90-PC

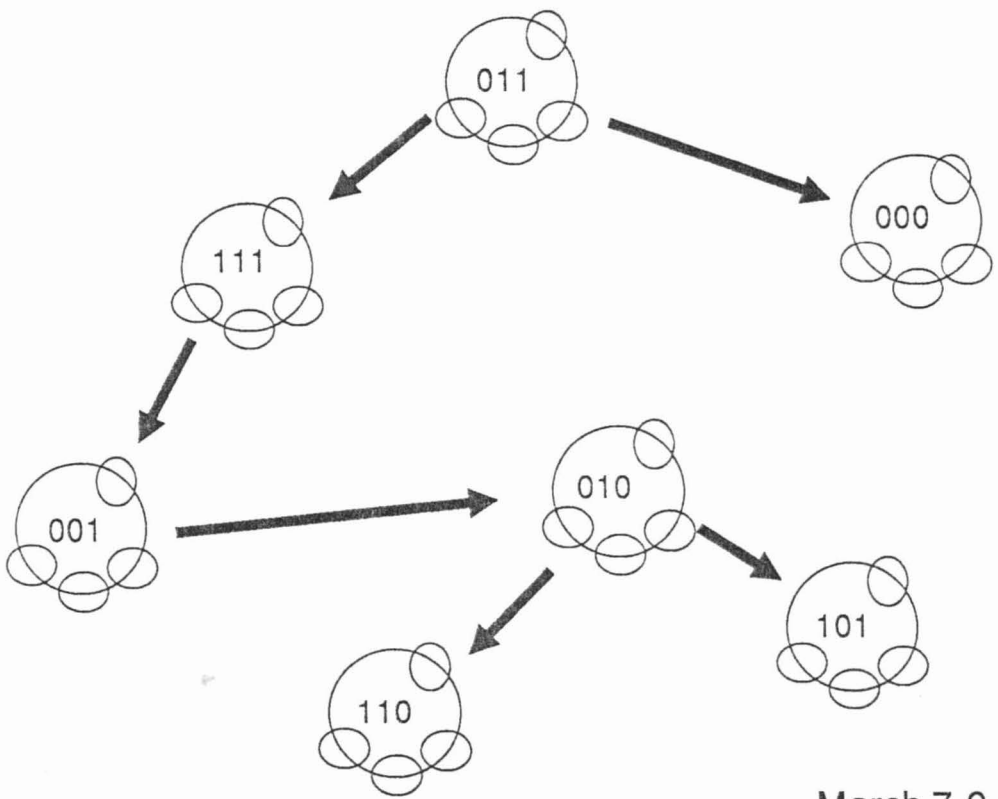
Databases and Parallel Architectures

Proceedings

PARBASE-90

International Conference on Databases, Parallel Architectures, and Their Applications

Edited by N. Rishe, S. Navathe, and D. Tal



March 7-9, 1990
Miami Beach, Florida

Sponsored by
Florida International University
in cooperation with IEEE and Euromicro

IEEE Computer Society Press

Institute of Electrical and Electronics Engineers, Inc.

A Predicate-calculus Based 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, USA

This paper proposes a non-procedural language for semantic databases and in particular for the Semantic Binary Model. The language is based on a first-order calculus.

1. Introduction

The semantic binary model [Rishe-88-DDF] 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 objects are classified into non-disjoint categories. Inheritance of properties of categories is determined by a graph of sub-categories and super-categories. 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 database aggregates information about abstract objects. Abstract objects stand for real entities of the user's world. The representation of abstract objects is transparent to the user and is unprintable. In addition to the abstract objects, the database contains, in a subservient role, concrete, or printable, objects. These are character strings, numbers, dates, *etc.*

This paper proposes a non-procedural language for semantic databases in general, and in particular for the Semantic Binary Model. The foundation of the language is a database interpretation of a first-order predicate calculus [Rishe-88-DDF]. The calculus is enriched with second-order constructs for aggregation (statistical functions), specification of transactions, parametrized query forms and other uses. The language is called SD-calculus (Semantic Database Calculus.)

Of special interest is the use of this language for specification of bulk transactions, including generation of sets of new abstract objects. This problem does not exist in the relational databases because there the user controls the representation of his objects by data, namely by key attributes of those objects. (*E.g.* persons might be represented in the relational model by their social security numbers, provided such numbers are unique, always exist for every person, and do not change with time. In the semantic models the user does not care how the persons are represented.)

Another especially interesting feature of the proposed language is the automatic intuitively-meaningful handling of null-values, *i.e.* of application of non-total functional relations.

The language proposed in this paper can also be used with most other semantic models: Abrial's Binary Model [Abrial-74], the IFO model [Abiteboul&Hull-84], SDM [Hammer&McLeod-81], SEMBASE [King-84], NIAM ([Nijssen-81], [Nijssen&VanBekkum-82], [Leung&Nijssen-87]), GEM [Tsur&Zaniolo-84], TAXIS [Nixon&*al.*-87], or the Entity-Relationship Model [Chen-76].

The examples in this paper refer to the schema of Figure 1. That schema describes some activities of a Dining Club.

The following syntactic notation is used in the definitions of syntactic constructs and in the examples: language keywords are set in **boldface**; the names of the relations and categories from the database are set in *UPPER-CASE ITALICS*; in syntax description templates, items to be substituted are set in *lower-case italics*.

2. Semantic Model Terminology

An **object** is any item in the real world. It can be either a concrete object or an abstract object as follows. A **value**, or a **concrete object**, is a printable object, such as a number, a character string, or a date. An **abstract object** is 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, offering of a course by an instructor), or an idea.

A **category** is any concept of the application's real world which is a unary property of objects. An object may belong to several categories at the same time. A **binary relation** is 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. *Notation*: "$x\ R\ y$" means that object x is related by the relation R to object y. 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 category C is the **domain** of R if it satisfies the following two conditions: (a) whenever xRy then x belongs to C; and (b) no proper subcategory of C satisfies (a). A category C is the **range** of R if it satisfies the following two conditions: (a) whenever xRy then y belongs to C; and (b) no proper subcategory of C satisfies (a). A relation R whose domain is C is *total* if *at all times* for every object x in C there exists an object y such that xRy.

This research has been supported in part by a grant from Florida High Technology and Industry Council

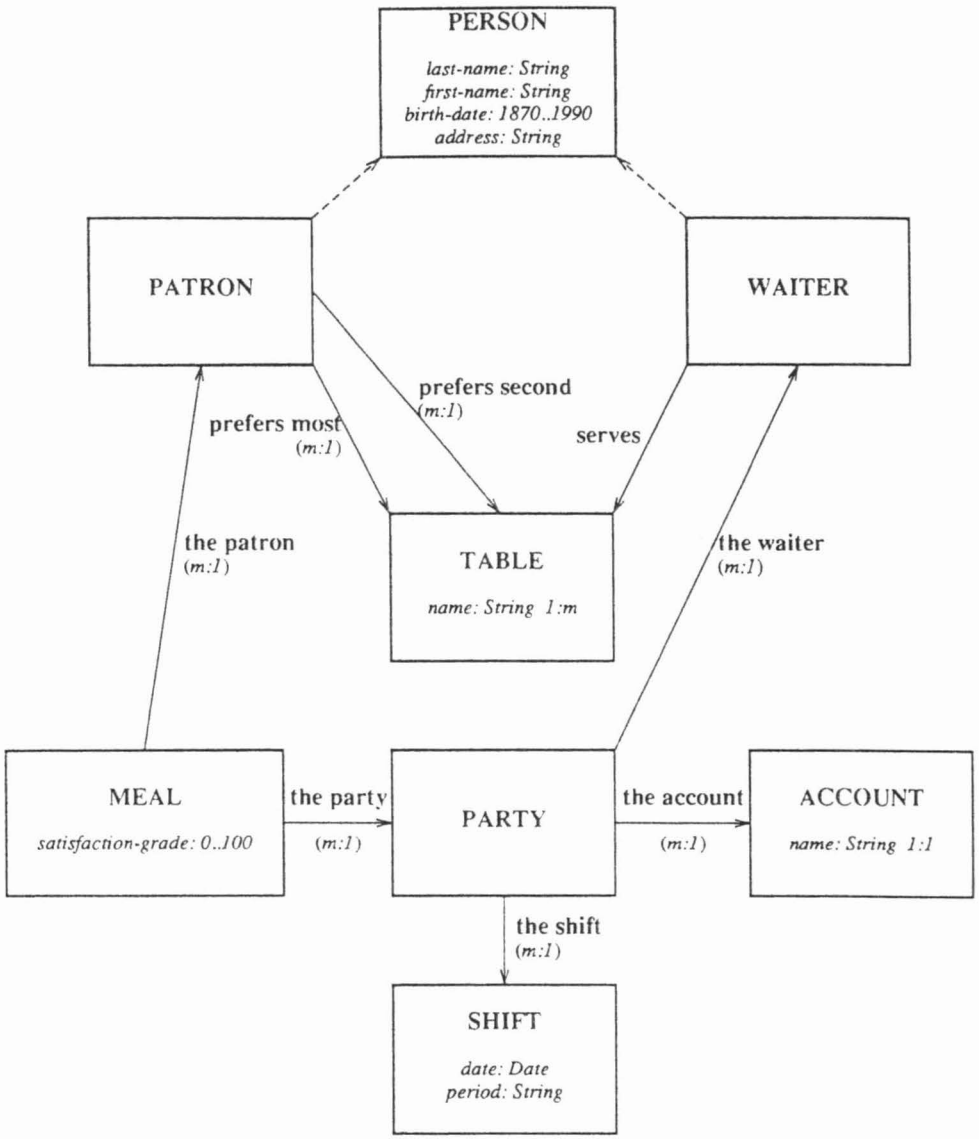


Figure 1-1. A binary schema for a dining-club application.

3. Preview of the language

Non-procedural language — a language in which the user specifies *what* is to be done without specifying *how* it is to be done.

Example 3-1.

What waiters have served every patron?

```
get waiter.LAST-NAME where
    (for every s in PATRON:
        exists meal in MEAL:
            ((meal THE-PATRON s) and
            (meal.THE-PARTY. THE-
                WAITER = waiter)))
```

4. First-order predicate calculus expressions

The First-order Predicate Calculus can be applied to semantic databases, if we regard the instantaneous database as a finite structure with binary relations, unary relations (categories), and functions (functional relations).

Expression — a combination of *constants, variables, operators*, and parentheses. The syntax and semantics are given below.

An expression may depend on some variables. When the variables are interpreted as some fixed objects, the expression can be evaluated with respect to a given instantaneous database, and will yield an object, abstract or concrete. The following are syntactic forms of expressions:

1. *constant*
   a. *number*
   b. *character-string* (in quotes)
   c. *Boolean value* (TRUE and FALSE)
   d. *Date*
2. *variable*

   A variable is a sequence of letters, digits, and hyphens. The first character must be a letter.
3. (*expression*)

   Parentheses in expressions may be omitted when no ambiguity results.
4. (*expression basic-binary-operator expression*)

   The basic binary operators are: +, –, *, /, >, <, ≥, ≤, =, ≠, **and**, **or**.

   Each operator may be used only when the expressions yield values of types appropriate for the operator. The only basic binary operators defined for abstract objects are '=' and '≠', which produce TRUE or FALSE as results.
5. (**if** *expression* **then** *expression*)

   "if e_1 **then** e_2" is equivalent to "(**not** e_1) **or** e_2"
6. (*expression relation expression*)

   The *relation* is a relation from the userview. The result is TRUE if the two objects are related by the *relation* in the instantaneous database.

   Example 4-1.

   (x *BIRTH-DATE* 1960)

   The value of this Boolean expression depends on the variable *x*.
7. (*basic-unary-operator expression*)

   The basic unary operators are: –, **not**.

   Example 4-2.

   (**not** (1>1)) = TRUE
8. (*expression* **is a** *category*)

   This Boolean expression yields TRUE when the object is in the *category* in the instantaneous database.

   Example 4-3.

   x **is a** *PATRON*
9. (*expression . functional-relation*)

   x.R is the object related by the relation *R* to *x*, it is the object *y* from the instantaneous database such that (*x R y*) gives *true*. Such an expression is called **dot-application**.

   Example 4-4.

   - x.*BIRTH-DATE*
   - e.*THE-PARTY.THE-WAITER.LAST-NAME*

   The dot-application is well-defined for total functional relations. The case of non-total functional relations will be discussed later.
10. (**exists** *variable* **in** *category* : *expression*)

    The ':' may be pronounced 'so that the following is true:'.

    The contained *expression* must be Boolean. The result is also Boolean.

    The result is TRUE when there exists at least one object in the category which satisfies the Boolean *expression*.

    The *expression* normally depends on the *variable*, but may also depend on additional variables. The resulting expression no longer depends on the *variable*.

    Interpretation:

    Let $a_1, a_2, \ldots, a_n$ be all the objects in the *category* in the instantaneous database.

    Let $e_1, e_2, \ldots, e_n$ be obtained from the *expression* by substituting each of $a_1, a_2, \ldots, a_n$ for all the occurrences of the *variable* in the *expression*.

    Then

    exists variable **in** *category* : *expression*

    is equivalent to

    e_1 **or** e_2 **or** ... **or** e_n

    Example 4-5.

    (**exists** x **in** *WAITER* : x.*BIRTH-DATE* = y)

    This is TRUE if there is at least one waiter who was born on date *y*. The whole expression depends only on the variable *y*.

    The keyword '**exists**' is called '**the existential quantifier**'.
11. (**for every** *variable* **in** *category* : *expression*)

    The ':' is pronounced 'the following is true:'.

    The *expression* must be Boolean. The result is also Boolean. It is TRUE when all the objects of the *category* satisfy the Boolean *expression*. The *expression* usually depends on the *variable*, and may also depend on additional variables. The resulting expression no longer depends on the *variable*.

    Interpretation:

    Let $a_1, a_2, \ldots, a_n$ be all the objects in the *category* in the instantaneous database.

    Let $e_1, e_2, \ldots, e_n$ be obtained from the *expression* by substituting each of $a_1, a_2, \ldots, a_n$ for all the occurrences of the *variable* in the *expression*.

Then

for every variable **in** *category* : *expression*

is equivalent to

e_1 **and** e_2 **and** ... **and** e_n

Example 4-6.

(**for every** x **in** *WAITER* : x.*BIRTH-DATE* = y)

This is TRUE if all the waiters were born in the date *y*. The whole expression depends only on the variable *y*.

The keyword 'for every' is called 'the **universal quantifier**'.

Note:

for every *variable* **in** *category* : *expression*

is equivalent to

not (**exists** *variable* **in** *category* : **not** *expression*)

Usage of variables:

The variable after a quantifier in a sub-expression should not be used outside that sub-expression. Although many versions of Predicate Calculus do not have this requirement, this requirement does not decrease the power of SD-calculus, but improves readability, prevents some typical errors in query specification, and simplifies the semantics.

Example 4-7.

WRONG:

(**exists** x **in** *PERSON*: x **is a** *PATRON*) **and** (x *BIRTH-DATE* 1970)

Here, *x* appears in the quantifier of the left sub-expression, but also appears in the right sub-expression. Logically, these are two distinct variables, and they should not be called by the same name 'x'.

To use the expressions correctly, we shall need to know what variables are *quantified* in an expression, and on what variables an expression depends.

Quantified variable: variable *v* is quantified in expression *e* if *v* has an appearance in *e* immediately after a quantifier.

Example 4-8.

The variable *v* is quantified in:

((z > 0) **or** (**exists** v **in** *PATRON*: v **is a** *WAITER*))

Expression *e* **depends on** variable *v* if *v* appears in *e* and is not quantified.

Example 4-9.

The following expression depends on *z* and *x*, but not on *y*.

((z > 0) **or** (**exists** y **in** *PATRON*: x = y.*BIRTH-DATE*))

Notation: when an expression *e* depending on variables x_1, x_2, ..., x_k is referred to (not in the actual syntax of the language), it may be denoted as

$$e(x_1, x_2, ..., x_k)$$

(In many sources, a variable on which an expression depends is called a **free variable** in that expression. An expression which depends on no variables is called a **closed expression**.)

Condition on variables $x_1, x_2, ..., x_k$ — a Boolean expression which depends on x_1, x_2,

Assertion — a Boolean expression which does not depend on any variable, that is, every variable is restricted by a quantifier.

Interpretation: for a given instantaneous database, the assertion produces *true* or *false*.

Example 4-10.

Assertion that every patron had a dinner at the club on 1-Jan-88:

for every p **in** *PATRON*:

exists meal **in** *MEAL*:

((meal *THE-PATRON* p) **and**

(meal.*THE-PARTY*. *THE-SHIFT*. *THE-DATE*=1-Jan-88))

5. Dot-application of non-total functional relations

If *f* is not total then *e.f* could be ambiguous. In order to provide a meaningful intuitive result, the dot-application '*e.f*' of a non-total functional relation *f* to an expression *e* is interpreted by the DBMS by analyzing the whole condition or assertion containing the dot-application.

Example 5-1.

Consider the following assertion which contains a dot-application of the non-total relation *BIRTH-DATE*.

for every y **in** *PATRON*: y.*BIRTH-DATE* > 1980

This assertion will be interpreted by the DBMS as

for every y **in** *PATRON*:

exists x **in** *Integer*: y *BIRTH-DATE* x **and** x > 1980

The quantification over the conrete category *Integer* is over the finite set of integers which happen to be present in the instantaneous database at the time of the expression's evaluation.

This interpretation of the dot-application of non-total functional relations can be defined formally as follows.

An expression $e.f$, where e is an expression and f is a database functional relation, is formally regarded as a syntactic abbreviation. Let $x_1, \ldots, x_k$ be the variables on which the expression e depends. For the above example, the only such variable is y.

Let ϕ be the largest sub-expression (within the whole assertion or condition) containing $e.f$ and still depending on all the variables $x_1, \ldots, x_k$, that is, none of these variables is quantified in the subformula ϕ. (ϕ may depend also on additional variables.) For the above example,

$$\phi = (y.BIRTH\text{-}DATE > 1980)$$

Let C be the range of f.

Let $\psi = \phi^x_{e.f}$. (That is, ψ is obtained from ϕ by substitution of a new variable x for all the occurrences of $(e.f)$ in ψ.) For the above example,

$$\psi = (x > 1980)$$

Then ϕ stands for:

$$(\text{exists } x \text{ in } C : ((e\ f\ x) \text{ and } \psi))$$

6. Queries

Specification of a query to retrieve a table, that is, a set of rows of values:

get *expression, ..., expression*

where (*condition-on-the-variables-on-which-the-expressions-depend*)

Interpretation of

$$\text{get } e_1, \ldots, e_n \text{ where } (\phi(x_1, \ldots, x_k))$$

The variables $x_1, \ldots, x_k$ are assigned all the possible tuples of objects from the instantaneous database which make $\phi(x_1, \ldots, x_n)$ *true*; the expressions $e_1, \ldots, e_n$ are evaluated for these tuples and the corresponding results are output. (The output is not printable if any of the expressions produces an abstract object.)

Example 6-1.

Who has been served by Waiter Smith?

get patron.*LAST-NAME* where

exists meal in *MEAL*:

(meal.*THE-PATRON*=patron and

meal.*THE-PARTY. THE-WAITER. LAST-NAME*='Smith')

Abbreviation:

Queries which output only one value may be specified without the "**where** *condition*" part, as:

get *expression*

(provided the *expression* depends on no variables).

Example 6-2.

The following is a yes-or-no query which displays 'TRUE' if every patron has eaten at least once at the club.

get (**for every** p **in** *PATRON*: **exists** meal **in** *MEAL*: p=meal.*THE-PATRON*)

Headings of output columns:

The columns in a table which is an output of ***get*** can be labeled:

get $heading_1 : e_1, heading_2 : e_2, \ldots, heading_n : e_n$ where *condition*

Example 6-3.

Print a table with two columns, which associates patrons to their waiters. Only last names are printed.

get Waiter-who-served: waiter.*LAST-NAME*, Patron-served: patron.*LAST-NAME* **where**

exists meal in *MEAL*:

meal.*THE-PATRON* = patron **and**

meal.*THE-PARTY.THE-WAITER* = waiter

When no heading for e_i is specified, then, by default, the following heading is assumed:

- if e_i ends in "*.relation*", then the heading is the *relation*;
- otherwise the heading is the number i.

Example 6-4.

The query

get x.*LAST-NAME*, x.*BIRTH-DATE* where x is a *PATRON*

produces a two-column printout with headings *LAST-NAME*, *BIRTH-DATE*

7. Other features

Other feautures of the language, beyond the limits of this short paper, include: aggregate operations: sum, count, average; shorthand notation for n-ary relationships; update transactions; parametric query forms; data definition language; specification of integrity constraints; and specification of userviews.

8. Implementation

We have implemented this language under the UNIX operating system. We have developed an experimental translator from this language into *C* with subroutine calls to our experimental semantic database management system. We have also implemented another translator for a large subset of this language under the MS-DOS operating system. The latter translator is intended for personal computers.

Acknowledgment

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

References

[Abiteboul&Hull-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.

[Chen-76] P. Chen. "The Entity-relationship Model: Toward a unified view of data." *ACM Trans. Databas Syst. 1*, 1, 9-36.

[Hammer&McLeod-81] M. Hammer and D. McLeod. "Database Description with SDM: A Semantic Database Model", *ACM Transactions on Database Systems*, Vol. 6, No. 3, pp. 351-386, 1981.

[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.

[King-84] R.King. "SEMBASE: A Semantic DBMS." Proceedings of the First Workshop on Expert Database Systems. Univ. of South Carolina, 1984. (pp. 151-171)

[Leung&Nijssen-87] C.M.R. Leung and G.M. Nijssen. From a NIAM Conceptual Schema into the Optimal SQL Relational Database Schema, Aust. Comput. J., Vol. 19, No. 2.

[Nijssen-81] G.M. Nijssen "An architecture for knowledge base systems", Proc. SPOT-2 conf., Stockholm, 1981.

[Nijssen&VanBekkum-82] G.M.A. Nijssen and J. Van Bekkum. "NIAM - An Information Analysis Method", in Information Systems Design Methodologies: A Comparative Review, T.W. Olle, et al. (eds.), IFIP 1982, North-Holland.

[Nixon&*al.*-87] B. Nixon, L. Chung, I. Lauzen, A. Borgida, and M. Stanley. Implementation of a compiler for a semantic data model: Experience with Taxis." In *Proceedings of ACM SIGMOD Conf.* (San Francisco), ACM, 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. 436 pages, hardbound. ISBN 0-13-196791-6.

[Tsur&Zaniolo-84] S. Tsur, C. Zaniolo. "An implementation of GEM — supporting a semantic data model on a relational backend." In *Proc. ACM SIGMOD Intl. Conf. on Management of Data, May 1984.*