MODELING AND QUERYING RECURSIVE DATA STRUCTURES I: INTRODUCTION

J.H. ter Bekke and J.A. Bakker
Delft University of Technology
Faculty of Information Technology and Systems
e-mail: {J.H.terBekke, J.A.Bakker}@its.tudelft.nl

ABSTRACT

Present recursive applications for recursive data structures require complex software packages; they cannot be specified in a declarative query language. Recursive queries are complex when a kind of data modeling is applied that emphasizes variable relationships instead of definite and inherent (structural) relationships: users must specify processing details as navigation and iteration.

Another kind of modeling supports definite relationships; an example is the relational model. Although this kind of modeling makes control statements in queries superfluous, the relational model still creates problems for end users because relationships between tables cannot be specified inherently; they are specified by relationships between key attributes. Consequently, users interested in data from diverse tables have to specify join operations, which offers opportunities for semantic errors such as joining over non-key attributes.

Also semantic data modeling is based on definite relationships, but contrary to the relational model it enables us to specify data structure in an inherent way. As a consequence join terms are superfluous: processing details can be derived by a software system interpreting semantic metadata.

Using family trees as an example, we compare the consequences of the two categories of data modeling mentioned above for the specification of non-recursive queries, whereas the following paper [21] will show that inherent specification of data structure is also a fundamental prerequisite for the declarative specification of recursive operations. The resulting processing is reliable and efficient as can be demonstrated by a working DBMS.

KEYWORDS

Query language, recursive data structure, recursive query processing, semantic data modeling, metadata, end user computing.

1. Introduction

A collection of objects O_1, O_2, ….. has a recursive definition when O_1 is defined on known concepts and the definition of O_{n+1} is based on the definition of O_n for positive n, so when the predecessor occurs as a property (or attribute) in the definition of each successor. Recursion can occur in data structures and in database applications as well. The present paper focuses on recursive data structures. We think that applications (recursive and non-recursive as well) should be specified as declarative queries, because that probably is the best way to support ad-hoc querying by end users.

Recursive data structures, for example simple hierarchies in family trees, have always played an important role in database management systems [7]; they were driving forces in the early stages of developing database management systems. However, the modeling of recursive data structures has been a major problem for many years. For example, in classical hierarchical models (i.e. models based on 1:n relationships) certain modifications must be carried out to represent a recursive data structure, which resulted in a complex way of querying recursive data structures. This will be discussed in more detail in section 2.

In general, classical hierarchical models require complex computer programs including functional recursion, nesting, navigation and iteration [5]. However, the simpler (linear) relational model, in which modeling with self-reference is allowed, does not offer practical query language solutions for recursive applications [5, 8]. Also here computer programs are needed for recursive applications with the same properties as before. Further, an extension of the relational concept with nested (Non First Normal Form) relations did not provide a solution to the problem [9, 13]. More recently, certain forms of recursion were proposed in the area of relational databases. Draft SQL3 [22] contains extensions by which a limited class of recursive problems can be formulated. These solutions are derived from recursive Datalog rules [6], but are difficult to be understood or specified by end users. Another disadvantage is that the user must be fully aware of and is responsible for the finiteness of the

recursion process. Later on, extended and adapted hierarchical models were introduced again for certain applications [1, 3]. Although they sometimes could offer faster solutions than relational systems, they still could not support all kinds of queries efficiently. Despite these diverse approaches, a fundamental breakthrough for recursive queries seemed to be unattainable. Although not proven, the general opinion in the field was that only complex programs could offer a solution for recursive queries.

It is our objective to demonstrate that simple query solutions, also for recursive queries on recursive data models [21], are possible when next to nesting, navigation and iteration also other procedural aspects as explicit join terms and ordering do not occur in query specifications. The relational model enabled us already by using simple structures to make iteration and navigation superfluous in declarative query specifications, but recursive applications require more from the semantics of a data model: in order to avoid procedural details in query specifications, software must be able to derive the required processing from a declarative query and guarantee process termination. The more recent development directed towards hierarchical models seems to be incompatible with these requirements.

Data structure and data processing can have different characteristics. It is possible that a data structure is recursive but the application is not, and vice versa. Earlier papers discussed recursive applications related to non-recursive data structures represented by graphs: solutions for critical path analysis [18], bill of materials applications [19] and longest path calculation [20] were presented. The last paper also describes how declarative recursive queries on graph data are translated into a well-ordered finite procedure. The present paper only discusses applications for simple recursive data structures, but a more complex recursive model for version management based on semantic concepts was presented earlier [16]. A further restriction is that we only discuss approaches in which a database is described on a conceptual level using a certain data model. Database applications should offer solutions without irreversibly changing the current contents of the database by query execution; therefore we do not discuss logic databases in which facts and rules together define the database [2, 11]. We also require that all data occurring in a database satisfy certain structural patterns and satisfy certain semantic constraints, which are determined by the underlying data model (for example hierarchical, network, relational or semantic model).

This paper is organized as follows. First, we discuss some data modeling alternatives for recursion. We distinguish two categories of modeling: approaches emphasizing variable relationships of objects (section 2) and approaches only allowing objects having definite relationships (section 3). The kind of data modeling is of crucial importance because it determines the opportunities for querying (section 4). Because the determination of the essential semantic relationships is decisive, no compromise whatsoever should be made there. Some examples of non-recursive queries will be specified using a recursive data structure for family trees. These examples will demonstrate that only definite relationships should be used in order to prevent pitfalls and misinterpretations. All applied semantic concepts have been implemented and extensively tested using the Xplain DBMS [17], version 5.7.

2. Modeling approaches emphasizing variable relationships

Many data models are based on view modeling. Examples are models allowing nested hierarchical structures. These models are similar in the modeling of recursion using 1:n relationships. This occurs for example in well-known hierarchical, network and object-oriented data models. A model with nested hierarchical structures (for example an object oriented model [1]) may allow the following data for a recursive data structure. Here the name and birth year of persons are represented through definite relationships, but the relationship between parent and children is defined with a hierarchical, thus variable, relationship. The following content is an example:

(Peter, 1963, {(John, 1988, {}), (Anne, 1990, {})})
(Susan, 1964, {(John, 1988, {})})
(Karen, 1965, {(William, 1989, {})})

This example illustrates a number of shortcomings of a nested data structure:

- Completeness and consistency of the data is unclear: A database system requires a closed world assumption; only facts are registered. No conclusion can be drawn from the absence of data; this absence can be accidental, but also be intended.
- The structure contains an undesirable asymmetry: E.g., Peter and Anne are differently modeled. It is easy to determine the children of a person, but it is difficult to determine the parents of a person: this requires software with nesting, navigation and iteration. The result of this determination is difficult to interpret.
- Essential relationships are missing and cannot be derived: From the structure alone it is unclear how many parents a child has. A child can have two parents (see parents of John), can have more parents, but also have one parent (see parent of William and Anne), or even zero (as Peter, Susan and Karen).
- Complex update properties: When a person becomes parent (here for example John), then the structure of others (here Peter and Susan) must be adapted.
- The structure contains redundancy: For example, the birth year of John occurs several times in the structure.

The consequences of an emphasis on variable relationships can be made visible. An example is the modeling of recursion according to the CODASYL network approach [4, 5, 7]. Figure 1a contains a simple recursive structure in a collection of persons. Each person can be parent of a number of children. This recursive 1:n relationship, modeled by a set type 'parent' in which 'person' is both member and owner record-type, is not allowed in many implementations of CODASYL networks [5].

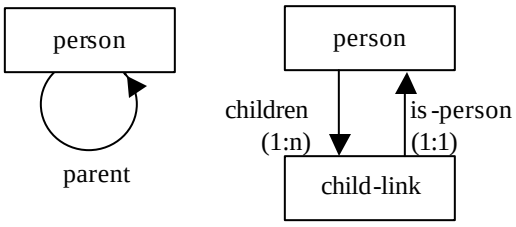

Figure 1. CODASYL recursive set (a) and implementation (b)

The required modification is mainly caused by the interpretation of the relationship between persons. This relationship is conceived as being 1:n, in other words with 1 person (the parent) correspond n other persons (the children). This interpretation, a presentation of an access path (an arrow) in the view, implies implementation problems that can be solved by the introduction of an additional link-record-type 'child-link' (figure 1b) not containing any data fields together with two set types defining the relationship between person and children and between child and person.

The CODASYL data manipulation language is based on programming concepts and therefore unsuitable for end users. The required model modifications deviate from the proper definition of recursion given before. Therefore recursive applications will inevitably become very complex programs. These problems become even worse when the relationship with two parents must be registered.

3. Modeling approaches only allowing definite relationships

The relational data model is the first data model that breaks with the tradition of nested structures: only linear structures are allowed. In this model only definite relationships (attributes) of an object are allowed. The earlier mentioned variable relationship can be considered as a derivation of this fundamental relationship. This approach enables us to specify declarative queries without nesting, navigation and iteration. A relational specification of recursive applications could be based on the following definition:

relation person
(name, birth_year, father, mother)

The relationship between parent and children is not defined by means of inherent metadata, but by primary and foreign key specifications (subset constraint specified by a REFERENCES statement in a CREATE TABLE statement)). Although the name of an attribute may be identical to the name of a relation, relational concepts do not allow us to specify recursive relationships in an inherent way as suggested by the following definition:

relation person
(name, birth_year, father_person, mother_person)

Consequently, recursive relationships are difficult to detect. Primary and foreign key values refer to attribute values. They do not refer to structures (i.e. the relation 'person'). The relation is therefore according to the earlier definition not a proper recursive definition, which of course has consequences for data manipulation. We illustrate this by some queries; the first one is retrieving the children and grand children of person "Henk", using the first mentioned relational definition of 'person':

```
(SELECT name, birth_year
 FROM person
 WHERE father = "Henk")
UNION
(SELECT x.name, x.birth_year
 FROM person x, person y
 WHERE x.father = y.name AND y.father = "Henk"
 OR x.mother = y.name AND y.father = "Henk");
```

We refrain from showing a solution for the retrieval of all descendants of a selected person. For such recursive problems programs containing embedded SQL are advised [5].

Another example is the retrieval of mothers:

```
SELECT m.name, m.birth_year
FROM person m
AND EXISTS ( SELECT *
             FROM person x
             WHERE x.mother = m.name);
```

Another modeling approach, also only applying definite relationships, is the semantic approach introduced in [10, 14] and applied in the Xplain-DBMS. This approach enables us to model family trees as follows:

type person = birth_year, father_person, mother_person.

Using the last definition, persons can be identified by a name; so there is no need to define 'name' as an attribute. The Xplain language enables us to specify queries such as the following one, dealing with the children and grand children of Henk:

get person *its* birth_year
    where father_person = "Henk"
    or father_person *its* father_person = "Henk"
    or mother_person *its* father_person = "Henk".

By default, the result of such retrievals always contains the object identifiers of retrieved objects, in this case the name of retrieved persons.

Another application is determining mothers. Now we first derive a Boolean attribute 'person *its* is_mother':

extend person *with* is_mother = *any* person
per mother_person.

get person *its* birth_year *where* is_mother.

These examples indicate that the exchange of concept interpretations (*type* versus *attribute*) is, as will be made clear later, of crucial importance for declarative solutions, in particular for recursive problems: it enables us to apply the *its* construct in an attribute path.

The presentation of recursive applications of the Xplain language will be postponed to a following paper [21], where we present some recursive applications, including a solution for finding all descendants of a certain person. Here, we continue with a discussion of modeling approaches and their consequences for data manipulation.

4. Discussion

Two kinds of relationships can be recognized in family trees, those between the attributes of a person and those between persons. In both hierarchical and network models variable relationships are used for the interrelationships between objects (for example 1:n relationships between persons) and definite relationships for the relationships within an object (for example: each person has a name and a birth year as well). There seem to be two fundamental approaches to the modeling of recursive relationships:

- Variable relationship
  For example: 'each person has a number of children'. This is expressed in an access path and not in data. Therefore it does not lead to metadata: the inverse relationship 'each person has a parent' cannot be derived by the software system; the user must specify a procedure for that purpose.
- Definite relationship
  For example: 'each person has a parent'. This is expressed in a recorded property (namely as an attribute of 'person') and can possibly lead to metadata. The inverse relationship can be derived by the software system (see [15] for an application in a view generator). This approach contains therefore also the possibilities that can be defined with variable relationships.

In classical hierarchical models designer and user (they must cope with different concepts like 'parent' and 'child') must manage the access to relationships, whereas in the relational and semantic model these relationships are managed by the software system. In the last two approaches users can consider 'parent' and 'child' as derivable concepts. This freedom of interpretation offers possibilities for a declarative query language.

An advantage of the relational and the semantic approach as well is that users are not hindered by predefined variable relationships. The user does not have to indicate that the system should deviate from predefined relationships (for example, when the inverse relationship must be considered). This implies that the responsibility for the proper usage of relationships is delegated to the query language concepts and the software system (i.e. the query language processor). Of course, an essential precondition is that the software has enough metadata to carry out the required checks and to take the decisions for a correct processing.

Both the relational model and the semantic model use only one kind of relationship, namely the definite relationship. The variable relationship is considered as a derivable relationship. However, contrary to the relational model, all relationships in a semantic model are specified in an inherent way (in structure, not by subset constraints); therefore they lead to metadata. This makes it possible to use all relationships in a uniform way: a type is defined by its attributes. This situation is comparable with mathematical set theory in which only one fundamental relationship occurs: a set is defined by its elements (the membership relationship in the first axiom [12]). The following paper [21] will refer to this situation and will present semantic solutions for querying recursive data structures.

References

[1] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier and S. Zdonik, Object-oriented Database System Manifesto, *Proceedings IEEE 1st International Conference on Deductive and Object-Oriented Databases*, Kyoto, (1989), pp. 40-57.

[2] F. Bancilhon and R. Ramakrishnan, An Amateur's Introduction to Recursive Query Processing Strategies, *Proceedings ACM SIGMOD*, (1986), pp. 16-52.

[3] R.G.G. Cattell, *Object Data Management*, Addison-Wesley, 1994.

[4] CODASYL *Database Task Group April 1971 Report*, ACM New York, April 1971.

[5] T. Conally, C. Begg, A. Strachan, *Database systems: A practical approach to design, implementation and management*, Addison-Wesley, Reading Mass. 1995.

[6] G. Gardarin and P. Valduriez, *Relational databases and knowledge bases*, Addison-Wesley, Reading Mass. 1989.

[7] D.M. Kroenke, *Database Processing: Fundamentals, Design, and Implementation* (sixth edition), Prentice Hall, Upper Sadle River, NJ, 1998.

[8] C.H. Papadimitriou, Database Metatheory: Asking the Big Queries, *Proceeedings PODS '95*, San Jose CA, (1995), pp. 1-10.

[9] J. Paredaens, D. Van Gucht, Converting Nested Algebra Expressions into Flat Algebra Expressions, *ACM Transactions on Database Systems*, **17**, 1 (1992), pp. 65-93.

[10] F. Rolland, *The essence of databases*, Prentice Hall, Hemel Hempstead, 1998.

[11] A. Rosenthal, S. Heder, U. Dayal and F. Manola, Traversal Recursion: A Practical Approach to Supporting Recursive Applications, *Proceedings ACM SIGMOD*, (1986), pp. 166-176.

[12] J.R. Shoenfield, Axioms of Set Theory, In: *Handbook of Mathematical Logic*, J. Barwise (ed.), North-Holland, Amsterdam (1977), pp. 321-344.

[13] D. Suciu, J. Paredaens, Any algorithm in the complex object algebra with powerset needs exponential space to compute transitive closure, *Proceedings 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, (1994), p 201.

[14] J.H. ter Bekke, *Semantic Data Modeling*, Prentice Hall, Hemel Hempstead, 1992.

[15] J.H. ter Bekke, Complex values in databases, *Proceedings International Conference on Data and Knowledge Systems for Manufacturing and Engineering* (DKSME '94), Hong Kong (1994), pp. 449-455.

[16] J.H. ter Bekke, Semantic modeling of successive events applied to version management, *Proceedings International Symposium on Cooperative Database Systems for Advanced Applications* (CODAS '96), Kyoto (1996), pp.32-39; also in: Cooperative Databases and Applications, World Scientific, Singapore, 440-447.

[17] J.H. ter Bekke, Advantages of a compact semantic meta model, *Proceedings 2nd IEEE Metadata Conference*, Silver Spring (1997). http://www.computer.org/conferen/proceed/meta97/papers/jterbekke/jterbekke.html.

[18] J.H. ter Bekke and J.A. Bakker, Recursive queries in product databases, *Proc. 5th Int. Conference on Flexible Query Answering Systems*, (FQAS 2002), Copenhagen, Denmark, October 27-29, 2002, LNCS Vol. 2522, T. Andreasen, A. Motro, H. Christiansen, H. Legind Larsen (Eds.), Springer-Verlag, Berlin-Heidelberg (2002), pp. 44-55.

[19] J.H. ter Bekke and J.A. Bakker, Content-driven specifications for recursive project planning applications, *Proceedings International Conference on Applied Informatics* (AI 2002), Innsbruck, Austria (2002), pp. 448-452.

[20] J.H. ter Bekke and J.A. Bakker, Fast Recursive Data Processing in Graphs Using Reduction, *Proceedings International Conference on Applied Informatics* (AI 2003), Innsbruck, Austria (2003), pp. 490-494.

[21] J.H. ter Bekke and J.A. Bakker, Modeling and Querying Recursive Data Structures II: A Semantic Approach, *Proceedings International Conference Artificial Intelligence and Soft Computing* (ASC 2003), Banff, Canada (2003).

[22] J.D. Ullman and J. Widom, *A First Course in Database Systems*, Prentice Hall, 1997.