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 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...
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

<table>
<thead>
<tr>
<th>COMPANY</th>
<th>TRAINER</th>
<th>COURSE DETAILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>Jack</td>
<td>xx0 1 75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xx1 1 52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xx2 2 79</td>
</tr>
<tr>
<td>IBM</td>
<td>Tim</td>
<td>xy1 1 82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xy2 2 79</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Karen</td>
<td>xx1 2 77</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xx2 2 81</td>
</tr>
</tbody>
</table>

### DEPT

<table>
<thead>
<tr>
<th>DEPT</th>
<th>UNIT</th>
<th>COURSE DETAILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>511 Software Engineering</td>
<td>Mark Apple 1 75</td>
</tr>
<tr>
<td></td>
<td>552 Basic Research</td>
<td>Karen Microsoft 1 82</td>
</tr>
<tr>
<td></td>
<td>578 Planning</td>
<td>Tim IBM 5 79</td>
</tr>
<tr>
<td>Development</td>
<td>650 Design</td>
<td>Karen Microsoft 1 75</td>
</tr>
<tr>
<td></td>
<td>780 Maintenance</td>
<td>Tim IBM 3 82</td>
</tr>
<tr>
<td></td>
<td>981 Planning</td>
<td>Mark Apple 2 81</td>
</tr>
</tbody>
</table>

### LOCATION

<table>
<thead>
<tr>
<th>COMPANY</th>
<th>ANNEX</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOSHIBA</td>
<td>North Building</td>
<td>Porchester Rd.</td>
</tr>
<tr>
<td>IBM</td>
<td>Maple House</td>
<td>Kendal Av.</td>
</tr>
<tr>
<td></td>
<td>Main Building</td>
<td>Danebury Rd.</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Pegasus House</td>
<td>Ashford St.</td>
</tr>
<tr>
<td></td>
<td>Queen’s Building</td>
<td>Park Rd.</td>
</tr>
</tbody>
</table>

### COURSE

<table>
<thead>
<tr>
<th>SUBJECT</th>
<th>TOPICS</th>
<th>COURSE_DURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Word</td>
<td>80</td>
</tr>
<tr>
<td>Excel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multimedia</td>
<td>Power Point</td>
<td>120</td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming</td>
<td>C++</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>JAVA</td>
<td></td>
</tr>
</tbody>
</table>

### CASH-POINT

<table>
<thead>
<tr>
<th>BANK</th>
<th>BRANCH</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barclays</td>
<td>386600</td>
<td>Ashford St.</td>
</tr>
<tr>
<td>NatWest</td>
<td>560045</td>
<td>Park Rd.</td>
</tr>
<tr>
<td></td>
<td>560038</td>
<td>Porchester Rd.</td>
</tr>
<tr>
<td>Lloyd’s</td>
<td>478202</td>
<td>Ashford St.</td>
</tr>
<tr>
<td></td>
<td>478210</td>
<td>Park Rd.</td>
</tr>
</tbody>
</table>

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 ⊆ U, where D_A is the domain of the attribute A. If D_A ⊆ 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 ≥ 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 attributes 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 \( L_{A_n \rightarrow A_j} \) 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, \( L_{A_n \rightarrow A_j} \) is defined as follows:

i) \( L_{A_n \rightarrow A_j} = A_n \) where \( A_j = A_n \)

ii) \( L_{A_n \rightarrow A_j} = A_n(L_{A_{n+1} \rightarrow A_j}) \), 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 \( \text{Attr}(R) = \{R_{a1}, R_{a2}, \ldots, R_{ap}, R_{n1}, \ldots, R_{nj}, \ldots, R_{nq}\} \) where:

- \( \{R_{a1}, R_{a2}, \ldots, R_{ap}\} \) are atomic attributes at nesting level 1 of relation \( R \) (p \( \geq 0 \)),
- \( \{R_{n1}, \ldots, R_{nj}, \ldots, R_{nq}\} \) are nested attributes at nesting level 1 of relation \( R \) (1 \( \leq i \leq q \)),
- \( R_{ni} = \begin{cases} R_{n1} & \text{if } k = 0 \\ R_{ni} & \text{for } k > 0 \end{cases} \)

\( m \) is the number of descendants’ attributes of nested attribute \( R_{mi} \).

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 \( \text{TRAINING} \) (Fig. 1) with tree structure in Fig. 2 is \( L_{\text{TRAINER} \rightarrow \text{CN}} = \text{TRAINER}(L_{\text{COURSE} \rightarrow \text{CN}}) = \text{TRAINER}(\text{COURSE}(\text{LC} \rightarrow \text{CN})) = \text{TRAINER}(\text{COURSE}(\text{LC} \rightarrow \text{CN})) \).

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 \( \text{COURSE} \) (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 \( \Theta \)-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 (\( \cup\cup \))**

The union of the two relations \( r \) and \( q \), \( r \cup \cup q \), is defined as follows:

**Definition 9 (Recursive Nested Union)**

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

\[
\begin{align*}
  r \cup q = \{ t | \exists t_r \in r : (t[Attr(R)] = t_r[Attr(R)]) \\
  \land (\exists t_q \in q : (t[Attr(R)] = t_q[Attr(R)])) \}
\end{align*}
\]

ii) Recursive union for nested relations (\( r \cup\cup q \))

\[
\begin{align*}
  r \cup\cup q = \{ t | (\exists t \in r) \land (\forall t_q \in q : (t[S(R)] \neq t_q[S(R)])) \\
  \land (\exists t_q \in q : (\exists t_r \in r : (t[S(R)] = t_r[S(R)])) \\
  \land (t[R_1] = t_r[R_1] \cup\cup t_q[R_1]) \land ... \land (t[R_n] = t_r[R_n] \cup\cup t_q[R_n])) \}
\end{align*}
\]

**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 (\(-\))**

\[
\begin{align*}
  TRAINING_1 - TRAINING_2
\end{align*}
\]

\[
\begin{align*}
  TRAINING_1 \cap \cap \cap TRAINING_2
\end{align*}
\]

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 \( r - q \)
  
  \( r - q = \{ t \mid (\exists t_r \in r) (\forall t_q \in q) (((\exists t_{\text{Attr}(R)}) = t_r[\text{Attr}(R)]) \land (t[R_a] = t_r[R_a]) \land \ldots \land (t[R_n] = t_r[R_n])) \lor \ldots \lor (t[S(R)] = t_r[S(R)]) \land (t[\text{Attr}(R)] = t_r[\text{Attr}(R)]) \} \)

- ii) Recursive difference for nested relations \( r' - q' \)
  
  \( r' - q' = \{ t \mid (\exists t_r \in r) (\forall t_q \in q) (((\exists t_{\text{Attr}(R)}) = t_r[\text{Attr}(R)]) \land (t[R_a] = t_r[R_a]) \land \ldots \land (t[R_n] = t_r[R_n])) \lor \ldots \lor (t[S(R)] = t_r[S(R)]) \land (t[\text{Attr}(R)] = t_r[\text{Attr}(R)]) \} \)

**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 shown in Fig. 4.

**The Recursive Nested Selection Operation (\( \sigma \))**

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 \( \{ c_{a1}, c_{a2} \} \) where \( c_{a1} = \{ c_{a11}, \ldots, c_{a1n} \} \) is a set of conditions which must be true for the atomic attributes \( R_{a1}, \ldots, R_{ak} \) of \( R \) respectively and \( c_{n1} = \{ c_{n11}, \ldots, c_{n1m} \} \) 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 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)** \( L_a \) is a project list of \( R \) if

- i) \( L_a \) is empty (the project list of an atomic attribute is empty).
- ii) \( L_a \) is of the form \( R_{a1}L_{a1}, \ldots, R_{am}L_{am} \), where \( L_{a1}, \ldots, L_{an} \) are project lists of nested attributes \( R_{a1}, \ldots, R_{am} \) respectively.

Then, the projection operation in a nested relation \( r \), \( \pi^L(r) \), 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) \( \pi^L(r) = r \)
- ii) \( \pi^L(r_{a1}, \ldots, r_{ak}, R_{a1}L_{a1}, \ldots, R_{am}L_{am}) = \{ t \mid (\exists t_r \in r) (((\exists t_{\text{Attr}(R)}) = t_r[\text{Attr}(R)]) \land (t[R_{a1}] = t_r[R_{a1}]) \land \ldots \land (t[R_{ak}] = t_r[R_{ak}]) \land (t[R_{n1}] = \pi^L(t[R_{n1}])] \land \ldots \land (t[R_{nm}] = \pi^L(t[R_{nm}])] \} \)

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

Definition 15 (Recursive Nested Selection)

\[ \sigma(rca_1, \ldots, cak) = \{ t | (\exists tr \in r) ((t[Attr(R) - \{Ra_1, \ldots, R ak\}] = tr[Attr(R) - \{Ra_1, \ldots, R ak\}]) \land (t[Ra_1] = tr[Ra_1]) \land c_1 = true) \land \ldots \land (t[Rak] = tr[Rak]) \land c_k = true) \} \]

\[ \sigma(rcn_1, \ldots, cnm) = \{ t | (\exists tr \in r) ((t[Attr(R) - \{Rn_1, \ldots, R nm\}] = tr[Attr(R) - \{Rn_1, \ldots, R nm\}]) \land (t[Rn_1] = \sigma(rcn_1)\cap cn_1 = \emptyset) \land \ldots \land (t[Rnm] = \sigma(rcn_m)\cap cn_m = \emptyset) \} \]

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

\[ \sigma(rcL) = \sigma(rca_1, \ldots, cak, cn_1, \ldots, cnm) = \sigma(rca_1, \ldots, cak) \cap \sigma(rcn_1, \ldots, cnm) \]

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.

The Recursive Unnest Operation (\(\mu\))

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

Definition 16 (Unnest List) \(L_o\) is an unnest list of \(R\) if it is of the form

\(R_o\), where \(R_o\) is a nested attribute of \(R\) at the top level.

\((R_1, \ldots, R_n)\) where \(L_o\) is a unnest list of the nested attribute \(R_1\).

Let Attr(R) be the set of all attributes of \(R\) and \(R_1\) a nested attribute of \(R\), at the top level of \(R\). Let also, \(t\) be a tuple in relation \(r\) and \(t\) a tuple in the result relation. Then, the unnest operation, \(\mu(t(r))\), is defined as follows (see also [13]):

Definition 17 (Recursive Unnest)

i) \(\mu(t(r_o)) = \{ t | (\exists t_e \in r) ((t[Attr(R) - R_o] = t_e[Attr(R) - R_o]) \land (t[R_o] = t_e[R_o])) \}

ii) \(\mu(t(L_o)) = \{ t | (\exists t_e \in r) ((t[Attr(R) - R_o] = t_e[Attr(R) - R_o]) \land (t[R_o] = t_e[R_o])) \}

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

The Recursive Nest Operation (\(\nu\))

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

Definition 18 (Nest List) \(L_r\) is a nest list of \(R\) if it is of the form

\(R_1, \ldots, R_n\) where \(L_r\) is a nest list of \(R_1\).

i) \((R_1, \ldots, R_n)\) where \(L_r\) is a nest list of \(R_1\).

ii) \((R_1, \ldots, R_n)\) where \(L_r\) is a nest list of the nested attribute \(R_1\).

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

Let also, \(t\) be a tuple in relation \(r\), \(t\) a tuple in the result relation and \(s\) a tuple of the new nested attribute \(A\). The nest operation, \(\nu(\text{TRAINING}_{\text{TRAINER(COURSE)}})\), is defined as follows (see also [13]):

Definition 19 (Recursive Nest)
The Recursive Nested Rename Operation ($\rho'$)

The rename operation takes a specified relation and returns another relation 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 a join operation. To perform such an operation, one can rename attributes in two relations having 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 $R = \{R_1, R_2, \ldots, R_m, A, B, \ldots, Z\}$, where $R_1, R_2, \ldots, R_m$ are atomic attributes and $A, B, \ldots, Z$ are nested attributes at the top level of relation $R$. Then, the rename operation, $\rho'$, of relation $r$ is defined as follows:

**Definition 20 (Recursive Nested Rename)**

i) Rename of an atomic attribute $R_i$ to $R_j$ at the top level of relation $R$

$$\rho[R_i \leftarrow R_j](R) = \{R_1, R_2, \ldots, R_j, \ldots, R_m, A, B, \ldots, Z\}$$

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

$$\rho[A \leftarrow A'](R) = \{R_1, R_2, \ldots, R_m, A_1, \ldots, Z\}$$

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 $R$

$$\rho'[A_i \leftarrow A'_i](R) = \{R_1, R_2, \ldots, R_m, A_1, \ldots, Z\}$$

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 $A_0 = A_i$).

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

$$\rho'[R_{a1} \leftarrow R'_{a1}, \ldots, R_{ak} \leftarrow R'_{ak}, R_{a1} \leftarrow R'_{a1}, \ldots, R_{mn} \leftarrow R'_{mn}, R_{i1} \leftarrow R'_{i1}, \ldots, R_{ip} \leftarrow R'_{ip}](R) = (p[R_{a1} \leftarrow R'_{a1}](p[R_{a2} \leftarrow R'_{a2}](\ldots(p[R_{a1} \leftarrow R'_{a1}](p[R_{a2} \leftarrow R'_{a2}](\ldots(p[R_{ai} \leftarrow R'_{ai}](p[R_{a1} \leftarrow R'_{a1}](p[R_{a2} \leftarrow R'_{a2}](\ldots))))))))))$$

where $R_{i1}, \ldots, R_{ip}$ are atomic attributes at the top level of relation $R$, $R_{a1}, \ldots, R_{mn}$ are nested attributes at the top level of relation $R$ and $R_{i1}, \ldots, R_{ip}$ are either atomic or nested attributes at lower levels (different, in general) of relation $R$ and $k, m, p \geq 0$. 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:

$$\rho'[UD \leftarrow UD', C \leftarrow C'](DEPT) = \rho[C \leftarrow C'(\rho'[UD \leftarrow UD')(DEPT)]) = (D, DN, UNIT, UNIT(UNIT(UNIT(UNIT(UNIT(UNIT)))))$$

The Recursive Nested Cartesian Product Operation ($\times'$)

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) $L = R_L L_i$ where $R_i$ is a nested attribute of $R$ and $L_i$ is a join path of $R_i$.

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. The join path is written in brackets and the symbols $\Downarrow$ and $\leftarrow$ are used to separate the levels.

**Example 13:** In relation DEPT (Fig. 1) an example of a join path is UNIT(COURSEDETAILS(TRACT)).

Let $r$ and $q$ be two nested (in general) relations with relation schemes $R$ and $Q$ respectively and let $\text{Attr}(R)$ be all the attributes (atomic and nested) of $R$, $\text{Attr}(Q)$ 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 $R_i$, $t$ a tuple in relation $r$, $q$ 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.
So, the Cartesian product of two relations \( r \) and \( q \) is defined as follows [13]:

**Definition 22 (Recursive Nested Cartesian Product)**

\[
\times (r, q) = \{ t | (\exists t_r \in r) (\exists t_q \in q) ((t[Attr(R)] = t_r[Attr(R)]) \land (t[Attr(Q)] = t_q[Attr(Q)])) \}
\]

...
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 \( \Theta \) the condition that they must satisfy. Assume, without loss of generality, that \( Y \) belongs to a deeper nesting level than \( X \) and \( L_{Y \rightarrow X} \) 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 \( \Theta \)-join operation of the two relations \( r \) and \( q \) is defined as follows:

**Definition 24 (Recursive Nested \( \Theta \)-Join)**

\[
\begin{align*}
r \bowtie_{q \rightarrow R} \Theta \qquad & = \qquad q''(r \times q) \quad \Theta \left( r_{x \rightarrow Y} \right) \end{align*}
\]

**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 (\( f \in \{ N-MAX, N-MIN, N-SUM, N-AVG, N-COUNT \} \)), 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 (\( f' \in \{ MAX, MIN, SUM, AVG, COUNT \} \)), \( r \) a nested relation, \( X \) an atomic or nested attribute at a lower nesting level of \( r \), \( Par(Y) \) 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)**

\[
\begin{align*}
f[X/Y](r) & = f'(\{t[X] \mid t \in Part(Y) \land t[X] \neq \text{null}) \end{align*}
\]

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.

\[
\pi''(\sigma''(DEPT_{D = 1}) \text{D, UD, TRN})
\]

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.

\[
\sigma''(DEPT_{D = CN})
\]

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.

\[
\nu''((\pi''((\sigma''((\Theta_{\text{LOCATION} > \text{CASH-POINT}}) \text{COMPANY,BANK,ADDRESS}) \text{ADDRESS})\text{COMPANY,BANK})))
\]

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.

\[
\pi''((\sigma''(TRAINING \bowtie_{COURSE} \text{COMPANY,BANK})) \text{COMPANY BANK}))
\]

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 \( \text{TRAINING} \) and at nesting level 1 in relation \( \text{COURSE} \)), 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.

\[
\pi_{\text{LOCATION}}((\pi_{\text{TRN}}(\text{TRAINING}_{\text{TRN} = \text{"Tim"}} \text{ OR TRN = "Karen"}))) \bowtie \pi_{\text{CASH-POINT}}(\text{COMPANY, BANK})
\]

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.

\[
\pi_{\text{TITLE}}(\text{COUNT}[\text{TOPICS/TITLE}] \leftarrow \text{MTOPICS})) \bowtie \pi_{\text{MTOPICS}}(\text{COUNT}[\text{TOPICS/TITLE}] \leftarrow \text{MTOPICS1})
\]

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 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:

\[
\pi_{\text{TITLE}, \text{MAX}[\text{SUBJECT}]}(\text{COURSE}_{\text{DURATION}, \text{TITLE}, \text{SUBJECT}, \text{SUBJECT}}); \text{SUBJECT} := \text{COUNT}[\text{TOPICS}](\text{SUBJECT} \leftarrow \text{COURSE})
\]

where \( \leftarrow \) is the subrelation constructor.

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

\[
\pi_{\text{TRAINING}}(\text{TRN} = \text{"Tim"} \text{ OR TRN} = \text{"Karen"}) ((\text{TRAINING}_{\text{TRN} = "Karen"}) ^{\pi_{\text{COURSE}}}) (\text{TRAINING}_{\text{TRN} = "Karen"})
\]

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 \( R_{n1}, R_{n2}, \ldots, R_{nk} \) 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 \( R_n \), \( \text{DOM}(R_n) \), is defined recursively as:

1. \( \text{DOM}(R_n) \subseteq D \), where \( D \) is the underlying database domain, for the special case where \( R_n \) is an atomic attribute.
ii) $\text{DOM}(R_{n_1}) \times \text{DOM}(R_{n_2}) \times \ldots \times \text{DOM}(R_{n_k})$, for $k \geq 1$, where $R_{n_1}, R_{n_2}, \ldots, R_{n_k}$ are atomic attributes of $R_n$.

iii) $\text{P}(\text{DOM}(R_{n_1})) \times \text{P}(\text{DOM}(R_{n_2})) \times \ldots \times \text{P}(\text{DOM}(R_{n_k}))$, for $k \geq 1$, where $R_{n_1}, R_{n_2}, \ldots, R_{n_k}$ 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 $RS = R(R_1S_1, R_2S_2, \ldots, R_nS_n)$, where $n \geq 1$, $R_1, R_2, \ldots, R_n$ are the attribute names of $R$, either atomic or nested and

$$S_i = \begin{cases} \emptyset & \text{if } R_i \text{ is an atomic attribute} \\ \{ (R_1S_{i_1}, R_2S_{i_2}, \ldots, R_nS_{i_n}) \mid \text{if } R_i \text{ is a nested attribute and } k \geq 1 \} & \text{where } 1 \leq i \leq n. \end{cases}$$

**Example 15:** The scheme of relation TRAINING (Fig. 1) is $\text{TRAINING (COMPANY TRAINER (TRN COURSE (CODE (CN Y))))}.$

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 $\Theta$-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.
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):

\[
\pi_{\text{tr}(\text{R})}(\text{R}) = \{ \text{t} | (\exists \text{t}_r \in \text{r}) \land (\forall \text{t}_q \in \text{q}) ((\text{t}[\text{Attr}(\text{R})] = \text{t}[\text{Attr}(\text{R})]) \land (\text{t}[\text{Attr}(\text{Q})] = \text{t}[\text{Attr}(\text{Q})])) \land (\text{t}[\text{Attr}(\text{R})] = \text{t}[\text{Attr}(\text{R})]) \land (\text{t}[\text{Attr}(\text{Q})] = \text{t}[\text{Attr}(\text{Q})])) \land \ldots \land (\text{t}[\text{Attr}(\text{R})] = \text{t}[\text{Attr}(\text{R})]) \}
\]

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, \( R_{\text{La}} = \emptyset \), for all \( i (1 \leq i \leq m) \), and the definition of the projection operation is reduced to:

\[
\pi_{\text{tr}(\text{R})}(\text{R}) = \{ \text{t} | (\exists \text{t}_r \in \text{r}) \land (\forall \text{t}_q \in \text{q}) ((\text{t}[\text{Attr}(\text{R})] = \text{t}[\text{Attr}(\text{R})]) \}
\]

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 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:

\[
\rho[A \leftarrow A'](\text{R}) = [\text{R}_{1}, \ldots, \text{R}_{k}, \ldots, \text{R}_{m}] \cup \text{L}_{A' \rightarrow A_k} \land B, \ldots, Z]
\]

This definition is reduced to:

\[
\rho[A \leftarrow A'](\text{R}) = [\text{R}_{1}, \ldots, \text{R}_{k}, \ldots, \text{R}_{m}, A', B, \ldots, Z]
\]

for the special case where the attribute to be renamed, \( A \), is an atomic attribute at the top level of relation \( R \), since \( \cup \text{L}_{A' \rightarrow A_k} \land A' = A' \) (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 \( L=\emptyset \) 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, \( \triangleright \text{d}^\triangleright(rL,qM) \), 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:

\[
\triangleright \text{d}^\triangleright(rL,qM) = \triangleright \text{d}^\triangleright(rL,qM) = \{ \text{t} | (\exists \text{t}_r \in \text{r}) \land (\exists \text{t}_q \in \text{q})\}
\]

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 :: = \( \rho \) [attribute-commalist1] (term)

attribute-commalist1 :: = fattribute ← fattribute | fattribute ← fattribute, attribute-commalist1

fattribute :: = attribute1 | function2(attribute1)

attribute1 :: = attribute | nested-aggregate-attribute

function2 :: = N-MAX | N-MIN | N-SUM | N-COUNT | N-AVG

term :: = relation | (expression)

nested-projection :: = \( \pi \) (term (attribute-commalist2)) | term

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 -> columnIndex -> 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 -> columnIndex -> tableEntry |

| selectEntryByStr: Selects all columns from a list of
columns using the given column tag.

\[
\text{selectEntryByStr} :: \text{string} \rightarrow \text{tableEntry} \rightarrow \text{tableEntry}
\]

|| selectEntryByStrLst: Selects a list of columns whose names are provided by a list of string tags.

\[
\text{selectEntryByStrLst} :: [\text{string}] \rightarrow \text{tableEntry} \rightarrow \text{tableEntry}
\]

|| selectEntryLstByStrLst: Selects entries from a given entry list whose names are provided by a list of tags.

\[
\text{selectEntryLstByStrLst} :: [\text{string}] \rightarrow [\text{tableEntry}] \rightarrow [\text{tableEntry}]
\]

|| tableProjection2: Selects a subset of table entries based on given column names after all recursive column names have been resolved.

\[
\text{tableProjection2} :: [\text{string}] \rightarrow \text{relationalTable} \rightarrow \text{relationalTable}
\]

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

\[
\text{flattenRelTable} :: \text{relationalTable} \rightarrow \text{relationalTable}
\]

|| flattenEntryList: Flattens a list of table entries by calling helper function flattenColumnList.

\[
\text{flattenEntryList} :: [\text{tableEntry}] \rightarrow [\text{tableEntry}]
\]

|| flattenColumnList: Flattens a list of columns by calling helper function flattenColumn.

\[
\text{flattenColumnList} :: [\text{columnType}] \rightarrow [\text{columnType}]
\]

|| flattenColumn: Flattens recursive columns.

\[
\text{flattenColumn} :: \text{columnType} \rightarrow [\text{columnType}]
\]

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

\[
\text{tableProduct2} :: \text{relationalTable} \rightarrow \text{relationalTable} \rightarrow \text{relationalTable}
\]

|| tableProduct3: Product of a table with an inner table of another table. All possible combinations of table entries are included.

\[
\text{tableProduct3} :: \text{relationalTable} \rightarrow (\text{string}, \text{relationalTable}) \rightarrow \text{relationalTable}
\]

|| tableProduct4: Applies the product of a table to a list of table entries for recursive application.

\[
\text{tableProduct4} :: \text{relationalTable} \rightarrow \text{string} \rightarrow [\text{tableEntry}] \rightarrow [\text{tableEntry}]
\]

|| tableProduct5: Applies the product of a table to a list of column numbers for recursive application.

\[
\text{tableProduct5} :: \text{relationalTable} \rightarrow \text{string} \rightarrow \text{tableEntry} \rightarrow \text{tableEntry}
\]

|| tableProduct6: Applies the product of a table to a recursive column which holds the required inner table.

\[
\text{tableProduct6} :: \text{relationalTable} \rightarrow \text{columnType} \rightarrow \text{columnType}
\]

|| tableProduct7: Applies the product of a table to a recursive column if the inner table is the one required.

\[
\text{tableProduct7} :: \text{relationalTable} \rightarrow \text{string} \rightarrow \text{columnType} \rightarrow \text{columnType}
\]

|| getRecTableNames: Gets the name of a table and calls getRecTableNames2 to get the names of all recursive tables.

\[
\text{getRecTableNames} :: \text{relationalTable} \rightarrow [\text{string}]
\]

|| getRecTableNames2: Gets table names recursively for a list of table entries.

\[
\text{getRecTableNames2} :: [\text{tableEntry}] \rightarrow [\text{string}]
\]

|| getRecTableNames3: Gets table names recursively for a list of columns.

\[
\text{getRecTableNames3} :: [\text{columnType}] \rightarrow [\text{string}]
\]

|| getRecTableNames4: Gets the name of a table column if it is an inner table.

\[
\text{getRecTableNames4} :: \text{columnType} \rightarrow [\text{string}]
\]

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)))"]\(\text{["D", NF ((=) 1)]d}\)

**Query 4:**
selectFrom["COURSE/TRAINING(COURSE/TRAINER(TRN))"]\(\text{["COURSE/TRAINING(COURSE/TRAINER(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).

\[
\text{tableProjection} \text{["COMPANY","BANK"]}
\text{(joinTables("TRAINING/LOCATION(ANNEX(ADDRESS))", "World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering Vol:4, No:3, 2010... "},
\text{ISNI:0000000091950263}
joinTables("COMPANY", selectFrom("COMPANY"))[("TRAINING(TRAINER(TRN))", SF(="Karen")])[("COMPANY", location)]("CASH-POINT(ADDRESS)", cashpoint))

REFERENCES