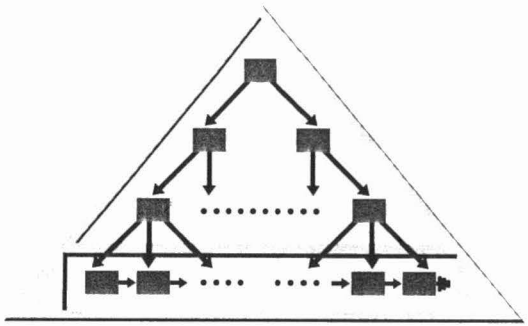
DATABASES:
Theory, Design, and Applications

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



A postconference publication based upon the proceedings of PARBASE-90, held in Miami Beach, Florida, on March 6–9, 1990

Sponsored by Florida International University

1951 1991 IEEE Computer Society Press The Institute of Electrical and Electronics Engineers, Inc.

Databases: Theory, Design, and Applications

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

A postconference publication based on the proceedings of PARBASE-90, held in Miami Beach, Florida, March 6-9, 1990

Sponsored by
Florida International University
in cooperation with IEEE and Euromicro

1951-1991

IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

A PREDICATE-CALCULUS BASED LANGUAGE FOR SEMANTIC DATABASES

Naphtali Rishe and Wei Sun

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

Abstract

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 predicate calculus enriched with second-order constructs for aggregation, specification of transactions, parametrized query forms and other uses. The language has a capability of 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 feature of the proposed language is the automatic intuitively-meaningful handling of null-values, i.e. of application of non-total functional relations.

About the authors

Naphtali Rishe is Associate Professor of Computer Science at the Florida International University. His Ph.D. is from Tel Aviv University. Rishe's expertise is in databases: design, semantic modeling, implementation, languages. He is the author of 16 papers in journals, 1 keynote paper, 24 papers in proceedings. E-mail: rishen@fiu.edu

Wei Sun received his Ph.D degree from the Department of Electrical Engineering and Computer Science, University of Illinois at Chicago in 1990. He is currently an Assistant Professor at the School of Computer Science, Florida International University, Miami. His research interests include query processing and optimization in distributed and object-oriented database systems, knowledge bases, semantic query optimization, and the interoperability in heterogeneous database systems. E-mail: weisun@fiu.edu.

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

0-8186-9165-4/91/0000/0168$01.00 © 1991 IEEE

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 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], [Verheijen&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.

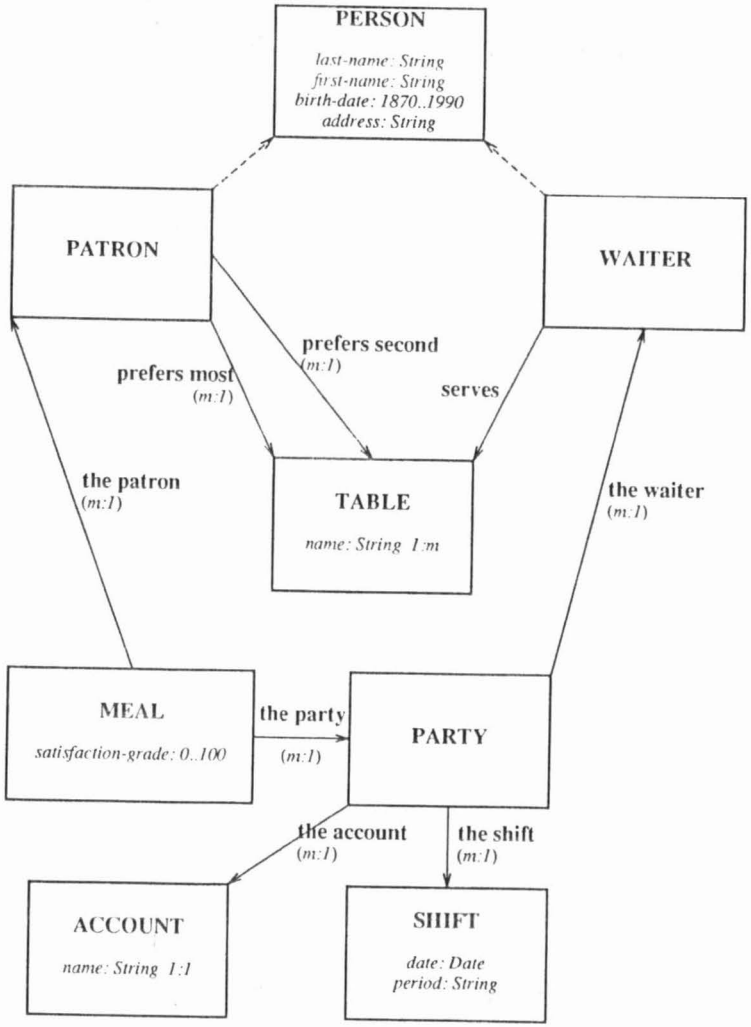


Figure 1-1. A binary schema for a dining-club application.

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*.

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)

Interpretation:

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

Let $e_1, e_2, ..., e_n$ be obtained from the *expression* by substituting each of $a_1, a_2, ..., a_n$ 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 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** 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 re the dot-application '$e.f$' of a non-total functional relation f to an expression e is interprete 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:

(exists x **in** C**:** $((e\ f\ x)$ **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

get $e_1, \ldots, e_n$ **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. SD-calculus with aggregate operations: sum, count, average

Defined here is a second-order extension to enable set operations, such as summation, counting, etc. This is done by extending the syntax of *expression* with the following **summation quantifier**. To make it clearer, we first use a common two-dimensional mathematical notation, and then translate it into the linear notation of the language.

$$\sum_{\substack{\textit{variables}\\ \textbf{where}\\ \textit{condition}}} \textit{expression}_1$$

Example 7-1.

The number of pairs (waiter, table) where the waiter serves the table.

$$\sum_{\substack{\text{waiter,table}\\ \textbf{where}\\ \text{waiter } \textit{SERVES} \text{ table}}} 1$$

(In the above formula, "1" is a constant function. Thus, *e.g.* $\sum_{i=1}^{5} 1$ equals 5. Adding up the constant "1" is thus the same as counting the object pairs satisfying the condition under the sum.)

The *variables* under Σ are quantified by the summation symbol. In addition to these *variables*, the *condition* and/or the *expression* $_1$ may depend on other variables.

Example 7-2.

The sum of the satisfaction grades given by patron *s*. The sum depends on the variable *s*, meaning *s* remains free in the sum.

$$\sum_{\substack{\text{meal}\\ \textbf{where}\\ \text{meal } \textbf{is a } \textit{MEAL} \textbf{ and}\\ \text{meal.}\textit{SATISFACTION-GRADE} \geq 0 \textbf{ and}\\ \text{meal } \textit{THE-PATRON} \text{ s}}} \text{meal.}\textit{SATISFACTION-GRADE}$$

(The sub-condition "*SATISFACTION-GRADE*≥0" is used to ensure that only existing grades are taken into account. Absent grade would fail this condition according to the algorithm of interpretation of nulls defined in a previous section.)

Interpretation:

Let e be an expression, and let $\phi(x_1,\ldots,x_n,y_1,\ldots,y_k)$ be a condition. Then the following is also an expression (it depends on the variables $y_1, \ldots, y_k$):

$$\sum_{\substack{x_1,\ldots,x_n\\ \textbf{where}\\ \phi(x_1,\ldots,x_n,y_1,\ldots,y_k)}} \left(e(x_1,\ldots,x_n,y_1,\ldots,y_k) \right)$$

When all the parameter-variables $y_1, \ldots, y_k$ are interpreted as some fixed objects, the **sum** yields a number. This number is the result of summation of the values of e computed for every tuple of objects $x_1, \ldots, x_n$ satisfying $\phi(x_1, \ldots, x_n, y_1, \ldots, y_k)$

The Σ acts like a quantifier for $x_1, \ldots, x_n$. Therefore, though the sub-expression e does depend on $x_1, \ldots, x_n$, the whole Σ expression does not. The variables $y_1, \ldots, y_n$ remain unquantified.

Alternative (linear) notation (we would not use the two-dimensional notation of Σ in a real computer language):

sum e
for $x_1, \ldots, x_n$
where $\phi(x_1, \ldots, x_n, y_1, \ldots, y_k)$

Abbreviation. When $x_1, \ldots, x_n$ are exactly the variables on which the expression e depends (that is, all x_i and none of y_i appear free in the expression e), the **for** clause may be omitted:

sum e **where** $\phi(x_1, \ldots, x_n, y_1, \ldots, y_k)$

Example 7-3.

For every patron who prefers most the Presidential Table, print his last name and the sum of his satisfaction grades.

get
patron.*LAST-NAME*,
(**sum** meal.*SATISFACTION-GRADE*
where meal *THE-PATRON* patron)
where
patron **is a** *PATRON* **and**

(patron.*PREFERS-MOST TABLE-NAME* 'Presidential')

Abbreviation for **count**:

count $x_1,...,x_n$ **where** ϕ $(x_1,...,x_n, y_1,...,y_k)$

stands for:

sum 1 **for** $x_1,...,x_n$ **where** ϕ $(x_1,...,x_n, y_1,...,y_k)$

Example 7-4.

How many patrons are there in the dining-club?

get (count p **where** p **is a** *PATRON*)

Abbreviation for **average**:

average *e* ...

stands for:

(**sum** *e* ...) / (**count** *e* ...)

Example 7-5.

What patrons give average satisfaction grade below 60?

get p.*LAST-NAME*
where p **is a** *PATRON*
and 60 > (**average** meal.*SATISFACTION-GRADE*
where meal *THE-PATRON* p)

8. Shorthand notation for n-ary relationships

Example 8-1.

Often we need to specify a condition like:

The waiter *i* worked for account *c* in shift *q*.

In SD-calculus this can be stated as:

exists party **in** *PARTY*:
party *THE-WAITER* i **and**
party *THE-ACCOUNT* c **and**
party *THE-SHIFT* q

The above statement can be written in a shorthand notation as:

PARTY (*THE-WAITER*: i, *THE-ACCOUNT*: c, *THE-SHIFT*: q)

Abbreviation:

category ($relation_1$: $expression_1$,..., $relation_k$: $expression_k$)

stands for

exists x **in** *category*:
(x $relation_1$ $expression_1$ **and**... **and** x $relation_k$ $expression_k$)

9. SD-Calculus for transactions

This section shows how SD-calculus can be used to specify transactions — creation of sets of objects, categorization and decategorization of objects, relating and unrelating objects, and so on.

A transaction in SD-calculus need not be restricted to the relationships of one object, but can work on sets of objects. One single operation can create a *set* of new objects, place them in categories, and relate them to different existing objects by *several* relations.

Creation of new abstract objects and relating them to existing or concrete objects:

insert into *category* ($relation_1$: $expression_1$, . . . , $relation_k$: $expression_k$)

where *condition*

- If no *where* clause is specified then only one new abstract object is created. This object is put into the *category* and related by the relations to the values of the expressions.

Example 9-1.

Create a new table named 'Tulip'

insert into *TABLE* (*NAME*: 'Tulip')

- Some of the names of the relations may be identical. This allows one object to be related to several objects by one relation (many-to-many or one-to-many.)
- If *where* clause is specified with a *condition* on variables $x_1, \ldots, x_n$, then for every tuple of values of the variables satisfying the *condition*, one new object is created and related accordingly.

Example 9-2.

Arrange for the Presidential Table's most preferring patron Jack Johnson to participate in the party dining at the IBM's expense and served by Waiter Smith in the evening shift of 1-Jan-88.

insert into *MEAL* (*THE-PATRON: s*, *THE-PARTY:* party) **where**

s.*PREFERS-MOST NAME* 'Presidential' **and**

s.*LAST-NAME*='Johnson' **and** s.*FIRST-NAME*='Jack' **and**
party.*THE-SHIFT.DATE*=1-Jan-88 **and**
party.*THE-SHIFT.PERIOD*='Evening' **and**
party.*THE-ACCOUNT.NAME*='IBM' **and**
party.*THE-WAITER.LAST-NAME*='Smith'

- The variables on which the *condition* depends must be those on which the *expressions* depend.

Example 9-3.

Arrange for all Presidential Table's most preferring patrons to participate in the party dining at the IBM's expense and served by Waiter Smith in the evening shift of 1-Jan-88.

insert into *MEAL* (*THE-PATRON: s*, *THE-PARTY:* party) **where**

s.*PREFERS-MOST NAME* 'Presidential' **and**
party.*THE-SHIFT.DATE*=1-Jan-88 **and**
party.*THE-SHIFT.PERIOD*='Evening' **and**
party.*THE-ACCOUNT.NAME*='IBM' **and**
party.*THE-WAITER.LAST-NAME*='Smith'

- when the *insert* statement calls for an insertion of a new object while there is already an object having the same relationships as those of the new object, the new object is not inserted.

Example 9-4.

If the table named 'Management' already exists, then the following command produces no effect:

insert into *TABLE* (*NAME*: 'Management')

Connection between existing abstract objects, between existing abstract objects and concrete objects, between existing abstract objects and categories:

connect $fact_1, \ldots, fact_k$ [**where** *condition*]

Each $fact_i$ is either

$$expression_i \quad category_i$$

or

$$expression_i \quad relation_i \quad expression'_i$$

Interpretation:

- If no *where* clause is specified, the values of the expressions are related by the *relations* and/or categorized by the categories.
- If a *where* clause is specified with a condition ϕ on variables $x_1, \ldots, x_n$, then for every tuple of values of the variables satisfying ϕ the values of the expressions are related and categorized as above.
- The variables on which the condition ϕ depends must be those on which the expressions depend.

Removal of connections and **removal of objects**:

disconnect $fact_1, \ldots, fact_k$ [**where** *condition*]

Interpretation:

- If no *where* clause is specified, the values of the expressions are unrelated and/or decategorized. Objects that are removed from all their categories, are removed from the database.
- If a *where* clause is specified with a condition ϕ on variables $x_1, \ldots, x_n$, then for every tuple of values of the variables satisfying ϕ the values of the expressions are unrelated and decategorized as above.
- The variables on which the *condition* depends must be those on which the expressions of the *facts* depend.

Correction of facts:

update $fact_1, \ldots, fact_k$ [**where** *condition*]

This is a combination of *disconnect* and *connect*. Before a connection *aRb* is made, the relationships *aRx* are removed for every *x*.

Example 9-5.

Increase by 10% all the grades given to Waiter Smith.

update meal *SATISFACTION-GRADE* 1.1×meal.*SATISFACTION-GRADE*

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

10. Query forms

Often the users ask similar queries which differ only in the values of some parameters. It is desirable that such queries be predefined in parametric form, and the the-users would supply only the values of the parameters.

Example 10-1.

What are the satisfaction grades of the patron whose name is *x*, where *x* is supplied by the end-user when the query runs?

Such a predefinition is called a '**query in parametric form**' or '**query form**'. It saves time on specification of similar queries, and allows the less-sophisticated end-users to use queries which can be specified only by more sophisticated users, such as programmers and analysts.

In SD-calculus, query forms are specified by the following syntax:

depending on *parameters*
get *expressions*
where *condition*

The condition and the expressions may depend on the parameters.

Example 10-2.

What are the satisfaction grades of the patron whose name is *x*, where *x* is supplied by the end-user when the query runs?

depending on x
get e.*SATISFACTION-GRADE*
where e.*THE-PATRON.LAST-NAME* = x

11. 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.

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.

[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.

[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.

[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.*

[Verheijen&VanBekkum-82] G.M.A. Verheijen 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.