 1. List the lands of usage a and their areas which are less than 35 kilometer from the center of a city and whose prices are less than 600,000 YEN/m^2 in the year 1981.

Q1. $(\exists l\hat{}/\text{LAND})(\exists a\hat{}/\text{AREA})(\exists p/\text{PRIC})(\exists d/\text{DIST})$
[LYP(l, #1981/YEAR, p) & LU(l, 'a'/USAG)
& LDA(l, d, a)
& LT(p, #60/PRIC) & LT(d, #35/DIST)].

The predicate LT(p, #60/PRIC) restricts values of the column PRIC in the relation LYP to less than 60, i.e., the price of lands are less than 600,000 YEN/m^2. Similarly, the predicate LT(d, #35/DIST) restricts values of the column DIST in the relation LDA to less than 35, i.e., the distances of lands from the center of a city are less than 35 kilometer. These predicates play the role of Codd's "restriction" operator.

A query against more than one relation with a universal quantification

Query 2. List the lands which are acquired in all the years, their usage and distance from the center of a city.

Q2. $(\exists l\hat{}/\text{LAND})(\exists u\hat{}/\text{USAG})(\exists d\hat{}/\text{DIST})(\forall y/\text{YEAR})$
$(\exists p/\text{PRIC})(\exists a/\text{AREA})$
[LYP(l, y, p) & LU(l, u) & LDA(l, d, a)].

Hereafter, we use an abbreviation in which variables that are not referred to in the query description are replaced by "@" symbol. For example, Q2 is abbreviated like Q2'.

Q2'. $(\exists l\hat{}/\text{LAND})(\exists u\hat{}/\text{USAG})(\exists d\hat{}/\text{DIST})(\forall y/\text{YEAR})$
[LYP(l, y, @) & LU(l, u) & LDA(l, d, @)].

A query with built-in arithmetic and aggregation functions

Query 3. List, for each land of usage a and acquired in the year 1981, its identifier, its price and the difference of the price with the average price computed on all lands acquired in the year 1981.

Q3. $(\exists Q/\text{*PRIC})(\exists l\hat{}/\text{LAND})(\exists p\hat{}/\text{PRIC})(\exists d\hat{}/\text{REAL})$
$(\forall q/Q)$
[LYP(l, #1981/YEAR, p) & LU(l, 'a'/USAG)
& LYP(@, #1981/YEAR, q)
& LET(d, sub(p, avg(Q)))].

Apart from the predicate constant LET(d, sub(p, avg(Q)), this query is decomposed into the following two formulas. One is $(\exists l\hat{}/\text{LAND})(\exists p\hat{}/\text{PRIC})$ [LYP(l, #1981/YEAR, p) & LU(l, 'a'/USAG)]. The result of this formula is a set of values of pairs $\langle l, p\rangle$ which satisfy the stipulation, that is, a set of lands of usage a and acquired in the year 1981. The other is $(\exists Q/\text{*PRIC})(\forall q/Q)$ [LYP(@, #1981/YEAR, q)]. This formula defines a set Q of prices to which corresponds a set of lands acquired in the year 1981. The result of Q3 yields the lands of usage a and corresponding prices together with the difference of the corresponding price with the average price computed on Q.

A query with a nested aggregation function and a Boolean condition

Query 4. What is the average number of lands per usage, which are acquired in the year 1981 and whose areas are not less than 100 square meter.

Q4. $(\exists LL/\text{**LAND})(\exists g\hat{}/\text{REAL})(\forall u/\text{USAG})(\exists L/LL)$
$(\forall l/L)(\exists a/\text{AREA})$
[LU(l, u) & LYP(l, #1981/YEAR, @)
& LDA(l, @, a) & GE(a, #100/AREA)
& LET(g, avg(ucount(LL)))].

In the query 4, it is required to calculate "an average number of lands." For this purpose, we have to introduce a variable LL whose value is a powerset of lands. This fact is explicitly represented by a prefix $(\exists LL/\text{**LAND})$. By means of a variable LL, the underliying specification is easily expressed by a nested aggregation function, i.e., avg(ucount(LL)). For example, if a value of a variable LL is a powerset $\{\{A1, A2, A3\}_a, \{A3, B2\}_b, \{D2\}_c\}$, where $\{\ \}_x$ indicates a set corresponding to a value x. Then, ucount(LL) yields a set $\{3, 2, 1\}$. Consequently, the function avg(ucount(LL)) yields 2. For more examples, [8, 9, 11] can be referred.

4. Virtual Relation and Query Transformation Algorithm

In the multi-layer logic for relational databases, virtual relations are defined through formulas which have the following form:

$$(\forall x_1/X_1)\ldots(\forall x_p/X_p)[(\exists x_{p+1}/X_{p+1})\ldots(\exists x_n/X_n)$$
$$(R_1 \# \ldots \# R_m) \rightarrow T(x_{i_1}, \ldots, x_{i_q})],$$

where # is either & or $\vee$, each of $R_1, \ldots, R_m$ is a formula containing predicate constants, base and virtual relations connected by either & or $\vee$. $T(x_{i_1}, \ldots, x_{i_q})$, where $i_1, \ldots, i_q \in \{1, \ldots p\{$, is an atomic formula containing variables $x_{i_1}, \ldots, x_{i_q}$ and denotes a virtual relation to be defined.

A query including virtual relations could not be reduced directly into the retrieval procedures or a sequence of operations in (an extended) relational algebra. To evaluate such a query, it is necessary to transform a query into one that contains no virtual relations.

The method for transforming a query is based on the idea of replacing virtual relations (if any) in the query by those defining formulas. The substitutions

are done repeatedly until a given query contains no virtual relation. Now, an overview of the query transformation algorithm is given. Generally, a query containing virtual relations is given as:

(1) $\ldots(Q_1x_1/X_1)\ldots(Q_px_p/X_p)\ldots[\ldots \mathrm{T}(x_{i_1},\ldots, x_{i_q})\ldots]$,

where $\mathrm{T}(x_{i_1},\ldots,x_{i_q})$ is a q-place virtual relation, $i_1,\ldots,i_q \in \{1,\ldots,p\}$, Q's are quantifiers of either $\forall$ or $\exists$, and ... indicates that the query may contain other variables, constants and/or functions.

The query transformation algorithm consists of the following steps.

Step 1. If the query (1) contains predicate constants which can be evaluated at this point, then evaluate them. Otherwise, simply go to next step.

Step 2. Find a virtual relation in a query. If there is, go to next step, otherwise go to "reduction procedure" which is discussed in Section 5.

Step 3. Let a formula defining a virtual relation $\mathrm{T}(y_{i_1},\ldots,y_{i_q})$ be given as:

(2) $\ldots(\forall y_1/Y_1)\ldots(\forall y_p/Y_p)[(\exists y_{p+1}/Y_{p+1})\ldots(\exists y_n/Y_n)[(\mathrm{R}_1\#\ldots\#\mathrm{R}_m)\rightarrow \mathrm{T}(y_{i_1},\ldots,y_{i_q})]$.

Rename the variable names so that the query (1) and the formula (2) share no variables in common.

Step 4. If a formula $(\forall y_1/Y_1)\ldots(\forall y_p/Y_p)\,\mathrm{T}(y_{i_1},\ldots,y_{i_q})\rightarrow(Q_1x_1/X_1)\ldots(Q_px_p/X_p)\,\mathrm{T}(x_{i_1},\ldots,x_{i_q})$ is a valid formula, then substitute $\mathrm{T}(x_{i_1},\ldots,x_{i_q})$ by $(\forall y_1/Y_1)\ldots(\forall y_p/Y_p)[(\exists y_{p+1}/Y_{p+1})\ldots(\exists y_n/Y_n][\mathrm{R}_1\#\ldots\#\mathrm{R}_m]$. As the result, the query (1) is transformed to the formula (3). The validity of the formula

$(Q_1'y_1/Y_1)\ldots(Q_p'y_p/Y_p)\mathrm{T}(y_{i_1},\ldots,y_{i_q})\rightarrow(Q_1x_1/X_1)\ldots(Q_px_p/X_p)\mathrm{T}(x_{i_1},\ldots,x_{i_q})$

is testable by the "implication conditions" shown in Table 1.

(3) $\ldots(Q_1\tilde{z_1}/Z_1)\ldots(Q_p\tilde{z_p}/Z_p)\ldots(\exists y_{p+1}/Y_{p+1})\ldots(\exists y_n/Y_n)[\ldots \tilde{\mathrm{R}_1}\#\ldots\#\tilde{\mathrm{R}_m}\ldots]$,

where $Q_j^{\sim}$, z_j, Z_j $(1\le j\le p)$, $\tilde{\mathrm{R}_1},\ldots,\tilde{\mathrm{R}_m}$, are respectively obtained from Q_j, x_j, X_j, y_j, Y_j, $\mathrm{R}_1,\ldots,\mathrm{R}_m$ by substitution rules shown in Table 1.

If a formula $(Q_1'y_1/Y_1)\ldots(Q_p'y_p/Y_p)\mathrm{T}(y_{i_1},\ldots,y_{i_q})\rightarrow(Q_1x_1/X_1)\ldots(Q_px_p/X_p)\mathrm{T}(x_{i_1},\ldots,x_{i_q})$

is not a valid formula, then try for other formulas defining $\mathrm{T}(y_{i_1},\ldots,y_{i_q})$.

Table 1. Implication conditions and substitution rules in the multi-layer logic. $\mathrm{D_c}$ denotes the domain of 'constant'.

(Q_ky_k/Y_k)	(Q_kx_k/X_k)	$(Q_k^{\sim}z_k/Z_k)$	Condition
$(\forall y_k/Y_k)$	$(\forall x_k/X_k)$	$(\forall z_k/X_k)$	$Y_k\supseteq X_k$
$(\forall y_k/Y_k)$	$(\exists x_k/X_k)$	$(\exists z_k/[X_k\wedge Y_k])$	$Y_k\wedge X_k\neq\phi$
$(\exists y_k/Y_k)$	$(\exists x_k/Y_k)$	$(\exists z_k/Y_k)$	$Y_k\subseteq X_k$
constant	$(\forall x_k/X_k)$	ϕ	$\mathrm{D_c}\subseteq X_k$
constant	$(\exists x_k/X_k)$	ϕ	$\mathrm{D_c}\subseteq X_k$
$(\forall y_k/Y_k)$	constant	ϕ	$\mathrm{D_c}\subseteq Y_k$
$(\exists y_k/Y_k)$	constant	ϕ	$\mathrm{D_c}\subseteq Y_k$

If a original query is true over an interpretation (σ,φ) then a transformed query is also true over (σ,φ). For details [11, 12] can be referred.

An example is given here. Suppose a virtual relation Largeland 81 (l, p, u) is defined which gives price and usage for each land acquired in the year 1981 and whose area is not less than 100 square meter. This relation is defined by the following formula:

$(\forall l/\mathrm{LAND})(\forall p/\mathrm{PRIC})(\forall u/\mathrm{USAG})[(\exists a/\mathrm{AREA})(\exists d/\mathrm{DIST})$
$[\mathrm{LU}(l,u)\&\mathrm{LYP}(l,\#1981/\mathrm{YEAR},p)\&\mathrm{LDA}(l,d,a)$
$\&\sim\mathrm{LT}(a,\#100/\mathrm{AREA})]\rightarrow\text{Large-land81}(l,p,u)]$.

Using this relation, we can describe the query 4 in Section 3 by the formula Q4', which is semantically equivalent to the formula Q4.

Q4'. $(\exists LL/{**}\mathrm{LAND})(\exists g\,\hat{}/\mathrm{REAL})(\forall u/\mathrm{USAG})$
$(\exists L/LL)(\forall l/L)(\exists p/\mathrm{PRIC})$
$[\text{Large-land81}(l,p,u)\&\mathrm{LET}(g,\mathrm{avg}(\mathrm{ucount}(LL)))]$.

The virtual relation Large-land81 has to be developed according to its definition.

Step 1. Although the query Q4' contains the predicate constant LET $(g, \mathrm{avg}(\mathrm{ucount}(LL))$, it can not be evaluated because the value of the variable LL is not determined at this point. So simply go to Step 2.

Step 2. Since the query Q4' contains the virtual relation Large-land81, it needs to be transformed. Go to Step 3.

Step 3. Rename variables l, p, u, a, d in the defining formula to l', p', u', a', d', and obtain

$(\forall l'/\mathrm{LAND})(\forall p'/\mathrm{PRIC})(\forall u'/\mathrm{USAG})[(\exists a'/\mathrm{AREA})$
$(\exists d'/\mathrm{DIST})[\mathrm{LU}(l',u')\&\mathrm{LYP}(l',\#1981/\mathrm{YEAR},p')$
$\&\mathrm{LDA}(l',d',a')\&\sim\mathrm{LT}(a',\#100/\mathrm{AREA})]$
$\rightarrow\text{Large-land81}(l',p',u')]$.

Step 4. The formula

$(\forall l'/\mathrm{LAND})(\forall p'/\mathrm{PRIC})(\forall u'/\mathrm{USAG})$
$\text{Large-land81}(l',p',u')\rightarrow(\forall u/\mathrm{USAG})(\forall l/L)(\exists p/\mathrm{PRIC})$
$\text{Large-land81}(l,p,u)$

is valid because each corresponding pair of variables satisfy the implication conditions. That is, $\mathrm{Dom}(l')\supseteq\mathrm{Dom}(l)$, $\mathrm{Dom}(p')\wedge\mathrm{Dom}(p)=\{\mathrm{PRIC}\}\neq 0$, $\mathrm{Dom}(u')\supseteq\mathrm{Dom}(u)$. Replacing Large-land81$(l, p, u)$ in the query by

$\mathrm{LU}(l',u')\&\mathrm{LYP}(l',\#1981/\mathrm{YEAR},p')$
$\&\mathrm{LDA}(l',d',a')\&\sim\mathrm{LT}(a',\#100/\mathrm{AREA})$,

we get the transformed query as follows:

TQ4. $(\exists LL/{**}\mathrm{LAND})(\exists g\,\hat{}/\mathrm{REAL})(\forall u/\mathrm{USAG})$
$(\exists L/LL)(\forall l/L)(\exists a/\mathrm{AREA})(\exists p/\mathrm{PRIC})(\exists d/\mathrm{DIST})$
$[\mathrm{LU}(l,u)\&\mathrm{LYP}(l,\#1981/\mathrm{YEAR},p)\&\mathrm{LDA}(l,d,a)$
$\&\sim\mathrm{LT}(a,\#100/\mathrm{AREA})\&\mathrm{LET}(g,\mathrm{avg}(\mathrm{ucount}(LL)))]$.

The final transformed query does not contain any virtual relations, thus it can be reduced into retrieval procedures. Section 5 concerns the reduction algorithm from a formula into a sequence of operations in (an extended) relational algebra.

5. Implementation of a Deductive Database System SBDS-F3

5.1 Overview of SBDS-F3

This section provides an overview of the database system with deductive mechanism SBDS-F3. It is based on the multi-layer logic for relational databases, and developed by modifying SBDS-F2 [7] that is based on the many-sorted logic. SBDS-F3 is implemented on top of the MRDOS operating system for Data General Corporation ECLIPSE S/130 computer system and programmed in FORTRAN IV (partially in assembler for efficiencies).

Our system is based on evaluational approach [2] in which a query is processed in two steps. First, a given query is transformed into one which involves no virtual relations. Implementation of this query transformation is discussed in Section 5.2. Second, a transformed query is reduced into a sequence of database access procedures. Since a formula in the multi-layer logic may contain variables whose values are sets, the Codd's relational algebra has to be extended. The reduction algorithm and an extended relational algebra are discussed in Section 5.3.

Fig. 2 illustrates the system structure in some details. SBDS-F3 can accept commands from two sources: console key board and data file specified. The commands fall into five categories. For each category, one to four commands are provided. They are summarized as follows.

(1) SR-, SD-, SI-, SS-command: commands for defining four kinds of domains, i.e. root domains, domains defined by disjunct, intersect and powerset relation. For details, [4, 5, 11] can be referred.

SR ⟨separator⟩⟨defined node name⟩
⟨separator⟩⟨data type⟩.

SD ⟨separator⟩⟨defined node name⟩
⟨separator⟩⟨parent node name⟩
⟨separator⟩⟨division code⟩.

SI ⟨separator⟩⟨defined node name⟩
⟨separator⟩⟨parent node name 1⟩
⟨separator⟩⟨parent node name 2⟩.

SS ⟨separator⟩⟨defined node name⟩
⟨separator⟩⟨parent node name⟩.

(2) KD-command: a command for defining a virtual relation.

KD ⟨separator⟩⟨formula for virtual relation⟩.

(3) QU, QV, QW-command: commands for defining a query.

QU ⟨separator⟩⟨formula for query⟩.
QV ⟨separator⟩⟨formula for query⟩.
QW ⟨separator⟩⟨formula for query⟩.

For a query defined by QU-command, only answers for the given query are printed out. For a query defined by QV-command, retrieval operations reduced and answers for the given query are listed. If a query is defined by QW-command, three types of information are given, i.e. retrieval operations, contents of a intermediate relation for each of the operations and answers for the given query [7, 11].

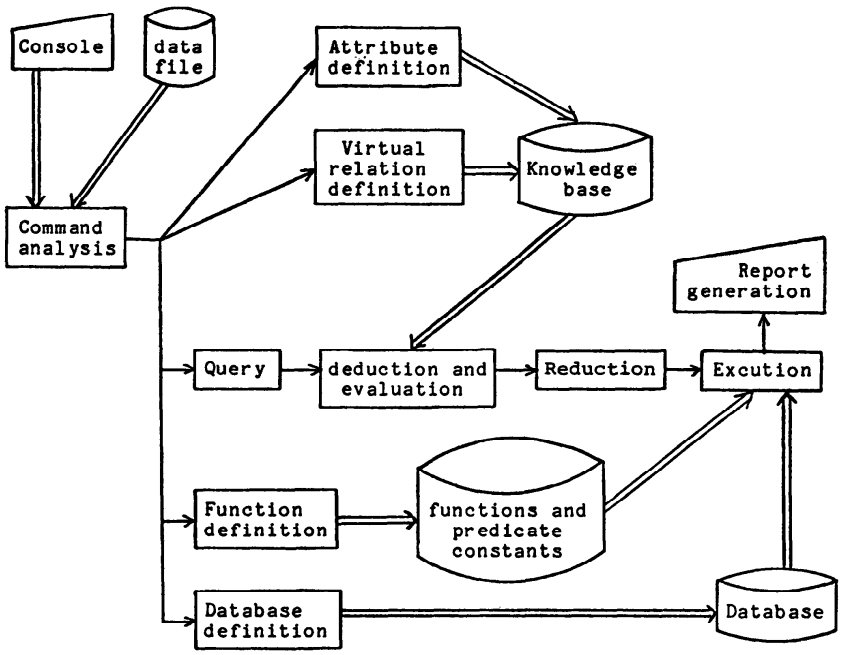

Fig. 2 SBDS-F3 system structure. → indicates a flow of processes and ⇒ indicates a flow of data, definitions of a virtual relation.

(4) FD-command: a command for defining a user definable function.

FD ⟨separator⟩⟨argument identifier list⟩
⟨separator⟩
⟨defined function⟩/⟨domain specification⟩

(5) RD-command: a command for defining a base relation.

RD ⟨separator⟩(⟨relation name⟩
⟨attribute list⟩)

5.2 Implementation of query transformation algorithm of SBDS-F3

The general flow-chart of query transformation algorithm for SBDS-F3 is shown in Fig. 3. This algorithm is based on the refutation procedure and depth first search strategy for time and storage efficiency. To find out all the data which satisfy a given query or negation of the query, a stack to retain information for backtracking (called B-stack hereafter) is provided.

First the given query is pushed into the B-stack, and negate the query. Next, atoms evaluable at this point are evaluated. Atoms corresponding to virtual relations are selected. If there are, formulas defining them are seeked from the knowledge base (a set of formulas defining virtual relations). In case, the seeking is not successful, "Aborted . . ." is printed out and processing

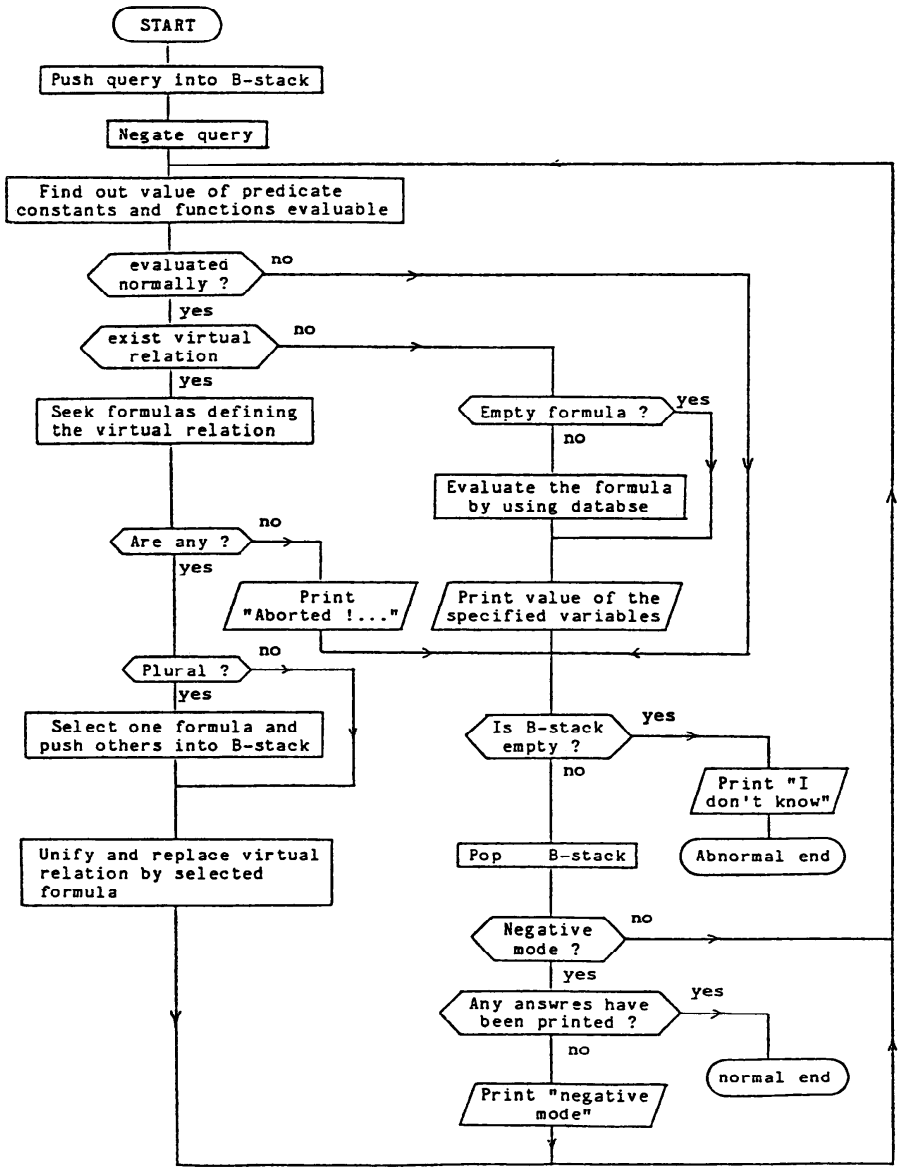

Fig. 3 The general flow-chart of the query transformation algorithm for SBDS-F3.

of the query is aborted, because there is no information corresponding to the selected virtual relation. If the seeked formulas are plural, only one of them is selected and others are pushed into the B-stack. The selected virtual relation is unified with and replaced by the seeked formula. Details of this process are discussed in Section 4 of this paper.

If there are no atoms corresponding to virtual relations, then it is tested whether the query is an empty formula or not. If the query is empty, then values of the specified variables are printed out. Otherwise it is evaluated by using a relational database, which is discussed in Section 5.3. By the process mentioned above, an answer for the given query is found.

The later process serves to find all the answers that satisfy the query. If the B-stack is not empty, pop the B-stack and test whether the B-stack is empty (negative mode) or not. If it is not empty and any answers have been printed out, then the given query is evaluated affirmatively i.e. normal end. Otherwise, "negative mode" is printed out and try to evaluate the query negatively.

5.3. Reducing a formula into relational database access procedures

The flowchart of the reduction algorithm is shown in Fig. 4. The algorithm which reduces a formula in the first order logic into a sequence of the operations in the relational algebra is discussed by E. F. Codd [1], R. Reiter [6] and C. L. Chang [2]. But their algorithm is not applicable to a formula in the multi-layer logic because it may contain variables whose values are sets. The reduction algorithm for the first order predicate logic is extended to manipulate the formula in the multi-layer logic by containing the "determine a set" operation explicitly.

Overview of the algorithm is as follows. First, an atom corresponding to a base relation is translated into the "load" operation, an operation to access a base relation. If the atom contain any constants and/or predi-

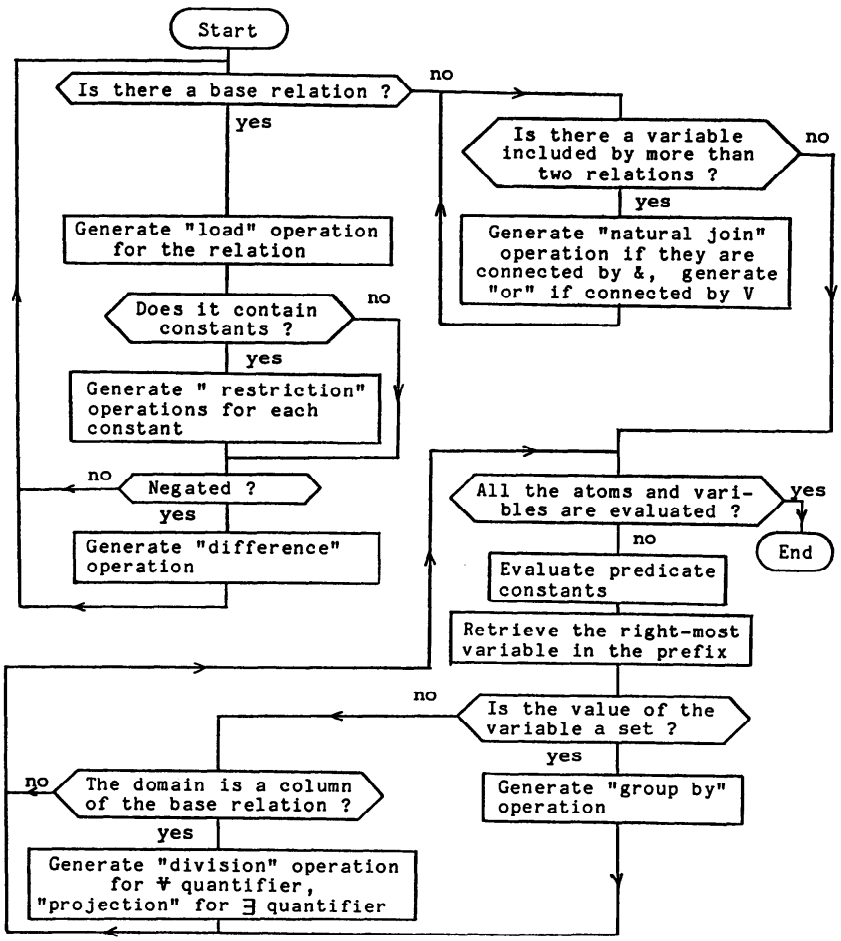

Fig. 4 The flowchart of the reduction algorithm of SBDS-F3.

cate constants evaluable, they are translated into the "restriction" operation. This process is repeated until all the atoms corresponding to base relations are translated. Next, all the variables included by more than two relations are translated into the "natural join" operation if the relations are connected by &, the "or" operation if the relations are connected by ∨ (Note that "intersection" operation is a special case of the "natural join" operation.).

The rest is translation of quantifiers, which consists of the following processes. First retrieve the right-most variable in the prefix. If its value is a set, then it is translated into the "determine a set" operation. Otherwise, if the domain of the variable is a column of a base relation, then the ∀ quantifier is translated into the "division" operation, the ∃ quantifier is into the "projection" operation. When a "LET" atom is evaluated, the "evaluate function" operation is generated. For more complete description, [8, 11] can be referred.

The notations of the operations generated from a formula are summarized below.

*LOAD ⟨relation name⟩[⟨work file name⟩ ⟨column identifier list⟩].

*OR [⟨work file name⟩] [⟨work file name⟩ ⟨column identifier list⟩], [⟨work file name⟩ ⟨column identifier list⟩].

*AND [⟨work file name⟩] [⟨work file name⟩ ⟨column identifier list⟩], [⟨work file name⟩ ⟨column identifier list⟩].

*DIFF [⟨work file name⟩] [⟨work file name⟩ ⟨column identifier list⟩], [⟨work file name⟩ ⟨column identifier list⟩].

*PROJ [⟨work file name⟩ ⟨column identifier list⟩] $⟨column identifier⟩.

*JOIN [⟨work file name⟩] [⟨work file name⟩ ⟨column identifier list⟩] ⟨⟨column identifier list⟩⟩ ⟨⟨column identifier list⟩⟩ [⟨work file name⟩ ⟨column identifier list⟩].

*REST ⟨work file name⟩: ⟨column identifier⟩ ⟨predicate constant⟩⟨constant⟩/ ⟨domain specification⟩.

*DIV [⟨work file name⟩⟨column identifier list⟩] /⟨column identifier⟩.

*DETM [⟨set name⟩]=⟨variable⟩ ={⟨work file name⟩: ⟨column identifier⟩ FOR EACH ⟨work file name⟩: ⟨column identifier list⟩}.

*DETM [⟨set name⟩]=⟨variable⟩ ={[⟨set name⟩]}.

*EVAL ⟨variable⟩⟨=⟨function name⟩.

*LOAD means to load the base relation indicated. *OR, *AND and *DIFF denote OR, AND and subtract operations, respectively. *PROJ, *JOIN, *REST and *DIV stand for projection, join, restriction and division operations, respectively. *DETM indicates the "determine a set" operation, which determines the value of a variable of level m ($m \geq 1$) and stores it in a set indicated. *EVAL indicates the "evaluate function" operation and means that the function indicated is evaluated and its value is retained in the variable. Strictly speaking, the value of the variable is equal to the value of the function as stated by Definition 15 of this paper. *EVAL also checks consistency of the sorts of the variables and functions, such as the function avg maps a set of integer or real into real.

Fig. 5 gives some results of SBDS-F3. In this figure, the 1st to 6th lines define the attribute names of relations. For example, the 1st line indicates that "LAND" is an attribute name and its sort is a string of alphabets and/or digits. The 2nd line denotes that "YEAR" is an attribute name and its sort is integer. The lines labeled 7], 8], 9] define the relations LYP, LU, LDA, respectively. The lines labeled 10] and 11] define virtual relations. For example, the line 10] defines the relation LAND81, which gives the lands acquired in the year 1981. The lines 12] to 15] stand for the query Q1, Q2, Q3 and Q4, respectively. For each query, the corresponding expressions in an extended relational algebra are given preceded by the symbol*.

6. Conclusion

In this paper, we define a formal language called the multi-layer logic for relational databases, an extended many-sorted logic, for uniform handling of complex queries. In the multi-layer logic, queries are expressed non-procedurally which contain operations whose arguments are grouped by some other variables and/or which involve nested aggregation functions. For some properties of the multi-layer logic, [11, 12] can be referred. In some cases, a query contains virtual relations. Query transformation algorithm for such a query is discussed. We also discuss implementation of a relational database system based on this logic and give some examples.

Since a formula in the multi-layer logic is too complicated for non-mathematically trained users to use, a user interface language GOING (a Graphics Oriented INteractive data lanGuage) is designed and implemented. For GOING, [9, 10, 11] can be referred.

Acknowledgements

I wish acknowledge all the members of Meetings of Information Systems for their encouragements and

```
      C   ###   Start of database definintion   ###
      C   ===   Definition of the attribute names   ===
 1]   SR        ;LAND,C
 2]   SR        ;YEAR,I
 3]   SR        ;PRIC,I
 4]   SR        ;USAG,C
 5]   SR        ;DIST,I
 6]   SR        ;AREA,I

      C   ===   The LYP relation has a row for each
      C   ===   land's identifier and year, giving its price.
 7]   RD        ; (LYP, LAND, YEAR, PRIC )
 1)     A1,        #1977,     #50
 2)     A1,        #1979,     #55
 3)     A1,        #1981,     #58
 4)     A2,        #1979,     #80
 5)     A2,        #1981,     #86
 6)     A3,        #1979,     #105
 7)     A3,        #1981,     #112
 8)     B1,        #1979,     #63
 9)     C1,        #1977,     #22
10)     C1,        #1979,     #25
11)     C2,        #1977,     #23
12)     C2,        #1979,     #26
13)     C2,        #1981,     #27
14)     D1,        #1981,     #40
15)     D2,        #1981,     #36
16)     D3,        #1979,     #18
17)     D3,        #1981,     #20
18)     %

      C   ===  The LU relation gives the usage of each land.
 8]   RD        ; ( LU, LAND, USAG )
 1)     A1,        a
 2)     A2,        a
 3)     A3,        a
 4)     A5,        b
 5)     B1,        b
 6)     B1,        c
 7)     C1,        c
 8)     C2,        a
 9)     C2,        c
10)     D1,        d
11)     D2,        b
12)     D2,        d
13)     D3,        d
14)     %

      C   ===  The LDA relation gives, for each land,
      C   ===   its area and the distance from the center
      C   ===   of a city in which it is located.
 9]   RD        ; ( LDA, LAND, DIST, AREA )
 1)     A1,        #40,       #300
 2)     A2,        #25,       #120
 3)     A3,        #7,        #110
 4)     B1,        #27,       #60
 5)     C1,        #10,       #50
 6)     C2,        #34,       #90
 7)     D1,        #20,       #55
 8)     D2,        #30,       #140
 9)     D3,        #28,       #80
10)     %

      C   ###   Definition of virtual relations   ###

      C   ===   The LAND81 lands are lands inquired
      C   ===   in the year 1981.
10]   KD        ; ( A l/LAND, A p/PRIC, A u/USAG,'
                    A a/AREA, A d/DIST ) '
            [ -( (LYP, l, #1981/YEAR, p) & (LU, l,u) '
                 (LDA, l, d, a) )'
              V (LAND81, l, p, u, a, d) ]

      C   ===   The Large land81 (LA_LA81) lands are
      C   ===   lands inquired in the year 1981 and whose
      C   ===   area is not less than 100 square meter.
11]   KD        ; ( A l/LAND, A p/PRIC, A u/USAG,'
                    A a/AREA, A d/DIST ) '
            [ -( (LU, l, u) & (LYP, l, #1981/YEAR, p) '
              & (LDA, l, d, a) & -(LT, a, #100/AREA) )'
             V (LA_LA81, l, p, u) ]

      C   ###   End of database definition   ###


      C   ===   Query 1 in Section 3   ===
12]   QU        ; ( E la^/LAND, E ar^/AREA,'
                    E pr/PRIC, E di/DIST ) '
            [ (LAND81, la, pr, a/USAG, ar, di) '
              &(LT, pr, #60/PRIC) & (LT, di, #35/DIST) ] ?
#LOAD  LYP  [ W 1:  1,  2,  3 ]
#REST  W 1:  2 = #1981/YEAR
#REST  W 1:  3 .LT. #60/PRIC
#LOAD  LU  [ W 2,  1,  2 ]
#REST  W 2:  2 = a/USAG
#LOAD  LDA  [ W 3,  1,  2,  3 ]
#REST  W 3:  2 .LT. #35/DIST
#JOIN  [ W 1 ]  [ W 1,  1,  2,  3 ]
       < 1 >< 1 > [ W 2,  1,  2 ]
#JOIN  [ W 1 ]  [ W 1,  1,  2,  3,  4 ]
       < 1 >< 1 > [ W 3,  1,  2,  3 ]
#PROJ  [ W 1,  1,  2,  3,  4,  5,  6 ] $  5
#PROJ  [ W 1,  1,  2,  3,  4,  6 ] $  3
#PROJ  [ W 1,  1,  2,  4,V  6 ] $  6
#PROJ  [ W 1,V  1,  2,  4,V  6 ] $  1

== ANSWER == ( la  ar )
 C2          90

      C   ===   Query 2 in Section 3   ===
13]   QU        ; ( E l^/LAND, E u^/USAG, E d^/DIST'
                    A y/YEAR )'
            [ (LYP, l,y,e ) & (LU, l,u )'
              & (LDA, l,d,e ) ] ?
#LOAD  LYP  [ W 1,  1,  2 ]
#LOAD  LU  [ W 2,  1,  2 ]
#LOAD  LDA  [ W 3,  1,  2 ]
#JOIN  [ W 1 ]  [ W 1,  1,  2 ]
       < 1 >< 1 > [ W 2,  1,  2 ]
#JOIN  [ W 1 ]  [ W 1,  1,  2,  3 ]
       < 1 >< 1 > [ W 3,  1,  2 ]
#DIV   [ W 1,  1,  2,  3,  4 ] / 2
#PROJ  [ W 1,  1,  3,V  4 ] $  4
#PROJ  [ W 1,  1,V  3,V  4 ] $  3
#PROJ  [ W 1,V  1,V  3,V  4 ] $  1

== ANSWER == ( l  u  d )
 A1          a          40
 C2          a          34
 C2          c          34

      C   === Query 3 in Section 3   ===
14]   QU        ; ( E Q/#PRIC, E l^/LAND, E p^/PRIC, '
                    E d^/REAL, A q/Q ) '
            [ (LYP, l, #1981/YEAR, p ) & (LU, l, a/USAG) '
              & (LYP, e, #1981/YEAR, q) '
              & (LET, d, sub(p,avg(Q)) ) ] ?
#LOAD  LYP  [ W 1,  1,  2,  3 ]
#REST  W 1:  2 = #1981/YEAR
#LOAD  LU  [ W 2,  1,  2 ]
#REST  W 2:  2 = a/USAG
#LOAD  LYP  [ W 3,  1,  2 ]
#REST  W 3:  1 = #1981/YEAR
#JOIN  [ W 1 ]  [ W 1,  1,  2,  3 ]
       < 1 >< 1 > [ W 2,  1,  2 ]
#PROJ  [ W 1,  1,  2,V  3,  4 ] $  3
#PROJ  [ W 1,V  1,  2,V  3,  4 ] $  1
#DETM  [ S 1 ] = Q = ( W 3:  2 )
#EVAL  d <= sub
#DIV   [ W 3,  1,  2 ] / 2

== ANSWER == ( l  p  d )
 A1          58   0.371E  1
 A2          86   0.317E  2
 A3          112  0.577E  2
 C2          27  -0.272E  2

      C   ===   Query 4' in Section 3   ===
15]   QU        ; ( E LL/#LAND, E e^/REAL,'
                    A u/USAG, E L/LL, A l/L ) '
            [ (LA_LA81, l, e, u) '
              & ( LET, s, avg(ucount(LL)) ) ] ?
#LOAD  LU  [ W 1,  1,  2 ]
#LOAD  LYP  [ W 2,  1,  2,  3 ]
#REST  W 2:  2 = #1981/YEAR
#LOAD  LDA  [ W 3,  1,  2,  3 ]
#REST  W 3:  3 .LT. #100/AREA
#JOIN  [ W 1 ]  [ W 1,  1,  2 ]
       < 1 >< 1 > [ W 2,  1,  2,  3 ]
#JOIN  [ W 1 ]  [ W 1,  1,  2,  3,  4 ]
       < 1 >< 1 > [ W 3,  1,  2,  3 ]
#PROJ  [ W 1,  1,  2,  3,  4,  5,  6 ] $  5
#PROJ  [ W 1,  1,  2,  3,  4,  6 ] $  4
#PROJ  [ W 1,  1,  2,  3,  6 ] $  6
#DETM  [ S 1 ] = L = ( W 1:  1  FOR EACH  W 1:  2 )
#DIV   [ W 1,  1,  2,  3 ] / 1
#DIV   [ W 1,  2,  3 ] / 2
#DETM  [ S 2 ] = LL = ( [ S 1 ] )
#EVAL  s <= avg

== ANSWER == ( s )
   0.200E  1
```

Fig. 5 Some results of SBDS-F3.

constructive comments.

I am grateful to Dr. N. Yonezaki, Enomoto Laboratory of Tokyo Institute of Technology, for many helpful discussions and providing me valuable decuments.

I would like to express my appreciation to Dr. K. Agusa, Ohno Laboratory of Kyoto University, for providing the SAFE editor system. The SAFE system is very useful and exclusively used for preparing this paper.

References

1. Codd, E.F. Relational completeness of data base sublanguages, *Data Base Systems, Courant Computer Science Symposia Series*, 6, (R. Rustin, Ed.), Prentice-Hall, (1971), 65–98.
2. Chang, C. L. DEDUCE 2, Futher Investigation on deduction in relational data base, *Logic and Data Bases* (H. Gallaire etc., ed.), Plenum Press, N.Y., (1978), 201–236.
3. Kunifuji, S. Second-order many-sorted Boolean logic for knowledge representation language, *IIAS of Fujitsu research report*, 14 (Feb. 1981).
4. Ohsuga, S. and Yamauchi, H. A technique for retrieving information using inference making, (in Japanese), *Joho Shori*, **18**, 8, (1977), 789–798.
5. Ohsuga, S. Perspectives on new computer systems of the next generation—a proposal for knowledge-based systems. *Journal of Information Processing*, 3, 3 (1980).
6. Reiter, R. On closed world data bases, *Logic and Data Bases* (H. Gallaire and J. Minker, Eds.), Plenum Press, New York, (1978), 55–76.
7. Udagawa, Y. and Ohsuga, S. The first order predicate logic as a goal-oriented programming language, *WGSE Meeting of IPSJ*, (in Japanese) (1980), 13–2.
8. Udagawa, Y. and Ohsuga, S. The multi-layer logic as a relational database query language and its reduction algorithm to relational procedures, (in Japanese) *Joho-Shori*, **23**, (1982), 634–642,
9. Udagawa, Y. and Ohsuga, S. Design and implementation of a database system based on the multi-layer logic, *Proc. of Advanced Database Symposium* of IPSJ, (Dec. 1981), 31–42.
10. Udagawa, Y. and Ohsuga, S. GOING—A Data Sublanguage Using a Graphics Display, *WGDB Meeting of IPSJ* (Mar. 1982), 29–3.
11. Udagawa, Y. A Study on Design and Implementation of a Database System Based on Predicate Logic, *Doctorial Thesis*, Tokyo University (Feb. 1982).
12. Udagawa, Y. and Ohsuga, S. Construction of SBDS-F3: a relational database with inference mechanism, *RIMS, Univ. of Kyoto, Kokyu-Roku*, **461**, (1982), 49–78.
13. Loveland, D. W. Automated theorem proving: a logical basis, *Fundamental Studies in Computer Science* 6, North-Holland (1978).

(Received Mar. 29, 1982)