A STRAIGHTFORWARD FORMALIZATION OF THE RELATIONAL MODEL

Timo Niemi and Kalervo Järvelin

University of Tampere
Department of Mathematical Sciences
Computer Science
P.O. Box 607
SF-33101 Tampere 10, Finland

ABSTRACT

There has been a lot of recent interest in the formalization of the relational data model (RDM). Many approaches may be characterized as ones oriented mainly towards declaring the components of the RDM and their interrelationships. Other approaches provide a tool for manipulating the components of RDM so that research topics on the model can be specified exactly. The latter approaches are based on formal specification methods such as denotational semantics or abstract data types. Some in the data base community experience them quite complex and cumbersome.

The goal of the approach of this paper is of the latter kind. However, special attention is being paid to avoid the complexity of the formal specification methods because our notations and definitions are based on set theory. We attempt to provide an exact, convenient and general tool for specifications and proofs concerning various topics like relational query languages, query optimization, relational data base restructuring, data base design, etc.

1. INTRODUCTION

Many different formulations of the relational data model [8] have been presented. They vary with respect to terminology, notation and the degree of precision and formality. The application environment determines the degree of precision and formality. Informal [27] and semi-formal definition are appropriate for end user quidance whereas formal and precise definitions facilitate exact specification of relational data base systems and proofs of theorems concerning RDM. This paper has been mainly intended for research work with RDM, and therefore the definition is formal and precise.

Specification methods with strong mathematical orientation have been applied (e.g. [7,23,13,31]) to define RDM or some of its parts exactly and formally. So far, the most formal specification methods have not been widely used in the data base area. Some in the data base community experience them complex and difficult because they often require mastering unfamiliar notations. In this paper we aim at giving mathematically exact definitions for the essential concepts of the relational data model without the complexity of the existing formal specification methods. The definitions are based on set theory so familiarity with the usual set theoretic notation is the only required background knowledge.

Delobel [12] points out well-foundedly that the word 'relation' is used sometimes confusingly in the data base literature to mean both the structural description of tuples (schema level) and the set of tuples (instance level). Due to this, the term relation_value, denoting the set of tuples, has been used in recent publications [15,26]. Very often in the context of RDM we must consider components belonging to schema and instance levels together. Therefore, in this paper we define conciously the components of RDM such as tuple, relation, relational data base so that they contain both the structural descriptions (schemata) and the instances. Likewise, the operations of the relational algebra are defined in such a way that the result relations of the operations contain their schemata, too. The fixed togetherness of the schema and instance level is appropriate in order to avoid separate instance/schema consideration.

The purpose of this paper is not to introduce all possible which may be included in the relational data model. Here, we are satisfied to introduce the framework which defines the essential concepts of RDM. We have tried to create such an approach that provides a flexible and mathematically sound tool for research on different features of RDM.

2. BASIC NOTATIONS

In addition to the usual set theoretic notation we need finite n-tuples in our formalism.

Notational_convention_1: The finite n-tuples are denoted between angle brackets, for example <a,b,c>.

Notational_convention_2: Two n-tuples r_1 and r_2 are concatenated by $\langle r_1,r_2\rangle$, for example if $r_1=\langle a,b\rangle$ and $r_2=\langle c,d,e\rangle$ then $\langle r_1,r_2\rangle=\langle a,b,c,d,e\rangle$.

Notational_convention_3: The ith component in the n-tuple r is denoted by r[i], for example if r=<a,b,c,d,e> then r[3]=c.

In the formalism we shall manipulate indices.

Definition_1: Indices are finite n-tuples consisting of natural numbers.

When we later on shall give formal definitions of concepts associated with RDM we take some new notational conventions.

3. RELATIONAL DATA MODEL

3.1. Tuple component and tuple

A relation consists of tuples and tuples of tuple components. As mentioned above tuples and tuple components must contain their descriptions, too. Here, the complete description both of a tuple component and of a tuple consists of five components.

The formal representation of a tuple component and of a tuple: They are represented as (r,X,AN,I_x,f_x) where

- r is an instance of the tuple component or of the tuple,
- X is a type of the tuple component or of the tuple,
- AN is a set of attribute names of the tuple component or of the tuple,
- I_x is a set of indices and
- $f_x:AN \rightarrow I_x$ is a naming function.

Consider first (r,X,AN,I_x,f_x) in the case that it describes a tuple component. We shall later on define the operation which constructs tuples from tuple components represented in this way. The component X is always some domain. Domains can be either application independent domains or application dependent domains. Application independent domains are general sets (primitive types) such as the sets of integers (int), real numbers (real), character strings (char) etc.

In data base applications we have sometimes situations where we want to emphasize that only some values are permitted in our data base. Thus, we define application dependent domains which are specific for a particular data base. Application dependent domains can be defined either

by expressing a predicate for some primitive type, for example $E\text{-}AGE=\{x \in int \mid 16 \leq x \leq 64\}$ or

by listing explicitely the elements of the set, for example COLOUR={red,blue,green}.

The instance of the tuple component is always some value of X, that is $r \in X$. For tuple components $|AN|=|I_x|=1$ (we denote by $|Z|$ the number of elements of Z). The set AN consiste only of one attribute name and the set I_x contains the index <1> only. Furthermore, the naming function f_x has the form: $f_x(a)=<1>$ where the attribute name a is the element of the set AN. For example, $t_1=(1112,int,\{E\text{-}CODE\},\{<1>\},f(E\text{-}CODE)=<1>)$ would be the complete description of one tuple component.

A tuple is constructed from a finite number of tuple components. For this purpose we define the generalized concatenation operation of tuple components and of tuples. In the definition we need the operation $\oplus$ which, in turn, is defined between the index set I_x and the non-negative integer n.

Definition 2: $I_x \oplus n = \{<i+n> \mid <i> \in I_x\}$

The generalized concatenation operation of tuple components and of tuples will also be applied later on in the definition of the relational algebra.

Definition 3: Let $t_1=(r_1,X_1,AN_1,I_{x_1},f_{x_1})$ and $t_2=(r_2,X_2,AN_2,I_{x_2},f_{x_2})$ be two tuple components or tuples such that $AN_1 \cap AN_2 = \emptyset$. The concatenation operation $<t_1,t_2>$ constructs the tuple $(<r_1,r_2>,X_1 \times X_2, AN_1 \cup AN_2, I_{x_1 \times x_2}, f_{x_1 \times x_2})$ where

$$I_{x_1 \times x_2} = I_{x_1} \cup (I_{x_2} \oplus |I_{x_1}|) \quad \text{and}$$

$$f_{x_1 \times x_2} : AN_1 \cup AN_2 \rightarrow I_{x_1 \times x_2}$$

$$f_{x_1 \times x_2}(a) = \begin{cases} f_{x_1}(a), & \text{if } a \in AN_1 \\ f_{x_2}(a) \oplus |I_{x_1}|, & \text{if } a \in AN_2 \end{cases}$$

The result of the concatenation operation is always a tuple. For example, let t_2=(SMITH,char,{E-NAME},{<1>},f(E-NAME)=<1>) be another tuple component. Now $<t_1,t_2>$ is the tuple (<1112,SMITH>,int×char,{E-CODE,E-NAME},{<1>,<2>},{f(E-CODE)=<1>, f(E-NAME)=<2>}).

It is typical of a tuple that in the sets AN and I_x there can be more than one element. Any tuple can be constructed by repeating the concatenation operation defined above finitely.

Definition 4: Let $t_1,t_2,t_3,\ldots,t_n$ be any tuple components then the tuple constructor $\bowtie_{i=1}^{n} t_i$ means the following sequence of concatenations $<\ldots<<t_1,t_2>,t_3>,\ldots,t_n>$.

For example, if t_3 is the tuple component (5819.20,real, {SALARY},{<1>},f(SALARY)=<1>) and t_1,t_2 are tuple components described above and $t_0 = \bowtie_{i=1}^{3} t_i$ then t_0 is the tuple

(<1112,SMITH,5819.20>,int×char×real,{E-CODE,E-NAME,SALARY}, {<1>,<2>,<3>},{f(E-CODE)=<1>,f(E-NAME)=<2>,f(SALARY)=<3>}).

Because the naming function f_x is bijective it has the inverse function which is denoted $f_x^{-1}: I_x \rightarrow AN$. For example, in the tuple t_0 $f_x^{-1}(<2>)$=E-NAME.

Consider the formal representation (r,X,AN,I_x,f_x) of the tuple. The component r is a finite n-tuple which has been constructed from the instances of the tuple components. In the set I_x

there is one index for each factor of the Cartesian product expressed in the component X. We utilize this feature in the following notational convention:

Notational convention 4: X_ξ $(\xi \epsilon I_X)$ means that type in X with which the index ξ is associated, for example in t_0 $X_{<3>}$=real.

We must be able to refer to the components of the formal representation of a tuple component and of a tuple separately. For this we use the selection function σ.

Definition 5: Let $t=(r,X,AN,I_X,f_X)$ be any tuple component or tuple then $\sigma_i(t)$ $(1 \leq i \leq 5)$ selects the ith component of the representation.

In the context of the representation of a relation we shall also apply the selection function σ (then i can vary from 1 to 7). For example, in t_3 above $\sigma_1(t_3)$=5819.20 and $\sigma_2(t_3)$=real.

3.2. The projection of a tuple

Many manipulations require reference to a single tuple component of the tuples.

Definition 6: Let $t=(r,X,AN,I_X,f_X)$ be any tuple and <i> an index $(<i> \epsilon I_X)$ then $t[<i>]=(r[i],X_{<i>},\{f_X^{-1}(<i>)\},\{<1>\},$ $f_t(f_X^{-1}(<i>))=<1>)$ where f_t is the naming function of the tuple component.

It is worth noting that t[<i>] gives the complete description of the tuple component, for example t_0[<3>]=(5819.20,real, {SALARY},{<1>},f(SALARY)=<1>), i.e. the tuple component t_3 above.

In the definitions of relational operations we apply often projections of tuples and therefore we define the projection below.

Definition 7: Let $t=(r,X,AN,I_X,f_X)$ be any tuple and $I=\{<i_1>,$ $<i_2>,\ldots,<i_n>\}$ any index set such that $I \subseteq I_X$ then the projection of the tuple

$$t[I]= \bowtie_{i \epsilon I} t[i].$$

For example, $t_0[<2>,<3>]=<t_0[<2>],t_0[<3>]>=(<$SMITH,5819.20$>,$ char×real,{E-NAME,SALARY},{<1>,<2>},{f(E-NAME)=<1>,f(SALARY)=<2>}) It is obvious that the projection produces the complete representation for the result tuple, too. We can also express the projection with attribute names instead of indices. Let A be a set of attribute names such that $A \subseteq AN$. We define that $t[A]=t[\cup_{i \epsilon A} f_X(i)]$. In our sample projection the set A was {E-NAME,SALARY}.

At the formal level we have two basic ways of defining and manipulating tuples of relations (see e.g. [30]). We can represent them either as labeled n-tuples or as ordered n-tuples. Labeled n-tuples can be represent in the form $\{a_1:d_1,a_2:d_2,\ldots,a_n:d_n\}$ where $a_1,a_2,\ldots,a_n$ are distinct attribute names and $d_1,d_2,\ldots,d_n$ are values from the corresponding domains, in other words, each value is explicitly associated with an attribute [26]. Thus, a labeled n-tuple consists of unordered attribute-value pairs.

Ordered n-tuples can be represented in the form $<d_1,d_2,\ldots,d_n>$ where the values $d_1,\ldots,d_n$ appear in the same order as their domains in the cartesian product of which the relation is a subset. In other words, the relative position in an ordered n-tuple is important. In our formalism above tuples are considered as ordered n-tuples because ordered n-tuples are the conventional background structure of the relation.

In the formalism one index is associated with each component of a tuple. The explicit use of indices gives us many benefits. They can be used effectively for example to check the compatibility of two attributes. However, we can use in our formalism attribute names instead of indices (see the definition of the projection of the tuple). This is possible because we have the bijective function between the set of attribute names and the index set. The following definition of the compatibility of attributes illustrates the utilization of the naming function.

Definition 8: In a tuple (r,X,AN,I_x,f_x) two attributes A (ϵAN) and B (ϵAN) are compatible iff $X_{f_x(A)}=X_{f_x(B)}$. Otherwise they are incompatible.

In the sample tuple t_0 where the component X=int×char×real the attributes E-NAME and SALARY are incompatible because $X_{f_x(\text{E-NAME})}=X_{<2>}=\text{char}$ and $X_{f_x(\text{SALARY})}=X_{<3>}=\text{real}$. The compatibility of attributes in two different tuples is defined analogically.

In the structural sense a relation consists of homogenous tuples. This requirement is expressed with the concept 'tuples of the same type'.

Definition 9: Let $t_1,t_2,\ldots,t_n$ be any tuples such that $t_i=(r_i,X_i,AN_i,I_{x_i},f_{x_i})$, i=1,...,n. If for any two tuples t_i,t_j ($i\neq j$) the following condition holds $X_i=X_j \wedge AN_i=AN_j \wedge I_{x_i}=I_{x_j} \wedge f_{x_i}=f_{x_j}$ then the tuples are of the same type.

In other words, tuples of the same type differ from each other only with respect to the component r (the instance of the tuple). Let t_4 and t_5 be the following tuples t_4=(<1155, TAYLOR,5500.40>,int×char×real,{E-CODE,E-NAME,SALARY},{<1>,<2>, <3>},{f(E-CODE)=<1>,f(E-NAME)=<2>,f(SALARY)=<3>}) and t_5=(<1100,DALE,5819.20>,int×char×real,{E-CODE,E-NAME,SALARY},

{<1>,<2>,<3>},{f(E-CODE)=<1>,f(E-NAME)=<2>,f(SALARY)=<3>})
then the tuples t_0, t_4, t_5 are of the same type because the only difference between them is r-component.

3.3. Functional dependencies

Functional dependencies (FDs) have been investigated and used widely in the relational data base theory (e.g. [9, 2, 6,30]). If AN is the set of attribute names for some relation and $A,B \subseteq AN$, then, according to Ullman, A functionally determines B if, at any moment, any two tuples in the relation with equal projections on attributes A also have equal projections on attributes B [30].

We define FDs first as time-dependent properties of a set of tuples of the same type. Let T be such a set, i.e. $T=\{(r_i,X,AN,I_X,f_X) \mid i=1,\ldots,n\}$, and $A,B \subseteq AN$ two attribute name sets. Now A functionally determines B means that, for any pair of tuples $t_i, t_j \in T$, if the projections $t_i[A]$ and $t_j[A]$ are equal, then also the projections $t_i[B]$ and $t_j[B]$ must be equal.

Definition_10: Let $T=\{(r_i,X,AN,I_X,f_X) \mid i=1,\ldots,n\}$ and $A \subseteq AN \wedge B \subseteq AN$. If $t_i, t_j \in T: t_i[A]=t_j[A] \Rightarrow t_i[B]=t_j[B]$ then A functionally determines B in T.

Notational_convention_5: An FD "A functionally determines B in T" is represented as FD=A→B.

We take following convention of referring to the components of this representation:

Notational_convention_6: Let FD=A→B be valid in T. Then left(FD)=A and right(FD)=B.

Definition_11: An FD=A→B is called <u>trivial</u> if $B \subseteq A$ and else <u>non-trivial</u>.

In the set T there may be several different FDs. We define the set of all valid FDs in T with the function FD-set(T).

Definition_12: Let $T=\{(r_i,X,AN,I_X,f_X) \mid i=1,\ldots,n\}$.
$FD\text{-}set(T)=\{A \rightarrow B \mid t_i, t_j \in T \wedge A,B \subseteq AN: t_i[A]=t_j[A] \Rightarrow t_i[B]=t_j[B]\}$.

The function FD-set(T) yields the set of all FDs valid in T. An example will be given below when defining the relation constructor. The function will be used below in the definition of relational algebra (RA) that produces result relations with all valid FDs included into the representation.

It should be noted that the set of FDs given by FD-set(T) is valid at the moment of its computation. This notion of FDs is not the same as the notion of time-invariant FDs of relations as defined by Ullman [30]. Later below we shall introduce the application dependent, time-invariant FDs (of relations) and methods to define them for the result relations of RA operations.

The candidate keys are used for unique identification of tuples. One among them is chosen as the (main) key of the relation. Keys have many applications in the relational data base theory [9,30]. A candidate key is such a set of attributes A that the tuple projections $t[A], t \in T$, uniquely identify the tuples t. Candidate keys are also minimal: no subset of its attributes has the unique identification property.

Definition 13: Let $T=\{(r_i,X,AN,I_X,f_X) \mid i=1,\ldots,n\}$.
$\text{cand-keys}(T)=\{A \mid A \subseteq AN \wedge A \rightarrow AN \in \text{FD-set}(T) \wedge \nexists B \subset A: B \rightarrow AN \in \text{FD-set}(T)\}$.

For example, let t_0, t_4 and t_5 be the tuples given above and $T=\{t_0,t_4,t_5\}$. Now $\text{cand-keys}(T)=\{\{\text{E-CODE}\},\{\text{E-NAME}\}\}$, i.e. attributes E-CODE and E-NAME are candidate keys in T.

The closure of a set of FDs (say Y) contains all FDs implied by the FDs in Y. The closure is derived using the inference rules in the following recursive definition, [2,3].

Definition 14: Let $T=\{(r_i,X,AN,I_X,f_X) \mid i=1,\ldots,n\}$ and $A \subseteq AN$, $A \neq \emptyset$ and $Y \subseteq \text{FD-set}(T)$. The closure Y^+ of Y contains the following FDs:

1. $Y \subseteq Y^+$
2. $A \rightarrow A \in Y^+$
3. $y \in Y^+ \wedge z \in Y^+ : \text{right}(y)=\text{left}(z) \Rightarrow \text{left}(y) \rightarrow \text{right}(z) \in Y^+$
4. $y \in Y^+ : \text{left}(y) \subseteq A \Rightarrow A \rightarrow \text{right}(y) \in Y^+$
5. $y \in Y^+ : A \subseteq \text{right}(y) \Rightarrow \text{left}(y) \rightarrow A \in Y^+$
6. $y \in Y^+ \wedge z \in Y^+ : \text{left}(y) \cup \text{left}(z) \rightarrow \text{right}(y) \cup \text{right}(z) \in Y^+$.

Note that FD-set(T) contains any closure inferred from $Y \subseteq \text{FD-set}(T)$.

3.4. Relation

In this section we consider the formal representation of a relation. A relation is constructed from tuples of the same type.

Definition 15: Let $T=\{(r_i,X,AN,I_X,f_X) \mid i=1,\ldots,n\}$ be a finite set of tuples of the same type and RN any relation name such that $RN \notin AN$. Then RN:T (The relation constructor operation) constructs the relation

$(\cup_i r_i, P(X), RN, AN, I_{p(x)}, f_{p(x)}, \text{FD-set}(T))$; where
P(X) is the power set of the type X,
$I_{p(x)}=\{<1>\} \cup \{<1,\xi> \mid \xi \in I_X\}$

$f_{p(x)}: AN \cup \{RN\} \rightarrow I_{p(x)}$

$$f_{p(x)} = \begin{cases} f_{p(x)}(RN) = <1> \\ f_{p(x)}(a) = <1, f_x(a)>, a \in AN \end{cases}$$

Let $T=\{t_0, t_4, t_5\}$ be a set of tuples described above, now the expression E=EMPLOYEE:T constructs for E the representation ({<1112,SMITH,5819.20>,<1115,TAYLOR,5500.40>,<1100,DALE,5819.20>}, P(int×char×real),EMPLOYEE,{E-CODE,E-NAME,SALARY},{<1>,<1,1>, <1,2>,<1,3>},{f(EMPLOYEE)=<1>,f(E-CODE)=<1,1>,f(E-NAME)=<1,2>, f(SALARY)=<1,3>},FD-set(T)). The FD-set(T) has been explicitly presented in Appendix. For example, the functional dependency fd={E-CODE}→{E-NAME,SALARY} belongs to the FD-set(T). In this case left(fd)={E-CODE} and the expression E-NAMEϵright(fd) is true.

This representation corresponds to the following tabular visualization of the relation.

EMPLOYEE(E-CODE,E-NAME,SALARY)

1112	SMITH	5819.20
1115	TAYLOR	5500.40
1100	DALE	5819.20

Figure 1. The tabular representation of the sample relation E.

We name the components of the relation analogically with the corresponding components of the tuple component and of the tuple. Any relation is described as seven-tuple $(r, X, RN, AN, I_x, f_x, FD)$, where

r is an instance of the relation (The term relation value has been used with the same meaning in [15, 26].),
X is a relation type,
RN is a relation name,
AN is a set of attribute names,
I_x is an index set associated with the relation type X,
$f_x: \{RN\} \cup AN \rightarrow I_x$ is a naming function and
FD is a set of functional dependencies.

The components of this representation have the following mathematical properties

1° $r \epsilon X$

2° $\{RN\} \cap AN = \emptyset$

3° $|I_x| = |AN| + 1$

4° The naming function $f_x: \{RN\} \cup AN \rightarrow I_x$ is bijective.

5° The relation type X has always the form: $P(D_1 \times \ldots \times D_n)$ where each domain D_i $(1 \leq i \leq n)$ is either some primitive type (int,char,real) or some application dependent domain.

6° The index set $I_x = \{<1>\} \cup \{<1,i> | i \epsilon \{1, \ldots, n\}\}$

7° $\bigcup_{i \epsilon FD} (left(i) \cup right(i)) \subseteq AN$

In fact, we can use the points 1^{o}-7^{o} as an implicit set of rules for constructing the formal representation of a relation.

Consider the components of this representation. The component r is a set whose elements are instances of tuples, i.e. it includes all rows of the tabular visualization of a relation. This means that $|r|$ expresses the cardinality of a relation. The cardinality of our sample relation is three. Although r itself is a set it is also an element in the set described in X (see point 1^{o}), i.e. the relation type describes always a set whose elements are sets. A relation is defined as a subset of the cartesian product $D_1 \times \ldots \times D_n$ whereas $P(D_1 \times \ldots \times D_n)$ denotes the set of all possible subset of the cartesian product $D_1 \times \ldots \times D_n$, i.e. a relation is always an element in this power set.

The index set I_x is associated with the relation type $X=P(D_1 \times D_2 \times \ldots \times D_n)$. In it the index $<1>$ is associated with the whole relation and the index $<1,i>$ is associated with the ith domain in the cartesian product $D_1 \times \ldots \times D_n$. Thus, if ξ is any index $(\xi \epsilon I_x)$ then X_ξ means that type with which the index ξ is associated. The naming function associates the relation name with the index $<1>$ and each attribute name with an index of type $<1,i>$ $(1 \leq i \leq n)$. In other words, we can use indices to express a correspondence between attribute names and their domains. The component AN contains all attribute names of the relation, i.e. $|AN|$ expresses the degree of the relation.

The component r in our representation describes the instance level. Also, the functional dependency is a concept at the instance level. However, it is described with attribute names which belong to the schema level. The components X, RN, AN, I_x and f_x describe the schema level. In addition, the component X describes the mathematical type of the relation.

We shall define a relational algebra (RA) where relations are described in this way. In other words, the definition is based on n-ary relations which contain also their complete descriptions. N-ary relations are the conventional background structure of the relational model. Quotient relations afford an alternative possibility of defining a relationally complete RA [16,31].

In the definition of the relational algebra we manipulate tuples and therefore we need the inverse operation for the construction operation of the relation. This inverse operation splits a relation into the set of its tuples.

Definition 16: Let $B=(r,X,R,AN,I_x,f_x,FD)$ be some relation such that $B=R:T$. Then $\sim B=T$.

The inverse operation for the construction operation of the sample relation E is $\sim E=\{t_0,t_4,t_5\}$ where each of the tuples t_0, t_4 and t_5 contains the five components described above.

It is easy to see that the following properties hold between the construction operation and its inverse operation: ~(A:T)=T and A:(~A:T)=A:T. We refer to the complete formal representation of a tuple t in the relation R simply by t∈~R.

3.5. Relational algebra

RA is conventionally defined to manipulate the operand relation instances and to produce a result relation instance [8,17,27]. The explicit manipulation of the schema level components has been left to the reader. Our RA produces complete schemas for the result relations. It is defined formally and exactly on the basis of the formalism developed above. This algebra has been extended with FDs from the algebra by Niemi [24]. Using indices in the description of the schema components does not mean any commitment to the memory representations of the relations.

In addition to the relation from above, we use P and D as sample relations to illustrate the RA operations and the relational data base. Formal representations for the relations P and D are given in the figures 2 and 3. Only the non-trivial FDs are described in the FD-components of the seven-tuples. The other FDs are denoted by "...".

P=({<1112,SMITH,5819.20>,<1115,TAYLOR,4000.00>,<1127,JONES,
5200.00>,<1100,DALE,5819.20>},P(int×char×real),PROVISION,
{E-CODE,E-NAME,P-PAY},{<1>,<1,1>,<1,2>,<1,3>},{f(PROVISION=<1>,
f(E-CODE)=<1,1>,f(E-NAME)=<1,2>,f(P-PAY)=<1,3>},{{E-CODE}→
{E-NAME,P-PAY},{E-NAME}→{E-CODE,P-PAY},{E-CODE,E-NAME}→
{P-PAY},...})

Figure 2. The sample relation P

D=({<SALES,1112>,<SALES,1100>,<SALES,1115>,<PROD,1127>},
P(char×int),DEPT,{D-NAME,D-EMP},{<1>,<1,1>,<1,2>},{f(DEPT)=<1>,
f(D-NAME)=<1,1>,f(D-EMP)=<1,2>},{{D-EMP}→{D-NAME},{D-EMP}→
{D-NAME,D-EMP},...})

Figure 3. The sample relation D

The operations union, intersection and difference of RA require that their operands are union-compatible [10]. This is defined as follows.

Definition 17: Two relations $R_1=(r_1,X_1,RN_1,AN_1,I_{x_1},f_{x_1},FD_1)$ and $R_2=(r_2,X_2,RN_2,AN_2,I_{x_2},f_{x_2},FD_2)$ are union-compatible iff $X_1=X_2$ and otherwise union-incompatible.

The relational algebra can be defined as follows. Using the relation constructor, this algebra finds all result relation FDs. As operands we use relations $R_1=(r_1,X_1,RN_1,AN_1,I_{x_1},f_{x_1},FD_1)$ and $R_2=(r_2,X_2,RN_2,AN_2,I_{x_2},f_{x_2},FD_2)$ which are any two relations. We use the name λ as the RN-component of the result relations.

Definition 18: intersection$(R_1,R_2)=\lambda:\{t|t\epsilon\sim(r_1\cap r_2,X_1,RN_1,AN_1,I_{x_1},f_{x_1},FD_1)\}$

Requirement: $X_1=X_2$.

Note that the inverse operation of the relation constructor ~ splits the seven-tuple into a set of tuples which do not have FD-components. The relation constructor : then finds the result relation FDs as defined above. Intersection requires union-compatibility of its operands but allows different attribute name sets. We have arbitrarily chosen to use the AN_1 component of the first operand in the result relation. It is equally possible to use the corresponding component of the second operand or to create new names by some transformation. As an example, intersection(E,P)=({<1112,SMITH,5819.20>,<1100, DALE,5819.20>},P(int×char×real),λ,{E-CODE,E-NAME,SALARY},{<1>, <1,1>,<1,2>,<1,3>},{f(λ)=<1>,f(E-CODE)=<1,1>,f(E-NAME)=<1,2>, f(SALARY)=<1,3>},FD_λ).

Definition 19: union$(R_1,R_2)=\lambda:\{t|t\epsilon\sim(r_1\cup r_2,X_1,RN_1,AN_1,I_{x_1},f_{x_1},FD_1)\}$

Requirement: $X_1=X_2$.

For example, union(E,P)=({<1112,SMITH,5819.20>,<1115,TAYLOR, 5500.40>,<1100,DALE,5819.20>,<1115,TAYLOR,4000.00>,<1127,JONES, 5200.00>},P(int×char×real),λ,{E-CODE,E-NAME,SALARY},{<1>,<1,1>, <1,2>,<1,3>},{f(λ)=<1>,f(E-CODE)=<1,1>,f(E-NAME)=<1,2>,f(SALARY)= <1,3>},FD_λ).

Definition 20: difference$(R_1,R_2)=\lambda:\{t|t\epsilon\sim(r_1-r_2,X_1,RN_1,AN_1,I_{x_1},f_{x_1},FD_1)\}$

Requirement: $X_1=X_2$.

For example, difference(E,P)=({<1115,TAYLOR,5500.40>},P(int× char×real),λ,{E-CODE,E-NAME,SALARY},{<1>,<1,1>,<1,2>,<1,3>}, {f(λ)=<1>,f(E-CODE)=<1,1>,f(E-NAME)=<1,2>,f(SALARY)=<1,3>},FD_λ).

Definition 21: projection$(R_1,A)=\lambda:\{t[A]|t\epsilon\sim R_1\}$
Requirement: $A\subseteq AN$.

This definition is based on the definition of tuple projection with a set of attribute names (see sec. 3.2). As an example, projection (E,{E-CODE,SALARY})=λ:{t[{E-CODE,SALARY}]|tε~E}=

({<1112,5819.20>,<1115,5500.40>,<1100,5819.20>}, P(int×real), λ, {E-CODE,SALARY},{<1> <1,1> <1,2>}, {f(λ)=<1>, f(E-CODE)=<1,1>, f(SALARY)=<1,2>}, {{E-CODE}→{SALARY}...}).

An equivalent way of defining the projection is based on an index set instead of a set of attribute names. Tuple projections with attribute names vs. indices are analogical to relation projections with attribute names vs. indices, respectively (see sec. 3.2 for comparison).

The restriction (or selection) operation may have any finite number of predicates testing the validity of operand tuples for inclusion into result. A set of predicates connected by Boolean operators is in a canonical form if it is a conjuction of predicates [17].

Notational convention 7: The predicates are represented as triples (a_i,ω_i,b_i), $i=1,\ldots,n$, where $a_i,b_i \in AN_1 \wedge X_{f_{x_1}(a_i)} = X_{f_{x_1}(b_i)} \wedge \omega_i \in \{=,\neq,\leq,<,>,\geq\}$.

The predicates define a comparison ω between two attributes which are identified by their names. To allow predicates with constant values we take the following convention.

Notational convention 8: In a predicate (a_i,ω_i,b_i) either a_i or b_i may be a constant value, denoted by $\underline{cons}_i$.

The constant value may be on either side of the predicate. However, below we consider only predicates of types (a,ω,b) and $(a,\omega,\underline{cons})$. In the latter case we require that $\underline{cons} \in X_{f_{x_1}(a)}$ i.e. the constant value must be compatible with the attribute. The set of predicates is denoted by $\bigcup_{i=1}^{n}(a_i,\omega_i,b_i)$.

Definition 22: $\underline{restriction}(R_1, \bigcup_{i=1}^{n}(a_i,\omega_i,b_i)) = \lambda:\{t \mid t \in \sim R_1 \wedge_{i=1}^{n} \sigma_1(t[\{a_i\}])\omega_i\sigma_1(t[\{b_i\}])\}$

Any Boolean predicate can be expressed as a disjunction of canonical forms. We leave it up to the reader to construct a generalized restriction that allows a disjunction of canonical forms as its predicate. As an example of restriction we restrict the relation E with predicates containing constants: $\underline{restriction}(E,\{(E\text{-}CODE,\leq,1110),(SALARY,=,5819.20)\}) = \lambda:\{t \mid t \in \sim E \wedge \sigma_1(t[\{E\text{-}CODE\}]) < 1110 \wedge \sigma_1(t[\{SALARY\}]) = 5819.20\} = (\{<1100,DALE,5819.20>\},P(int \times char \times real),\lambda,\{E\text{-}CODE,E\text{-}NAME,SALARY\},\{<1>,<1,1>,<1,2>,<1,3>\},\{f(\lambda)=<1>,f(E\text{-}CODE)=<1,1>,f(E\text{-}NAME)=<1,2>,f(SALARY)=<1,3>\},FD_\lambda)$.

Definition 23: $\underline{product}(R_1,R_2) = \lambda:\{<t_1,t_2> \mid t_1 \in \sim R_1 \wedge t_2 \in \sim R_2\}$
Requirement: $AN_1 \cap AN_2 = \emptyset$

For example, product(E,D)=({<1112,SMITH,5819.20,SALES,1112>, <1112,SMITH,5819.20,SALES,1100>,<1112,SMITH,5819.20,SALES,1115>, <1112,SMITH,5819.20,PROD,1127>,<1115,TAYLOR,5500.40,SALES,1112>, <1115,TAYLOR,5500.40,SALES,1100>,<1115,TAYLOR,5500.40,SALES,1115>, <1115,TAYLOR,5500.40,PROD,1127>,<1100,DALE,5819.20,SALES,1112>, <1100,DALE,5819.20,SALES,1100>,<1100,DALE,5819.20,SALES,1115>, <1100,DALE,5819.20,PROD,1127>},P(int×char×real×char×int),λ, {E-CODE,E-NAME,SALARY,D-NAME,D-EMP},{<1>,<1,1>,<1,2>,<1,3>, <1,4>,<1,5>},{f(λ)=<1>,f(E-CODE)=<1,1>,f(E-NAME)=<1,2>,f(SALARY)= <1,3>,f(D-NAME)=<1,4>,f(D-EMP)=<1,5>},FD_λ) . The attribute names in the operands must be unique.

Definition 24: $\underline{join}(R_1,R_2,\bigcup_{i=1}^{n}(a_i,\omega_i,b_i))=\lambda:\{<t_1,t_2>|t_1\epsilon\sim R_1\wedge t_2\epsilon\sim R_2\wedge\bigwedge_{i=1}^{n}\sigma_1(t_1[\{a_i\}])\omega_i\sigma_1(t_2[\{b_i\}])\}$

where each ω_i is some comparison operator $(\omega_i\epsilon\{=,\neq,...\})$

Requirements: 1. $a_i\epsilon AN_1\wedge b_i\epsilon AN_2$, i=1,...,n,

2. $X_{f_{x_1}(a_i)}=X_{f_{x_2}(b_i)}$, i=1,...,n , and

3. $AN_1\cap AN_2=\emptyset$.

For example, join(E,D,{(E-CODE,=,D-EMP)})=λ:{<t_1,t_2>|$t_1\epsilon\sim$E$\wedge$ $t_2\epsilon\sim$D$\wedge\sigma_1(t_1[\{$E-CODE$\}])=\sigma_1(t_2[\{$D-EMP$\})]$}=({<1112,SMITH,5819.20, SALES,1112>,<1115,TAYLOR,5500.40,SALES,1115>,<1100,DALE,5819.20, SALES,1100>},P(int×char×real×char×int),λ,{E-CODE,E-NAME,SALARY, D-NAME,D-EMP},{<1>,<1,1>,<1,2>,<1,3>,<1,4>,<1,5>},{f(λ)=<1>, f(E-CODE)=<1,1>,f(E-NAME)=<1,2>,f(SALARY)=<1,3>,f(D-NAME)=<1,4>, f(D-EMP)=<1,5>},FD_λ) .

The second requirement guarantees compatibility of the attributes in the predicates. The third requirement guarantees the uniqueness of the attribute names in the result relation. The join-operation may be generalized in the same way as the restrict-operation to allow predicates that are disjunctions of canonical forms. This generalization is not presented here.

The natural join is a common variant of join. In natural join the comparison operators ω_i, (i=1,...,n), are always equality operators (=). Usually the natural join is taken on operand attributes having common names. In our representation the intersection $AN_1\cap AN_2$ determines the common attributes of relations R_1 and R_2. If preserved into the result, the common attributes would be equal. Only the other of them is preserved.

Definition 25: $\underline{n\text{-}join}(R_1R_2)=\lambda:\{<t_1,t_2[AN_2-AN_1]>|t_1\epsilon\sim R_1\wedge t_2\epsilon\sim R_2\wedge\forall b\epsilon AN_1\cap AN_2:t_1[\{b\}]=t_2[\{b\}]\}$

Requirement: $AN_1\cap AN_2\neq\emptyset$.

In case $AN_1\cap AN_2=\emptyset$ we must express the corresponding attributes in R_1 and R_2.

We leave it up to the reader as an exercise to define the division-operation. The RA defined above is relationally complete [10] even without division. If required, a sequence of product, projections and differences may be used instead of a division [10].

The operations of our RA can be combined into sequences as in any other RA. For example, projection (join(restrict(E,{(SALARY,>, 5600.00)}),D,{(E-CODE,=,D-EMP)}),{E-NAME,D-NAME}) produces the relation ({<SMITH,SALES>,<DALE,SALES>},P(char×char),λ, {E-NAME,D-NAME},{<1>,<1,1>,<1,2>},{f(λ)=<1>,f(E-NAME)=<1,1>, f(D-NAME)=<1,2>},{{E-NAME}→{D-NAME},{E-NAME}→{E-NAME,D-NAME},...}).

3.6. Inference rules for time-invariant FDs of result relations

The notion of FDs so far used is a time-varying property of a tuple set. However, in the relational data base theory the main interest is in time-invariant FDs as properties of relations [3, 6,12,30].

Definition 26: Let $time_1$ and $time_2$ be any moments of time. Let R_1 be any relation with relation name RN and functional dependencies FD_1 at $time_1$ and $y \in FD_1$. Let T be any set of new tuples and the updated relation $R_2 = RN_1 :\sim R_1 \cup T$ at $time_2$. Let FD_2 be the FD-set of R_2. If at $time_2$ $y \in FD_2$ then y is a time-invariant FD.

Notational convention 9: The time-invariant FDs of R_1 are denoted by $FD\text{-}i_1$.

Obviously, the time-invariant FDs of relation R_1 are included into the FD-component FD_1 of R_1, i.e. $FD\text{-}i_1 \subseteq FD_1$.

The time-invariant FDs are dictated by the application area of the relational data base and they are (required to be) valid irrespective of the particular instances of the relations at some moment. Normally the time-invariant FDs also bear some meaning with respect to the application. On the contrary, many of the time-varying FDs are purely occasional.

We define the candidate keys of a relation on the basis of time-invariant FDs. The selector function σ_i, i=1,...,7, is used analogically to the selector function σ_i for tuples.

Definition 27: Let $R_1=(r_1,X_1,RN_1,AN_1,I_{x_1},f_{x_1},FD\text{-}i_1)$.

$$rel\text{-}keys(R_1)=\{A \mid A \subseteq AN_1 \wedge A \rightarrow AN_1 \in (\sigma_7(R_1))^+ \wedge \nexists B \subset A : B \rightarrow AN_1 \in (\sigma_7(R_1))^+\}.$$

We apply the definition of closure above in the context of the time-invariant FDs, too.

Below we present inference rules for transforming time-invariant operand FDs and those definitely produced by the RA-operations into the result relations. The rules define an FD-set for the result of each RA operation. This set is substituted for the set defined by the relation constructor. The rules do not require inspecting the instances of the operand or result relations. The rules have been proved in [21].

Intersection requires that we are able to express the FDs of one operand in terms of corresponding attributes of the other operand. For this purpose the function fd-renaming is defined below. Let $R_1=(r_1,X_1,RN_1,AN_1,I_{x_1},f_{x_1},FD_1)$ and $R_2=(r_2,X_2,RN_2,AN_2,I_{x_2},f_{x_2},FD_2)$ be two union-compatible relations.

Because union-compatible relations are subsets of the same Cartesian product (i.e. $X_1=X_2$) we can establish attribute name correspondences through their indices and naming functions

and inverse naming functions. For any attribute name $b \in AN_2$ the expression

$$f_{x_1}^{-1}(f_{x_2}(b))$$

yields the corresponding attribute in AN_1. For example, the relations E and P are union-compatible and such an expression gives SALARY and P-PAY as corresponding attributes. Using this correspondence the FDs in FD_2 may be expressed with corresponding attribute names of AN_1.

Definition 28: Let $R_1=(r_1,X_1,RN_1,AN_1,I_{x_1},f_{x_1},FD\text{-}i_1)$ and $R_2=(r_2,X_2,RN_2,AN_2,I_{x_2},f_{x_2},FD\text{-}i_2)$ be relations such that $X_1=X_2$.

$$\text{fd-renaming}(f_{x_1},f_{x_2},FD\text{-}i_2)=\{\cup_{i\in A} f_{x_1}^{-1}(f_{x_2}(i)) \rightarrow \cup_{j\in B} f_{x_1}^{-1}(f_{x_2}(j)) \mid A\rightarrow B\in FD\text{-}i_2\}$$

The time-invariant FDs for the result relations are as follows. In the definitions we use $R_1=(r_1,X_1,RN_1,AN_1,I_{x_1},f_{x_1},FD\text{-}i_1)$ and $R_2=(r_2,X_2,RN_2,AN_2,I_{x_2},f_{x_2},FD\text{-}i_2)$. In this case the result relations of RA operations contain the following FDs.

Definition 29: In the result relation of intersection(R_1,R_2)

$$FD\text{-}i=FD\text{-}i_1 \cup \text{fd-renaming}(f_{x_1},f_{x_2},FD\text{-}i_2).$$

Definition 30: In the result relation of difference(R_1,R_2)

$$FD\text{-}i=FD\text{-}i_1 .$$

Definition 31: In the result relation of union(R_1,R_2)

$$FD\text{-}i=\{A\rightarrow B \mid A\subseteq AN_1 \wedge B\subseteq A\}.$$

Definition 32: In the result relation of product(R_1,R_2)

$$FD\text{-}i=FD\text{-}i_1 \cup FD\text{-}i_2 .$$

Definition 33: In the result relation of projection(R_1,A)

$$FD\text{-}i=\{y \mid y\in FD\text{-}i_1^{+} \wedge \text{left}(y)\cup\text{right}(y)\subseteq A\} .$$

Definition 34: In the result relation of restriction$(R_1, \cup_{i=1}^{n}(a_i,\omega_i,b_i))$

$$FD\text{-}i=FD\text{-}i_1 \cup \{\{a_i\}\rightarrow\{b_i\},\{b_i\}\rightarrow\{a_i\} \mid \omega_i='=' \wedge a_i\neq \underline{cons} \wedge b_i\neq \underline{cons}\}$$
$$\cup \{D\rightarrow\{a_i\} \mid D\subseteq AN_1 \wedge a_i\neq \underline{cons} \wedge b_i=\underline{cons} \wedge \omega_i='='\}.$$

The result contains the time-invariant FDs of the operand and, in addition, for each predicate with $\omega_i='='$ a set of new time-invariant FDs. If neither $a_i=$ cons nor $b_i=$ cons then this set contains mutual FDs between the attributes. If one of the attributes (here b_i) is cons then all attribute sets $D\subseteq AN_1$ functionally determine the other attribute (a_i).

In the join operation, where predicates with $a_i \epsilon AN_1$ and $b_i \epsilon AN_2$ are the only possible ones, a set of mutual dependencies similar to those in restriction is created for predicates with ω_i='='. In the natural join this is unnecessary because the duplicate attributes are not preserved into the result.

Definition 35: In the result relation of $join(R_1,R_2, \bigcup_{i=1}^{n} (a_i, \omega_i, b_i))$

$FD\text{-}i=FD\text{-}i_1 \cup FD\text{-}i_2 \cup \{\{a_i\}\rightarrow\{b_i\},\{b_i\}\rightarrow\{a_i\} | \omega_i='='\}$

Definition 36: In the result relation of $n\text{-}join(R_1,R_2)$ $FD\text{-}i=FD\text{-}i_1 \cup FD\text{-}i_2$.

These rules are limited so that at each operation, only the operand attribute names, time-invariant operand FDs and the parameters of the operation (e.g. predicates, attribute sets) are the premises taken into account. Other information that might be available in the application area of the data base, in the data base schema or in the possible subexpressions producing the operands is bypassed. This raises the following notes:

- The result relation of union has only trivial FDs. This is because union-compatibility is the only criterion required for taking a union.
- Identification of common subexpressions [18] (e.g. union(restrict(R_1,P_1),restrict(R_1,P_2))) would sometimes allow inferring (or preserving) additional FDs into result relations. Some techniques for identifying and transforming these into equivalent expressions are presented in [29].
- Application dependent domains sometimes allow inferring that one, none or all tuples in the operands of a restriction or join satisfy the predicate or that the comparison operator ω in some predicate may be changed to be '=' (if not so already) without affecting the correctness of the result. These obviously would allow inferring additional time-invariant FDs.

It should be noted that FDs are but one type of dependencies recognized in the relational data base literature. Similar inference rules may be defined for the other dependency types (e.g. multivalued dependencies [14]), too.

3.7. Relational data base

Our formal representation of a relation defined in this paper affords a possibility of representing the relational data base in such a way that it contains the description (schema) of the data base, too. A relational data base consists of relations and its schema consists of schemata of these relations. The seven-tuple representation of a relation contains both the

instance and the schema of a relation. Therefore, a relational data base and its description can be represented as a set of seven-tuples. Because the relational data base schema must define the time invariant properties of relations we have time invariant FD's in relations. Ausiello et al. call functional dependencies static constraints in their 6-tuple representation of a relational data base schema [4,5].

Definition 37: Let $(r_i, X_i, RN_i, AN_i, I_{x_i}, f_{x_i}, FD\text{-}i_i)$ be the seven-tuple representation of the relation rel-i then a relational data base (rdb) can be defined as follows.

rdb={rel-1,rel-2,...,rel-n}

A relational data base is now described exactly as a set where we can apply set-valued expressions in the normal way. For example, the expression $\{z|z\epsilon rdb \wedge P(z)\}$ gives the set of relations for which the predicate P is true.

Next we give examples on how we can flexibly express different kinds of things with the formalism. Our sample data base consists of relations E,D,P above, i.e. rdb={E,D,P} .

Sample expression-1: $|rdb|$ gives the number of relations in the relational data base rdb. In our sample data base $|rdb|=3$.

Sample expression-2: $x\epsilon rdb \wedge \sigma_3(x)=RN$ gives the seven-tuple representation of that relation which has the name RN. In the sample data base the expression $x\epsilon rdb \wedge \sigma_3(x)=EMPLOYEE$ gives the relation E.

Sample expression-3: Let RN_1 and RN_2 be two relation names in the relational data base. Now, the expression

$$\{a|a\epsilon\sigma_4(x_1)\cap\sigma_4(x_2)\wedge x_1\epsilon rdb:\sigma_3(x_1)=RN_1\wedge x_2\epsilon rdb:\sigma_3(x_2)=RN_2\}$$

yields the common attribute names of the relations with relation names RN_1 and RN_2. The empty set as the result of the evaluation of the expression indicates that there are not such attribute names.

Sample expression-4: $\cup_{x\epsilon rdb}\ \sigma_4(x)$ defines the content of the relation data base. In our sample data base $\cup_{x\epsilon rdb}\ \sigma_4(x)=$ {E-CODE,E-NAME,SALARY,P-PAY,D-NAME,D-EMP} .

Sample expression-5: $\cup_{x\epsilon rdb}\ \sigma_3(x)$ contains all relation names in the relational data base. In the sample data base $\cup_{x\epsilon rdb}\ \sigma_3(x)=$ {EMPLOYEE,PROVISION,DEPT}.

Sample expression-6: $\bigcup_{x \in rdb} \sigma_7(x)$ gives all functional dependencies in the relational data base.

Sample expression-7: Let A be any set consisting of attribute names. Now, the expression

$$\{x \mid x \in rdb \wedge A \subseteq \sigma_4(x)\}$$

defines those relations from the relational data base which contain the attributes in A. If A={E-CODE,E-NAME} then in our sample data base this expression gives the set {E,P}.

Sample expression-8: Let RN be any relation name. The expression

$$\{x \mid x \in rdb \wedge x_1 \in rdb : \sigma_3(x_1) = RN \wedge \sigma_2(x) = \sigma_2(x_1)\}$$

selects those relations from the relational data base which are union-compatible with the relation whose name is RN. In our sample data base $\{x \mid x \in rdb \wedge x_1 \in rdb : \sigma_3(x_1) = EMPLOYEE \wedge \sigma_2(x) = \sigma_2(x_1)\} = \{P\}$.

Sample expression-9: $\{x \mid x \in rdb \wedge 2 < |\sigma_1(x)| < 5 \wedge |\sigma_4(x)| < 3\}$ contains those relations whose cardinality is between two and five and whose degree is smaller than three. In our sample data base the result of this expression is {D} .

Sample expression-10: Let A be a set of attribute names. Now,

$$\{x \mid x \in rdb \wedge A \in cand\text{-}keys(\sim x)\}$$

gives those relation where A is a possible candidate key. The empty set indicates that there is not such a relation in the data base.

Sample expressions do not form a complete list of what we can define with the formalism. They are primarily aimed at demonstrating how we can express different features of RDM exactly and conveniently. Our ultimate purpose is to provide an efficient tool to facilitate precise specifications and proofs concerning:

relational query languages,
relational data base restructuring [22], [28]
consistency of mappings in a relational DBMS architecture [1],
query optimization [22],
design of relational data bases and other topics.

4. CONCLUSIONS

We have presented a straigthforward formalization of the relational model in such a way that it provides a general and exact tool for specifications and proofs concerning various topics in the RDM research area. For this purpose we have defined all the components and operations of the relational model consciously so that the schema and instance level descriptions remain together. This formalism, except for the FD-component has been applied in the exact specification of relational query language and data restructuring. It is possible to adapt the formalism for different situations e.g. other dependency types may be substituted for the FD-component. In the structural sense, the construction operations introduced in this paper can be used to represent unnormalized relations or power set type relations [19,20] so that they contain their descriptions, too [25].

There are many papers on the RDM (e.g.[11,26]) which aim at precise representation of its components and their inter-relations. In addition to this we provide a formalism that allows flexible manipulation of the components. This mean that it can be used for exact specification and proofs concerning RDM. Because the formalism is based on set theory it avoids the complexities and heavyness of notations typical of many formal specification methods borrowed from programming language definitation.

APPENDIX

FD-set(T)={{E-CODE}→{E-CODE},{E-NAME}→{E-NAME},{SALARY}→{SALARY},
{E-CODE,E-NAME}→{E-CODE},{E-CODE,E-NAME}→{E-NAME},{E-CODE,
E-NAME}→{E-CODE,E-NAME},{E-CODE,SALARY}→{E-CODE},{E-CODE,SALARY}→
{SALARY},{E-CODE,SALARY}→{E-CODE,SALARY},{E-NAME,SALARY}→
{E-NAME},{E-NAME,SALARY}→{SALARY},{E-NAME,SALARY}→{E-NAME,SALARY},
{E-CODE,E-NAME,SALARY}→{E-CODE},{E-CODE,E-NAME,SALARY}→{E-NAME},
{E-CODE,E-NAME,SALARY}→{SALARY},{E-CODE,E-NAME,SALARY}→{E-CODE,
E-NAME},{E-CODE,E-NAME,SALARY}→{E-CODE,SALARY},{E-CODE,E-NAME,
SALARY}→{E-NAME,SALARY},{E-CODE,E-NAME,SALARY}→{E-CODE,E-NAME,
SALARY},{E-CODE}→{E-NAME},{E-CODE,SALARY}→{E-NAME},{E-CODE}→
{SALARY},{E-CODE,E-NAME}→{SALARY},{E-NAME}→{E-CODE},{E-NAME,
SALARY}→{E-CODE},{E-NAME}→{SALARY},{E-CODE}→{E-NAME,SALARY},
{E-NAME}→{E-CODE,SALARY},{E-CODE,E-NAME}→{SALARY,E-NAME},{E-CODE,
E-NAME}→{SALARY,E-CODE},{E-CODE,E-NAME}→{SALARY,E-CODE,E-NAME},
{E-CODE}→{E-CODE,E-NAME,SALARY},{E-NAME}→{E-CODE,E-NAME,SALARY}}

REFERENCES

[1] ANSI/X3/SPARC, Study group data base management Systems, Interim Report, FDT 7(2), 1975.

[2] Armstrong, W.W., Dependency Structures of Data Base Relationships. Proc. Information Processing 74, North Holland, 1974, 580-583.

[3] Arora, A.K. and Carlson, C.R., The Information Preserving Properties of Relational Database Transformations. Proc. 4th VLDB Conf., West Berlin, Sept. 1978, 352-359.

[4] Ausiello, G., Batini, C. and Moscarini, M., Conceptual relations between databases transformed under join and projection. Mathematical foundations of computer science 1980. Lecture Notes in Computer Science, 1980, 123-136.

[5] Ausiello, G., Batini, C. and Moscarini, M., On the equivalence among data base schemata, Proc. Int. conf. on data bases, University of Aberdeen, 1980, 34-36.

[6] Beeri, C., Bernstein, P.A. and Goodman, N., A Sophisticate's introduction to database normalization theory. Proc. 4th VLDB Conf. West Berlin, Sept. 1978, 113-124.

[7] Björner, D., Formalization of data base models, in Abstract Software Specifications (ed. D. Björner), Lecture Notes in Computer Science, 1980.

[8] Codd, E.F., A relational model of data for large shared data banks, Comm. ACM 13(6), 1970, 377-397.

[9] Codd, E.F., Further normalization of the relational model, In: Randall, R. (ed.), Data Base Systems, Prentice-Hall, 1972, 33-64.

[10] Codd, E.F., Relational completeness of data base sublanguages. In: Randall, R. (ed.), Data Base Systems, Prentice-Hall, 1972, 65-98.

[11] Date, C., A formal definition of the relational model, ACM Sigmod record, 13(1), 1982, 18-29.

[12] Delobel, D., An overview of the relational data theory, Proc. IFIP80, North-Holland, 1980, 413-426.

[13] Emden, M.H. and Maibaum, T.S.E., Equations compared with clauses for specification of abstract data types, Proc. Formal bases for data bases,Toulouse, Dec. 1979.

[14] Fagin, R., Multivalued dependencies and a new normal form for relational databases, ACM TODS 2(3) 1977, 262-278.

[15] Final Report of the ANSI/X3/SPARC DBS-SG Relational Database Task Group (eds. M. Brodie and J. Schmidt), ACM Sigmod record, 12(4), 1982, 1-62.

[16] Furtado, A.L. and Kerschberg, L., An algebra of quotient relations, Proc. ACM Sigmod, Toronto, August 3-5, 1977, 1-8.

[17] Gosh, S.P., Data base organization for data management, Academic Press, New York, 1977.

[18] Hall, P.A.V., Common subexpression identification in general algebraic systems. Rep. # UKSC 0060, IBM UK Scientific Centre, Nov. 1974.

[19] Jaeschke, G. and Schek, H.J., Remark on the algebra of non first normal form relations, Proc. ACM Symposium on principles of database systems, Los Angeles, March 29-31, 1982, 124-138.

[20] Jaeschke, G., An algebra of power set type relations, Heidelberg Scientific Centre, TR 82.12.002., Dec. 1982.

[21] Järvelin, K., Finding functional dependencies for intermediate relations of relational algebra expressions, Proc. First Scandinavian research seminar on information modelling and data base management, University of Tampere, 1982, 407-441.

[22] Kangassalo, H., Jaakkola, H., Järvelin, K., Lehtonen, T. and Niemi, T., System D- An integrated tool for systems design, implementation and data base management, Proc. IFIP WG 8.1 Working Conference on automated tools for information systems design and development, New Orleans, Jan. 1982, 67-83.

[23] Neuhold, E.J. and Olnhoff, Th., The Vienna Development Method (VDM) and its use for the specification of a relational data base system, Proc. IFIP 1980, North-Holland, 1980, 3-16.

[24] Niemi, T., A relational algebra for manipulating relations and their schemas together, Proc. 2nd Scandinavian research seminar on information modelling and data base management, Tampere, Jan. 1983. To appear.

[25] Niemi, T., A seven-tuple representation for hierarchical data structures, Information Systems, 8(3), 1983. To appear.

[26] Pirotte, A., A precise definition of basic relational notions and of the relational algebra, ACM Sigmod record, 13(1), 1982, 30-45.

[27] Sandberg, G., A primer on relational database concepts, IBM systems journal, 20(1), 1981.

[28] Shneiderman, B., Thomas, G., An architecture for automatic relational database system conversion, ACM TODS 7(2), 1982, 235-257.

[29] Smith, J. and Chang, P.Y-T., Optimizing the Performance of the Relation Data Base Interface. Univ. of Utah, 1975.

[30] Ullman, J., Principles of data base systems, Computer Science Press, Maryland, 1980.

[31] Tompa, F.W., Practical example of specification of abstract data types, Acta Informatica, Vol. 13, 1980, 205-224.