NF-NR: A Practical Normal Form for Nested Relations

Tok-Wang Ling Ling-Ling Yan*
Dept. of Information Systems and Computer Science
National University of Singapore

Abstract

We propose a normal form for nested relations, called NF-NR, which removes undesirable anomalies from a nested relational database schema. Both functional dependencies and multivalued dependencies are considered. NF-NR reduces to 3NF/4NF if the nested relation considered is actually a flat relation. Especially, NF-NR removes global redundancies among a set of nested relations. Two approaches to NF-NR database design, namely the restructuring rules approach and the ER approach, are discussed. We relate NF-NR to ER-NF, a normal form of ER defined earlier, by defining a simple mapping from an ERD in ER-NF to a set of nested relations in NF-NR. This approach effectively removes ambiguities and redundancies on a semantic level and hence gives a set of nested relations with clean semantics and yet in good normal form. A set of desirable properties for any normal form for nested relations are described and an evaluation of several existing normal forms is given based on this set of properties. The evaluation shows that NF-NR improves over previously proposed normal forms in various aspects and is a more practical normal form for nested relations.

1. Introduction

A **non-first normal form relation(**NF^2 **relation)** is a relation whose attributes may not be atomic values, rather, they may be *repeating groups*. Codd recognized immediately that non-first normal form relations may contain some redundancy which may cause some updating anomalies [3]. One process that attempts to remove undesirable updating anomalies and redundancy from a relation is called **normalization**. The first step in the normalization process is to remove the existence of non-atomic attributes by flattening the relation. The resulting relation is called **a first normal form (1NF)** relation or **flat relation**. Theory and practice in relational database design are mostly based on

* Currently with the Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada.

1NF relation[10, 16]. The normalization process of 1NF relations is well established[10]. However, many applications are more suitably represented by NF^2 relations than by flat relations. Examples of such applications are picture data processing, CAD and some business data processing applications whose data is hierarchically structured. The motivations behind considering nested relation as a database model are discussed in [4, 11, 13, 15]. Research work in semantic data model and object-oriented data model shows that the capability of a data model to represent and manipulate complex structures is highly desirable[1].

A **nested relation** (or called **not-necessarily normalized relation**) is a relation whose attributes are either atomic values or nested relations. [14, 15] extend the relational algebra and calculus to manipulate nested relations. There are numerous prototype systems built based on nested relation data model[5]. However, nested relations may suffer from anomalies. These anomalies must be removed by a normal form. In [11, 12, 13, 9], normal forms for nested relations are discussed. These normal forms are based on their own conceptualizations of the model of nested relation and the real world application to be modeled. Consequently, the definitions given differ from one another.

In this paper, we define the concept of nested relation and investigate various anomalies that may appear in this model. A normal form for nested relation, NF-NR, is proposed. Our normal form deals with functional dependencies as well as multivalued dependencies and removes identified anomalies from nested relations. Two approaches to designing NF-NR databases are given. First, the approach of restructuring the nested relations. We give a set of restructuring rules to be used to transform nested relation(s) into NF-NR relation(s). Examples are given to show the use of the rules. Second, the entity-relationship approach to NF-NR database design. This approach is based on the earlier work on normal form for ER model[7]. The capability of ER model in capturing more semantics makes this approach to NF-NR database design promising. An evaluation of previously proposed normal forms for nested relations is given. We describe a set of properties that a "good" normal form for nested relations should have. The evaluation, carried out based on this set of properties, shows that NF-NR is a more practical normal form compared with other normal forms.

The rest of the paper is organized as follows. In section 2, definitions of various concepts in nested relation are given. The anomalies that may occur in nested relations are identified. In section 3, a normal form for nested relations, NF-NR, is defined. Examples are given to show that NF-NR removes undesirable anomalies. In section 4, two approaches to NF-NR database, namely the approach of restructuring and that of ER technique, are discussed. In section 5, an evaluation of normal forms for nested relations is presented. In section 6, we present the conclusions.

2. Nested Relations: Definition and Anomalies

Data modeling people have long noticed that the restriction that attributes must be "atomic" decreases the modeling power of relational data model greatly. Non-atomic attributes arise naturally from many applications. It is desirable to have non-atomic attributes supported in relational data model. However, by introducing this concept, the problem of anomalies, which was well solved for flat relations, will appear again. In this section, we define the concept of nested relation and discuss various anomalies that may appear in this model.

2.1. Definition of Nested Relation

Definition 2.1. A **nested relation** is defined as follows:

1. Let $A_1, \ldots, A_n$ $(n>0)$ be distinct atomic attribute names. Then $R(A_1, \ldots, A_n)$ is a nested relation, where R is a distinct name called the **name** of the nested relation.

2. Let $B_1, \ldots, B_m$ $(m>0)$ be distinct names, each B_j $(j=1,m)$ is either an atomic attribute name or nested relation but at least one B_i $(1 \le i \le m)$ is an atomic attribute name. Then

$$R(B_1, \ldots, B_m)$$

is a nested relation, where R is a distinct name called the **name** of the nested relation.

By the 1 of Definition 2.1, a first normal form (1NF) relation is also a nested relation. Since atomic attribute names and nested relation names are distinct, we can simply refer to a nested relation $R(B_1, \ldots, B_m)$ by its name R without any ambiguity.

In this paper, we often use A_i's to represent atomic attribute names and R_i's to represent nested relations. For a clean presentation, we write a nested relation in the following form:

$$R(A_1, \ldots, A_n, R_1, \ldots, R_m)$$

where $n>0$, $m \ge 0$, A_i's (i=1,n) are distinct atomic attribute names and R_j's (j=1,m) are distinct nested relations. A_i's are also referred to as **single-valued attributes.** R_j's are referred to as **nested attributes**. Both single-valued attributes and nested attributes of a nested relation are refereed to as **attributes** of the nested relation. Hence the term "all attributes of a nested relation" means all the single-valued attributes and nested attributes of the nested relation. Usually, the term atomic attribute is used in contrast with the term composite attribute. We will explain the term composite attribute in

Section 4.2.1. In the rest of this paper, the two terms, atomic attribute and singled-valued attribute, will be used as the same unless specially specified.

Definition 2.2. Let A's be atomic attribute names and R's be nested relations. A **tuple** and an **instance** of a nested relation are defined inductively as follows:

1. Let $R(A_1, \ldots, A_n)$ where n > 0, be a nested relation (which is also in 1NF). A tuple of R is in the form of

$$t = (a_1, \ldots, a_n)$$

where a_i $(1 \le i \le n)$ is a value from the domain of A_i. An instance of R is a set of tuples of R.

2. Let $R(A_1, \ldots, A_n, R_1, \ldots, R_m)$ where $n > 0$, $m \ge 0$, be a nested relation. A tuple of R is in the form of

$$t = (a_1, \ldots, a_n, r_1, \ldots, r_m)$$

where a_i $(1 \le i \le n)$ is a value from the domain of A_i and r_j $(1 \le j \le m)$ is an instance of R_j. An instance of R is a set of tuples of R.

Definition 2.3. Let $R(A_1, \ldots, A_n, R_1, \ldots, R_m)$, where $n > 0$, $m \ge 0$, be a nested relation. The **visible attribute set** of R, denoted as *V_ATTR(R)*, is defined as

$$V\_ATTR(R) = \{A_1, \ldots, A_n, R_1, \ldots, R_m\}$$

Definition 2.4. Let $R(A_1, \ldots, A_n, R_1, \ldots, R_m)$, where $n > 0$, $m \ge 0$, be a nested relation. The **nested attribute set** of R, denoted as $N\_ATTR(R)$, is defined as

$$N\_ATTR(R) = \{R_1, \ldots, R_m\}$$

Notice that if R is a relation without nested attribute, i.e., R is in 1NF, then we have $N\_ATTR(R) = \varnothing$.

Definition 2.5. Let $R(A_1, \ldots, A_n, R_1, \ldots, R_m)$, where $n > 0$, $m \ge 0$, be a nested relation. The **complete atomic attribute set** of R, denoted as $ATTR(R)$, is defined as follows:

$$ATTR(R) = \{A_1, \ldots, A_n\} \cup ATTR(R_1) \cup \cdots \cup ATTR(R_m)$$

Notice that if R is a relation without nested attribute, i.e., R is in 1NF, then we have $ATTR(R) = \{A_1, \ldots, A_n\}$.

Definition 2.6. Let R be a nested relation. The **ground relation** for R is a flat relation

$$R_G(B_1, \ldots, B_k)$$

where $\{B_1, \ldots, B_k\} = ATTR(R)$. The **instance of ground relation** of R, R_G, is constructed as follows:

1. if $N\_ATTR(R) = \varnothing$, then $R_G = R$.
2. if R is $R(A_1, \ldots, A_n, R_1, \ldots, R_m)$ where $(n > 0, m > 0)$, then for each R-tuple:

$$t = (t_1, \ldots, t_n, r_1, \ldots, r_m)$$

the following tuples are in R_G:

$$(t_1, \ldots, t_n, r^1{}_1, \ldots, r^1{}_{j_1}, \ldots, r^m{}_1, \ldots, r^m{}_{j_m})$$

where $(r^i{}_1, \ldots, r^i{}_{j_i}) \in (r_i)_G$ for i = 1,m.

A nested relation can be a nested attribute in another nested relation. This is a way of organizing data. Ultimately data is represented by atomic values. Especially when dependencies among values are considered, we need to study on atomic attribute basis as if they are in a flat relation. The complete atomic attribute set defined above gives all the atomic attributes involved in a nested relation. The ground relation provides a "platform" on which we can discuss all the data dependencies.

Definition 2.7. Let $R(A_1, \ldots, A_n, R_1, \ldots, R_m)$, where $n > 0$, $m \geq 0$, be a nested relation. The **single-valued attribute set** of R, denoted as $ATTR_s(R)$, is defined as

$$ATTR_s(R) = \{A_1, \ldots, A_n\}$$

Definition 2.8. The **level** of a nested relation R, denoted as $Level(R)$, is defined as follows:

1. If R is not a nested attribute in another nested relation, then $Level(R) = 0$.

2. If R is a nested attribute in nested relation R' where $Level(R') = v$, then $Level(R) = v+1$.

Definition 2.9. The **path** of a nested relation R, denoted as $Path(R)$, is a list of names of nested relations defined as follows:

1. if $Level(R) = 0$, $Path(R) = \varnothing$.

2. if $Level(R) > 0$, and R is a nested attribute in a nested relation R', then $Path(R) = Path(R') \cdot R'$, where $\cdot$ denotes list concatenation.

Definition 2.10. Let $R(A_1, \ldots, A_n, R_1, \ldots, R_m)$, where $n>0, m\geq 0$, be a nested relation and $t = (a_1, \ldots, a_n, r_1, \ldots, r_m)$ be a tuple of R. For any $X \in V\_ATTR(R)$, the **projection** of t on X, denoted as $t[X]$, is defined as follows:

1. If $X = A_i$ $(1 \leq i \leq n)$, then $t[X] = a_i$.
2. If $X = R_j$ $(1 \leq j \leq m)$, then $t[X] = r_j$.

In 1NF relational model, every relation is with level 0 and hence an empty path. Especially, one relation will have exactly one instance in the database. In nested relational model, however, a relation can be a nested attribute in another relation. Assume relation R is a nested attribute in relation R'. The projection of an R'-tuple on this nested attribute R will be an instance of relation R. Hence, a relation may have multiple instances in the database. However, most existing languages for nested relations does not introduce higher order construct such as variables ranging over relations, nor do people think in higher order concepts when nested relational model is used. The key concept here is path. Each instance of a nested relation with level > 0 has a non-empty path via which it can be located and accessed.

Definition 2.11. Let $R(A_1, \ldots, A_n, R_1, \ldots, R_m)$, where $n>0, m\geq 0$, be a nested relation and $t = (a_1, \ldots, a_n, r_1, \ldots, r_m)$ be a tuple of R. For a list of attributes $L = <X_1, \ldots, X_p>$ such that $X_i \in V\_ATTR(R)$ for i = 1, p, the **projection** of t on L, $t[L]$, is a tuple defined as follows:

$$t[L] = (t[X_1], \ldots, t[X_p])$$

Definition 2.12. Let R be a nested relation and let t_1, t_2 be two tuples from R. Let $X \in V\_ATTR(R)$. We say t_1 and t_2 **agree** on X in R if the followings hold:

1. If $X \in ATTR_s(R)$, then $t_1[X] = t_2[X]$.

2. If $X \in N\_ATTR(R)$, then $t_1[X]$ and $t_2[X]$ satisfy the followings:

a). for any tuple $t_1' \in t_1[X]$, there exists $t_2' \in t_2[X]$ s.t. t_1' and t_2' agree on all attributes of X.

b). for any tuple $t_2' \in t_2[X]$, there exists $t_1' \in t_1[X]$ s.t. t_2' and t_1' agree on all attributes of X.

Definition 2.13. Let R be a nested relation and $M \subseteq V\_ATTR(R)$. For any two tuples $t_1, t_2 \in R$, they **agree** on M iff they agree on all attributes in M.

Example 2-1. The following is a nested relation:

$$R(A, R_1(B, C))$$

According to the definitions given above, we have the followings:

$$V\_ATTR(R) = \{A, R_1\}$$

$$N\_ATTR(R) = \{R_1\}$$

$$ATTR_s(R) = \{A\}$$

$$ATTR(R) = \{A, B, C\}$$

$$Level(R) = 0, \quad Level(R_1) = 1$$

$$Path(R) = \varnothing, \quad Path(R_1) = R$$

A tuple of R can be:

$$t = (a_1, r_1)$$

where r_1 can be an instance of relation R_1 as follows:

$$r_1 = \{(b_1, c_1), (b_2, c_2), (b_3, c_3)\}$$

An instance of R consists of a set of tuples similar to the above example. Take tuple t for example, we'll have:

$$t[A] = a_1$$

$$t[R_1] = r_1 = \{(b_1, c_1), (b_2, c_2), (b_3, c_3)\}$$

The ground relation of R is a relation

$R_G(A, B, C)$

Consider from tuple *t*, we know that the following tuples will be in R_G:

$(a_1, b_1, c_1), (a_1, b_2, c_2), (a_1, b_3, c_3)$

2.2. Anomalies in Nested Relation

Since flat relation is a special case of nested relation where no nested attribute is presented, nested relation in general involves similar anomalies identified for 1NF relations. This is illustrated in the following example.

Example 2-2. Consider the following nested relation

$SUPPLY(S\#, Sname, SP(P\#, Pname, P\_color, Quantity))$

This is the famous supplier-part example. Each supplier supplies part(s), each part may be supplied by several suppliers. If the above scheme is used, we may be encountered with some **anomalies**. If a part is not supplied by any supplier, information about it cannot be entered. This is the **insertion anomaly**. If the last supplier of a part is deleted, information about this part will be missing. This is the **deletion anomaly**. If the color of a part is to be changed, we must make sure that all the appearances of this information are consistent, knowing that a part can be supplied by multiple suppliers. This is the **rewriting anomaly**. These anomalies are generally referred to as **update anomalies**.

The fact that attributes can be relations also introduces other types of anomalies that do not happen in flat relations. This is illustrated in the following example.

Example 2-3. Consider the following nested relation

$TIME\_TABLE(Teacher\#, CLASSROOM(Room\#, SUBJECT(Day, Time, Subject\#)))$

This nested relation is intended to be a time table for teacher's use. Assume we have the following dependency:

$Teacher\#, Day, Time \rightarrow Room\#, Subject\#$

In the above scheme, this dependency is not well reflected. To change the time of subject *s* taught by teacher *t*, we have to search through all the classroom information, find the subject, and then make the change. We refer to such case as **path anomaly**. Intuitively, the following scheme is a better representation of the real world information:

$TIME\_TABLE'(Teacher\#,\ SUBJECT(Day,\ Time,\ Subject\#,\ Room\#))$

3. A Normal Form for Nested Relation(NF-NR)

In order to remove the anomalies that may exist in a nested relation, we define a normal form for it. In this section, this normal form, called NF-NR, will be described.

Definition 3.1. Let R be a nested relation. Let $A \subseteq ATTR_s(R)$ and $B \subseteq V\_ATTR(R)$. B is **extended functional dependent** on A, denoted as $A \Rightarrow B$, if whenever two tuples of R agree on all attributes in A, they also agree on all attributes in B.

Theorem-1. Let R be a nested relation and A be a set of single-valued attributes. We have the following properties for extended functional dependency:

1) Let R' be a nested attribute of R. $A \Rightarrow R'$ in R implies $A \twoheadrightarrow ATTR(R')$ in R_G.

2) If B is a set of single-valued attributes, then $A \Rightarrow B$ iff $A \rightarrow B$.

3) If B is a set of single-valued attributes and M is a set of attributes, nested or single-valued, then $A \rightarrow B$, $B \Rightarrow M$ imply $A \Rightarrow M$.

Proof.

The proofs of these properties follow directly from the definitions of extended functional dependency, functional dependency, multivalued dependency, and ground relation of R.

□

With the above theorem, all extended functional dependencies(EFDs) are reduced to functional dependencies(FDs) or multivalued dependencies(MVDs) in the ground relation. Apparently, the above theorem ensures that the inference rules for FDs and MVDs consist a **complete** set of inference rules for EFDs.

Definition 3.2. Let A be a set of single-valued attributes and B a set of attributes, nested or single-valued. An extended functional dependency $A \Rightarrow B$ is **partial** if there exists an $A' \subset A$ such that $A' \Rightarrow B$. Otherwise we say that $A \Rightarrow B$ is a **full** extended functional dependency.

Definition 3.3. Let A be a set of single-valued attributes and B a set of attributes, nested or single-valued. If there exists a set of single-valued attributes C such that

$$A \rightarrow C,\ C\ -/->\ A,\ C\ =>\ B$$

then we say that B is **transitively extended functional dependent** on A or **transitively dependent** on A, for short.

Definition 3.4. Let R be a nested relation and let $K \subseteq ATTR_s(R)$. The concept of keys are defined as follows:

case 1: *Level(R) = 0.*

K is a **candidate key** of R iff

i) $K\ =>\ V\_ATTR(R)$ and

ii) no proper subset of K satisfies the i) above.

One of the candidate keys is designated to be **primary key** of R, denoted as K_R. In this case, each candidate key defines an **absolute key**. The primary key of R defines the **primary absolute key** of R, $K^{abs}{}_R$, i.e.

$$K^{abs}{}_R\ =\ K_R$$

case 2: $Level(R)\ =\ v\ (v>0)$ and $Path(R)\ =\ R_0 \cdots\ R_{v-1}$.

K is a **candidate key** of R iff

i) $K^{abs}{}_{R_{v-1}}$, K => V_ATTR(R), and

ii) no proper subset of K satisfies the i) above.

One of the candidate keys is designated as **primary key**, denoted as K_R. Let $P_K \subseteq K^{abs}{}_{R_{v-1}}$. If P_K satisfies the following:

a). $P_K \bigcup K_R\ => V\_ATTR(R)$ and

b). no proper subset of P_K satisfies a) above.

Then $P_K \bigcup K_R$ is a **candidate absolute key**. One of the candidate absolute keys is chosen as the **primary absolute key** of R, i.e.

$$K^{abs}{}_R\ =\ P_K{}' \bigcup K_R$$

for some $P_K{}' \subseteq K^{abs}{}_{R_{v-1}}$.

In the following discussion, all the MVDs we talk about are in the context of $ATTR(R)$ and R_G.

Although we organize data into different structures, MVDs are preserved. We do not consider embedded MVD. Does Embedded MVD cause any problem then? The answer is "yes" if no constraint is added. Consider $R(A, B, C, D)$ with $ABC \rightarrow D$ and $A \multimap B \mid C$. Obviously upon deletion/insertion/update of tuples, certain checking is required. How to remove this kind of problem is not clear in 1NF relations or nested relations. In [8], an analysis of the real meaning of MVDs is given. The paper interprets the practical meaning of embedded MVDs with real world examples and captures them by using ER techniques. Especially, the result of this analysis shows that embedded MVDs may arise due to the wrong grouping of attributes into relations and integrity constraints among relations. Using a semantically rich design tool such as ERD can help to avoid such cases and hence give a clear interpretation to embedded MVDs.

Definition 3.5. Let R be a nested relation with $Level(R) = 0$. The **basic dependencies** of R, BD(R) is defined as follows:

1. Let $A, B \subseteq ATTR_s(R)$. Nontrivial functional dependencies of the form $A \rightarrow B$ where $A = K_R$ or B is part of a key of R, are in BD(R).

2. For each nested attribute R_1 of R, extended functional dependency $K_R \Rightarrow R_1$ is in BD(R).

3. Let R_1 be a nested attribute of R and R' be the nested relation constructed by including K_R as single-valued attribute(s) in R_1. Then $BD(R') \subset BD(R)$.

Definition 3.6. A nested relation R with Level(R) = 0 is in the **normal form for nested relation(NF-NR)** iff:

1. R has at least one key.

2. All the single-valued attributes of R form a 3NF relation with all keys of R as its keys.

3. If there is no nested attribute in R and R is all-key, then R is also in 4NF.

4. For any nested attribute R' of R and any key K of R, the followings hold:

   a. there exists no A, $A \subset ATTR(R')$, such that $K \multimap A$.

   b. R' is not transitively dependent on K.

5. For any nested attribute R_1 of R, form a nested relation R_1' by adding $K^{abs}{}_R$ into R' as new single-valued attribute(s). Then R_1' is in NF-NR.

6. For any nested attribute R_1 of R, $K^{abs}{}_R \cap K^{abs}{}_{R_1} = \varnothing$ or $K^{abs}{}_R \subset K^{abs}{}_{R_1}$.

7. Any non-trivial extended functional dependency in R can be derived from BD(R) by using the axioms for FD, MVD, and the three properties of extended functional dependency described in

Theorem-1.

NF-NR removes certain undesirable anomalies in a nested relation. This is illustrated in the following examples.

Example 3-1. Consider the SUPPLY relation in Example 2-2:

$$SUPPLY(S\#,\ Sname,\ SP(P\#,\ Pname,\ P\_color,\ Quantity))$$

We have the following dependencies:

$$S\# \rightarrow Sname$$

$$P\# \rightarrow Pname,\ P\_color$$

$$S\#,\ P\# \rightarrow Quantity$$

S# is the absolute key of relation SUPPLY. Consider the relation formed by including *S#* as a single-valued attribute in SP:

$$SP'(S\#,\ P\#,\ Pname,\ P\_color,\ Quantity)$$

The key of this relation is *{S#, P#}*. However, the relation is not in NF-NR because the following dependency is partial:

$$S\#,\ P\# \rightarrow Pname,\ P\_color$$

This violates condition 4 in Definition 3.6. Split the relation into two relations:

$$SUPPLY'(S\#,\ Sname,\ SP'(P\#,\ Quantity))$$

$$PART(P\#,\ Pname,\ P\_color)$$

Now each relation is in NF-NR. Apparently, by doing so, we can remove all the anomalies mentioned in Example 2-2.

Example 3-2. Consider the following relation:

$$EMP(Emp\_name,\ Child\_name,\ SALARY\_HISTORY(Date,\ Amount))$$

with extended functional dependencies:

$$Emp\_name \Rightarrow SALARY\_HISTORY$$

i.e.

$$Emp\_name \twoheadrightarrow Date, Amount$$

The key of this relation is $\{Emp\_name, Child\_name\}$. However, the extended functional dependency

$$Emp\_name, Child\_name \Rightarrow SALARY\_HISTORY$$

is partial. This violates condition 4 in Definition 3.6. Using complementation rule on the MVD

$$Emp\_name \twoheadrightarrow Date, Amount$$

we actually have

$$Emp\_name \twoheadrightarrow Child\_name$$

With this fact in mind, we know that the following is a better scheme for EMP relation:

$$EMP(Emp\_name, CHILD(Child\_name), SALARY\_HISTORY(Date, Amount))$$

where *CHILD* is a newly introduced name.

Example 3-3. Condition 6 in Definition 3.6 removes path anomaly in Example 2-3. Consider this example again:

$$TIME\_TABLE(Teacher\#, CLASSROOM(Room\#, SUBJECT(Day, Time, Subject\#)))$$

We have the following dependency:

$$Teacher\#, Day, Time \rightarrow Room\#, Subject\#$$

The absolute key for the relation *TIME_TABLE* is *Teacher#*, that for CLASSROOM is {*Teacher#,Room#*}. The absolute key for the relation SUBJECT is {*Teacher#*, *Day*, *Time*}. Notice that condition 6 in Definition 3.6 is violated here. The following relation is in NF-NR:

$$TIME\_TABLE'(Teacher\#, SUBJECT(Day, Time, Room\#, Subject\#))$$

The following theorems give more formal insight into NF-NR.

Theorem-2.

(1) All 3NF and BCNF relations which are not all-key are in NF-NR.

(2) All 4NF and 5NF relations are in NF-NR.

(3) Any relation which is not in 3NF, is not in NF-NR either.

Proof.

The result can be proven directly by the definitions of these normal forms.

□

Theorem-3. For any nested relation R which is in NF-NR or with $Level(R) = v (v > 0)$ and $Path(R) = R_0 \cdots R_v$ where R_0 is in NF-NR, the following facts hold:

1. for any nonprime attribute $A \in ATTR_s(R)$, $K^{abs}{}_R \rightarrow A$ is a full functional dependency.

2. if Level(R) > 0 and $K^{abs}{}_R \subseteq ATTR_s(R)$, then the functional dependency $K^{abs}{}_R \rightarrow K^{abs}{}_{R_v}$ holds.

Proof.

We first prove 1 in the above theorem. If $Level(R) = 0$, then the conclusion is straightforward from condition 2 in Definition 3.6. If $Level(R) > 0$, then the conclusion is obvious from conditions 5 and 2 in Definition 3.6.

The proof for 2 in this theorem is as follows. By assumption, we can write R as

$$R(K^{abs}{}_R, B_1, \ldots, B_k)$$

Let $Path(R) = R_0 \cdots R_v$. Consider the following relation:

$$R'(K^{abs}{}_{R_v}, K^{abs}{}_R, B_1, \ldots, B_k)$$

By definition, $K^{abs}{}_R$ is the primary key of relation R'. From condition 5 in Definition 3.6, we know that this relation must be in NF-NR. Notice that in R', $K^{abs}{}_{R_v}$ is a (set of) single-valued attribute. Directly we have

$$K^{abs}{}_R \rightarrow K^{abs}{}_{R_v}$$

□

The 1 in Theorem-3 takes after the similar property of 3NF for flat relation: no non-prime

attribute should be partially dependent on a key. In the context of nested relations, we talk about absolute key rather than primary key. An interesting point is that, a functional dependency holds no matter in R or in R_G. Although we organize data into different structures, functional dependencies are preserved.

Condition 6 in Definition 3.6 displays a special feature of nested relation: some nested attributes can have keys that contain only single-valued attributes in this nested attribute. Consider relation R with $Path(R) = R_0 \cdots R_v$ and $K^{abs}{}_R \subseteq ATTR_s(R)$. The 2 in Theorem-3 ensures that if this is the case, then each instance of R will appear at most once in one relation instance of R_v.

We view this feature from database design level. Assume we use ER approach[2]. Nested relation supports a better modeling of 1:m and m:m attributes than relational model. The 2 in Theorem-3 together with condition 6 in Definition 3.6 ensures that NF-NR can capture these two types of attributes properly. This will be further illustrated in section 4.2.

Theorem-4. Let R be an NF-NR relation in which one of the followings holds:

1. $N\_ATTR(R) \neq \varnothing$ and $K_R = ATTR_s(R)$.
2. $N\_ATTR(R) = \varnothing$ and there exists a non-prime single-valued attribute.

Then there exist no $A, B \subset K_R$ where $A \neq \varnothing$, $B \neq \varnothing$, $A \bigcap B = \varnothing$ and $A \bigcup B = K_R$, such that $A \multimap B$.

Proof.

We consider two cases.

case 1: there exist nested attributes in R and $K_R = ATTR_s(R)$.

Let R_1 be a nested attribute. If there exist A, B: $A, B \subset K_R$ where $A \neq \varnothing$, $B \neq \varnothing$, $A \bigcap B = \varnothing$ and $A \bigcup B = K_R$, such that $A \multimap B$, then we have the followings:

$$A \multimap B, \quad AB \multimap ATTR(R_1)$$

Trivially we have $A \multimap A$, by additivity of MVD, we have $A \multimap AB$. By transitivity of MVD, we have $A \multimap ATTR(R_1)$, i.e. $A \Rightarrow R_1$. This contradicts condition 4 in Definition 3.6.

case 2: there exists no nested attribute and there exists a non-prime single-valued attribute D.

Let $E = ATTR_s(R) - K_R$. Again we assume that there exist A, B: $A, B \subset K_R$ where $A \neq \varnothing$, $B \neq \varnothing$, $A \bigcap B = \varnothing$ and $A \bigcup B = K_R$, such that $A \multimap B$. From $A \multimap B$, by complementation, we have

$$A \text{ } -\!\!>\!\!> E \tag{1}$$

From the condition in this case, we also have

$$AB \rightarrow E \tag{2}$$

Using the coalescence rule of FD and MVD on (1) and (2), we have $A \rightarrow E$. Since $D \in E$, we also have $A \rightarrow D$. This contradicts condition 2 in the definition of NF-NR since a non-prime single-valued attribute, namely D, is partially dependent on a key AB.

□

Theorem-4 reflects a motivation behind normalizing nested relations, namely, to remove redundancy to the "best" extent. All set-valued attributes should be represented by nested attributes as far as possible. Consider the nested relation EMP in Example 3-2 again:

$$EMP(Emp\_name,\ Child\_name,\ SALARY\_HISTORY(Date,\ Amount))$$

The primary key of EMP is $\{Emp\_name, Child\_name\}$ and we have

$$Emp\_name \text{ } =\!\!> SALARY\_HISTORY$$

From which we can easily get

$$Emp\_name \text{ } -\!\!>\!\!> Child\_name$$

From Theorem-4, EMP is not in NF-NR. Another way to view this problem is, $Child\_name$ should be represented as a nested attribute rather than single-valued. As mentioned in Example 3-2, if we do so, EMP will be in NF-NR.

Definition 3.7. A set of nested relations S is said to be in **normal form for set of nested relations** iff

(1) Each nested relation in S is in NF-NR.

(2) Given any relation $R \in S$ and any non-trivial extended functional dependency $A \text{ } =\!\!> B$ in $BD(R)$, $A \text{ } =\!\!> B$ cannot be derived from the union of the set of basic dependencies of all the relations in S with B removed from relation R, by using the axioms for FDs and MVDs and the properties of extended functional dependency given in Theorem-1.

Notice that the second condition in the above definition ensures that there is no redundancy(due

to redundant dependencies) among the NF-NR relations in a database. The following example illustrates this fact.

Example 3-4. Consider a database consisting of the following nested relations:

$$R_1(A,\ B,\ R_{11}(C)), \quad R_2(B,\ R_{21}(D)), \quad R_3(A,R_{31}(D))$$

with $A \rightarrow B$, A $-$>> C, B $-$>> D, A $-$>> D.

Notice that all the MVDs above are in the context of *ABCD*. Clearly, each relation is in NF-NR. However, this database is not in NF-NR since the dependency A $-$>> D can be derived from $A \rightarrow B$ and B $-$>> D. In this case, R_3 is redundant and can be removed from the database.

4. NF-NR Relational Database Design

Three approaches to NF-NR database design can be considered. First, synthesizing. This approach should start from a given set of EFDs and generate a set of nested relations that is in NF-NR and covers all the EFDs. EFDs are basically FDs or MVDs. With the presence of MVD, synthesizing can be very difficult(if feasible at all). Second, restructuring. This approach starts from a given set of nested relations and tries to convert them into NF-NR relations by using heuristic restructuring rules. This approach was first discussed in [9] and will be further discussed in this section. Third, using Entity-Relationship technique. This approach was well investigated in [9] and will be further discussed later in this section.

4.1. Restructuring Rules for NF-NR relational Database Design

Restructuring for normalizing flat relations is basically decomposition. The major goal is to remove transitive or partial functional dependencies and undesirable MVDs. Similarly, when normalizing nested relations, we also need to remove transitive and partial extended functional dependencies. However, from Definition 3.6, we see that NF-NR requires more than elimination of these undesirable dependencies. For example, condition 6 in Definition 3.6 is to ensure that no path anomaly will happen. Moreover, due to the richer structure, these undesirable features happen in various forms. Restructuring will include nesting, unnesting as well as decomposition of nested relations. Given below are some rules of thumb.

(Rule1): remove transitive EFDs by decomposition

If a non-prime single-valued attribute or a nested attribute of a nested relation R is transitively

(extended)dependent on attributes A where $A \subset K^{abs}{}_R \bigcup ATTR_s(R)$*, then decompose the relation to eliminate this transitive dependency.*

It is easy to see that Rule1 can also be used to remove undesirable partial EFD since partial EFD is a special case of transitive EFD. Splitting may happen in many ways. It is worth mentioning that choosing a correct way to decompose is a nontrivial issue since certain splitting will cause the loss of EFDs. As we only intend to give some heuristic rules, we do not look further into this problem. The following example shows the usage of Rule1.

Example 4-1. Consider the following nested relation:

$$R(A,\ B,\ C,\ R'(D,\ E))$$

with $A \rightarrow B,\ B \rightarrow C,\ C => R'$.

A is the key of this relation. R is not in NF-NR due the fact that both C and R' are transitively dependent on A. To remove these, we apply Rule1 twice. First, we apply Rule1 to R to split it into the following relations:

$$R_1(A,\ B,\ C)$$

$$R_2(C,\ R'(D,\ E))$$

Next, we apply Rule1 again to split R_1 further to get:

$$R_{11}(A,\ B)$$

$$R_{12}(B,\ C)$$

$$R_2(C,\ R'(D,\ E))$$

Now they are a set of NF-NR relations.

(Rule2): remove partial EFD by moving attributes

Let R be a nested relation with Level(R) > 0 and A be any attribute(s) in R. If there is a B such that $B => A$ *where B is a (set of) single-valued attribute from the relations on Path(R), move A from R to the nested relation in which R is a nested attribute.*

Rule2 removes partial dependencies. This in turn helps in avoiding *over-nesting*. Attributes

should be kept as close to the "root" relation as possible. The "promotion" of an attribute may happen more than once, every time moving the attribute closer to the right position for it. This is illustrated in the following example.

Example 4-2. Consider nested relation:

$$R(A,\ R_1(B,\ C,\ R_{11}(D,\ E,\ F)))$$

with $A\ -{>>}\ DEF$, i.e., $A\ {=>}\ R_{11}$.

The absolute key of R is A. Obviously, BC is the key of R_1. Construct relation $R'(A,\ B,\ C,\ R_{12}(D,\ E,\ F))$. In this relation, the only key is ABC. However, $ABC\ {=>}\ R_{11}$ is not a full dependency. According to condition 5 and 3 in Definition 3.6, R is not in NF-NR.

Apply Rule2 to R, R_{11} can be promoted from R_1 to R. Hence we get the following NF-NR scheme for R:

$$R(A,\ R_1'(B,\ C),\ R_{11}(D,\ E,\ F))$$

(Rule3): removing partial EFD by nesting single-valued attributes

Let R be a nested relation in which either $N\_ATTR(R) \neq \varnothing$ and $K_R = ATTR_s(R)$ or $N\_ATTR(R) = \varnothing$ and there exists a non-prime single-valued attribute. If there is an MVD, $A\ -{>>}\ B$, where $A,\ B \neq \varnothing$, $A \bigcup B = K_R$, $A \bigcap B = \varnothing$, then make a new nested attribute of R, R', such that $ATTR(R') = ATTR_s(R') = B$.

Rule3 is a direct application of Theorem-4. The following example illustrates the use of this rule.

Example 4-3. Consider the nested relation

$$R(A,\ B,\ C,\ R'(D,\ E,\ F))$$

with dependency $A\ {=>}\ R'$.

The key of this relation is ABC. R is not in NF-NR since R' does not fully depend on ABC. As a matter of fact, $A\ {=>}\ R'$ implies $A\ -{>>}\ BC$. Apply Rule3, we can rewrite the scheme of R as

$$R(A,\ R''(B\ ,C),\ R'(D,\ E,\ F))$$

Now it is in NF-NR.

(Rule4): removing partial EFD by nesting attributes

Let R be a nested relation of any level. Let A be a set of single-valued attributes of R such that $A \subset K_R$, *and* R' *be a nested attribute of R. If* $(K^{abs}{}_R - A) => R'$ *and no proper superset of A satisfies this property, then restructure R as follows: make* $(K_R - A)$ *the only single-valued attribute(s) of R,* R' *remains to be a nested attribute. The rest of the attributes, single-valued or nested, form a nested attribute.*

Example 4-4. Consider the nested relation

$$R(A, B, R_1(C), D, E, R_2(F))$$

with $AB \rightarrow D$, $AB => R_2$, $A => R_1$, $A \rightarrow E$. The key of this relation is AB. R is not in NF-NR since $AB => R_1$ is not a full dependency. Apply Rule4, we get:

$$R(A, R_1(C), R'(B, D, E, R_2(F)))$$

Now we apply Rule2 to promote E to get the following NF-NR relation:

$$R(A, R_1(C), R''(B, D, R_2(F)), E)$$

(Rule5): restructuring to satisfy condition 6 of NF-NR(Definition 3.6)

Let R be a nested relation and R' *be a nested attribute of R. Let* $S = K^{abs}{}_{R'} - ATTR_s(R')$. *If* $S \neq \emptyset$ *and* $S \subset K^{abs}{}_R \cap ATTR_s(R)$, *then restructure R as follows:*

a). R' *is still a nested attribute of R.*

b). Make $ATTR_s(R) = S$.

c). The rest attributes of R, single-valued or nested, form a new nested attribute of R.

Example 4-5. Consider the following relation:

$$R(K, A, B, R_1(C, D), R_2(E, F))$$

with $KA => R_1$, $KB => R_2$.

The key of relation R is KAB but R is not in NF-NR due to the obvious partial EFDs. Use

Rule4, we first restructure R into the following:

$$R(K,\ A,\ R_1(C,\ D),\ R'(B,\ R_2(E,\ F)))$$

Now the absolute key of relation R is KA while that for relation R' is KB. Notice that condition 6 in NF-NR definition is violated. R is still not in NF-NR. Apply Rule5 we get the following:

$$R(K,\ R''(A,\ R_1(C,\ D)),\ R'(B,\ R_2(E,\ F)))$$

Now R is in NF-NR.

(Rule6): splitting of nested attribute

Let R be a relation and R′ be its nested attribute which does not contain any nested attribute and is all key. If ATTR(R′) can be partitioned into three subsets, A(≠∅), B(≠∅), and C(might be empty), such that $KC\ {-}{>>}\ A|B$ *where K is a (set of) single-valued attribute(s) of R, remove C from R′ and make it a (set of) single-valued attribute(s) in R, decompose R′(with C removed) into two nested attributes of R,* R'_1 *and* R'_2*, such that* $ATTR(R'_1) = ATTR_s(R'_1) = A$ *and* $ATTR(R'_2) = ATTR_s(R'_2) = B$.

Applying this rule will remove structures in which condition 4 of NF-NR definition is violated. This is shown by the following example.

Example 4-6. Consider the following relation:

$$R(A,\ R_1(B,\ C,\ D))$$

where $AB\ {-}{>>}\ C|D$.

The key of relation R is A. Relation R_1 is all key and contains no nested attributes. Add attribute A into relation R_1 to get the following relation:

$$R_1'(A,\ B,\ C,\ D)$$

Relation R_1' is all key and it is not in NF-NR since the existence of MVD $AB\ {-}{>>}\ C|D$ violates condition 4 in Definition 3.6. According to condition 5 in Definition 3.6, R is not in NF-NR. Using Rule6, we can restructure relation R into the following:

$$R(A,\ B,\ R_{11}(C),\ R_{12}(D))$$

Now it is in NF-NR.

In Rule6 we do not consider cases where R' has nested attributes. If it does, then we need to use other rules. First, we use Rule2 to promote these nested attributes. If the promotion is successful, then we use Rule6. This is illustrated in the example below.

Example 4-7. Consider the nested relation

$$R(A,\ R_1(B,\ C,\ R_{11}(D)))$$

with $A\ -{>>}\ B, A\ -{>>}\ C, A\ -{>>}\ D$.

We transform it into an NF-NR relation as follows.

step 1. Use Rule2, R_{11} is promoted, we get

$$R(A,\ R_1(B,\ C),\ R_{11}(D))$$

step 2. Use Rule6, the nested attribute $R_1(B,\ C)$ is split, we get an NF-NR relation

$$R(A,\ R_1'(B),\ R_1''(C),\ R_{11}(D))$$

(Rule7): Merging relations

Consider two NF-NR relations:

$$R_1(K_{R_1},\ A_0, \ldots,\ A_n,\ T_0, \ldots,\ T_m)$$

$$R_2(K_{R_2},\ B_0, \ldots,\ B_k,\ S_0, \ldots,\ S_t)$$

A_i's (i=0,n) and B_j's (j=0,k) are single-valued attribute names. T_u's (u=0,m) and S_v's (v=0,t) are nested relations. K_{R_1} and K_{R_2} are the primary keys for R_1 and R_2, respectively. Assume $K_{R_1} \subseteq K_{R_2}$ and there exists no extended functional dependency that involves attributes from both R_1 and R_2. Let

$$\{B'_0, \ldots,\ B'_{k'}\} = \{B_0, \ldots,\ B_k\} - \{A_0, \ldots,\ A_n\}$$

and

$$\{S'_0, \ldots,\ S'_{t'}\} = \{S_0, \ldots,\ S_t\} - \{T_0, \ldots,\ T_m\}$$

Merge relations R_1 and R_2 to get relation R as follows:

1) If $K_{R_1} \subset K_{R_2}$, then

$$R(K_{R_1}, A_0, \ldots, A_n, T_0, \ldots, T_m, R'(K_{R_2} - K_{R_1}, B'_0, \ldots, B'_{k'}, S'_0, \ldots, S'_{t'}))$$

2) If $K_{R_1} = K_{R_2}$, then

$$R(K_{R_1}, A_0, \ldots, A_n, B'_0, \ldots, B'_{k'}, T_0, \ldots, T_m, S'_0, \ldots, S'_{t'})$$

Rule7 helps to avoid unnecessary fragmentation of relations. This is illustrated in the following example.

Example 4-8. Consider the following relations

$$R_1(A, D, R_{11}(F, G)) \text{ and } R_2(A, B, C)$$

with $A \rightarrow D$, $A => R_{11}$, $AB \rightarrow C$.

The primary key for R_1 is A, that for R_2 is AB. There is no functional dependency among non-prime attributes of R_1 and R_2. Using Rule7, we can merge R_1 and R_2 into the following relation:

$$R(A, D, R_{11}(F, G), R_2'(B, C))$$

R is in NF-NR.

Rules 1-7 are "rules of thumb". They can be applied whenever applicable. For a restructuring approach as discussed above, two problems are interesting to consider. First, given a set of relations and EFDs, is it always possible to restructure it into a set of NF-NR relations using heuristic rules such as Rules 1-7? To answer this question, we face the same difficulty encountered when synthesizing approach is investigated. The major problem is still given by the presence of MVDs. A formal investigation into this question is not in the scope of this paper.

Second, does the process of restructuring give unique solution? The answer is no. This is shown by the following example.

Example 4-9. Consider the nested relation

$$R(A, B, R_1(C), R_2(D), E)$$

with $A \text{ }->>\text{ } C$, $B \text{ }->>\text{ } D$, $AB \rightarrow E$.

Obviously AB is the key. R is not in NF-NR due to the two partial MVDs $AB \text{ }->>\text{ } C$ and $AB \text{ }->>\text{ } D$. In this example, we go by two steps. Step 1, apply Rule4 according to $A \text{ }->>\text{ } C$ to get:

$R(A,\ R_1(C),\ R_3(B,\ R_2(D),\ E))$

Notice we have dependency $B\ =>\ R_2$ but B does not contain the absolute key of R_3, namely AB. So we need step 2: apply Rule1 to get the followings:

$R'(A,\ R_1(C),\ R_3(B,\ E))$

$R''(B,\ R_2(D))$

Now they are in NF-NR. The result of restructuring of this relation is not unique. If in step 1, we apply Rule4 according to $B\ -\!>>\ D$, we can have:

$R(B,\ R_2(D),\ R_3'(A,\ R_1(C),\ E))$

Then we apply Rule1 in relation R_3', we'll have the following relations as the final result:

$R'(B,\ R_2(D),\ R_3'(A,\ E))$

$R''(A,\ R_1(C))$

From the above example, we see that the application of Rule1-7 does not necessarily give unique result. However, the significance of this set of rules is not that it establishes a well-defined restructuring procedure for normalizing nested relations, but is that it gives practical heuristics and provides some insight into the normalization tasks.

4.2. ER Approach to NF-NR Nested Relational Database Design

The task of designing "good" nested relational database can be made easier if more semantics is available. In [7], a normal form for ERD is proposed. In [9], this ER normal form is used to design normal form nested relations. The approach consists of normalizing an ER diagram and converting a normalized ER diagram into a set of NF-NR relations. Normalizing an ER diagram effectively removes ambiguities, anomalies and redundancies on a semantic level. By converting a normalized ER diagram into a set of nested relations, we can obtain a database schema with clean semantics and is yet in good normal form. As suggested by the theorems in [9], designing good nested relational database using ER technique might be the most promising approach to the problem. We further discuss this approach in this section.

4.2.1. Entity-Relationship Approach

The entity-relationship(ER) approach for database design was proposed by [2]. This approach captures real world semantics with the concepts of entity and relationships among entities. An **entity** is an object which exists in our minds and can be distinctly identified. Entities can be classified into **entity types**(or entity sets) which are sets that contain entities satisfying a set of predefined common properties.

Let $E = \{E_1, \ldots, E_n\}$ be a set of entity types. A **relationship set** R over E is defined as:

$$R \subseteq \{ < e_1, \ldots, e_n > \mid e_i \in E_i,\ 1 \le i \le n \}$$

Elements in R satisfy a set of predefined common properties and are called **relationships**.

The structural properties of an entity type E (or a relationship set R) are represented by **attributes**. An attribute A of E(or R) is a mapping:

$$A\colon E(or\ R) \rightarrow V_1 \times \cdots \times V_n$$

where V_i's are value sets. An **atomic attribute** always has $n = 1$. When $n \ge 2$, A is called a **composite attribute**.

In ER approach, we distinguish among four types of attributes, according to the nature of the mappings defined. These are: one-to-one(1:1), many-to-one(m:1), one-to-many(1:m) and many-to-many(m:m). If an attribute is 1:m or m:m, it is called **multivalued attribute**. Otherwise, it is a **single-valued attribute**.

A minimal set of attributes $K = \{A_1, \ldots, A_m\}$ of an entity type E is called a **key of entity type** E if it defines a one-to-one mapping:

$$M_e : E \rightarrow V_1 \times \cdots \times V_m$$

where V_j $(1 \le j \le m)$ is the value set of attribute A_j. An entity may have multiple keys. One of them is designated as the **identifier** of entity type E. Sometimes an entity set does not have its own key(see identifier dependency in the next paragraph). In this case, the identifier of the entity set may include identifiers of other entity set(s).

Let $K = \{B_1, \ldots, B_s\}$ be a minimal set of identifiers of some entity types participating in a relationship set R. K is called a **key of the relationship set** R if it defines a one-to-one mapping

$$M_r : R \rightarrow V_1 \times \cdots \times V_s$$

where V_k $(1 \le k \le s)$ is the value set of B_k. A relationship set may have multiple keys. One of them is

designated as the **identifier of relationship set** *R*. In ER approach, different semantics of relationship can be captured. These are introduced below.

1. weak relationship sets

Some entity type does not have an identifier of its own but has to be identified by its relationship with other entities. Such relationship set is called **identifier dependent relationship set**. There may be entity type whose existence depends on another entity type with which it is related via a relationship. Such relationship is called **existence dependent relationship set**. Notice that identifier dependency is also an existence dependency. In both cases, the entity type is called a **weak entity type** and the relationship set is called a **weak relationship set**. Entity type which is not a weak entity type is called a **regular entity type**.

2. ISA relationship sets

If each entity in one entity type E_1 is also in another entity type E_2, E_1 and E_2 are related by an **ISA relationship set**.

3. UNION, INTERSECT, and DECOMPOSE relationship sets

If an entity type is the union(or intersection) of some other entity types, such a relationship is called **UNION (or INTERSECT) relationship set**. If an entity type can be partitioned or decomposed into several other entity types, such a relationship is called a **DECOMPOSE relationship set**.

The relationships described above are special relationships. These relationships have no attribute associated. Relationship sets that are not special relationships are called **regular relationship sets**.

The result of modeling a real world application in ER approach is an **entity-relationship schema**. Usually, this schema is represented by an **entity-relationship diagram (ERD)**. An ERD provides graphical representation of all the entity types, relationship sets and attributes. For the purpose of this paper, we omit the technical detail about ERD. Hereafter, we use the term ERD to refer to an entity-relationship schema.

4.2.2. A Normal Form for ERD

Data dependencies are part of the real world semantics. In [8], an analysis of data dependencies is given. The analysis shows that data dependencies should be modeled precisely early in the design stage for a correct and complete database representation of semantics. In [7], a normal form for ERD, ER-NF, is defined. The goal is to define and represent a clean semantics by taking into consideration certain data dependencies. Here we give the outline of the definition for ER-NF.

Let *E* be an entity type with identifier *K*. The **basic dependencies of an entity type** *E*, $BD(E)$,

is defined as follows:

1) For each key K_1 of E which is not an identifier, $K_1 \rightarrow K,\ K \rightarrow K_1 \in BD(E)$.

2) For each m:1 attribute A of E, $K \rightarrow A \in BD(E)$.

3) For each 1:m multivalued attribute A of E, $A \rightarrow K \in BD(E)$.

4) For each 1:m and m:m multivalued attribute A of E, $K -\!\!>\!\!> A \in BD(E)$.

5) No other FD or MVD is in $BD(E)$.

An entity type E in an ERD is in **entity normal form(E-NF)** if all FDs and MVDs, which only involve attributes of E, can be derived from $BD(E)$ by using the inference rules for FDs and MVDs[10].

Let R be an relationship set with identifier K. And F be the set of FDs which only involve the identifiers of entity types participating in R. The **basic dependencies of a relationship set** R, $BD(R)$, is defined as follows:

1) For each key K_1 of R which is not an identifier, $K_1 \rightarrow K,\ K \rightarrow K_1 \in BD(R)$.

2) For each m:1 attribute A of R, $K \rightarrow A \in BD(R)$.

3) For each 1:m multivalued attribute A of R, $A \rightarrow K \in BD(R)$.

4) For each 1:m and m:m multivalued attribute A of R, $K -\!\!>\!\!> A \in BD(R)$.

5) Let $A \rightarrow B \in F$ be a full dependency where A is a set of identifiers of entity types participating R and B is the identifier of some entity type participating in R. If A is a key or B is part of a key of R, then $A \rightarrow B \in BD(R)$.

6) No other FD or MVD is in $BD(R)$.

A relationship set R in an ERD is in **relationship normal form (R-NF)** if all the FDs and MVDs, which only involve attributes of R and identifiers of entity types participating in R, can be derived from $BD(R)$ by using the inference rules for FDs and MVDs.

Let D be an ERD. The **set of basic dependencies of ERD** D, $BD(D)$, is defined as the union of the sets of basic dependencies of all entity types in D and those of all the relationship sets in D. An ERD D is in **ER normal form (ER-NF)** if it satisfies the following conditions:

1) all attribute names are distinct and of different semantics.

2) every entity type and relationship set in D is in E-NF and R-NF, respectively.

3) All the FDs and MVDs are implied by $BD(D)$.

4) Every relationship set R in D which has no attribute associated satisfies two conditions. First, R is not the result of join of any two other relationship sets. Second, R is not the result of join of any

three other relationship sets.

The process of normalizing an ERD goes on a semantic level. Guidelines and steps for this process are given in [7].

4.2.3. Representing an ERD with a set of nested relations

When translating an normal form ERD into a set of nested relations, the special relationship sets ISA, UNION, INTERSECT and DECOMPOSE can be represented by inclusion constraints. Moreover, *role names* must be assigned to certain attributes when a cycle is present in an ERD. These issues are well discussed in [7, 9] and will not be discussed further. The process of generating nested relations for entity types and regular/weak relationship sets in an ERD is given below.

ALGORITHM-1. generating nested relations for entity types

Let D be an ERD, nested relations for entity types in D are generated as follows:

1. For each entity type A which is either regular or a weak entity type that identifies itself, we construct a nested relation E_A as follows:

   a). All the single-valued attributes of A are single-valued attributes in E_A.

   b). The identifier and keys of A are the absolute key and candidate keys of E_A, respectively.

   c). Each m:m multivalued attribute B (which can be a composite attribute) is a nested attribute R_b of E_A such that $ATTR_s(R_b) = ATTR(R_b) = B$. Relation R_b is all key and has absolute key $K^{abs}{}_{E_A} \bigcup B$.

   d). Each 1:m multivalued attribute C (which can be a composite attribute) is a nested attribute R_c of E_A such that $ATTR_s(R_c) = ATTR(R_c) = C$. Relation R_c is all key and has absolute key C.

2. For each entity type B which is identifier dependent on another entity type, construct relation E_B as in 1 above except b). Instead, those attributes of B that are involved in the identifier of B will form the key of nested relation E_B, K_B. The absolute key of B will be constructed in the next step.

3. For each weak entity type with key K_B and let A be the entity(strong or weak) that B depends on. Then make E_B a nested attribute of E_A. If the dependency between B and A is an existence dependency, then E_B has absolute key K_B. If the dependency is an identifier dependency, then E_B has absolute key $K_B \bigcup K^{abs}{}_{E_A}$.

end of ALGORITHM-1.

ALGORITHM-2. generating nested relations for regular relationship sets.

Let D be an ERD. For each regular relationship set S in D, we construct a nested relation R_S as follows:

1. All the identifiers of the entity types participating in S and all the single-valued attributes of S are the single-valued attributes of R_S.

2. The identifier, keys of S are the absolute/primary key and keys of R_S. All the 1:1 attributes of S are also keys of R_S.

3. Each m:m attribute B(which can be a composite attribute) forms a nested attribute R_b of R_S such that $ATTR_s(R_b) = ATTR(R_b) = B$. R_b is all key and is with absolute key $B \cup K_{R_s}$.

4. Each 1:m attribute C(which can be a composite attribute) forms a nested attribute R_c of R_S such that $ATTR_s(R_c) = ATTR(R_c) = C$. R_c is all key and is with absolute key C.

5. If $E \rightarrow F$ is a non-trivial full dependency which only involve attributes in $ATTR(R_S)$, where E is not a key of S, then record this dependency as an integrity constraint on R_S.

end of ALGORITHM-2.

We give a simple example to show the result of converting an ERD to nested relations by using the above two algorithms.

Example 4-10. Consider the ER scheme represented by the diagram in Fig.4-1. Entity type A has identifier *a#*, an m:1 attribute a_1 and an m:m composite attribute a_{23} consisting of a_2 and a_3. Entity type B has identifier *b#*, an m:1 attribute b_1 and an m:m attribute b_2. Entity type B is existence dependent on entity type A. Entity type C has identifier *c#* and an m:1 attribute *c*. There is a relationship set involving entity types *A* and *C* with an m:1 attribute *d*.

Applying ALGORITHM-1 and ALGORITHM-2 to this ERD, we get the following database schema:

$$E_A(a\#,\ a_1,\ R_1(a_2,\ a_3),\ E_B(b\#,\ b_1,\ R_2(b_2))$$

$$E_C(c\#,\ c)$$

$$R_{C-A}(c\#,\ a\#,\ d)$$

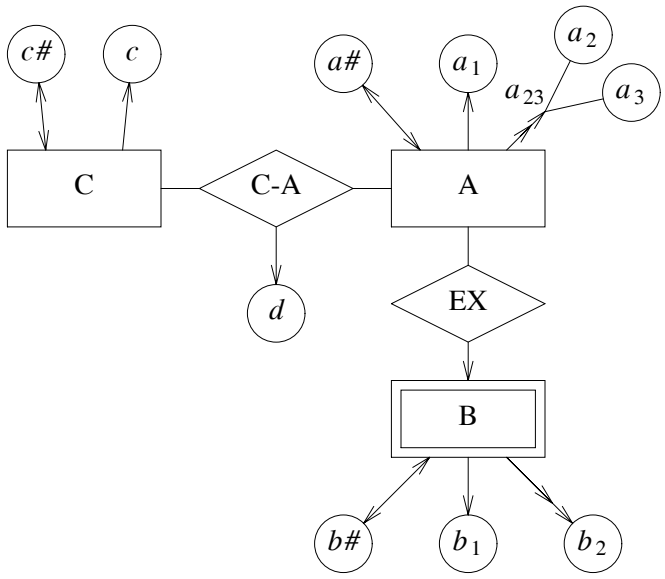

Fig.4-1. An Entity-Relationship Schema

4.2.4. NF-NR and ER-NF

Using ALGORITHM-1 and ALGORITHM-2 to generate nested relations from an ERD, we see that the model OF NESTed relations captures the following semantics:

1. **multivalued attributes**

Multivalued attributes of entity types and relationship sets are represented as nested attributes. 1:m attributes are distinguished from m:m attributes by different absolute keys.

2. **weak entity types**

A weak entity type is represented as a nested attribute in the nested relation representing the entity it depends on. Identifier dependency and existence dependency are captured by different absolute keys.

The following theorem is a formal expression of the fact that NF-NR is capable of capturing the semantics represented by an ERD in ER-NF.

Theorem-5. Let D be an ERD in ER-NF and T be the set of nested relations generated for D by ALGORITHM-1 and ALGORITHM-2. Then the followings hold:

1) All relations in T are in NF-NR.

2) For each entity type A in D, if the nested relation $E_A \in T$ has no nested attributes, it is in 5NF.

3) For each relationship set S in D, if the nested relation $R_S \in T$ has no nested attributes, it is in 3NF or 5NF.

4) T is in the normal form for set of nested relations.

A very similar theorem is given in [9]. The proof is rather straightforward. We omit the proof in this paper.

5. An Evaluation of Normal Forms for Nested Relations

Several normal forms for nested relations were proposed[11, 12, 9]. These normal forms are proposed based on different concepts about nested relation. In this section, we evaluate three normal forms, NF-NR, whose predecessor is the NF-N3 in [9], NF, proposed in [11] and NNF, proposed in [12]. One interesting question is: what is the criteria based on which we can say that a normal form is "good"? In this section, we describe several desirable properties of normal forms for nested relation. The evaluation is carried out based on these arguments.

5.1. Types of Dependencies

There are two types of well-known dependencies, namely, functional dependency and multi-value dependency. By definition, both FD and MVD deal with the dependencies among attribute values. We refer to them as **value-based dependencies**. The concept of extended functional dependency(EFD) in this paper allows the right hand side of a dependency to be a nested relation. However, with Theorem-1, we see that EFDs are basically FDs or MVDs, i.e., they are still value-based.

In [11], another type of dependency, which allows the left hand side of an EFD to be a relation, is proposed. The following example was given in [11] to show the meaning of this type of dependency.

Example 5-1. Consider the relation

$$U - TRIANGLE(3 - POINTS(X, Y), AREA, COLOR)$$

which records the size and color of triangles. One tuple in this relation might be:

$$(\{(1, 1), (3, 1), (3, 3)\}, 2, red)$$

In [11], the dependency

$$3-POINTS \rightarrow AREA,\ COLOR \tag{5-1}$$

was identified according to the fact that one triangle has one area and one color. Notice that 3-POINTS is a relation itself. Semantically, this relation should contain exactly three tuples but this constraint was not presented in the original example.

We distinguish between relation and data. Relation is a way of organizing data while data are values from real world. Hence real world data dependencies should be value-based, e.g. FDs and MVDs. No data organization can be involved. In this sense, dependency (5-1) is difficult to justify. Actually, the attribute 3-POINTS is intended to be an atomic attribute with a type such as "3 ordered integer pairs" rather than a nested attribute. The distinction was not made by [11]. This is why the U-TRIANGLE relation becomes nonsense after being flattened. The problem can be solved by introducing complex data types into database. Another way is to introduce an identifier into the relation(similar to introducing EMP# for identifying employees) as follows:

$$U-TRIANGLE'(TRIANGLE\_ID,\ 3-POINTS(X,\ Y),\ AREA,\ COLOR)$$

This way is fine when we talk about polygons. However, if only triangles are considered, this way is still wrong since there is no way to make sure that the nested relation $3-POINTS$ contains exactly three tuples! It seems that the only correct way to model this case is to use the following relation:

$$R(X_1,\ Y_1,\ X_2,\ Y_2,\ X_3,\ Y_3,\ Area,\ Color)$$

with the following dependency:

$$X_1, Y_1, X_2, Y_2, X_3, Y_3 \rightarrow Area,\ Color$$

This fits into our definition of EFD.

In NNF[12], all the dependencies considered are basically FDs or MVDs which are value-based. However, NNF does not distinguish between FD and MVD and hence has the tendency of over-nesting and unnecessary splitting. For example, consider a relation

$$R(A,\ B,\ C)$$

where $A \rightarrow C$, $A \twoheadrightarrow B$. According to NNF, the result of normalizing this nested relation should be:

$$R(A,\ R_1(B),\ R_2(C)) \tag{5-2}$$

while according to our definition of NF-NR, the result should be:

$$R(A,\ R_1(B),\ C) \tag{5-3}$$

Intuitively, (5-3) is a better form since it is simpler and is a more precise reflection of real world dependencies. It should be very interesting if the algorithms given in [12] can be extended to include FDs.

5.2. Relationship with Normal Forms for flat Relations

Two properties of a normal form for nested relations are desirable.

Property 1: *normal form for nested relations should reduce to normal form for flat relations when the relation under consideration is flat.*

This is naturally desirable in accordance with the fact that nested relation is a stronger model than flat relation in that the latter is a reduced case of the former. According to conditions 2 and 3 in Definition 3.6, NF-NR satisfies this property. Both NF in [11] and NNF in [12] satisfy this property(in the case of flat relations, they all reduce to 4NF). However, since NNF does not distinguish between FD and MVD, which usually appear in flat relations, it reduces to 4NF but does not allow the existence of any non-trivial FD. The set of flat relations normalized to satisfy reduced NNF will be unnecessarily fragmented.

Property 2: *A normalized nested relation can be represented by a set of flat relations which cover the same EFDs and is in good normal form.*

Treatment for value-based dependencies is well established for flat relation cases. Since we continue to study value-based dependencies in nested relation model, results from normal forms for flat relations should be used to justify the quality of a new normal form for nested relations. According to the definition given in this paper, NF-NR satisfies this property since every NF-NR relation can be decomposed into a set of 3NF/4NF relations. By this we ensure that the undesirable anomalies are eliminated. Both NF[11] and NNF[12] satisfy this property. However, the result of decomposition might be quite different from one another. Intuitively, decomposing NF-NR relations into flat relations will give less fragmented result in most cases.

5.3. Practicality and Feasibility

Identifying keys in flat relations is important only because a relational DBMS supports certain indexing structures to enforce key dependencies. In a nested relational database, enforcing data dependencies is more complicated. However, at least there must be a counterpart of key in flat relation defined for nested relation.

The fact that a counterpart for key in relational databases should exist for nested relation model is reflected in all three normal forms. However, only in NF-NR, it is explicitly formulated with the concept of *absolute key*. As shown by previous discussion in this paper, absolute key plays the same role in nested relation model as key plays in flat relations. With this concept, NF-NR enforces desired dependencies while giving more reasonable representations of real world semantics. This is shown by the following example.

Example 5-2. Consider the following NF-NR relation

$$STUDENT(student\#,\ name,\ C(course\#,\ grade))$$

Note that we have $student\#,\ course\# \rightarrow grade$ and $student\# \rightarrow name$. The absolute key for relation C is {student#, course#}. By enforcing this absolute key, we enforce all the desired data dependencies. In this sense, STUDENT relation is a proper representation of real world semantics.

NF does not allow the existence of functional dependencies which are defined among the key of a relation and the attributes of a nested attribute. Hence the relation *STUDENT* is not in NF. In fact, the NF definition given in [11] is not correct because no unnormalized relation can satisfy the condition (3). Due to the FD $student\# \rightarrow name$(which will be taken as an MVD) in STUDENT, it is not in NNF either.

Another unique feature that makes NF-NR more practical and feasible is its relationship with ER-NF described by Theorem-5. With this relationship, redundancies and ambiguities can be removed on a semantic level. The set of nested relations generated from an ERD in ER-NF will have a clean semantics and is yet in good normal form. NF[11] and NNF[12] do not provide similar design tools.

5.4. Versatility

Normalization may restrict the capability of a relational model in expressing real world dependencies while removing anomalies. For example, there are 3NF relations which do not have BCNF representations. A practical normal form should hence balance between the two goals, i.e. to remove anomalies as far as possible without sacrificing too much the modeling capability. In both NF and NNF, all the single-valued attributes of a normalized nested relation must be in BCNF. This is not required by NF-NR(we go up to 3NF only). Hence, there are relations that cannot be expressed by NF or NNF but can be expressed in NF-NR. Since BCNF is more restricted than 3NF, the reverse is not true. In this sense, NF-NR improves versatility over NF and NNF.

5.5. Elimination of Global Redundancy

A normal form for nested relations, just like that for flat relations, should be able to eliminate redundancies in set of relations as well as those in a single relation. Similar work was done for flat relations by [6] where an *improved third normal form* was proposed to eliminate redundancies among a set of flat relations. In this paper, a normal form for set of nested relations is defined. Example shows that this definition removes global redundancies from a set of nested relations. NF and NNF do not provide similar definitions.

6. Conclusions

In this paper, a normal form for nested relations, NF-NR, is presented. We first give a detailed definition for nested relations. Our definition is an extension of the normal forms for flat relation by allowing an attribute to be a nested relation as well as atomic. The definition naturally reduces to that of flat relation when no nested attribute is involved. Undesirable anomalies in nested relations are identified. NF-NR is designed to remove all these anomalies. In accordance with the fact that nested relation is an extended case of flat relation, NF-NR is an extension of normal forms for flat relations. It reduces to normal form for flat relation when the relation under consideration is flat. Properties of NF-NR are shown by several theorems.

Two approaches to NF-NR database design are discussed in detail. The first approach considered is that by using restructuring rules. Seven heuristic rules are given to transform nested relation(s) into NF-NR relation(s). The use of these rules is shown by examples. This approach, although ad hoc, takes advantage of various features of NF-NR and is practically simple and flexible. The second approach considered is that by using ER techniques. We relate NF-NR to ER-NF, a normal form proposed earlier in [7], by defining a procedure to map an ERD in ER-NF into a set of nested relations. A theorem is given to show that the set of nested relations thus generated is in NF-

NR. The advantage of this approach is that anomalies and redundancies are effectively removed when an ERD is normalized. This normalization is carried out with ER data model which is capable of capturing more semantics. By converting an ERD in ER-NF into a set of nested relations in NF-NR, we obtain a data base schema in nested relations which has a clean semantic and is yet in good normal form. This approach is very promising.

An evaluation of several existing normal forms, NF-NR, NF in [11] and NNF in [12] is given. The evaluation goes along five dimensions, namely, types of data dependencies considered, relationship with normal forms for flat relations, practicality and feasibility, versatility, and elimination of global redundancy among a set of nested relations. NF-NR considers FDs as well as MVDs and is better than NNF in which only MVDs are considered. There is a close and well-defined relationship between NF-NR relations and flat relations in good normal form. This justifies NF-NR as a good extension to normal forms for flat relations. Its concept of *absolute key* and relationship with ER-NF also make NF-NR a more practical and feasible normal form for nested relations. NF-NR is more versatile than NF and NNF. Finally, different from both NF and NNF, NF-NR provides a definition to remove global redundancies among a set of nested relations. The result of our evaluation shows that NF-NR is a more practical normal form for nested relations.

REFERENCES

[1] S. Abiteboul, M. Scholl. From simple to sophisticated languages for complex objects. *in* [5].

[2] P. P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transaction on Database Systems*. 1(1), (1976).

[3] E. F. Codd. Further Normalization of the Data Base Relational Model. In *Data Base Systems.* Randell Rustin eds, Prentice Hall(1972).

[4] G. Jaeschke, H. J. Schek. Remarks on the Algebra of Non first Normal Form Relations. *ACM Symposium on Principles of Database Systems*. Los Angeles(1982).

[5] Special Issue on Nested Relations. *IEEE Data Engineering*. Aug(1988).

[6] T. W. Ling, F. W. Tompa, T. Kameda. An Improved Third Normal Formal for Relational Database. *ACM Transaction on Database Systems.* 6(2), (1981).

[7] T. W. Ling. A Normal Form for Entity-Relationship Diagrams. *Proc. 4th International Conference on Entity-Relationship Approach*(1985).

[8] T. W. Ling. An analysis of multivalued and join dependencies based on the entity-relationship approach. *Data and Knowledge Engineering*. vol 1, (1985).

[9] T. W. Ling. A Normal Form for Sets of Not-Necessarily Normalized Relations. *Proc. 22nd Annual Hawaii International Conference on Systems Science*(1989).

[10] D. Maier. *The Theory of Relational Databases*. Pitman(1983).

[11] A. Makinouchi. A Consideration on Normal Form of Not-Necessarily Normalized Relation in the Relational Data Model. *3rd International Conference on VLDB*. Japan(1977).

[12] Z. M. Ozsoyoglu, L. Y. Yuan. A New Normal Form for Nested Relations. *ACM Transaction on Database Systems.* 12(1), (1987).

[13] M. A. Roth, H. F. Korth. The Design of ¬1NF Relational Databases into Nested Normal Form. *ACM SIGMOD*(1987).

[14] M. A. Roth, H. F. Korth, D. S. Batory. SQL/NF, A Query Language for ¬1NF Relational Databases. *Information Systems*. 12(1), (1987).

[15] M. A. Roth, H. F. Korth, A. Siberschatz. Extended Algebra and Calculus for Nested Relational Databases. *ACM Transaction on Database Systems*. 13(4), (1988).

[16] J. D. Ullman. *Principles of Database Systems*. Second edition, Computer Science Press(1982).