Specification and Result Representation of Complex Navigational Queries for Object Databases

Alexander A. Bukatov
Computer Center
Rostov State University, Russia
baa@rnd.runnet.ru

Dmitry A. Zastavnoy
Computer Center
Rostov State University, Russia
dzast@oberon.rnd.runnet.ru

Abstract

We propose an advanced query language for object databases that gets facilities to specify the whole paths consisting of database objects associated with themselves by any relationships. The article includes both the introduction to high-level navigational language and the way of representation of complex queries for providing an access to the query results from host programming language. Also we figure out the implementation tasks.

1. Introduction

This article introduces an query language proposed for accessing data from object databases at high-level navigational style. We will consider the following aspects concern with our query:

- the concept of high-level navigation based on relationship linking database objects and expressed in the form of identifying expression which specify sets of relationship paths in the database (high-level or primary navigation);
- a semiformal identifying expressions definition that is proposed for the aim of identifying database object, and three kinds of expression values: partial, total and mixed ones;
- the query language along that provides the possibilities to access data stored in database objects;
- the way for representation of the query results by complex structure of flat tables. It is used for performing the secondary navigation inside it and fetching the held data from it by a host programming language; an actual procedure for database query calculation, which is based on the concept of contexts, introduced by the author.

Proceedings of the Workshop on Computer Science and Information Technologies CSIT'99
Moscow, Russia, 1999

The query language is considered from three points of view. At first, the formal definition of our language based on high-level navigation through relationships is provided. Then, the problem of the representation of the results of complex and complicated queries is in focus of our consideration. We expect to present the way for representation such results, which is convenient for accessing and manipulating data from host languages. Besides, the concept of the secondary navigation, i.e. the navigation inside this representation of a query result is argued. The last point of view covers implementation tasks.

Our general approach to data retrieving by the query expression is the following: a query expression specifies an image of some part of database. This image is being kept by in accordance with the navigational origin of the data.

The preliminary results of our investigations have been presented at works [1,2,3,4]. The work [4] proposes an example of applying our approach to data modeling for CASE systems. The discussion of problems concerned with object query languages can be found in [5,6] as well as in notable work [7], that supposes the similar approach. The descriptions of existing standard (ODMG 2.0) and the most well known object systems, such as Postgres, are published in [8,9].

2. The extensional graph concepts

The native and wide spread way for visualized representation of the database state is based on the notion of an extensional graph. The extensional graph, by our suggestions, is to be basis for the investigations of the advanced query language for object databases, because it provides the most suitable manner for user's cognitivity of associations between database objects and expressing complex database queries.We consider the extensional graph consists of nodes and directed hedges between them, which are corresponded to database objects and relationships instances respectively. The nodes are labeled by the names, each of them is formed by the pair <object type, object name>, and hedges are

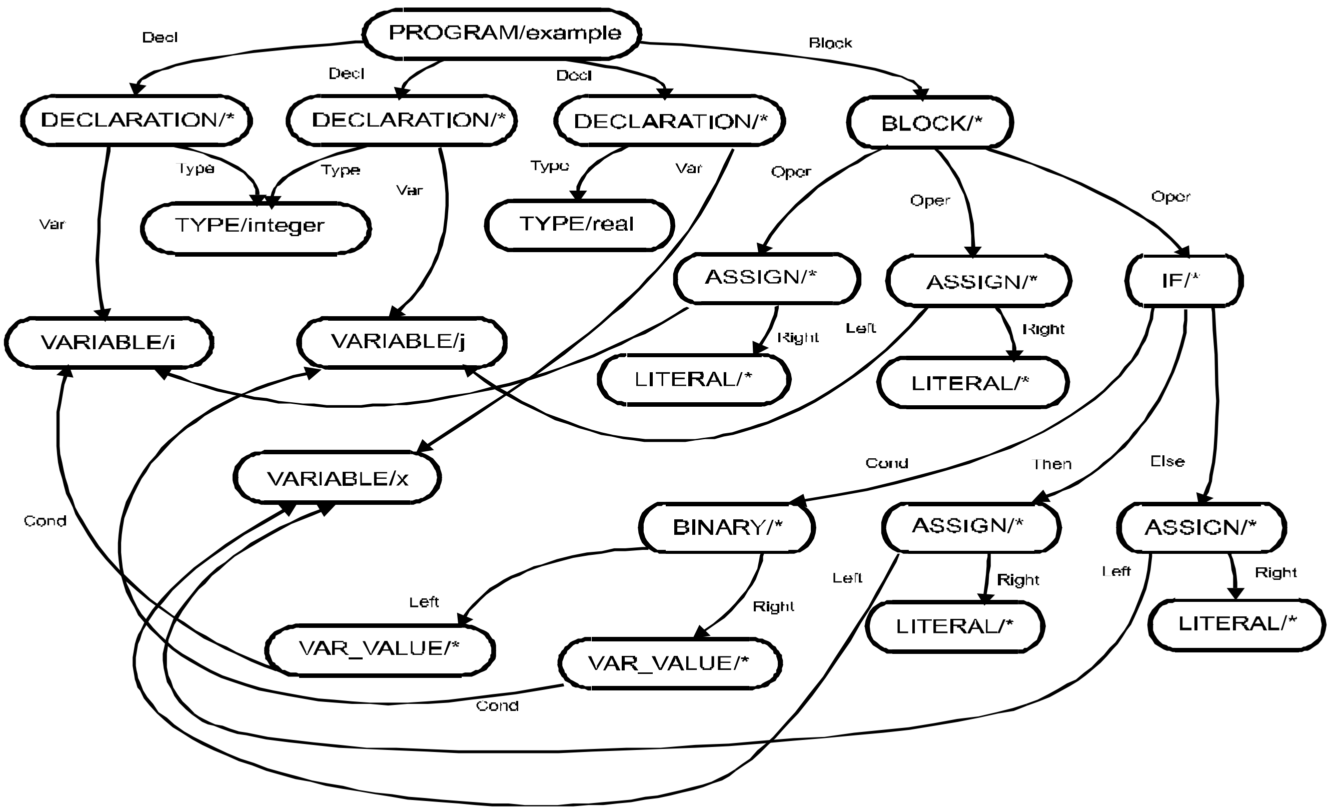

Figure 1. The example of extensional graph

labeled by relationships names. Any hedges are directed from the objects which is introduced as the owner of relationship instance to the target object.

The Figure 1 shows an example of the extensional graph for the database that holds the representation of the program shown on Figure 2. These examples were primary discussed at work [4].

```
program example;
        var     i , j : integer;
                x : real;
begin
        i := 0;
        j := 1;
        if ( i && j )   then    x := 2.3;
                        else    x := 5.7;
end.
```

Figure 2. Simple program

We can see on Figure 1, some nodes are labeled by asterisks instead of their names; it means these objects do not have any specific names in the application domain, and they are anonymous. Nevertheless, all anonymous objects are distinguished among themselves from the viewpoint of their identifications.

A simple path in extensional graph is introduced as the list, which consists of node names and hedges names linking these nodes. The name of the object type are involved for the fist object in this path only, because we suggest that each type name can be determined by knowing the name of previous type and the relationship name in the list ([3]). If the relationship names are out of our considerations, these names can also be missed in formal notations.

Some examples of simple paths in extensional graph shown on Figure 1 are suggested below.

IF/* ° BINARY/* ° Left/* ° VARIABLE/i

IF/* ° BINARY/* ° Right/* ° VARIABLE/i

IF/* ° ASSIGN/* ° VARIABLE/x

etc.

It is obvious, the fist and the second paths are very similar to each other. In the other sections we define the query language that provide the means for specifying the sets of objects by specifying the set of simple paths which are unified by the similarity of they structure. Also it will be proposed how such sets can be represented, stored and accessed by means of host programming languages is the most convenient manner.

2. Identifying expressions and queries

In this sections we define the structure and the values of identifying expression in semiformal manner. The identifying expressions, or chains, are the bases for our query language. We restrict the score of our consideration by dealing with the expressions, which include the navigation through relationships. The other operations of the language are union, filter, reverse navigation, and getting collection values, which are described at [3]. We consider the general case of identifying expression definition, which is detailed in the real-system query language.

The identifying chain is an expression

$T\bullet L_1\bullet L_2\bullet\ldots\bullet L_N$, (*)

where T is the objects type name, L_j are the relationships names, $j \geq 0$. Than, j-level of the chain (*) is defined as Ò chain for j = 0, and L_j for $j > 0$. Left j-subchain of the chain $T\bullet L_1\bullet L_2\bullet\ldots\bullet L_N$ is introduced as $T\bullet L_1\bullet\ldots\bullet L_j$ expression, and the right j-subchain of the same chain is $L_{j+1}\bullet\ldots\bullet L_N$ expression.

Partial value of the chain

It is assumed there are two sets, which are associated with each database type. The first is the set of type instances, i.e. database objects of this type. It is denoted by *ext*(T) notion. The second set denoted by *int*(T) consists of all relationship names which are defined for the objects of the type.

The value L(o), or the instance of L relationship of o source object, is the set of objects, which are named as target objects. So, if $o \in$ *ext*(T), $L \in$ *int*(T), than L(o) is $\{o'_1,o'_2, \ldots ,o'_K\}$.

The partial value of the chain is defined by the following rules:

Val [T] = ext(T) (one-level chain)

$\mathrm{Val}\,[\,T\bullet L_1\bullet L_2\bullet\ldots\bullet L_N\,] = \bigcup\limits_{o\in \mathrm{Val}\,[T\bullet L_1\bullet L_2\bullet\ldots\bullet L_{N-1}]} L_N(o)$ (multi-level chain)

It should be noted, the group of objects forming the value of multilevel chain is not the set in formal sense because it can include the duplicates of objects, therefore $\cup$ symbol of union operation denotes the including the same objects into new "set" and summarizing their multiplicities. The multiplicity of an object in a set is equal to the number of its duplicates. It is useful to note sets with the grouped duplicates: $\{o_1^{k1},o_2^{k2}, \ldots , o_n^{kn}\}$, where k_j is the multiplicity of o_j object in the set.

Total value of the chain

Now the definition of the total value of a expression is given. The simple (separated) expression is defined as the $T/n^\circ L_1/n_1{}^\circ L_2/n_2.\ \ldots\ {}^\circ L_N/n_N$ expression, where n_j are object names. It is obviously, the separated expression can be interpreted as the notion of any simple path in an extensional graph. At the matter of fact, separated expressions are values.

We define the total value of the chain as a set of simple expression, which are the simple paths generated by this identifying chain. The set of these simple paths is the generalized path in the extensional graph.

Mixed value of the chain

Let us consider $T\bullet L_1\bullet L_2\bullet\ldots\bullet L_N$ chain, and the partial value is calculated for its j-left subchain; its partial value is kept in $\{ o_1^{k1}, o_2^{k2} , \ldots , o_n^{kn}\}$ result set. Now we can evaluate the total values of $o_j\bullet L_{j+1}\bullet\ldots\bullet L_N$ for each o_j^{kj} object of the set. The union of all values calculated by this way forms the j-mixed value of the chain.

The value of j-level of a chain is defined as the partial value calculated but not grouped for left j-subchain.

While o_j^{kj} objects can have their multiplicities, the $o_j\bullet L_{j+1}\bullet\ldots\bullet L_N$ expressions inherit the multiplicities of o_j^{kj} objects. We should keep in mind the following properties:

$o_j^k\bullet L_{j+1}\bullet\ldots\bullet L_N = (o_j\bullet L_{j+1}\bullet\ldots\bullet L_N)^k$,

and for a separated expression:

$(o_j{}^\circ o_{j+1}{}^\circ\ldots{}^\circ o_N)^k = (o_j{}^k){}^\circ (o_{j+1}{}^k){}^\circ\ldots{}^\circ (o_N{}^k)$

Database queries

A database query is an extended identifying expression that includes lists of the object field names, which are required:

$T : (f_1,\ldots,f_K)\bullet L_1 : (f_{11},\ldots,f_{K1})\bullet\ldots\bullet L_N : (f_{N1},\ldots,f_{KN})$ (**)

The partial, the total and the mixed values of extended expressions are defined by the same way. For example, the total value of the extended expression (**) can be denoted as the group of all extended simple paths:

$\{ o:(f_1,\ldots,f_K){}^\circ o_1:(f_{11},\ldots,f_{K1}){}^\circ\ldots{}^\circ o_N:(f_{N1},\ldots,f_{KN}) \}$

All queries can be divided into two groups: the queries which consist of the specification of object field names at the last level only, and the queries, in which the field lists are at the any levels. The queries of these groups are introduced as the reduced and the complex queries correspondingly.

In addition to field names, the queries can include object function names. These object functions are intended to perform some data processing. Unfortunately we have too many space to provide the consistent and complete description on this matter.

3. The representation of query results

Now let us consider the ways of representation of data that are specified by expressions of our query language.

It should be noted the ways that are involved to represent the values of queries of the reduced and the complex kinds are different. Than, our consideration is based on the assumption that only data fields are required. If all of the object names of all paths are required, the total values must be calculated for each query.

Performing the reduced query is provided by calculating the partial value of the query chain. In other words, the result of the query execution consists of object names, their multiplicities and field data being specified in the query. The result is simply represented with the flat two-dimensional tables. Any access to data stored in it and the data manipulation can be provided by the simple and powerful technique of cursor that is well-known in relational technology.

At first glance, there is a simple and obvious way for the representation of complex navigational queries with the n+1 – length chain. It is provided by the two-dimensional table, that consists of n+1 columns and the rows; each of them holds one simple path. However, even the simplest example of the cluster-like relationship structure shown on Figure 3 demonstrates the evident deficiencies of such representation. At the matter of fact, the generalized path on this Figure is formed by the { $o \circ o_1$, $o \circ o_2$, $o \circ o_3$, $o \circ o_4$ } family of simple paths; and their representation with a flat table is turned out to be unnormalized in the sense of the relational database theory; and it causes the well-known troubles with inconsistency, etc.

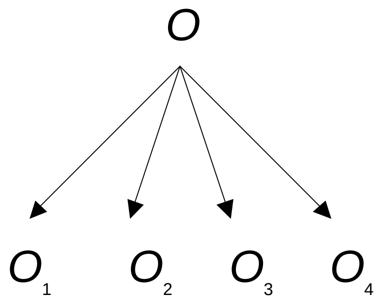

Figure 3. Cluster-like structure of relationships

The last example forces us to suggest the other method of representation of complex navigational queries at the manner that reflects the navigational nature of the queries more suitable and carefully. By other words, each path that forms the result of the query must maintain its path properties in the low-level representation of the query result. We should keep in mind that the representation holds the result specified by a high-level set-oriented query expression; but further this representation is accessed and manipulated with low-level value-oriented means of the modern programming languages. It is quite important for developing interface from a host language to the database system.

We do not intend to describe such interface ad-hoc, but it is necessary to introduce the basic data structures required for the complex query result representation, and methods of access.

Now the concept of secondary navigation is proposed. While the primary navigation is intended for the aim of specifying data at high-level manner, the secondary navigation is required to manage access to the internal representation of the query result. It can be noted the simple kind of secondary navigation is applied implicitly at the relational technology as the cursor technique.

From the implementation point of view, the definition of query values provided at previous sections has a lack; it is caused by its indetermination. Therefore the notion of simple paths should be made more precise.

Let’s describe the rules for representation of values of complex queries:

1. the value of each level is stored in a flat table that consists of object names, their multiplicities and data of fields;

2. the current row is directed in each table with a pointer that can be moved sequentially from one row to other;

3. the left and right neighbor objects are pointed for each object stored in the table;

4. these pointers to neighbor objects can be moved through left and right objects to look over all objects which are target objects (right neighbors) or which are the source objects (left neighbors) for the current object of any given level.

If the pointers to neighbors are stated automatically, the current path will be stated in the representation. Moreover, any movement of the pointer to current object or the pointer to current neighbor in any level will form a new path in the representation.

4. Contexts and context queries

Now let us describe the aspects of the implementation of the query language proposed at previous sections. Our investigation is based on the notion of contexts.

The context is a data structure, which holds the result set that is formed by result of any level of a query with the pointed current objects. In other words, the result set of any context keeps the flat table of data. The contexts are managed by an interpreter of queries. There can be a few contexts managed by the interpreter, but only one may be the current. Interpreter’s contexts form a stack, so each upper context “remembers” its lower context.

Now let’s consider the procedure of execution of the query of general case, i.e. $T \bullet L_1 \bullet L_2 \bullet \ldots \bullet L_N$. It is no matter, weather the chain is extended one or not.

The calculation procedure starts from processing T level. This level is evaluated as context-free; it means if there is any current context holding any result set, than this context is destroyed (dropped from the stack) and replaced with the new context. The new context is to store in itself the result set for objects of T level.

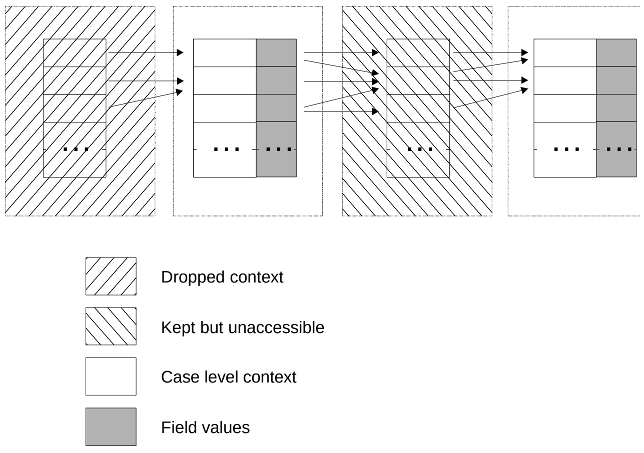

Figure 4. The contexts for extended chain

The other levels are calculated as context ones: each of them requires the data from its previous context. Let's assume the current context holds the objects of j-l level of the chain, and o_i is the current objects in the context. Objects of the next level are evaluated by the following way. All o_i objects are looked through, and the values for all $o_i \bullet L_j$ expressions are calculated and placed into the new result set of a new context. After level evaluating this new context becomes the current one. The sequence of j+1 steps described above forms the direct step of query execution. The next part of the execution is named as a reverse step, it will be discussed later. After all levels of the chain are calculated, any context laid on the stack can be accessed for fetching object data consequently. Besides, the left and right neighbor objects are set to form the whole path. In addition to looking through the own objects stored in the own context's result set, it is possible to reset the pointers from one current neighbor object to another.

There are some differences between calculations of partial and total values. At first, while the total value is formed of the set of objects only, therefore no need to keep all lower contexts. But the multiplicities of objects must be calculated, so grouping objects is involved. When calculating total values of chains, all contexts must be kept; moreover, forming paths requires storing identifiers of the left neighbor object for each object; in other words, the object should "remember" its left neighbors. At last, if an extended chain is to be calculated, the contexts for the levels which does not include the field list should not be accessible, but they cannot be removed from the stack, because they are required for the forming the parts of paths for the other levels. The Figure 4 shows the contexts generated for the query extended by the lists of fields for second and forth levels.

When the common value is calculated, the reverse step of the procedure must be processed. We should keep in mind the direct step forms the unproductive paths also, which are the values of any left subchains, but are not the values of the whole chain. In other words, these paths are short ones, and therefore the objects which form they must be removed from result sets of all contexts starting with the upper context to the lower. If an object from any context (except the last context) does not have any right neighbors, it is removed from the result set.

Context queries

The queries introduced in Section 2, are the global ones because such queries identify database objects absolutely. Neither the formal definition nor the real procedure does require any information on objects identified by other queries. But sometimes it can be useful to identify objects that continue the paths calculated before. By other words, database objects can be specified by relation with known-yet objects. In order to provide such facility, we propose the notion of context chains. The context chain is any right chain. Let us assume C chain is the $C_L \bullet C_R$ composition of two chains. If C_L is calculated and its last level forms $\{oj^{kj}\}$ result set, than C_L chain is called the context chain which context is generated by any o_j object from the result set; the value of this context chain is introduced as a value of $oj^{kj} \bullet C_R$ expression.

The definition of the context chain notion does not require an existence of any left subchain, but it involves the result set from which objects the navigational paths are continued. Besides, the type matching is intended for correctness of query performing, but the description of type aspects including type resolution during query execution is out of our presentation caused by space limitation.

At the matter of fact, the calculation of the whole $T \bullet L_1 \bullet \ldots \bullet L_N$ chain can be divided in N+1 steps. At first, T chain is calculated. Then, the context chains for all Lj level are calculated step-by-step. All objects that form the result paths for the chain will be identified and stored in the stack of contexts. But the objects, which lay on the unproductive paths, will not be removed from the lower result sets because reverse step of the procedure is not provided.

The most significant application of the context chains deals with recursive chains and subqueries evaluation. A subquery is the query that is called inside the body of any object function. Some functions are applied to current objects of the context, and the usage of context chains gets the possibility to identify both the current object along and all objects on incident paths.

5. The implementation

This section figures out the main features of the implementation of the system that provides the possibilities suggested above.

A simple prototype system, that realized some of introduced concepts, has been implemented by the author. Now more complete system is under implementation; we hope that system will provide all of the facilities.

Our system is an interpreter of commands, which include:

- objects type definition commands;
- objects manipulation commands (adding, dropping objects, etc);
- queries;
- the sources for function bodies and stored modules (i.e. the unions of commands stored in the database and retrieved from it for execution);
- commands for context management and fetching information from their;
- other commands.

The architecture of the system is the following. The interpreter is implemented at the top of relational system ORACLE for Sun platform. All low-level operation for data representation, storing and retrieving is performed by the means of SQL language.

The most significant and interesting features of the implementation are the implementation of query calculation and the representation of the query result.

The query execution includes two basis functions.

Type resolution whose aim is to determine the type of database objects which are to be retrieved from database. The problem of type resolution is faced if all type names are not specified in query source and therefore the type names must be evaluated before data access. Such circumstances can be taken place when untyped objects (which are identified by their names only) or reverse relationships are specified. The authors' investigations on this matter are briefly described at [3] work. In general, the type resolution problem is not trivia and requires additional researching.

When object type are determined data are retrieved from relational database by generating and executing the SQL queries. This procedure is simple, so it is no need to describe it.

The representation of query result is provided by the following way. Each level of paths is stored in relational table that consists of object names, object internal identifier, values of specified fields, and the identifier of the objects which is the left neighbor of given object. After complex query execution each context from the stack of contexts can be become as the current context and the current object in its result set are pointed. In the other contexts the current objects can be set to form the whole path by the different ways: either automatically or by user with some commands.

Forming path and getting neighbor objects are performed by relational queries which force the joins between neighbor objects stored in the result sets.

6. Conclusion

In this paper we have not described some features of suggested query language and data modal. At first, the precise notation of object types is omitted due to space limitation as well as data manipulation language. Besides, we do not discuss the concepts of object functions, the interface with host languages and the commands for manipulating contexts. At last, we propose the concept of stored query result sets, but any details of these concepts are out of our current presentation.

Some of the omitted details were presented in the previous works, and others require an additional investigation. Authors hope, they will be described in future papers.

References

1. Zastavnoy D.A.. “High-level Declarative Query Language for Object-Oriented Database”. In: “*Advances in Databases and Information Systems*. St.Petersburg, 1997, V.2,pp. 26-28.

2. Bukatov A., Zastavnoy D.. “Programming Support System for Supercomputer nCube based on Object-Oriented DBMS” (in Russian). In: “*Mathematics Modeling and Information Technologies*”. Kislovodsk. 1997.

3. Zastavnoy D.A. “A Concepts of Polymorphism for High-Level Query Language for Object Databases” (in Russian). In: “*Computer Technologies for Engineering and Management”*. Taganrog.1997.

4. Bukatov A., Zastavnoy D.. “High-level Navigational Language for Querying Complex Data Objects and its Application to CASE Systems”. In: Pavol Navrat. Haruki Veno (ed) “*3th Joint Conference on Knowledge-Based Software Engineering”*. Pp. 103-107. (IOS Press, Amsterdam. 1998)

5. Bertino B., Negri M., Pelagatti G., Sbattella L.. “Object-Oriented Query Language: The notion and the Issues”. *IEEE Transaction on Knowledge and Data Engineering*. 1992; 4.

6. Won Kim. A Model of Queries for Object Oriented Databases. In Proceegings of VLDB. Amsterdam. 1989.

7. Barclay P.J., Kennedy J.B.. “A Conceptual Language for Querying Object Oriented Data”. In Proceedings of BNCOD-94.

8. Cattell R.G.G, Barry D.K. (ed) “The Object Database Standart: ODMG 2.0”. Morgan Kaufmann publ. San Francisco, 1997.

9. Stonebraker M., Kemnitz G.. “The Postgres Next-Generation Database Management System”. *ACM Communications*, 1991; 34:27-38.