World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:3, 2010

A Generalised Relational Data Model

Georgia Garani

***Abstract*—**A generalised relational data model is formalised for the representation of data with nested structure of arbitrary depth. A recursive algebra for the proposed model is presented. All the operations are formally defined. The proposed model is proved to be a superset of the conventional relational model (CRM). The functionality and validity of the model is shown by a prototype implementation that has been undertaken in the functional programming language Miranda.

***Keywords*—**nested relations, recursive algebra, recursive nested operations, relational data model.

I. Introduction

A number of researchers have made proposals to relax the First Normal Form (1NF) assumption using the Non-First Normal Form (N1NF) relations to solve problems in applications such as text processing, engineering design systems and office automation and thus, overcome a number of limitations imposed by the apparently reasonable restriction that 1NF causes.

The use of a N1NF model eliminates many problems since it enables data about an object to be represented within one relation rather than distributing it over several relations. One major advantage is the fact that join operations which are substantially expensive in terms of execution time can be avoided.

The N1NF relational database model, or more simply the nested relational model, allows relations to have attributes which can have non-atomic values i.e., the latter are themselves relations, subrelations of the relation to which they belong. The N1NF model provides the basis for the object-oriented database model. Many models [1]-[6] have been defined since 1977, when Makinouchi [7] proposed, for the first time, the relaxation of the 1NF assumption. A list of relevant references can be found in [8].

In spite of the large number of N1NF models, only a few query languages have been proposed for the management of N1NF relations (e.g., [9]-[11]) by reason of its difficulty. These are extensions of existing query languages, SQL and Query by Example; an example is QBEN [9], a Query by Example language for nested tables which allows the formulation of complex queries.

The N1NF database models that have been developed so far can be divided into two categories. Models of the first category are called non-recursive models (e.g., [3], [6], [12]) and those of the second category are called recursive models (e.g., [1], [4], [5], [13]-[15]). The two approaches are distinguished by the recursive or non-recursive nature of the operators that have been defined by the distinct researchers. The difference is that recursive operators can be applied repeatedly to the subrelations at the different levels of a relation, whereas the non-recursive operators cannot. In section II of this paper the superiority of the recursive models compared to the non-recursive ones is explained and justified.

A Nested Relational Model (NRM) is formalised in this paper for the representation of nested data. A nested recursive algebra (NRA) for the NRM is proposed. All the operations are formally defined, including also the rename operation for nested relations. NRM is proved to be a superset of the CRM.

The rest of the paper is organised as follows. In section II a survey of the most important N1NF recursive models is presented. The running example of the paper is presented in section III. The basic concepts and terminology which are used in this paper are given in section IV. The NRA is presented in section V. In section VI the ease of use of the NRA is demonstrated by a number of examples. The components of the NRM are described in section VII. In section VIII the NRM is proved to be a superset of the CRM. Finally, conclusion is presented in section IX. Appendices I and II are provided with the formal syntax of the NRA and a sample code of the prototype implementation.

II. Literature Survey

Recursive algebraic definitions in nested models are undoubtedly preferable to the corresponding non-recursive ones. This is based on the following facts:

1) The non-recursive algebras allow operations only on entire tuples. In contrast, recursive algebras allow the direct manipulation of tuples either at the top level or at lower levels of the nested relations.

2) When an attribute at a lower nesting level of the nested relation needs to be accessed, because it participates in an operation expressed in a non-recursive algebra, one or more unnesting operations need to be applied resulting in the creation of many additional tuples. The non-recursive operation can then be performed and finally the relation is nested again. However, one of the main motivations for a model consisting of nested relations is the reduction in the number of tuples processed.

3) In the non-recursive algebras, queries can become long and complicated, whereas in the recursive algebras queries

Manuscript received May 28, 2007.

G. Garani is with the Computer Science and Telecommunications Department, Technological Educational Institute, Larissa, 41110, Greece, (phone: +30 2410 684344, fax: +30 2410 684387, e-mail: garani@teilar.gr).

will be shown to be compact, simpler and more naturally expressed.

4) Restructuring operations are not required with recursive algebras unlike non-recursive ones.

5) Traditional query optimisation techniques can be used with recursive algebras. In contrast, nest and unnest operations which have to be used frequently in non-recursive algebras, are not, in general, inverse operations. Therefore, traditional query optimisation techniques can be applied to queries which are expressed using recursive algebras since recursive operations can be performed at any nesting level without using nest or unnest operations [14].

However, it has been shown that the recursive and non-recursive algebras are equivalent in expressive power [13].

Previous research work dealing with N1NF recursive models includes Abiteboul and Bidoit's model [1] who proposed a Non-First Normal Form database model called the Verso model which allows data restructuring. Arbitrary projections can be achieved but they usually require a restructuring of the original relation. Two versions of the selection operation are defined, a simple version of the selection operation, the Verso-selection and an extension of the selection, called the "super-selection" which can be expressed by the Verso-selection, projection, and join operations. The restriction operation is itself restricted in that it can be applied only to the "root" of the format. The Cartesian product operation requires the first operand to be an instance over a flat relation and this is again a significant weakness. Furthermore, the key feature of their model, the restructuring operation, cannot reconstruct entirely the structures of the relations without loss of information, even when using a combination of all three transformations, root and branch permutations, compactions and extensions. As a result, the potentiality of the operation is limited to a restricted spectre of cases.

In Roth, Korth and Silberschatz's model [4] the Partitioned Normal Form (PNF) property is defined for nested relations. A relation R is in PNF if all the atomic attributes of R form a key for the relation and recursively, each relation-valued attribute of the relation is also in PNF. The simplicity and clarity of relations in PNF is apparent, as well as the fact that relations in PNF have some good properties compared to other relations. However, in general, relations in PNF impose two important restrictions, that there is at least one atomic attribute at every nesting level of the relation and also that relation-valued attributes cannot be part of the key. Two new operators, nest and unnest, are added to the basic set of operators. This approach has a number of limitations as presented in [16], [17]. Furthermore, the algebraic operators are defined in such a way that works within the class of PNF relations and therefore, they are closed only under PNF relations. In addition, projection, selection, join and Cartesian product operations cannot be applied to subrelations of nested relations.

A recursive algebra for nested relations is defined by Colby in [13]. Nest and unnest operations can be applied to subrelations directly, without transforming any other attribute of the relation by the assistance of a nest and an unnest list. The PNF assumption is not made. Arbitrary algebraic expressions in lists (select lists, project lists etc.) of the operators, such as comparisons of values of compatible attributes situated at different nesting levels in a relation cannot be supported.

An improved version of the algebra proposed by Schek and Scholl in [5] is presented by Deshpande and Larson in [18]. Two new operators, the subrelation constructor which transforms the interior of a nested relation and the PNF-Transformer which transforms recursively a nested relation into a nested relation in PNF are defined. Clearly, the invocation of the subrelation constructor one or more times in the formulation of queries increases the execution time to answer queries. The operators of the algebra are defined in such a way to preserve PNF property. Aggregate functions are also included in their algebra. Comparisons in the selection operation can only take place if the attributes that participate in the selection predicate are in the path starting at the root of R and ending at the subrelation identified by the pathname.

Levene in [14] presents the nested Universal Relation (UR) Model which forms an extension of the classical UR model to nested relations in order to solve the problem of incomplete information. One of the main features that his algebra provides, is the fact that the user does not need to know the structure of the nested relations in order to express a query in that algebra. Null values are also taken into consideration in the formalised proposed model. All the basic operators of the algebra are defined extensively. The problem of defining the join operation of two nested relations is solved with the insertion of empty nodes.

Liu and Ramamohanarao present also an algebra for nested relations in [19]. However, their algebra provides a restricted and complicated approach to the problem, since the following constraints must be satisfied: i) the selection operator considers only selection-comparable nodes, ii) the join operator can be performed only between two relations that have atomic attributes at the top levels.

Further interesting work by Buneman, Naqvi, Tannen and Wong [20] simplifies Colby's algebra and easily allows one to express all the recursive operations. Particularly, they present a language for structures in which nested relations and complex objects may be freely combined. They proved that their language coincides in expressive power with the nested relational languages proposed by [5], [6] and [13]. However, it has to be noted that only the semantics of the constructs that could be used in the language are studied and not the practical aspects of the design of syntax for query languages.

III. THE RUNNING EXAMPLE OF THE PAPER

The running nested database example of the paper consists of five nested relations TRAINING, DEPT, LOCATION, CASH-POINT and COURSE (Fig. 1).

TRAINING

COMPANY	TRAINER			
	TRN	COURSE		
		CODE	C	
			CN	Y
Apple	Jack	xx0	1	75
			2	76
	Mark	xy1	1	82
			3	82
		xy2	2	79
IBM	Tim	xy1	3	82
		xx2	5	79
			4	82
Microsoft	Karen	xx1	2	77
			2	81

DEPT

D	DN	UNIT				
		UN	UD	COURSE_DETAILS		
				TRN	COMPANY	C
						CN Y
1	Research	511	Software Engineering	Mark	Apple	1 75
						2 76
						5 79
		552	Basic Research	Karen	Microsoft	1 82
						2 79
				Tim	IBM	5 79
		678	Planning	Mark	Apple	2 76
						4 82
2	Development	650	Design	Karen	Microsoft	1 75
		780	Maintenance	Tim	IBM	3 82
				Mark	Apple	2 76
		981	Planning	Jack	Apple	2 81
						3 82
						5 79

LOCATION

COMPANY	ANNEX	
	BUILDING	ADDRESS
TOSHIBA	North Building	Porchester Rd.
IBM	Maple House	Kendal Av.
	Main Building	Danebury Rd.
Microsoft	Pegasus House	Ashford St.
	Queen's Building	Park Rd.

COURSE

C		COURSE_DURATION	TITLE	SUBJECT
CN	Y			TOPICS
1	75	80	Computer Skills	Access
2	77			Word
				Excel
2	82	120	Multimedia	Power Point
3	82			Internet
2	79	20	Programming	C++
				JAVA

CASH-POINT

BANK	BRANCH	
	SORT_CODE	ADDRESS
Barclays	386600	Ashford St.
NatWest	560045	Park Rd.
	560038	Porchester Rd.
Lloyd's	478202	Ashford St.
	478210	Park Rd.

Fig. 1 The running nested database example

Relation TRAINING holds data about courses and trainers provided by IT companies. Semantically, the attributes of the TRAINING relation have the following meaning: COMPANY - company name, TRN - trainer name, CODE - course code, CN - course number Y - year in which the course was taken. A specific course can be identified uniquely by both course number (CN) and year (Y); a specific course consists of a number of different topics (see rel. COURSE) which can be given by different trainers belonging to different companies.

Relation DEPT holds data about departments of a company as well as trainers who have given courses to the staff of these departments. The semantics of the attributes of relation DEPT are: D - department number, DN - department name, UN – unit number, UD – unit description, TRN – trainer name, COMPANY – company name, CN - course number and Y - year in which the course was taken. Relation DEPT is a modified version of relation DEPT in [5].

Relation LOCATION contains data about branches of different companies. The attributes of relation LOCATION have the following semantics: COMPANY –company name, BUILDING – building name and ADDRESS – street name.

Relation CASH-POINT has data about cash-points that different banks own. The semantics of the attributes of relation CASH-POINT are: BANK – bank name, SORT_CODE – sort code of the branch and ADDRESS – street name.

Relation COURSE contains data about the different courses that took place. Semantically, the meaning of the attributes of relation COURSE is: CN - course number, Y - year in which the course was taken, COURSE_DURATION – course duration (number of hours), TITLE – course title and TOPICS – course topics.

IV. BASIC CONCEPTS AND TERMINOLOGY

In order to introduce the Nested Relational Model (NRM) in the next section it is necessary to present firstly the basic concepts and terminology that are going to be used. Some of the following definitions have been used before by the database community. However, a repetition of these definitions at the present point is necessary for completeness. Moreover, some terms and notation are introduced for the first time in the present paper in order to provide the essential formalisation of the presented model.

Definition 1 ***(Atomic or flat attributes and relation-valued or nested attributes)*** Let U be the set of elementary values (i.e., reals, integers etc.) and the value null. An attribute A is atomic or flat if $D_A \subseteq U$, where D_A is the domain of the attribute A. If $D_A \subseteq P(U)$ where P is the power set, then A is a relation-valued or nested attribute.□

Relation-valued attributes or nested attributes can be considered as subrelations of the relations to which they belong.

Definition 2 ***(Non-first normal form relations or nested relations)*** Non-first normal form relations or nested relations are relations which contain relation-valued attributes or nested attributes.

In this paper, relations with atomic attributes only will be called *flat relations*, whereas relations that contain relation-valued attributes or atomic attributes will be referred to as *nested relations*. In other words, flat relations are considered as special cases of nested relations.

Attr(R) is the set of attributes of relation r with scheme name R i.e., $Attr(R) = \{R_1, R_2, ..., R_n\}$, where $n \geq 1$ and R_1, R_2, ..., R_n are the attributes of R, either atomic or nested.

Definition 3 ***(Tree structure)*** Every nested relation r with relation scheme R can be represented as a tree with root node R. All the nested attributes of the relation are the non-leaf nodes of the tree and all the atomic attributes form the leaf nodes of the tree. □

The tree structure is a very useful representation of a nested relation since the scheme of a nested relation can become complex and so, the tree offers a clear graphical representation of the nested structure.

Example 1: The tree structure of the TRAINING relation (Fig. 1) is shown in Fig. 2.

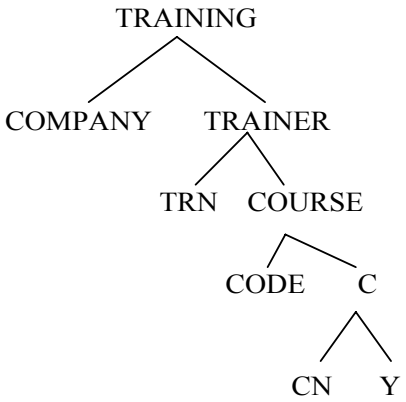

Fig. 2 Tree representation of relation TRAINING

Definition 4 ***(Nesting levels of a relation)*** The number of nesting levels of a relation is equal to the maximum number of nodes to be passed through starting from the root to reach any atomic attribute in the tree representation. The root of the relation is by definition at nesting level 0. □

Example 2: The nesting levels of relation TRAINING (Fig. 1) are 4.

Consequently, the nesting level of an attribute in a relation can be computed by counting the number of nodes which must be passed through from the root node to get to that attribute. For example, atomic attribute TRN of relation TRAINING is at nesting level 2.

Definition 5 ***(Common attributes between two relations)*** Two (flat or nested) relations have an atomic attribute in common if they both contain an atomic attribute which has the same name and domain in both relations. Two nested relations have a nested attribute in common if they both contain a nested attribute which has the same name and the same scheme (the same atributes with the same names defined over

the same domains).

The above definition can be applied recursively for nested attributes containing one or more nested attributes.

Definition 6 ***(Path)*** Let $L_{A_n \to A_j}$ be the path of nested or atomic attribute A_j belonging to nested attribute A_n, which is a child of the root of relation R. Then, $L_{A_n \to A_j}$ is defined as follows:

i) $L_{A_n \to A_j} = A_n$, where $A_j = A_n$

ii) $L_{A_n \to A_j} = A_n(L_{A_{n+1} \to A_j})$, where A_{n+1} is an attribute of A_n either equal to or containing A_j. □

Then, the set of all attributes (atomic and nested) of R can be defined as $Attr(R) = \{R_{a1}, R_{a2}, \ldots, R_{ap}, R_{n1}, \ldots, R_{ni}, \ldots, R_{nq}\}$ $= \{R_{a1}, R_{a2}, \ldots, R_{ap}, R_{n1}, \ldots, \bigcup_{k=0}^{m} L_{R_{ni} \to R_{ni_k}}, \ldots, R_{nq}\}$ where:

- $R_{a1}, R_{a2}, \ldots, R_{ap}$ are atomic attributes at nesting level 1 of relation R ($p \geq 0$),
- $R_{n1}, \ldots, R_{ni}, \ldots, R_{nq}$ are nested attributes at nesting level 1 of relation R ($1 \leq i \leq q$),
- $R_{ni_k} = \begin{cases} R_{ni} \text{ for } k = 0 \\ R_{ni_k} \text{ for } k \neq 0 \text{ (i.e., an attribute that has nested attribute } R_{ni} \text{ as its ancestor)} \end{cases}$

m is the number of descendants' attributes of nested attribute R_{ni}.

The path is used for the definition of an attribute in a nested relation, in contrast to flat relations, since the whole path of an attribute is needed in order to identify that specific attribute.

Example 3: The path of the atomic attribute CN of the nested relation TRAINING (Fig. 1) with tree structure in Fig. 2 is $L_{TRAINER \to CN} = TRAINER(L_{COURSE \to CN}) = TRAINER(COURSE(L_{C \to CN})) = TRAINER(COURSE(C(L_{CN \to CN}))) = TRAINER(COURSE(C(CN)))$.

From the above example, it is apparent that the name of an attribute by itself is not enough in general to uniquely identify the attribute, since in nested relations an attribute is fully defined by reference to both its name and its position in the tree structure of the relation in which it belongs. In addition, there are cases in which two common attributes belong in the same relation but in different subrelations, as for example in the result relation of a join operation. Consequently, the only way for the two attributes to be distinguished from one another is by their paths. Therefore, the path of an attribute shows whether the attribute belongs to a nested attribute or not, as well as the nesting level of it. The path of an attribute identifies the attribute uniquely.

Definition 7 ***(Two nested relations having the same scheme)*** Two nested relations have the same scheme if they contain only common attributes (atomic and/or nested) -see Definition 5. □

An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a key for that entity set [21]. For the case of a nested database model, entity sets are nested relations and the definition of the key must be expanded in order to support nested attributes as well.

Informally, a nested relation can have either atomic or nested attributes or even a combination of atomic and nested attributes as a key. Semantically, a nested attribute is a key of a nested relation, when each set of values of the nested attribute that belongs to the same tuple, uniquely identifies that tuple. That implies that each of these set of values of the nested attribute distinguishes, as an entirety, solely the tuple in which it belongs.

Formally, the definition of a key of a nested relation is given below:

Definition 8 ***(Key of a nested relation)*** The key of a nested relation r with relation scheme R, can be a set K consisting of atomic and/or nested attributes of R such that for any two tuples t_i and t_j in the relation the following constraint is valid at all times: $t_i[K] \neq t_j[K]$, where $i \neq j$ and with the additional property that removing any attribute from K leaves a set of attributes that is not a key of R. □

Example 4: The key of relation COURSE (Fig. 1) is the nested attribute C.

It is considered that an approach where nested attributes are allowed to be part of key attributes is an important benefit for a nested model. Nested models, where nested attributes are not allowed to be part of key attributes, have a significant limitation, since relations, as the one presented in Fig. 1, cannot be supported. Therefore, there are cases that are not covered by such an approach.

The PNF assumption presupposes that nested attributes cannot form part of a key in a nested relation, a significant restriction of a nested database model, as explained in section II.

Consequently, in the nested model defined in the present paper, the relaxing of the restriction that other nested models impose, to allow nested attributes as part of the key, is a considerable extension and thus, an important benefit that the NRM offers.

V. The Nested Relational Algebra (NRA)

A new nested relational algebra is defined in this chapter, called the Nested Relational Algebra, NRA. Relations in NRA can be nested to any finite depth.

A. Operations in the NRA

Union, difference, intersection, projection, selection, unnest, nest, rename, Cartesian product, natural join and Θ–join operations are formally defined using recursive definitions for nested relations. The "base case" of each recursive operator has the same definition as the non-recursive one; i.e., the recursive definition can be reduced to the non-recursive one when relations do not contain any nested attributes. For each definition, an example is presented in order to make it more comprehensive. The recursive rename operation for nested relations is also formally defined for the first time.

For the recursive nested union, difference and intersection operations let r and q be two nested (in general) relations with relation schemes R and Q respectively. Let also, the two relations have the same relation scheme i.e., R = Q = {S(R), R_1, R_2, …, R_n} where S(R) is the set containing all the key nested attributes and all the atomic attributes of R and Q (the same for the two relations) and {R_1, R_2, …, R_n} are the non-key nested attributes of R and Q. Assume also that Attr(R) is the set of all attributes (atomic and nested) of the two relations, t_r is a tuple in relation r, t_q is a tuple in relation q and t is a tuple in the result relation.

The Recursive Nested Union Operation ($\cup^{\smile}$)

The union of the two relations r and q, $r \cup^{\smile} q$, is defined as follows:

Definition 9 (*Recursive Nested Union*)

i) Non-recursive union for flat relations ($r \cup q$)

$r \cup q = \{ t| ((\exists t_r \in r) (t[Attr(R)] = t_r[Attr(R)]))$
$\vee ((\exists t_q \in q) (t[Attr(R)] = t_q[Attr(R)]))\}$

ii) Recursive union for nested relations ($r \cup^{\smile} q$)

$r \cup^{\smile} q = \{ t| ((\exists t \in r) \wedge (\forall t_q \in q) (t[S(R)] \neq t_q[S(R)]))$
$\wedge ((\exists t \in q) \wedge (\forall t_r \in r) (t[S(R)] \neq t_r[S(R)]))$
$\wedge ((\exists t_r \in r) (\exists t_q \in q) ((t[S(R)] = t_r[S(R)] = t_q[S(R)])$
$\wedge (t[R_1] = t_r[R_1] \cup^{\smile} t_q[R_1]) \wedge \ldots \wedge (t[R_n] = t_r[R_n] \cup^{\smile} t_q[R_n]))\}$ □

Example 5: Let relations TRAINING_1 and TRAINING_2 (Fig. 3) be two modified versions of relation TRAINING (Fig. 1) having the same scheme. In both relations, TRAINING_1 and TRAINING_2, S(TRAINING_1) = S(TRAINING_2) = COMPANY.

The union of the two relations, according to the above definition, is shown in Fig. 3.

The Recursive Nested Difference Operation (-˙)

TRAINING_1

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Jack	1	75
		2	76
	Mark	1	82
		3	82
		2	79
IBM	Tim	3	82
		5	79
		4	82
Microsoft	Karen	2	77
		2	81

TRAINING_2

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Jack	6	82
		2	76
	Mark	3	82
		2	79
Toshiba	Tim	5	84
Microsoft	Karen	2	77
		2	81

TRAINING_1 $\cup^{\smile}$ TRAINING_2

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Jack	1	75
		2	76
		6	82
	Mark	1	82
		3	82
		2	79
IBM	Tim	3	82
		5	79
		4	82
		5	84
Microsoft	Karen	2	77
		2	81
Toshiba	Tim	5	84

TRAINING_1 -˙ TRAINING_2

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Jack	1	75
	Mark	1	82
IBM	Tim	3	82
		5	79
		4	82

TRAINING_1 $\cap^{\frown}$ TRAINING_2

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Jack	2	76
	Mark	3	82
		2	79
Microsoft	Karen	2	77
		2	81

Fig. 3 Examples of the recursive nested union, difference and intersection operations

The difference of the two relations r and q, r -¯ q, is defined as follows:

Definition 10 (*Recursive Nested Difference*)

i) Non-recursive difference for flat relations (r - q)

$r - q = \{ t | (\exists t_r \in r) (\forall t_q \in q) ((t[Attr(R)] = t_r[Attr(R)]) \wedge (t[Attr(R)] \neq t_q[Attr(R)]))\}$

ii) Recursive difference for nested relations (r –¯ q)

$r -^{-} q = \{ t | ((\exists t_r \in r) (\forall t_q \in q)$

$((t[S(R)] = t_r[S(R)] - t_q[S(R)]) \wedge (t[R_1] = t_r[R_1]) \wedge \ldots \wedge (t[R_n] = t_r[R_n])))$

$\vee ((\exists t_r \in r), (\exists t_q \in q) ((t[S(R)] = t_r[S(R)] = t_q[S(R)])$

$\wedge (t[R_1] = t_r[R_1] -^{-} t_q[R_1]) \wedge \ldots \wedge (t[R_n] = t_r[R_n] -^{-} t_q[R_n]))) \}$ □

Example 6: The difference of the two relations TRAINING_1 and TRAINING_2 is shown in Fig. 3.

The Recursive Nested Intersection Operation (∩⌢)

The intersection of the two relations r and q, $r \cap^{\frown} q$, is defined as follows:

Definition 11 (*Recursive Nested Intersection*)

i) Non-recursive intersection for flat relations ($r \cap q$)

$r \cap q = \{ t | (\exists t_r \in r) (\exists t_q \in q) (t[Attr(R)] = t_r[Attr(R)] = t_q[Attr(R)])\}$

ii) Recursive intersection for nested relations ($r \cap^{\frown} q$)

$r \cap^{\frown} q = \{ t | (\exists t_r \in r) (\exists t_q \in q) ((t[S(R)] = t_r[S(R)] \cap t_q[S(R)])$

$\wedge (t[R_1] = t_r[R_1] \cap^{\frown} t_q[R_1]) \wedge \ldots \wedge (t[R_n] = t_r[R_n] \cap^{\frown} t_q[R_n]))\}$ □

Example 7: The intersection of the two relations TRAINING_1 and TRAINING_2 is shown in Fig. 3.

The Recursive Nested Projection Operation (π^{π})

Let r be a nested (in general) relation with relation scheme R. Let also, $\{R_{a1}, \ldots, R_{ak}\}$ be the subset of atomic attributes at the top level of R which are going to be projected and $\{R_{n1}, \ldots, R_{nm}\}$ the subset of nested attributes of R which are going to be projected either fully or partially on attributes belonging to these nested ones (k, m ≥ 0).

In order to define the projection operation, the term project list needs to be defined firstly. In general, a project list is a list of project paths. A project path of an attribute which is going to be projected is the path of that attribute (see Definition 6).

Definition 12 (*Project list*) L_{π} is a project list of R if

i) L_{π} is empty (the project list of an atomic attribute is empty).

ii) L_{π} is of the form $(R_{n1}L_{n1}, \ldots, R_{nm}L_{nm})$, where $L_{n1}, \ldots, L_{nm}$ are project lists of nested attributes $R_{n1}, \ldots, R_{nm}$ respectively. □

Then, the projection operation in a nested relation r, $\pi^{\pi}(rL_{\pi})$, where t_r is a tuple in relation r and t is a tuple in the result relation, is defined as follows:

Definition 13 (*Recursive Nested Projection*)

i) $\pi(r) = r$

ii) $\pi^{\pi}(r(R_{a1}, \ldots, R_{ak}, R_{n1}L_{n1}, \ldots, R_{nm}L_{nm})) = \{ t | (\exists t_r \in r)$

$((t[R_{a1}] = t_r[R_{a1}]) \wedge \ldots \wedge (t[R_{ak}] = t_r[R_{ak}])$

$\wedge (t[R_{n1}] = \pi^{\pi}(t_r[R_{n1}]L_{n1})) \wedge \ldots \wedge (t[R_{nm}] = \pi^{\pi}(t_r[R_{nm}]L_{nm})))\}$ □

Example 8: Given relation TRAINING_1 (Fig. 3) consider the following query: "Retrieve the course numbers for the courses that each company has run". The result relation is

COMPANY	TRAINER C CN
Apple	1, 2
	1, 3, 2
IBM	3, 5, 4
Microsoft	2, 2

Fig. 4 π^{π}(TRAINING_1(COMPANY, TRAINER(C(CN))))

shown in Fig. 4.

The Recursive Nested Selection Operation (σ^{σ})

Let r be a nested (in general) relation with relation scheme R and let $R_a = \{R_{a1}, \ldots, R_{ak}\}$ and $R_n = \{R_{n1}, \ldots, R_{nm}\}$ be the subsets of all atomic and nested attributes of R respectively that participate in the selection operation, where k and m are less than or equal to the number of atomic and nested attributes at the top level in the relation R, respectively. Let also, c be a set of conditions in R, which is of the form $\{c_a, c_n\}$ where $c_a = \{c_{a1}, \ldots, c_{ak}\}$ is a set of conditions which must be true for the atomic attributes $R_{a1}, \ldots, R_{ak}$ of R respectively and $c_n = \{c_{n1}, \ldots, c_{nm}\}$ is a set of conditions that must hold for the nested attributes $R_{n1}, \ldots, R_{nm}$ of R respectively. When both sets of conditions are applied simultaneously then, the result is obtained by computing the intersection of the two results. In addition, the condition can be no matter complicated, as for example equality of nested attributes. If two multi-valued nested attributes are compared for equality, they are treated as sets so, since each nested attribute is, in fact, a relation, equal tuples are searched at the level of the nested relations.

In order to define the selection operation, the term *select list* needs to be defined firstly. In general, a select list is a list of select paths. A select path of an attribute that is going to participate in the selection, is the path of that attribute (see Definition 6). The select list is defined recursively.

Definition 14 (*Select list*) L_{σ} is a select list of R if

i) L_{σ} is empty (all the atomic attributes of relation r have empty select lists).

ii) L_{σ} is of the form $(R_{n1}L_{n1}, \ldots, R_{nm}L_{nm})$ where $L_{n1}, \ldots, L_{nm}$ are select lists of nested attributes $R_{n1}, \ldots, R_{nm}$ respectively. □

Then, a selection operation of the relation r, where t_r is a tuple in relation r and t is a tuple in the result relation, is

defined as follows:

Definition 15 (*Recursive Nested Selection*)

$\sigma(r_{ca1, \ldots, cak}) = \{ t | (\exists t_r \in r)$
$((t[Attr(R) - \{R_{a1}, \ldots, R_{ak}\}] = t_r[Attr(R) - \{R_{a1}, \ldots, R_{ak}\}])$
$\wedge ((t[R_{a1}] = t_r[R_{a1}]) \wedge c_{a1} = true)$
$\wedge \ldots \wedge ((t[R_{ak}] = t_r[R_{ak}]) \wedge c_{ak} = true))\}$
$\sigma^{\sigma}(r_{cn1, \ldots, cnm}L_{\sigma}) = \{ t | (\exists t_r \in r)$
$((t[Attr(R) - \{R_{n1}, \ldots, R_{nm}\}] = t_r[Attr(R) - \{R_{n1}, \ldots, R_{nm}\}])$
$\wedge (t[R_{n1}] = \sigma^{\sigma}(t_r[R_{n1}]_{cn1}L_{n1}) \neq \varnothing)$
$\wedge \ldots \wedge (t[R_{nm}] = \sigma^{\sigma}(t_r[R_{nm}]_{cnm}L_{nm}) \neq \varnothing))\}$

In the general case, the selection operation can be defined as the intersection of the two previously defined cases as follows:

$\sigma^{\sigma}(r_cL_{\sigma}) = \sigma^{\sigma}(r_{ca1, \ldots, cak, cn1, \ldots, cnm}L_{\sigma}) = \sigma(r_{ca1, \ldots, cak}) \cap \sigma^{\sigma}(r_{cn1, \ldots, cnm}L_{\sigma})$ □

Example 9: Given relation TRAINING_1 (Fig. 3) consider the following query: "Find all the information of the TRAINING_1 relation of those courses that have been given by trainers Mark or Tim during the year 1982". The result is shown in Fig. 5.

COMPANY	TRAINER		
	TRN	C	
		CN	Y
Apple	Mark	1	82
		3	82
IBM	Tim	3	82
		4	82

Fig. 5 $\sigma^{\sigma}(TRAINING\_1_{((TRAINER(TRN) = 'Mark' \text{ OR } 'Tim') \text{ AND } (TRAINER(C(Y)) = 82))})$

The Recursive Unnest Operation (μ^{μ})

Let r be a nested (in general) relation with relation scheme R.

Definition 16 (*Unnest list*) L_{μ} is an unnest list of R if it is of the form

R_i, where R_i is a nested attribute of R at the top level.

(R_iL_i) where L_i is an unnest list of the nested attribute R_i. □

Let Attr(R) be the set of all attributes of R and R_i a nested attribute of R, at the top level of R. Let also, t_r be a tuple in relation r and t a tuple in the result relation. Then, the unnest operation, $\mu^{\mu}(r_{L\mu})$, is defined as follows (see also [13]):

Definition 17 (*Recursive Unnest*)

i) $\mu(r_{Ri}) = \{ t | (\exists t_r \in r) ((t[Attr(R) - R_i] = t_r[Attr(R) - R_i]) \wedge (t[R_i] \epsilon t_r[R_i]))\}$

ii) $\mu^{\mu}(r_{RiLi}) = \{ t | (\exists t_r \in r) ((t[Attr(R) - R_i] = t_r[Attr(R) - R_i]) \wedge (t[R_i] = \mu^{\mu}(t_r[R_i]_{Li})))\}$ □

COMPANY	TRAINER		
	TRN	CODE	
			CN Y
Apple	Jack	xx0	1 75
			2 76
	Mark	xy1	1 82
			3 82
	Mark	xy2	2 79
IBM	Tim	xy1	3 82
	Tim	xx2	5 79
			4 82
Microsoft	Karen	xx1	2 77
			2 81

Fig. 6 $\mu^{\mu}(TRAINING_{TRAINER(COURSE)})$

Example 10: The result of unnesting relation TRAINING (Fig. 1) on the COURSE attribute, i.e., $\mu^{\mu}(TRAINING_{TRAINER(COURSE)})$, is shown in Fig. 6.

The Recursive Nest Operation (v^{v})

Let r be a nested (in general) relation with relation scheme R.

Definition 18 (*Nest list*) L_v is a nest list of R if it is of the form

i) $(R_1, \ldots, R_n)$ where $R_1, \ldots, R_n$ are attributes of R, either atomic or nested at the top level of R.

ii) (R_iL_i) where L_i is a nest list of the nested attribute R_i. □

Let Attr(R) be the set of all attributes of R and $A_n = \{R_1, \ldots, R_n\}$ the set of attributes of R that are going to be nested to form a new nested attribute A.

Let also, t_r be a tuple in relation r, t a tuple in the result relation and s a tuple of the new nested attribute A. Then, the nest operation, $v^{v}(r_{Lv \rightarrow A})$, is defined as follows (see also [13]):

Definition 19 (*Recursive Nest*)

i) $v(r_{An \to A}) = \{ t | (\exists t_r \in r) ((t[Attr(R) - A_n] = t_r[Attr(R) - A_n])$
$\wedge (t[A] = \{s[A_n] \mid (s \in r) (s[Attr(R) - A_n] = t_r[Attr(R) - A_n])\})\}$

ii) $v^v(r_{(RiLi) \to A}) = \{ t | (\exists t_r \in r) ((t[Attr(R) - R_i] = t_r[Attr(R) - R_i])$
$\wedge (t[R_i] = v^v(t_r[R_i]_{Li \to A}))\}$ □

Example 11: In order to return to relation TRAINING (Fig. 1) from the relation $\mu^{\mu}(TRAINING_{TRAINER(COURSE)})$ of Fig. 6, a nest operation needs to be performed, i.e., $v^v(\mu^{\mu}(TRAINING_{TRAINER(COURSE)})_{TRAINER(CODE,C) \to TRAINER(COURSE)})$.

The Recursive Nested Rename Operation (ρ^{ρ})

The rename operation takes a specified relation and returns another that is identical to the given one except that at least one of its attributes has a different name ([22]). The rename operation is useful before or after performing a number of operations, as for example for cases when there are duplicate names in the result relation after performing a join operation of two relations, or when the Cartesian product operation is performed between two relations having attributes with the same name. When a rename operation takes place only the heading of the relation changes, the body (instance) remains the same.

Let r be a nested (in general) relation with relation scheme $R = \{R_1, R_2, \ldots, R_i, \ldots, R_n, A, B, \ldots, Z\}$, where $R_1, R_2, \ldots, R_i, \ldots, R_n$ are atomic attributes and A, B, …, Z are nested attributes at the top level of relation R.

Then, the rename operation, ρ^{ρ}, of relation r is defined as follows:

Definition 20 (*Recursive Nested Rename*)

i) Rename of an atomic attribute R_i to $R_{i'}$ at the top level of relation R

$\rho[R_i \leftarrow R_{i'}](R) = \{R_1, R_2, \ldots, R_{i'}, \ldots, R_n, A, B, \ldots, Z\}$

ii) Rename of a nested attribute A to A′ at the top level of relation R

$\rho[A \leftarrow A'](R) = \{R_1, R_2, \ldots, R_i, \ldots, R_n, \bigcup_{k=0}^{m} L_{A' \to Ak}, B, \ldots, Z\}$

where m is the number of attributes that are descendants of A and for m = 0, $A' = A_0$ (atomic attribute at the top level of R) and case (ii) reduces to case (i).

iii) Rename of an atomic or nested attribute A_i to A_i' at a lower level of relation R

$\rho^{\rho}[A_i \leftarrow A_i'](R) = \{R_1, R_2, \ldots, R_i, \ldots, R_n, A, A_1, \ldots, \bigcup_{k=0}^{m} L_{A \to Ai'k}, B, \ldots, Z\}$, where A_1 is a child attribute of nested attribute A, A_i is an attribute at a lower level of relation R belonging to nested attribute A and m is the number of descendants that A_i has (m = 0, when atomic, in which case $A_{i'0} = A_{i'}$). □

When more than one attribute has to be renamed the definition is recursive, as follows:

$\rho^{\rho}[R_{a1} \leftarrow R'_{a1}, \ldots, R_{ak} \leftarrow R'_{ak}, R_{n1} \leftarrow R'_{n1}, \ldots, R_{nm} \leftarrow R'_{nm}, R_{l1} \leftarrow R'_{l1}, \ldots, R_{lp} \leftarrow R'_{lp}](R) =$

$(\rho^{\rho}[R_{lp} \leftarrow R'_{lp}](\ldots(\rho^{\rho}[R_{l1} \leftarrow R'_{l1}](\rho[R_{nm} \leftarrow R'_{nm}](\ldots(\rho[R_{n1} \leftarrow R'_{n1}](\rho[R_{ak} \leftarrow R'_{ak}](\ldots(\rho[R_{a1} \leftarrow R'_{a1}](R))))))))))$

where $R_{a1}, \ldots, R_{ak}$ are atomic attributes at the top level of relation R, $R_{n1}, \ldots, R_{nm}$ are nested attributes at the top level of relation R and $R_{l1}, \ldots, R_{lp}$ are either atomic or nested attributes at lower levels (different, in general) of relation R and k, m, p ≥ 0. The names of the attributes having primes denote the new names that these attributes are going to be renamed.

Example 12: Consider the relation DEPT (Fig. 1) and let attribute UD be renamed as UD′ and attribute C as C′. Then, the rename operation is defined as follows:

$\rho^{\rho}[UD \leftarrow UD', C \leftarrow C'](DEPT) = \rho^{\rho}[C \leftarrow C'](\rho^{\rho}[UD \leftarrow UD'](DEPT)) =$

$\rho^{\rho}[C \leftarrow C'](\{D, DN, UNIT, UNIT(UN), UNIT(UD'), UNIT(COURSE\_DETAILS), UNIT(COURSE\_DETAILS(TRN)), UNIT(COURSE\_DETAILS(COMPANY)), UNIT(COURSE\_DETAILS(C)), UNIT(COURSE\_DETAILS(C(CN))), UNIT(COURSE\_DETAILS(C(Y)))\}) =$

{D, DN, UNIT, UNIT(UN), UNIT(UD′), UNIT(COURSE_DETAILS), UNIT(COURSE_DETAILS(TRN)), UNIT(COURSE_DETAILS(COMPANY)), UNIT(COURSE_DETAILS(C′)), UNIT(COURSE_DETAILS(C′(CN))), UNIT(COURSE_DETAILS(C′(Y)))}

The Recursive Nested Cartesian Product Operation ($\times^{\times}$)

Let R be a relation scheme of relation r.

Definition 21 ***(Join path)*** L is a join path of R if either:

(i) L is empty or

(ii) $L = R_i L_i$ where R_i is a nested attribute of R and L_i is a join path of R_i. □ ([3])

The join path can be represented as a branch of the tree structure of some nested relation R starting from a child of the root of the tree and going down to some node of the tree that represents either an atomic or nested attribute. In other words, the join path consists of all the nodes that are passed in order to reach a specific attribute.

Example 13: In relation DEPT (Fig. 1) an example of a join path is UNIT(COURSE_DETAILS(TRN)).

Let r and q be two nested (in general) relations with relation schemes R and Q respectively and let Attr(R) be all the attributes (atomic and nested) of R, Attr(Q) all the attributes (atomic and nested) of Q and L a join path of R. Let, also, R_i be a nested attribute of R, L_i a join path of R_i, t_r a tuple in relation r, t_q a tuple in relation q and t a tuple in the result relation. The Cartesian product operation can be applied either at the top level of both relations or between a lower nesting level of a relation and the top level of another relation. The first case is exactly the same as the standard Cartesian product for flat relations.

COMPANY	(TRN (CODE C BANK BRANCH))						
	TRN	(CODE C BANK BRANCH)					
		CODE			BANK	BRANCH	
			CN	Y		SORT_CODE	ADDRESS
Apple	Jack	xx0	1	75	Barclays	386600	Ashford St.
			2	76			
		xx0	1	75	NatWest	560045	Park Rd.
			2	76		560038	Porchester Rd.
		xx0	1	75	Lloyd's	478202	Ashford St.
			2	76		478210	Park Rd.
	Mark	xy1	1	82	Barclays	386600	Ashford St.
			3	82			
		xy1	1	82	NatWest	560045	Park Rd.
			3	82		560038	Porchester Rd.
		xy1	1	82	Lloyd's	478202	Ashford St.
			3	82		478210	Park Rd.
		xy2	2	79	Barclays	386600	Ashford St.
		xy2	2	79	NatWest	560045	Park Rd.
						560038	Porchester Rd.
		xy2	2	79	Lloyd's	478202	Ashford St.
						478210	Park Rd.
. . .	. . .	. . .	.	.	. . .	. . .	. . .

Fig. 7 $\times^{\times}$ (TRAINING(TRAINER(COURSE)), CASH-POINT)

So, the Cartesian product of two relations r and q is defined as follows [13]:

Definition 22 (*Recursive Nested Cartesian Product*)

$\times (r, q) = \{ t \equiv (t[Attr(R)], t[Attr(Q)])|$

$(\exists\ t_r \in r)\ (\exists\ t_q \in q)\ ((t[Attr(R)] = t_r[Attr(R)]) \wedge (t[Attr(Q)] = t_q[Attr(Q)]))\}$

$\times^{\times} (rL, q) = \times^{\times} (r(R_iL_i), q) \equiv \times^{\times} (q, r(R_iL_i)) =$

$\{ t|\ (\exists\ t_r \in r)\ ((t[Attr(R) - \{R_i\}] = t_r[Attr(R) - \{R_i\}]) \wedge (t[R_i] = \times^{\times} (t_r[R_i]L_i, q)))\}$ □

The commutative property is satisfied, as is the case in the CRM. Thus, it is always valid that

$\times^{\times} (rL, q) \equiv \times^{\times} (q, rL)$

Example 14: The Cartesian product operation is performed between the COURSE attribute of relation TRAINING and the CASH-POINT relation (Fig. 1). Due to the large number of tuples in the result relation, only a part of it is displayed in Fig. 7.

The Cartesian product operation is not often a semantically meaningful operation, as can be seen from the above example. However, it helps in defining the join operation, since the join is a special case of a Cartesian product operation and for this reason it is included here.

The Recursive Nested Natural Join operation ($\triangleright\triangleleft^{\triangleright\triangleleft}$)

The natural join operation is formally defined in [23]. The main definition is given here for completeness reasons.

Definition 23 (*Recursive Nested Natural Join*)

Let r and q be two nested relations with relations schemes R and Q respectively and let $A = \{A_0, A_1, \ldots, A_j\}$ be the set of all common attributes that the two relations have, where A_0, A_1, …, A_j are atomic or nested attributes either at the top or lower levels in the two relations.

Then, the natural join of relations r and q, $\triangleright\triangleleft^{\triangleright\triangleleft} (r, q)$, is defined as follows:

$\triangleright\triangleleft^{\triangleright\triangleleft} (r, q) = \triangleright\triangleleft^{\triangleright\triangleleft} (s_jL_{sjAj}, s'_jL_{s'jAj})(\ldots(\triangleright\triangleleft^{\triangleright\triangleleft} (s_1L_{s1A1}, s'_1L_{s'1A1})(\triangleright\triangleleft^{\triangleright\triangleleft} (rL_{rA0}, qL_{qA0}))))$ where $\triangleright\triangleleft^{\triangleright\triangleleft} (rL_{rA0}, qL_{qA0}) = x_1$, $\triangleright\triangleleft^{\triangleright\triangleleft} (s_1L_{s1A1}, s'_1L_{s'1A1}) = x_2$, …, $\triangleright\triangleleft^{\triangleright\triangleleft} (s_jL_{sjAj}, s'_jL_{s'jAj}) = x_{j+1}$ and (s_1, s'_1),…, (s_j, s'_j) pairs, are subrelations of x_1, …, x_j respectively with their root node being the first different nodes along the paths to the common attributes A_1, …, A_j respectively.

The Recursive Nested Θ-Join Operation ($\triangleright\triangleleft_{\Theta}^{\triangleright\triangleleft}$)

The Θ-join operation is a special case of the join operation where the two relations are joined on the basis of some comparison operator other than equality.

It can be expressed by applying a selection operation to the result of the Cartesian product operation of two relations. The Cartesian product is applied at the top levels of the two nested relations and then, a recursive nested selection operation follows which compares two attributes in the resulting relation. The two attributes need not be at the same nesting

level in the resulting relation.

Let r and q be two nested (in general) relations with relation schemes R and Q respectively. Let also, X and Y be two atomic attributes belonging to relations R and Q respectively and Θ the condition that they must satisfy. Assume, without loss of generality, that Y belongs to a deeper nesting level than X and $L_{\sigma Y' \to Y}$ is the select path of Y starting at node Y′ which is at the same nesting level as X (when X and Y are at the same nesting level the select path is empty). So, the recursive nested Θ–join operation of the two relations r and q is defined as follows:

Definition 24 (*Recursive Nested Θ-Join)*

$r \rhd\lhd_{\Theta}^{\rhd\lhd} q = \sigma^{\sigma}((r \times q)_{X \Theta Y} L_{\sigma Y' \to Y})$ □

B. Functions

Aggregate functions for nested relations have not been discussed in any other model presented in section II, but in [18]. Aggregate functions are redefined below.

Let f be a nested aggregate function (f ϵ {N-MAX, N-MIN, N-SUM, N-AVG, N-COUNT}, where N-MAX, N-MIN, N-SUM, N-AVG and N-COUNT are the nested versions for the corresponding aggregate functions MAX, MIN, SUM, AVG and COUNT for flat relations), f′ an aggregate function for flat relations (f′ ϵ {MAX, MIN, SUM, AVG, COUNT}), r a nested relation, X an atomic or nested attribute at a lower nesting level of r, Par the parent attribute of atomic attribute Y of r (Y is at the same or higher nesting level than X and it is the attribute over which attribute X is summarised) and X/Y denotes that attribute X is summarised over attribute Y. Then, f[X/Y](r) is defined as follows:

Definition 25 (*Nested Aggregate Function*)

$f[X/Y](r) = f'(\{t_i[X] \mid t_i \in t, t \in Par(Y) \wedge t_i[X] \neq null\})$ □

Note: Attribute X can be a nested attribute only when the nested aggregate function f is N-COUNT. For all other cases, X attribute must be an atomic attribute.

For an example see Query 6 in section VI.

VI. Management of Nested Data

The nested algebra presented in section V, is a well-defined and formalised nested algebra where data restructuring operations are avoided. In this section, examples are provided to show the ease of use of the NRA. Relations have no restrictions on the number of nesting levels they can contain. The nested model presented, provides a better way of representing and querying complex data as demonstrated by the queries that follow since they are compact and do not require nest, unnest or any other restructuring operations for the manipulation of nested data.

A number of examples are presented that contain only operations on nested data, demonstrating how this model works and functions. Queries refer to the nested database example described in section III (Fig. 1). For some queries, comparisons are made with other proposed models.

Query 1: What are the descriptions of the units that belong to department 1 and who are the trainers who have given courses to staff members of these units? Display also the value for the department.

$\pi^{\pi}((\sigma^{\sigma}(DEPT_{D=1}))$ D, UD, TRN)

A projection operation on a selected part of the DEPT relation is needed to answer the above query. Three attributes of the relation are projected which can be found at different nesting levels; attribute D at nesting level 1, attribute UD at nesting level 2 and attribute TRN at nesting level 3. However, the projection operation takes place as normal, without changing the structure of the relation using unnest and nest operations and thus, the nesting arrangement of the relation is maintained in the resulting relation as well. Therefore, in the resulting relation, D, UD and TRN are still at nesting levels 1, 2 and 3 respectively, as in the input relation DEPT.

Query 2: Find the tuples with course numbers equal to the number of the department for the whole tuple.

$\sigma^{\sigma}(DEPT_{D=CN})$

The above query shows the advantage of the selection operation proposed in section V that allows arbitrary expressions to be specified in the select condition, as for example equality of values of attributes that are not at the same nesting level in the relation, without unnesting and nesting the relation. The query is expressed algebraically in exactly the same way as if the two compared attributes were at the top level of the original relation.

Query 3: Find the names of the banks and the companies that are situated at the same road.

$\nu^{\nu}((\mu^{\mu}(\pi^{\pi}((LOCATION \rhd\lhd^{\rhd\lhd} CASH\text{-}POINT)$

$COMPANY,BANK,ADDRESS))_{(ADDRESS)})_{(COMPANY,BANK)\to(COMPANY\ BANK)})$

In this example, and in similar cases, nest and unnest operations are necessary since they can restructure the relations and as a result, present the same data in a different format that is required by the given query.

However, extra nest and unnest operations are avoided in the above query since the natural join and projection operations are defined recursively in the NRA.

In Abiteboul and Bidoit's model [1] this query cannot be performed since the two relations that participate in the natural join operation do not have any common attributes at the top level.

Query 4: Find the names of the trainers that have given the "Computer Skills" training course.

$\pi^{\pi}((\sigma^{\sigma}(TRAINING \rhd\lhd^{\rhd\lhd} COURSE)_{TITLE= \text{"Computer Skills"}})$ TRN)

One can easily see the advantage of joining subrelations which are at different nesting levels (in this example, the subrelation C at nesting level 3 in relation TRAINING and at nesting level 1 in relation COURSE), without the need to unnest and nest the data and without any other restructuring operations assumed by other proposed models (e.g., [1], [13]). The above example shows that NRA provides a simple way of answering queries, since even just the algebraic solution of the query can be translated naturally to the above well-phrased query; moreover, the query does not distinguish between

nested and flat relations, as the query would be expressed in the same way if the two relations, TRAINING and COURSE, were flat relations. This is explained by the recursive nature of the NRA operations.

In contrast, in Levene's model [14] the natural join can be applied only if relation COURSE is extended with two empty nodes at levels 1 and 2 so that the common attribute C to appear at the same nesting level 3 in both relations. Then, the two relations are *joinable*, according to Levene's definition and therefore, can be joined.

Query 5: Find the names of the banks which are located on the same road as the companies for which Tim or Karen have worked for, together with the names of these companies.

$\pi^{\pi}((((\sigma^{\sigma}(\text{TRAINING}_{(\text{TRN= "Tim" OR TRN = "Karen"})})) \rhd\lhd^{\rhd\lhd}$ LOCATION) $\rhd\lhd^{\rhd\lhd}$ CASH-POINT) COMPANY, BANK)

This query requires two natural join operations. However, since the natural join defined in [23] can be performed between any possible relations sharing common attributes, it does not involve any preliminary checks to determine if the two operand relations are qualified for the natural join. In other models, for example in [1], [13], it is not certain if the natural join operation can be performed between a nested relation and the output of the natural join of two nested relations, since, as explained in section II of this paper, for each of these models the natural join operation is subject to some restrictions. On the other hand, in NRA any possible combination of relations, sharing at least one common attribute, can be joined.

This query also demonstrates how complex queries can be answered easily in the query language proposed in this paper.

Query 6: What is the title of the course that has the maximum number of different topics? Display also the number of different topics that this course has.

π^{π}(COURSE(TITLE, N-COUNT[TOPICS/TITLE]←MTOPICS)) $\rhd\lhd^{\rhd\lhd}$

$\pi^{\pi}(\pi^{\pi}$(COURSE(TITLE, N-COUNT[TOPICS/TITLE]←MTOPICS1))

(MAX(MTOPICS1)) ←MTOPICS)

Aggregate functions for nested attributes have been defined in subsection B of section V.

The above query is expressed in the NRA using the following steps:

1. In the original relation COURSE, the number of different topics per title is computed, it is named MTOPICS1 and projected on TITLE and MTOPICS1 attributes.

2. From the result of step 1, MAX (MTOPICS1) is computed, named MTOPICS and projected.

3. In the original relation COURSE, the number of different topics per title is computed, it is named MTOPICS and projected on TITLE and MTOPICS.

4. The results of steps 2 and step 3 are joined together.

It is noteworthy that if the relation COURSE was a flat relation then, the SUMMARIZE operation would be used to produce the same result in combination with the traditional aggregate functions COUNT and MAX.

It must be said that this query or any other query containing aggregate functions on nested attributes cannot be expressed in any other relational model discussed in section II apart from [18] with the use of an additional operator, the subrelation constructor, as follows:

π[TITLE, MAX[SUBJECT′]] ($\oint$ (C, COURSE_DURATION, TITLE, SUBJECT, SUBJECT′); SUBJECT′ := COUNT[TOPICS](SUBJECT) $\oint$ (COURSE))

where $\oint$ is the subrelation constructor.

Query 7: Find all trainers who have given more courses than Karen has.

$\pi^{\pi}((\sigma^{\sigma}$

$(\pi^{\pi}((\nu^{\nu}(\mu^{\mu}(\pi_t^{\pi}(\text{TRAINING(TRN,COURSE)})_{\text{COURSE}})$

$_{\text{(CODE, C)}\rightarrow\text{COURSE}}))$(TRN,N-COUNT[CN/TRN]←MCN))

$\times^{\times}$ $\pi^{\pi}((\sigma^{\sigma}(\text{TRAINING}_{\text{TRN= "Karen"}}))$

(N-COUNT[CN/TRN]←MCN1)))$_{\text{MCN > MCN1}}$) TRN)

Two copies of the TRAINING relation are needed for this query in order to perform the Cartesian product operation between them. However, to make the query simpler, a projection operation is applied to the first copy of the relation and an aggregate function is also used to count the number of nested tuples which corresponds to the number of different courses that each trainer (TRN) has given. Moreover, an unnest and then a nest operation are also used to a projected part of the original relation to convert the relation to the right one, before the computation of the aggregate function. With the second copy of the relation, a projection is performed on a selection of the relation. The same aggregate function is also used here, applied to the same attribute as before. The Cartesian product is performed afterwards between a binary relation and a unary one containing only one tuple.

Once again, the above query can demonstrate the expressive power of the proposed nested model and the facility in stating complex queries. This query, as the previous one, cannot be expressed in any other nested relational model presented in section II apart from [18], yet with the problem discussed above.

VII. The Nested Relational Model (NRA)

The components of the NRM are described below.

A. Data types-Domains

Domains are data types of arbitrary internal complexity ([22]). Therefore, such domains can consist of relation-type values. Attributes defined on that domains are relation-valued attributes, that is, they contain values that are relations. The domain of a nested attribute is defined recursively below.

Assume that R_{n1}, R_{n2}, …, R_{nk} are, in general, all the atomic and nested attributes that belong to nested attribute R_n and P is the powerset of a set S.

Definition 26 (*Nested attribute domain*) The domain of a nested attribute R_n, DOM(R_n), is defined recursively as

i) DOM(R_n) $\subseteq$ D, where D is the underlying database domain, for the special case where R_n is an atomic attribute.

ii) $DOM(R_{n1}) \times DOM(R_{n2}) \times \ldots \times DOM(R_{nk})$, for $k \geq 1$, where $R_{n1}, R_{n2}, \ldots, R_{nk}$ are atomic attributes of R_n.

iii) $P(DOM(R_{n1})) \times P(DOM(R_{n2})) \times \ldots \times P(DOM(R_{nk}))$, for $k \geq 1$, where $R_{n1}, R_{n2}, \ldots, R_{nk}$ are nested attributes of R_n, in general.

B. Databases

In the NRM, databases are sets of nested relations. Nested relations do not satisfy the 1NF assumption. A database example in the NRM is shown in Fig. 1.

C. Structures

Definition 27 (*Nested Relation Scheme*) The scheme of a relation R in the NRM is defined recursively as $RS = R(R_1S_1, R_2S_2, ..., R_nS_n)$, where $n \geq 1$, $R_1, R_2, ..., R_n$ are the attribute names of R, either atomic or nested and

$$S_i = \begin{cases} \varnothing \text{ (empty set)} & \text{if } R_i \text{ is an atomic attribute} \\ (R_{i1}S_{i1}, R_{i2}S_{i2}, ..., R_{ik}S_{ik}) & \text{if } R_i \text{ is a nested attribute and } k \geq 1 \end{cases}$$

where $1 \leq i \leq n$.

Example 15: The scheme of relation TRAINING (Fig. 1) is TRAINING (COMPANY TRAINER (TRN COURSE (CODE C (CN Y)))).

D. Relational Operators

The set of conventional relational comparison operators of the CRM, $\{=, \neq, <, \leq, >, \geq\}$, is also supported in the NRM.

E. Operations

The union, difference, intersection, projection, selection, rename, Cartesian product, natural join and Θ-join recursive operations of the NRM have been defined formally in section V. Two additional operations, nest and unnest, have also been defined in the NRM.

F. Functions

The set of functions in the CRM is also supported in the NRM.

VIII. Mapping the CRM to the NRM

NRM is reduced to the CRM when restricted to support only flat relations, in a way similar to the approach of Paredaens and Van Gucht [24]. In this section, the components of the CRM are going to be mapped to the NRM that have been described in section VII, in order to prove that the NRM is a proper superset of the CRM.

A. Data types - Domains

Proposition 1: The set of domains in the CRM is a proper subset of the set of domains in the NRM.

Proof: The nested attribute domain is defined recursively (Definition 26). Therefore, for the special cases i) where k=0 i.e., the attribute is atomic or ii) where $k \geq 1$ i.e., the attribute is nested consisting of atomic attributes only (which can be considered as a flat relation), the nested attribute domain definition of the NRM is reduced to the atomic attribute domain definition of the CRM.

Consequently, since the set of domains in the NRM can be reduced, for specific special cases, to the set of domains in the CRM, the former is a proper superset of the set of domains in the CRM.

B. Databases

Proposition 2: The set of databases in the CRM is a proper subset of the set of databases in the NRM.

Proof: Databases in the NRM have been introduced in order to relax the 1NF assumption that is satisfied in the CRM. Thus, the 1NF assumption of flat relations is a special case of the general N1NF assumption which characterises relations in the NRM. By definition, a flat relation is also a relation of the nested model. Therefore, the set of databases in the NRM is a proper superset of the set of databases in the CRM.

C. Structures

Proposition 3: The set of structures in the CRM is a proper subset of the set of structures in the NRM.

Proof: The definition of the scheme in the NRM is given recursively (Definition 27). For the special case, where S_i, for all i, is equal to the empty set, the definition is reduced to that of the CRM, since all attributes of the relation are atomic.

D. Relational Operators

Proposition 4: The set of relational comparison operators in the CRM is isomorphic to the set of relational operators in the NRM (i.e., for every comparison operator in the CRM there is a corresponding comparison operator in the NRM).

Proof: The proof is omitted for obvious reasons.

E. Operations

In the following, it is shown by a number of propositions that each operation in the NRM is an extended operation of the relevant operation in the CRM. Before this is done, some preliminary discussion is necessary, regarding the effect of relational operations to the key of relations.

Let *Unary* be a unary operation and let $R_1 = Unary(R_0)$. Then, the first obvious remark is that this operation does not have any effect on the key of R_0 i.e., the key of R_0 remains the same. The second one is that the key of R_0 is not inherited to R_1. These observations apply to any data model, and to the CRM as well. As an example of the second remark, consider a flat relation R_0 and assume that K is its primary key. Then, the CRM select operation $R_1=\sigma_F(R_0)$, also yields a flat relation, R_1. Since R_1 is a subset of R_0, it follows that it does not contain two distinct tuples with identical values for K. However, it is not implied by this fact that K is also the key of R_1, it is only the user who may specify what the key of R_1 is.

As another example, let the scheme of R_0 be $R_0(K, A, B)$, where K is its key. If $R_1=\pi_{A,B}(R_0)$, it is known that R_1 does not contain duplicate tuples and, definitely, it is again the user who may specify its key.

Hence, the conclusion is that a unary CRM operation does not affect the key (if defined) of the input relation and it does

not propagate it to the result relation. This same conclusion can also be drawn for binary operations of the CRM. Subsequently, the same conclusion can be drawn for any operation in any data model, therefore for all the operations of the NRM as well.

Proposition 5: The union operation in the NRM is an extended version of the union operation in the CRM.

Proof: The union operation in the NRM is defined recursively (Definition 9). From the recursive definition, it is deduced that for the special case where the relations are in 1NF format, the definition is reduced to the non-recursive union definition for flat relations (case i), since the relations do not contain any nested attributes. This definition then, is the definition of the union operation in the CRM.

Proposition 6: The difference operation in the NRM is an extended version of the difference operation in the CRM.

Proof: The proof is similar to that of Proposition 5.

Proposition 7: The intersection operation in the NRM is an extended version of the intersection operation in the CRM.

Proof: The proof is similar to that of Proposition 5.

Proposition 8: The projection operation in the NRM is an extended version of the projection operation in the CRM.

Proof: From Definition 13 (case ii):

$\pi^{\pi}(r(R_{a1}, \ldots, R_{ak}, R_{n1}L_{n1}, \ldots, R_{nm}L_{nm})) = \{ t| (\exists\, t_r \in r)$
$((t[R_{a1}] = t_r[R_{a1}]) \wedge \ldots \wedge (t[R_{ak}] = t_r[R_{ak}])$
$\wedge\ (t[R_{n1}] = \pi^{\pi}(t_r[R_{n1}]L_{n1})) \wedge \ldots \wedge (t[R_{nm}] = \pi^{\pi}(t_r[R_{nm}]L_{nm}))\}$.

For the special case where relation r is flat, since all attributes of relation r are atomic, $R_{ni}L_{ni} = \varnothing$, for all i ($1 \le i \le m$), and the definition of the projection operation is reduced to:

$\pi^{\pi}(r(R_{a1}, \ldots, R_{ak}, R_{n1}L_{n1}, \ldots, R_{nm}L_{nm})) = \pi^{\pi}(r(R_{a1}, \ldots, R_{ak})) =$
$\{ t| (\exists\, t_r \in r) ((t[R_{a1}] = t_r[R_{a1}]) \wedge \ldots \wedge (t[R_{ak}] = t_r[R_{ak}]))\}$,

which is the definition of the projection operation in the CRM. So, the projection operation in the NRM is an extended version of the projection operation in the CRM.

Proposition 9: The selection operation in the NRM is an extended version of the selection operation in the CRM.

Proof: The proof is similar to that of Proposition 8. For the special case where relation r is flat, since all attributes of relation r are atomic, L is empty and Definition 15 is reduced to $\sigma^{\sigma}(r_cL_{\sigma}) = \sigma(r_c) = \sigma(r_{ca1, \ldots, cak})$ which is the traditional selection operation for flat relations in the CRM.

Proposition 10: The rename operation in the NRM is an extended version of the rename operation in the CRM.

Proof: From Definition 20-case (ii), the rename of a nested attribute at the top level of a relation is:

$\rho[A \leftarrow A'](R) = \{R_1, R_2, \ldots, R_i, \ldots, R_n, \bigcup_{k=0}^{m} L_{A' \to Ak}, B, \ldots, Z\}$.

This definition is reduced to:

$\rho[A \leftarrow A'](R) = \{R_1, R_2, \ldots, R_i, \ldots, R_n, A', B, \ldots, Z\}$, for the special case where the attribute to be renamed, A, is an atomic attribute at the top level of relation R, since $\bigcup_{k=0}^{m} L_{A' \to Ak} = A'$ (m=0 i.e., there are not any descendants of A). This is equivalent to the rename operation in the CRM.

Proposition 11: The Cartesian product operation in the NRM is an extended version of the Cartesian product operation in the CRM.

Proof: Case (i) or case (ii) for L=Ø of Definition 22 is the traditional Cartesian product operation for flat relations in the CRM.

Proposition 12: The natural join operation in the NRM is an extended version of the natural join operation in the CRM.

Proof: The natural join which operates for cases where the common atomic or nested attributes belong to different subrelations and at different nesting levels in the two relations), $\rhd\lhd^{\rhd\lhd}$ (rL, qM), is defined in [23].

The natural join can be reduced to the conventional natural join for flat relations if the special case is assumed, where the common attributes are atomic attributes at the top level of the two relations. Formally, the definition for L and M empty, is reduced to:

$\rhd\lhd^{\rhd\lhd}$ (rL, qM) = $\rhd\lhd$ (r, q) = $\{ t| (\exists\, t_r \in r) (\exists\, t_q \in q)$
$((t[Attr(R_i)] = t_r[Attr(R_i)])$
$\wedge\ (t[Attr(Q_i)] = t_q[Attr(Q_i)])$
$\wedge\ (t[R_{i1}] = t_r[R_{i1}] = t_q[Q_{i1}]))\}$

which is the traditional definition of the natural join operation in the CRM.

F. Functions

Proposition 13: The set of functions in the CRM is isomorphic to the set of functions in the NRM.

Proof: The proof is omitted for obvious reasons.

G. Mapping Synopsis

Proposition 14: The NRM is a superset of the CRM.

Proof: This is a result of Propositions 1-13 since, as it has been explained in subsection B of section VIII, in order to prove that a database model is a superset of another database model, it is necessary and sufficient to prove that every property of the latter (data types, databases, structures, operators, operations and functions) is also a property of the former.

IX. Conclusion

In this paper, an algebra (NRA) and a database model (NRM) have been defined for nested relations of arbitrary nesting levels. All the operators have been recursively defined. As a result, there is no need to flatten the nested relations when a series of operations are executed and so the data redundancy and duplication caused by unnesting relations is avoided. Furthermore, the representation of the data is claimed to be in a "natural form". Thus, it is easier for users to understand when working with the data, since even complex objects can be modelled in one relation. A number of example queries have been expressed in the NRA to demonstrate its

functionality. In addition, the model has been proved to be consistent with the CRM.

In appendix I, a formal syntax of the NRA is given. In appendix II, a sample code of the prototype implementation can be found as well as a number of examples, presented in this paper, coded in Miranda, which demonstrate the functionality and validity of the model. It should be stated that the prototype serves satisfactory as a proof of concept. Specifically, the nested rename, projection, selection and Cartesian product operations have been fully implemented. The nested join operation has been partially implemented. Particularly, only one nested column is allowed at each nesting level and the join operator allows joining on only one pair of columns; reasonable assumptions within the framework of a prototype.

Future work includes the study of optimisation techniques for the efficient evaluation of complex queries. The definition of an extension of SQL to support the nested features of NRM is also, another research direction. The incorporation of spatial data to NRM is an additional challenge. The NRM can also be used as a basis to build an algebra for supporting nesting structures in XML (similarly to the FLWR expressions of XQUERY).

APPENDIX I

Formal syntax of the NRA

expression
:: = one-relation-expression| two-relation-expression
one-relation-expression
:: = nested-renaming | nested-selection | nested-projection
two-relation-expression
:: = nested-projection binary-operation expression
nested-renaming
:: = ρ_t^{ρ} [attribute-commalist1] (term)
attribute-commalist1
:: = fattribute ← fattribute | fattribute ← fattribute, attribute-commalist1
fattribute
:: = attribute1 | function2(attribute1)
attribute1
:: = attribute | nested-aggregate-attribute
attribute
:: = basic-attribute | nested-attribute
basic-attribute
:: = atomic-attribute
nested-aggregate-attribute
:: = function1[attribute/basic-attribute]
function1
:: = N-MAX | N-MIN | N-SUM | N-COUNT | N-AVG
function2
:: = MAX | MIN | AVG | COUNT | SUM
term
:: = relation | (expression)
nested-projection
:: = π_t^{π} (term (attribute-commalist2)) | term
attribute-commalist2
:: = fattribute | fattribute, attribute-commalist2
binary-operation
:: = $\cup^{\cup}$ | $\cap^{\cap}$ | $-^{-}$ | $\times^{\times}$ | $\triangleright\triangleleft^{\triangleright\triangleleft}$ | $\triangleright\triangleleft_{\Theta}^{\triangleright\triangleleft}$
nested-selection
:: = σ^{σ}(term $_{comparison}$)
comparison
:: = attribute-term | attribute-term logical-operator comparison
logical-operator
:: = AND | OR | AND NOT | OR NOT
attribute-term
:: = FAA θ FAA
FAA
:: = constant | atomic-attribute | attribute-term | nested-aggregate-attribute
θ
:: = < | > | = | <= | >= | ≠

APPENDIX II

A sample of the prototype implementation

A small part of the code, that has been developed in Miranda, is listed in this section which contains basic functions for selection, projection and Cartesian product operators.

```
|| isColumnTag: Simple method to identify a column using its tag.
isColumnTag :: string -> columnType -> bool

|| resolvePath: Creates full path name for columns in a table.
resolvePath:: relationalTable -> relationalTable

|| resolvePath2: Creates full path names for a list of entries given a string and a depth.
resolvePath2 :: string -> num -> [tableEntry] -> [tableEntry]

|| resolvePath3: Creates full path names for a list of columns given a string and a depth.
resolvePath3 :: string -> num -> tableEntry -> tableEntry

|| resolvePath4: Creates full path names for a column given a string and a depth.
resolvePath4 :: string -> num -> columnType -> columnType

|| rpar: Generates a given number of closing parenthesis.
rpar :: num -> string

|| selectCol: Used to select a column recursively based on its tag.
selectCol :: string -> columnType -> tableEntry

|| selectEntryByStr: Selects all columns from a list of
```

columns using the given column tag.
selectEntryByStr :: string -> tableEntry -> tableEntry

|| selectEntryByStrLst: Selects a list of columns whose names are provided by a list of string
|| tags.
selectEntryByStrLst :: [string] -> tableEntry -> tableEntry

|| selectEntryLstByStrLst: Selects entries from a given entry list
|| whose names are provided by a list of tags.
selectEntryLstByStrLst :: [string] -> [tableEntry] -> [tableEntry]

|| tableProjection2: Selects a subset of table entries based on given
|| column names after all recursive column names have been resolved.
tableProjection2 :: [string] -> relationalTable -> relationalTable

|| flattenRelTable: Flattens a relational table by calling helper
|| function flattenEntryList.
flattenRelTable :: relationalTable -> relationalTable

|| flattenEntryList: Flattens a list of table entries by calling helper function flattenColumnList.
flattenEntryList :: [tableEntry] -> [tableEntry]

|| flattenColumnList: Flattens a list of columns by calling helper function flattenColumn.
flattenColumnList :: [columnType] -> [columnType]

|| flattenColumn: Flattens recursive columns.
flattenColumn :: columnType -> [columnType]

|| tableProduct2: Product of two relational tables at the top level.
|| All possible combinations of table entries are included.
|| No recursive application involved.
tableProduct2 :: relationalTable -> relationalTable -> relationalTable

|| tableProduct3: Product of a table with an inner table of another table.
|| All possible combinations of table entries are included.
tableProduct3 :: relationalTable -> (string, relationalTable) -> relationalTable

|| tableProduct4: Applies the product of a table to a list of table entries
|| for recursive application.
tableProduct4 :: relationalTable -> string -> [tableEntry] -> [tableEntry]

|| tableProduct5: Applies the product of a table to a list of table columns
|| for recursive application.
tableProduct5 :: relationalTable -> string -> tableEntry -> tableEntry

|| tableProduct6: Applies the product to a recursive column which holds the
|| required inner table.
tableProduct6 :: relationalTable -> columnType -> columnType

|| tableProduct7: Applies the product to a recursive column if the inner table is
|| the one required.
tableProduct7 :: relationalTable -> string -> columnType -> columnType

|| getRecTableNames: Gets the name of a table and calls getRecTableNames2
|| to get the names of all recursive tables.
getRecTableNames :: relationalTable -> [string]

|| getRecTableNames2: Gets table names recursively for a list of table entries.
getRecTableNames2 :: [tableEntry] -> [string]

|| getRecTableNames3: Gets table names recursively for a list of columns.
getRecTableNames3 :: tableEntry -> [string]

|| getRecTableNames4: Gets the name of a table column if it is an inner table.
getRecTableNames4 :: columnType -> [string]

Examples

The numbering of the queries below refers to the example queries found in section VI.

Query 1:

selectFrom ["D", "DEPT(UNIT(UD))", "DEPT(UNIT(COURSE_DETAILS(TRN)))"][("
D", NF ((=) 1))]d

Query 4:

selectFrom["COURSE/TRAINING(COURSE/TRAINER(TRN))"][("COURSE/TRAINING(COURSE/TRAINER(COURSE/COURSE(TITLE)))", SF ((=) "Computer Skills"))] (joinTables("C", t)("C", course))

Query 5:

A revised version of the query is given, due to the fact that relevant implementation is missing (only for 'Karen', since OR has not been implemented).

tableProjection["COMPANY","BANK"] (joinTables("TRAINING/LOCATION(ANNEX(ADDRESS))",

joinTables("COMPANY",selectFrom["COMPANY"][("TRAINING(TRAINER(TRN))", SF((=)"Karen"))]t)("COMPANY",location))("CASH-POINT(BRANCH(ADDRESS))", cashpoint))

REFERENCES

[1] S. Abiteboul, and N. Bidoit, "Non First Normal Form Relations: An Algebra Allowing Data Restructuring," *Journal of Computer and System Sciences*, vol. 33, no. 3, pp. 361-393, 1986.

[2] P.C. Fisher, and S.J. Thomas, "Operators for Non-First-Normal Form Relations," in *Proc. of the 7th IEEE International Conference on Computer Software and Applications*, Chicago, 1983, pp. 464-475.

[3] G. Jaeschke, and H.J. Schek, "Remarks on the Algebra of Non First Normal Form Relations", in *Proc. of the ACM Symposium on Principles of Database Systems*, Los Angeles, 1982, pp. 124-138.

[4] M.A. Roth, H.F. Korth, and A. Silberschatz, "Extended Algebra and Calculus for Nested Relational Databases," *ACM Transactions on Database Systems*, vol. 13, no. 4, pp. 389-417, 1988.

[5] H.-J. Schek, and M.H. Scholl, "The Relational Model with Relation-Valued Attributes," *Information Systems*, vol. 11, no. 2, pp. 137-147, 1986.

[6] S.J. Thomas, and P.C. Fischer, "Nested Relational Structures," *International Journal of Artificial Intelligence*, vol. 3, pp. 269-307, 1986.

[7] A. Makinouchi, "A consideration on Normal Form of Not-Necessarily-Normalized Relations in the Relational Data Model," in *Proc. of the 3rd International Conference on Very Large Data Bases*, Tokyo, 1977, pp. 447-453.

[8] V. Tannen, "Tutorial: Languages for Collection Types," in *Proc. of the 13th ACM Symposium on Principles of Database Systems*, Minneapolis, 1994, pp. 150- 154.

[9] N.A. Lorentzos, and A. Dondis, "Query by Example for Nested Tables," in *Proc. of the 9th International Conference on Database and Expert Systems Applications*, Vienna, 1998, pp. 716-725.

[10] M.A. Roth, H.F. Korth, and D.S. Batory, "SQL/NF: A Query Language for ¬1NF Relational Databases," *Information Systems*, vol. 12, no. 1, pp. 99-114, 1987.

[11] L. Wegner, S. Thelemann, S. Wilke, and R. Lievaart, "QBE-like Queries and Multimedia Extensions in a Nested Relational DBMS," in *Proc. of the International Conference on Visual Information Systems*, Melbourne, 1996, pp. 437-446.

[12] G. Özsoyoglu, Z.M. Özsoyoglu, and V. Matos, "Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions," *ACM Transactions on Database Systems*, vol. 12, no. 4, pp. 566-592, 1987.

[13] L.S. Colby, "A Recursive Algebra for Nested Relations," *Information Systems*, vol. 15, no. 5, pp. 567-582, 1990.

[14] M. Levene, "The Nested Universal Relation Database Model," Lecture Notes in Computer Science 595, Berlin: Springer-Verlag, 1992.

[15] Hong-Cheu Liu, and K. Ramamohanarao, "Multiple Paths Join for Nested Relational Databases." in *Proc. of the 5th Australian Database Conference*, 1994, pp. 30-44.

[16] M. Levene, and G. Loizou, "Correction to Null Values in Nested Relational Databases by M.A. Roth, H.F. Korth and A. Silberschatz," *Acta Informatica*, vol. 28, pp. 603-605, 1991.

[17] A. Tansel, and L. Garnett, "On Roth, Korth, and Silberschatz's Extended Algebra and Calculus for Nested Relational Databases," *ACM Transactions on Database Systems*, vol. 17, no. 2, pp. 374-383, 1992.

[18] V. Deshpande, and P.A Larson, "An Algebra for Nested Relations with Support for Nulls and Aggregates," Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Tech. Rep. CS-91-16, 1991.

[19] Hong-Cheu Liu, and K. Ramamohanarao, "Algebraic Equivalences among Nested Relational Expressions," in *Proc. of the 3rd International Conference on Information and Knowledge Management*, Gaithersburg, 1994, pp. 234-243.

[20] P. Buneman, S. Naqvi, V. Tannen, and L. Wong, "Principles of Programming with Complex Objects and Collection Types," *Theoretical Computer Science*, vol. 149, no. 1, pp. 3-48, 1995.

[21] J.D. Ullman, *Principles of Database and Knowledge-Base Systems*. New York: Computer Science Press, 1995.

[22] C.J. Date, *An Introduction to Database Systems*, 2nd ed. New York: Addison-Wesley, 2000.

[23] G. Garani, and R. Johnson, "Joining nested relations and subrelations," *Information Systems*, vol. 25, no. 4, pp. 287-307, 2000.

[24] J. Paredaens, and D. Van Gucht, "Converting Nested Algebra Expressions into Flat Algebra Expressions," *ACM Transactions on Database Systems*, vol. 17, no.1, pp. 65-93, 1992.

Georgia Garani received a BSc degree in physics from Aristotle University of Thessaloniki, Greece and MSc and Ph.D. degrees in computer science from King's College and Birkbeck College respectively, University of London, UK. She worked as a teaching assistant at Birkbeck College and as a visiting lecturer at the University of North London, UK. She is currently an assistant professor at the Department of Informatics and Telecommunications of the Higher Technological Educational Institute of Larisa, Greece. She is involved in a series of research projects. Her research interests include temporal and spatial databases, image databases and data mining.