88-TM

ISMM

MINI AND MICROCOMPUTERS

From Micros to Supercomputers

MIMI '88

Miami Beach, Florida, U.S.A.
December 14-16, 1988

Editor: M.H. Hamza

A Publication of
The International Society for
Mini and Microcompters - ISMM

ISBN 0-88986-107-2

ACTA PRESS

ANAHEIM * CALGARY * ZURICH

Transaction-management System in a Fourth Generation Language for Semantic Databases

Naphtali Rishe

School of Computer Science
Florida International University —
The State University of Florida at Miami
University Park, Miami, FL 33199

This paper presents a fourth generation language for semantic databases. This language is an extension of Pascal. Our implementation of the language includes an interesting transaction-management mechanism.

1. Introduction

The semantic binary database model ([Rishe-88-DDF], [Rishe-87-RM]) represents the information of an application's world as a collection of elementary facts of two types: unary facts categorizing objects of the real world and binary facts establishing relationships of various kinds between pairs of objects. The graphical database schema and the integrity constraints determine what sets of facts are meaningful, i.e. can comprise an instantaneous database (the database as may be seen at some instance of time.)

The essence of the fourth-generation data manipulation languages is the structured access to the database. (This is contrasted with earlier data manipulation languages, which provide no automatic loops to process bulks of information in the database, but only single commands to access one item at a time. As a result, the programmer was left with the responsibility of "navigating" between different data items in the database.)

This paper presents a fourth generation language for semantic databases. This language is an extension of Pascal. Our implementation of the language includes an interesting transaction-management mechanism.

A transaction is a set of interrelated update requests to be performed as one unit. Transactions are generated by programs and by interactive users. A transaction can be generated by a program fragment containing numerous update commands, interleaved with other computations. However, until the last command within a transaction is completed, the updates are not physically performed, but rather accumulated by the DBMS. Upon completion of the transaction the DBMS checks its integrity and then physically performs the update. The partial effects of the transaction may be inconsistent. Every program and user sees the database in a consistent state: until the transaction is committed, its effects are invisible.

2. The Semantic Binary Model

This section describes the Semantic Binary Model.

The semantic binary database model represents information of an application's world as a collection of elementary facts of two types: unary facts categorizing objects of the real world and binary facts establishing relationships of various kinds between pairs of objects.

The purpose of the model is to provide a means of simple natural data-independent flexible and non-redundant specification of information and its semantics.

A variant of the semantic binary model was first introduced in [Abrial-74]. Since then several modifications were published, including [Bracchi-76] and [Rishe-87-RM]. The concepts of the semantic binary model are close to those of the Functional Data Model (FDM) [Shipman-81], Semantic Data Model (SDM) [Hammer-81], and IFO [Abiteboul-84]. A comprehensive description of the Semantic Binary Model appears in [Rishe-88-DDF]. Formal mathematical semantics of the semantic binary model is defined in [Rishe-87-DS] using the methodology proposed in [Rishe-86-DN]. Design issues of semantic binary schemas are studied in [Rishe-87-RM] and [Rishe-88-DDF]. User languages for the Semantic Binary Model are studied in [Rishe-86-PS] and [Rishe-88-DDF].

One of the major advantages of the relational database model, as compared to the network and hierarchic models, was the independence of the logical *data* from the physical aspects of data storage. The semantic binary model went one step forward towards the independence of the actual *information* from its logical data representation. Among the semantic advantages of the semantic binary model relative to the relational model are the following:

- All the information is composed of the elementary facts describing the real world, so no normalization of a semantic binary schema is needed;
- No category (type) of objects needs to have a key. A key is collection of attributes which are never *null* and which universally identify the objects of the category. (Instead of having a fixed inflexible key, in the semantic model different objects of the category may be identifiable by different attributes or by different relationships with objects of perhaps other categories. In the real world, keys almost never exist.)
- Objects are not logically replaced by their keys, when these exist. So a value of a key is changeable with no influence on the other information about this object in the database.
- An object may belong to several categories simultaneously.
- Properties which are common to several categories, can be specified just once.
- It is conceptually simple and schemas can be easily explained to owners of the information to be stored in the database, who may have no computer knowledge but must approve the conceptual schema.

A definition of the model's concepts follows.

* This work has been supported in part by a grant from the Florida High Technology and Industry Council.

2.1. Categories

Object – any item in the real world. It can be either a concrete object or an abstract object as follows.

Value, or **Concrete Object** – a printable object, such as a number, a character string, or a date. A value can be roughly considered as representing itself in the computer, or in any formal system.

Abstract Object – a non-value object in the real world. An abstract object can be, for example, a tangible item (such as a person, a table, a country), or an event (such as an offering of a course by an instructor), or an idea (such as a course). Abstract objects cannot be represented directly in the computer.

This term is also used for a user-transparent representation of such an object in the Semantic Binary Model.

Category – any concept of the application's real world which is a unary property of objects. At every moment in time such a concept is descriptive of a set of objects which possess the property at that time.

Unlike the mathematical notion of a set, the category itself does not depend on its objects: the objects come and go while the meaning of the category is preserved in time. Conversely, a set *does* depend on its members: the meaning of a set changes with the ebb and flow of its members.

Categories are usually named by *singular* nouns. An object may belong to several categories at the same time.

Disjoint categories – Two categories are *disjoint* if no object may simultaneously be a member of both categories. This means that at every point in time the sets of objects corresponding to two disjoint categories have empty intersection.

Subcategory – A category is a *subcategory* of another category if at every point in time every object of the former category should also belong to the latter. This means that at every point in time the set of objects corresponding to a category contains the set of objects corresponding to any subcategory of the category.

Abstract category – a category whose objects are always abstract.

Concrete category, **category of values** – a category whose objects are always concrete. Many concrete categories, such as *NUMBER, STRING,* and *BOOLEAN,* have constant-in-time sets of objects. Thus, those concrete categories are actually indistinguishable from the corresponding sets of all numbers, all strings, and the Boolean values ({TRUE, FALSE}).

2.2. Binary Relations

Binary Relation – any concept of the application's real world which is a binary property of objects, that is, the meaning of a relationship or connection between two objects. At every moment in time, the relation is descriptive of a set of pairs of objects which are related at that time. The meaning of the relation remains unaltered in time, while the sets of pairs of objects corresponding to the relation may differ from time to time; when some pairs of objects cease or begin to be connected by the relation.

Notation: "$x\ R\ y$" means that object x is related by the relation R to object y.

Types of binary relations:

- A binary relation R is **many-to-one** (**m:1**, **functional**) if at no point in time xRy and xRz where $y \neq z$.
- A binary relation R is **one-to-many** (**1:m**) if at no point in time xRy and zRy where $x \neq z$.
- Relations which are of neither of the above types are called **proper many-to-many** (**m:m**).
- A binary relation which is both *m:1* and *1:m* (always) is called **one-to-one** (**1:1**).
- A binary relation is **proper m:1** if it is *m*:1 and not 1:1.
- A binary relation is **proper 1:m** if it is 1:*m* and not 1:1.

Domain and **range** of a binary relation:

A category C is the domain of R if it satisfies the following two conditions:

a. whenever xRy then x belongs to C (at every point in time for every pair of objects); and

b. no proper subcategory of C satisfies (a).

A category C is the range of R if:

a. whenever xRy then y belongs to C (at every point in time for every pair of objects); and

b. no proper subcategory of C satisfies (a).

92 – 95

a. an object e in the category R', and

b. binary relationships $eR_1x_1, \ldots, eR_nx_n$

3. Fourth-generation Extension of Pascal

The following **syntactic notation** is used herein.

- Language keywords are set in **boldface**.
- In syntax description templates, items to be substituted are set in *lower-case italics*.

The following is a fourth-generation extension of Pascal for structured access to databases.

1. **Global parameters** – among the global parameters of a program, such as INPUT and OUTPUT, there are the names of the database and of the userview. The database will be accessed through the userview during the execution of the program. The userview will also be accessed during the compilation of the program, in order to check for the correct usage of the names of the categories and relations and to correctly interpret the program's commands.

2. **Data type** *ABSTRACT* – a new *basic* data type, in addition to *INTEGER, BOOL, REAL, CHAR,* enumerated types, and *STRING.* (The type *STRING* is not defined in the standard Pascal, but is used, sometimes with a different name, in most practical versions of Pascal.)

   The variables of type *ABSTRACT* will contain abstract objects. (Practically, these variables will contain logical references to abstract objects. The referencing, however, is transparent to the user.) The variables of this type are called **abstract variables**.

   The abstract variables cannot be printed. They cannot receive a value through a *read* instruction. There are no constants of type *ABSTRACT*.

   Assignment to the abstract variables can be done from other abstract variables, or from the data base, or by the instruction *create* as discussed later.

   In expressions, the only meaningful operation on arguments of type *ABSTRACT* is the test for their equality. The equality test, "=", produces TRUE if the two arguments are one and the same object in the database.

3. **Extended expressions.** There are new operators which can be used in Pascal expressions:

   (i) (*expression-of-type-ABSTRACT* **is a** *category-from-the-userview*)

   This Boolean expression gives TRUE when the left-side sub-expression is evaluated into an object which is a member of the category on the right side. The membership test is done according to the information in the instantaneous database at the run time of the program.

   (ii) (*expression relation-from-the-userview expression*)

   This Boolean expression gives TRUE when the two sub-expressions yield objects participating in the relation in the instantaneous database. The types of the sub-expressions must be consistent with the relation. For example, if the relation is between abstract objects and real numbers, then the type of the left sub-expression must be *ABSTRACT* and the type of the right sub-expression must be *REAL*.)

   Instead of one of the sub-expressions, the keyword **null** may appear. Then the Boolean expression would give TRUE if the object yielded by the remaining sub-expression is related by the relation to *no* object in the instantaneous database.

   (iii) (*expression. functional-relation-from-the-userview*)

   Reminder: a functional relation is an m:1 relation. It relates every object of its domain to at most *one* object of its range.

   The expression $x.R$ produces the object related by the relation R to x, that is, the result is the object y from the instantaneous database such that (x R y) is TRUE.

   If no such object y exists, then a *null* object results, which can cause a subsequent execution-time error.

4. **Atomic database manipulations.**

   (i) **create new** *abstract-variable* **in** *abstract-category-from-the-userview*
   - A new abstract object is *created* in the database;
   - this object is placed into the specified category (the database is updated to reflect this fact);
   - [a reference to] this object is *assigned* to the specified variable.

   (ii) **categorize**: *expression-of-type-ABSTRACT* **is a** *category*

   The expression is evaluated to produce an *existing* instantaneous data base object, and this object is inserted into the specified category (in addition to other categories the object may be a member of).

   (iii) **decategorize**: *expression-of-type-ABSTRACT* **is no longer a** *category*

   The object is removed from the category.

   The object is also automatically removed from the sub-categories of the category. (Otherwise the database would become inconsistent.)

   The object is also automatically removed from the relations whose domains or ranges are categories of which the object is no longer a member. (This automatic removal saves programming effort. This removal is also necessary to maintain the consistency of the database.)

   If after the decategorization the object would not belong to any category in the database, then the object is removed from the database.

   (iv) **relate**: *expression relation expression*

   A new fact is added to the database: a relationship between the objects yielded by the two expressions.

   (v) **unrelate**: *expression relation expression*

   This has the reverse effect of the instruction **relate**.

   (vi) *expression.relation* := *expression*

   The assignment statement

   $x.R := y$

   means:
   - For every z, unrelate x R z;
   - then relate x R y.

5. The **for** statement.

   for *variable* **in** *category*
   where *boolean-expression* **do** *statement*

   The *statement* after **do**, which may be a compound statement, will be performed once for every object which belongs to the *category* and satisfies the *boolean-expression*.

   The **for** statement is *functionally* equivalent to the following algorithm.

   Let *VEC* of length L be the vector of all the *category*'s objects in the instantaneous database. The vector is arranged in an arbitrary order, transparent to the user. Then the equivalent algorithm for the **for** statement is:

```
for i := 1 to L do
    begin
    variable := VEC [i];
    if boolean-expression
    then statement
    end
```

Abbreviation:

In the **for** statement, the "**in** *category*" part may be omitted. In this case, by default the category is assumed to be a special category *OBJECT* which is regarded as the union of all the abstract categories in the database. Thus, the body of the loop will be executed for every abstract object (in the instantaneous database) satisfying the condition of the loop. Practically, the condition may explicitly or implicitly restrict the loop to one category.

6. The transaction statement.

   transaction *compound-statement*
   The effects of "**transaction** S" are:

   (i) While S is being executed, the program containing the transaction statement and all the other concurrent programs see the database in its instantaneous state just before S.

   (ii) All the updates are logically performed instantly when S is completed, provided the new instantaneous database would not violate the integrity constraints and no error-condition is raised.

   Note:
   Among the advantages of this statement is the following:

   At an intermediate state, the instantaneous information could be incomplete, which could bring failure of an integrity constraint and incorrect comprehension of the data base by concurrent programs.

   A database update statement which is not embedded in a transaction statement is regarded as one transaction.

7. **Error exit.**

   When the system fails to perform a transaction due to an error, such as a violation of an integrity constraint, it notifies the program by invoking

   procedure Transaction-error-handler (error-description: String)

   The body of this procedure can be specified in the program by the user. This allows the programmer to decide what to do in case of error. If the procedure is not defined by the user in the program, then, by default, the system will insert the following specification of the body of this procedure:

```
procedure Transaction-error-handler (error-description:
    String);
    begin
    writeln ('The program was terminated by the
        default transaction error handler when a tran-
        saction failed with the following error condi-
        tion:', error-description);
```

stop

end

4. Principles of implementation

Every real-world object in the database is user-transparently represented by a unique integer identifier. The categories (types) and relations of the schema are also treated as objects and hence have unique identifiers associated with them. Information in the database can then be represented using two kinds of tuples, denoted xC and xRy, where x is the identifier associated with a object, C and R are the identifiers associated with a category or a relation respectively, and y is either an identifier corresponding to a real-world object or a mathematical object (a number or a text string). "xC" indicates that the object x belongs to the category C. "xRy" indicates that the object x is associated with the object y by the relation R. Logically, the instantaneous database is a set of such tuples.

A completed transaction is composed of a set of facts to be deleted from the database, a set of facts to be inserted into the database, and additional information needed to verify that there is no interference between transactions of concurrent programs. If the verification produces a positive result, then the new instantaneous database is: ((the-old-instantaneous-database) − (the-set-of-facts-to-be-deleted)) $\cup$ (the-set-of-facts-to-be-inserted)

The execution of a transaction statement is composed of four stages:

1. transaction accumulation – This is a run of th program segment comprising the transaction. No updates are done physically in the database during this stage, but rather the results of the updating instructions are accumulated in the sets D (the set of facts to be deleted from the database) and I (the set of facts to be inserted into the database.) Also the results of the inquiries into the database performed during this stage are accumulated in a set V (information to be verified to validate concurrency.) The result of the transaction accumulation, i.e. the outcome (V,D,I) of the run of the program segment, is called the *accumulated transaction*.

2. integrity validation – at this time the system checks whether if the update were to be made, *i.e.* the set D deleted and I inserted, the resulting database would not violate the integrity constraints. No action update is done at this stage, but rather an algorithmic decision is made by examining the sets D, I, the constraints, and in some cases by getting some additional information from the database.

3. concurrency validation. It is verified that the results of inquires kept in V are still the same, i.e. have not been changed by the concurrent transactions. (This operation does not actually require recalculation of the inquires.)

4. execution of the accumulated transaction – the new instantaneous database is: ((the-old-instantaneous-database) − D $\cup$ I.

We have implemented this language in a prototype DBMS at the University of California, Santa Barbara ([Vijaykumar-87], [Jain-87]). Our implementation allows single-processor multi-user parallel access to the database. Optimistic concurrence control is used.

Currently, at Florida International University, we are working on a project, financed by the state government, to extend our semantic DBMS implementation into a massively-parallel very-high-throughput database machine [Rishe-88-AMPDM], to be composed of many (thousand[s]) processors, each equipped with a permanent storage device and a large cache memory.

Acknowledgment

The author gratefully acknowledges the advice of Narayanan Vijaykumar, Li Qiang, Nagarajan Prabhakaran, Doron Tal, and David Barton.

References

[Abiteboul-84] S. Abiteboul and R. Hull. "IFO: A Formal Semantic Database Model", Proceedings of ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1984.

[Abrial-74] J.R. Abrial, "Data Semantics", in J.W. Klimbie and K.L. Koffeman (eds.), *Data Base Management*, North Holland, 1974.

[Bracchi-76] Bracchi,G., Paolini, P., Pelagatti, G. "Binary Logical Associations in Data Modelings". In G.M. Nijssen (ed.), *Modeling in Data Base Management Systems.* IFIP Working Conference on Modeling in DBMS's, 1976.

[Jain-87] A. Jain. Design of a Binary Model Based DBMS and Conversion of Binary Model Based Schema to an Equivalent Schema in Other Major Database Models. M.S. Thesis, University of California, Santa Barbara, 1987.

[Rishe-86-PS] N. Rishe. "Postconditional Semantics of Data Base Queries." *Mathematical Foundations of Programming Semantics.* Proceedings of the International Conference on Mathematical Foundations of Programming Semantics, April 1985, Manhattan, Kansas (ed. A. Melton); Lecture Notes in Computer Science, vol. 239. Springer-Verlag, 1986. (pp 275-295.)

[Rishe-86-DN] N. Rishe. "On Denotational Semantics of Data Bases." *Mathematical Foundations of Programming Semantics.* Proceedings of the International Conference on Mathematical Foundations of Programming Semantics, April 1985, Manhattan, Kansas (ed. A. Melton); Lecture Notes in Computer Science, vol. 239. Springer-Verlag, 1986. (pp 249-274.)

[Rishe-87-RM] N. Rishe. "On Representation of Medical Knowledge by a Binary Data Model." *Mathematical Modelling*, vol. 8, 1987. (pp. 623-626)

[Rishe-87-DS] N. Rishe, *Database Semantics.* Technical report TRCS87-002, Computer Science Department, University of California, Santa Barbara, 1987.

[Rishe-88-DDF] N. Rishe. *Database Design Fundamentals: A Structured Introduction to Databases and a Structured Database Design Methodology.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[Rishe-88-AMPDM] N. Rishe, D. Tal, and Q. Li. "Architecture for a Massively Parallel Database Machine" *Microprocessing and Microprogramming.* The Euromicro Journal. 1988, accepted.

[Shipman-81] D.W. Shipman. "The Functional Data Model and the Data Language DAPLEX", *ACM Transactions on Database Systems*, v. *6*, no. 1, 140-173, 1981.

[Vijaykumar-87] N. Vijaykumar. Toward the Implementation of a DBMS based on the Semantic Binary Model. M.S. Thesis, University of California, Santa Barbara, 1987.