ASSOCIATING TYPES WITH DOMAINS OF RELATIONAL DATA BASES

Michel LACROIX*
Alain PIROTTE*+

1. Introduction.

In the DB field, there are two interpretations (or perceptions or schools of thought) concerning the relational model:

(a) one interpretation considers that the relational model contribution to the DB field essentially consists in the presentation of a clear and simple notion of (flat) file, and that many users will be very satisfied to describe their data structures as tables;

(b) another interpretation considers the relational model as a (reasonable) support for a (weak) entity relationship model.

Our work clearly belongs to category (b), the minority view in terms of the number of DB people who adhere to it. This paper shows how the introduction of scalar types enhances the semantic expressiveness of the relational model.

2. Domain-oriented Approach to the Relational Model of Data.

We have studied a version of the relational model which stresses more than the work of others the importance of the "data base domains", while remaining essentially compatible with the original work of Codd.

Thus, a relational schema consists of a collection of domains, a collection of structure definitions for relations or relation schemes and a collection of consistency constraints [4].

A relation scheme is noted $R(A_1:D_1,\ldots,A_n:D_n)$ where the A_i's are the role attributes and the D_i's are the data base domains. At every moment, the value of R is a set of n-tuples $(d_1,\ldots,d_n)$ where $d_i \in D_i$ for $1 \leqslant i \leqslant n$.

The "domain orientation" of the model has both data description and data manipulation aspects.

On the data description side, an entity-relationship (or object-predicate) perception of information is supported by choosing user-oriented data base domains, like employees, departments, companies, etc. instead of machine-oriented domains like strings, integers and booleans. Data base domains represent the important objects about which the data base contains information. Relations represent associations and properties of objects. Relation attributes express the role played by objects in relations: they are analogous for example to case markers such as prepositions in the structure of natural language sentences.

When a domain appears at several places in a relational schema, then this fact expresses in an explicit manner semantic information which cannot be expressed with the attributes only, since attributes are all different in a relation.

As some objects are not naturally assimilated to values as required in the relational model, we were led to introduce domains of "surrogates" in our model, that is, domains whose objects do not have a simple external representation.

A type is defined by each domain. If subtypes are not considered, then domains are disjoint and each object has a unique type which characterizes its membership in a domain. (If subtypes were considered, an object could belong to several domains and have several types).

On the data manipulation side, the domain orientation is supported in languages by banning the tuple variables and by introducing typed domain variables.

Several domain oriented languages were designed which can be more or less directly related to a domain relational calculus (DRC) just as constructs in tuple oriented languages can be more or less directly related to the (tuple) relational calculus of Codd [5].

In DRC, each data base domain defines a type for variables and constants. Type-checking rules constrain the operations of the language and filter

*Philips Research Laboratory, Brussels, Belgium
+Computer Corp. of America, Cambridge, MA, USA

out certain classes of meaningless queries by treating them as syntactically ill-formed.

The predicates of DRC are of two kinds:
- usual binary comparison predicates like =, ≠, <, ≪ etc. whose arguments are constants or variables; a binary predicate is well-formed only if its two arguments are of the same type and if the operation is legal for objects of the corresponding data base domain;
- 2^n-1 predicates are defined for each relation with n attributes; they have between 1 and n arguments and correspond to all possible ways of selecting between 1 and n attributes in the associated relation. Each predicate is in fact a membership predicate for a projection of the relation. It is well-formed only if the type of each of its arguments (constants or variables) corresponds to the type defined by the associated data base domain. A well-formed predicate is true if the tuple made of its arguments belongs to the associated relation projection. It is false otherwise.

Other operations of DRC are derived from the predicate calculus: conjunction, disjunction, negation, implication, quantifiers.

The range of a quantifier is restricted by the type of the quantified variable: "for all x" really means "for all values for x taken from the data base domain associated with the type of x".

Thus, if user-oriented data base domains like employees, departments, etc. are chosen when the relational data base is designed, type checking in DRC will be done on objects which are meaningful for the application, and in a way which will help users construct meaningful queries. In most other relational query languages, type checking is either done on uninterpreted strings and numbers or not done at all.

Besides DRC, we designed the FQL language which imposes a semantic control on queries, by a restricted definition of quantification, disjunction and conjunction, in addition to a control of types at the level of data base domains.

We also designed the ILL language whose syntactic structure is based on a form of natural language sentences, where objects appear as noun phrases, and properties and associations as verb phrases, and where a structure of related properties and associations is expressed as a nested structure of relative clauses [3].

In general, we found that queries in domain-oriented languages often have a reasonable English paraphrase. On the contrary, in general, the only faithful paraphrase of a query with tuple variables is in terms of table scanning, rows and columns (particularly so when quantifiers are involved) [2].

3. Background and Related Work.

As they are viewed here, types play essentially two roles:

(a) they flag objects, and variables ranging on objects, as belonging to certain categories;

(b) they constrain language operations by type checking so as to treat as syntactically incorrect expressions which denote operations considered as meaningless.

Such a view of types has become classical for scalar types in programming languages. Other aspects of the definition of such types for which any language definition has to make precise decisions include:
- the conversion (or transfer or coercion), automatic or not, of objects of a type into objects of another type;
- the representation of objects of any given type in terms of machine oriented objects like numbers strings for which machine operations are available.

Thus, in our domain oriented view of relational data bases, the definition of each data base domain includes the operations applicable on objects of the corresponding type, the conversion rules and the physical representation of objects of the type (in general there is a representation for the user interface and another representation for machine storage).

It is important to remark that the types considered in our view of relational data bases support semantic categories independently of the representation of objects of the type. Thus, for example, if the data base domains include a domain of employee numbers and a domain of department numbers, then the corresponding types will have nothing in common (except if the contrary is explicitly stated, of course) even if the objects of both data base domains happen to be represented by strings of digits. For example, it will be syntactically incorrect to compare for equality an employee number and a department number.

In his extensions of PASCAL, Schmidt has defined "interpreted types" [6] which permit to free the definition of a type from inherited properties of underlying types. For example, in standard PASCAL, if employee numbers and department numbers are both declared as types defined as subsets of integers, then there is no way to syntactically forbid the comparison of an employee number and a department number. Brodie [1] also proposed a similar view of interpreted types.

On the contrary, most discussion of types in the data base literature concentrate, not on the provision for supporting semantic categories, but on problems of representation and conversion (for example: the conversion of data items between a CODASYL schema and a COBOL subschema, or the dynamic conversion of objects between INGRES and the C language, the host language of QUEL, etc.)

Our approach to the relational model clearly goes in the direction of being more explicit about the semantics of the application domain. Viewing relations as predicates on objects suggests that, when designing a data base, data base domains should not be associated together in a relation if such an association does not model a meaningful semantic association in the application domain.

However, even if domains can be viewed as representing entities, the model remains relational

in that all objects are eventually modelled by values. Modelling all kinds of objects by values remains essentially less powerful than an entity relationship model where at least two kinds of objects are available (entities and property values).

For example, in the relational model, if two department numbers are equal, then this can be reasonably interpreted as meaning that both department numbers represent the same object. But, if two salary values are equal, then this can simply represent the fact that the salaries of two persons, otherwise completely unrelated, happen to be equal. The relational model is simply not powerful enough to support the difference between identity (the fact that two language constructs refer to the same entity) and the mere equality of two values.

4. References.

1. BRODIE M. L., "The Application of Data Types to Data Base Semantic Integrity", Technical Report TR-833, University of Maryland, October 1979.

2. LACROIX M., PIROTTE A., "Example Queries in Relational Languages", MBLE Technical Note N107, January 1976. Revised April 1978.

3. LACROIX M., PIROTTE A., "ILL: an English Structured Query Language for Relational Data Bases", Proc. IFIP TC-2 working conference on modelling in data base management systems, Nice (January 1977), Nijssen editor, North-Holland (1977), p. 237-260.

4. LACROIX M., PIROTTE A., "Domain-oriented Languages", Proc. International conference on very large data bases, Tokyo (October 1977), IEEE (1977), p. 370-378.

5. PIROTTE A., "High Level Data Base Query Languages", In: Logic and data bases, Gallaire and Minker editors, Plenum Press (1978), p. 409-436.

6. SCHMIDT J. W., "Type Concepts for Database Definition", In: Databases: improving usability and responsiveness, B Shneiderman Ed., academic Press (1978) p. 215-244.