Regular Paper

On Query Transformation for Non-First-Normal-Form Relational Databases

KUNITOSHI TSURUOKA*

Simple extended tuple relational calculus is proposed for non-first-normal-form (NF2) relational databases, as a way of facilitating the understanding of query representation. In order to allow NF2 queries over first-normal-form (1NF) relational databases, a query transformation algorithm is also proposed. This transforms a query for a nested relation into one for an unnested relation. Additionally, the equivalence of transformed queries is proved in the sense that SELECT and UNNEST are commutable. The proposed extended tuple relational calculus and query transformation algorithm can serve as a basis for the safe implementation of NF2 relations over 1NF relations.

1. Introduction

Much attention has been paid to non-first-normal-form (NF2) relational databases in recent years. Some researchers have studied NEST/UNNEST operations [5-7, 14], NF2 dependencies [2, 6, 10], NF2 relational algebra and query languages [3-5, 11-13], and NF2 relational calculus [8]. From a practical viewpoint, NF2 relations are important for such applications as office form management [16-19], information retrieval [14], and artificial intelligence, where repeating groups are usually the basic units of concern.

Several approaches to the processing of NF2 relations can be considered. Research is being done on how to use NF2 relations for an internal (physical) model [13], as well as for external and conceptual models. From my point of view, however, the use of first-normal-form (1NF) relations for the internal model is currently preferable (The difference between 1NF and 3NF-5NF is not important here.). The reasons are as follows:

1. Information sharing among several applications is easier for 1NF relations, since many applications prefer very simple data structures.
2. Several different views can be derived from 1NF relations more easily than from NF2 relations.
3. Techniques for NF2 file implementation are still immature. For example, the relational storage technique cannot efficiently handle cases in which different types of tuples are stored on the same page, and variable-length updates occur.

For these reasons, the NF2 conceptual/external model, based on the 1NF internal model, is an important contribution to the current technology.

*C & C Systems Research Laboratories, NEC Corporation, 4-1-1 Miyazaki, Miyamae-ku, Kawasaki, 216, Japan.

As a means of implementing NF2-type queries over 1NF relations, query transformation from NF2 format into 1NF format should be considered, since only 1NF queries are executable for internal 1NF relations. Additionally, a query transformation must be consistent in the sense that it preserves the meaning of the original query. Recent work on queries for NF2 relations, however, has not been very closely concerned with this transformation or its properties. Actually, nested queries based on nested relational algebra have been proposed [1, 11-14]. The nested structure of the query commands, however, seems rather procedural and not very easy to understand. Query transformation based on nested tuple relational calculus is considered in Kiyama and Nakano [8]. However, a model that directly represents the nested relations structure has not been proposed.

First, I specify the NF2 data model by using a LISP-like notation, which accurately captures the NF2 relation structure. It can be a good tool for fully computable NF2 relation processing. Next, a simple extension to tuple relational calculus (TRC) is proposed in order to specify extended-tuple relational calculus (ETRC). Query expression, based on ETRC, that has a flat (not nested) syntax is also specified. My ETRC is different from other NF2 algebras [3-5, 11-13], since the ETRC has a flat query syntax, and can handle both NF2 schemas and NF2 instances uniformly by using a formalized list structure. Queries based on ETRC are easy to understand, and help users to comprehend the query semantics properly. Next, I propose an algorithm that transforms an ETRC query into a TRC query that works for unnested (1NF) relations. At the same time, the equivalence between ETRC and the transformed TRC is proved, thus ensuring the validity of the equation UNNEST (SELECTN(R))=SELECTU (UNNEST(R)).

Here, R is a nested relation, SELECTN is a select operator for the nested relation, and SELECTU is another select operator for an unnested relation. This equation ensures that, instead of executing an NF2 query directly, we can obtain the same result by issuing an equivalent 1NF query for internal 1NF relations. That is, the equation ensures the safe implementation of NF2 relations over 1NF relations.

2. Basic Ideas

This section roughly sketches the basic concepts of the proposed ideas without using exact notation.

NF2 relational algebra (union, intersection, selection, projection, etc.) and its properties have been quite widely studied [3, 5, 6, 13]. It is said that the equation

UNNEST (SELECT(R))=SELECT (UNNEST(R)) (*)

does not hold for "select" if the selection condition includes unnested attributes, since the same selection condition is not applied to the nested and unnested relations [5].

This paper proposes ETRC and an algorithm that transforms a selection condition for an NF2 relation into one for an unnested relation. The two resulting conditions are equivalent in the sense that they satisfy the above equation (*). Figure 1 shows an example. SELECTN is a query in the form of ETRC, where conditions called "descendant-ancestor relationship" between tuples (for example T22≺T11) are used. Such a condition means that there is a subcomponent-component relationship between two tuples. That is, a subtuple (such as T22) is a component of a component of . . . a supertuple (such as T11). ETRC SELECTN can be transformed into TRC SELECTU, which is used for the unnested (1NF) relation. The two selects satisfy the equation

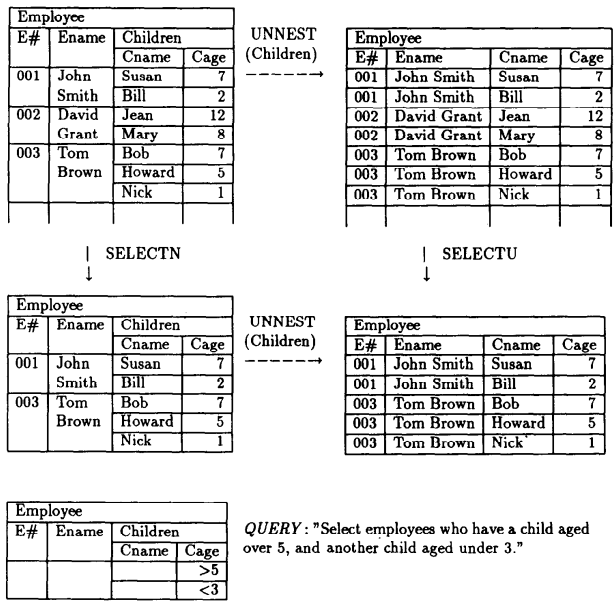
UNNEST (SELECTN(Employee))
=SELECTU(UNNEST(Employee)).

3. NF2 Relations and Operations

3.1 NF2 Relations

The following are brief definitions and notations for NF2 relations. The descriptions below are based on LISP notation. Each object is expressed as a list, and some LISP functions such as "append" and "cons" are used. At first, an (extended) relation is defined.
relation::=(schema instance)
schema::=attributeset | (append attributeset schemaset)
attributeset::=(attribute) | (cons attribute attributeset)
schemaset::=(schema) | (cons schema schemaset)

Instance is defined below. An attribute, which has a simple value, is the usual object defined for relational



SELECTN : {T11 | (T11∈Employee) and (∃T21∈Children) and (∃T22∈Children); (T21≺T11) and (T22≺T11) and (T21(Cage)>5) and (T22(Cage)<3)}
SELECTU : {U11 | (U11∈Employee) and (∃U12∈Employee) and (∃U13∈Employee); (U12(E#)=U11(E#)) and (U13(E#)=U11(E#)) and (U12(Cage)>5) and (U13(Cage)<3)}

Fig. 1 Extended Tuple Relational Calculus and Query Transfor mation.

databases (not defined here). An important point is that we require a schema to have at least one attribute (this comes from the primary key requirement specified later).

Next, the value and an (extended) tuple are defined.

instance::=tupleset
tupleset::=(tuple)|(cons tuple tupleset)
tuple::=valueset|(append valueset instanceset)
valueset::=(value)|(cons value valueset)
instanceset::=(instance)|(cons instance instanceset)

A value corresponds to an attribute value for the usual relational database term (not defined here). Note that an instance includes multiple instances for a "lower-level" schema. (Actually, an instance should be a "set" rather than a "list" as defined here. For our purpose, however, the difference is not important.) Figure 2 shows an example for the objects defined above. The middle part of Fig. 2 shows the list notation for a nested relation structure. The lower part of Fig. 2 shows the nested structure as a tree.

Some functions are also defined for convenience. Uppercase characters are used to show variables, while lower-case characters show keywords.

(owner M)=O, if M is a member of O (if there is an expression of the form (O . . . M . . .)); or (owner M) =null, if M has no owner.

(val TUPLE ASSET)=VALINSSET

where ASSET is a sublist of a schema, and VALINSSET is a sublist of TUPLE corresponding to ASSET. The schema should have an instance of which TUPLE is a member.

(ancestor D)=(A1 A2 . . . Ai . . .), if there is a sequence of the form Ai=(owner . . . (owner (owner D)) . . .); or

(ancestor D)=null otherwise.

Here, Ai is called an "ancestor" of D, and D is called a "descendant" of Ai(i=1, 2, . . .).

(compid C)=null, if (owner C)=null; or

(compid C)=(member-sequence-number), if (owner (owner C))=null; or

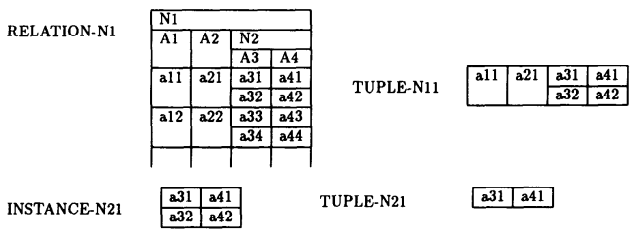
(compid C)=(Ck Ck-1 . . . C2 C1), if (compid (owner C))=(Ck-1 Ck-2 . . . C2 C1), and Ck is the member-sequence-number of C within its owner.

Compid means the component-id of an object, which uniquely identifies the component within a relation.

(schema INSTANCE)=SCHEMA

where INSTANCE is an instance of SCHEMA. This means that we can uniquely identify the schema of an instance.

For example, in Fig. 2,




```
RELATION-N1=(SCHEMA-N1 INSTANCE-N1)
  SCHEMA-N1=(ATTRIBUTE-A1 ATTRIBUTE-A2 SCHEMA-N2)
    SCHEMA-N2=(ATTRIBUTE-A3 ATTRIBUTE-A4)
  INSTANCE-N1=(TUPLE-N11 TUPLE-N12)
    TUPLE-N11=(VALUE-A11 VALUE-A21 INSTANCE-N21)
      INSTANCE-N21=(TUPLE-N21 TUPLE-N22)
        TUPLE-N21=(VALUE-A31 VALUE-A41)
        TUPLE-N22=(VALUE-A32 VALUE-A42)
    TUPLE-N12=(VALUE-A12 VALUE-A22 INSTANCE-N22)
      INSTANCE-N22=(TUPLE-N23 TUPLE-N24)
        TUPLE-N23=(VALUE-A33 VALUE-A43)
        TUPLE-N24=(VALUE-A34 VALUE-A44)
```

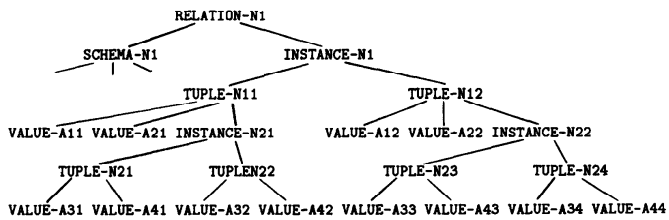


Fig. 2 An NF2 Relation.

(owner TUPLE-N22)=INSTANCE-N21,
(val TUPLE-N12 (ATTRIBUTE-A1 SCHEMA-N2))
=(VALUE-A12 INSTANCE-N22),
(ancestor TUPLE-N23)=(INSTANCE-N22 TUPLE-N12 INSTANCE-N1 RELATION-N1),
(compid TUPLE-N23)=(1 3 2 2), (see the tree in Fig. 2),
(schema INSTANCE-N22)=SCHEMA-N2.

Here, relationships are shown implicitly by suffixes (for example, if RELATION-R, SCHEMA-R and INSTANCE-R are used, there is an implicit relationship that RELATION-R=(SCHEMA-R INSTANCE-R).

Since a tuple is an extended row for an instance, a primary key is needed. A primary key for a schema is defined as two functions:

(keyattributes SCHEMA-R)=(primary-key-attributes-of-SCHEMA-R)
(keyvalues TUPLE)=(values-of-keyattributes)

Keyvalues must be unique within an instance of which TUPLE is a member. Note that keyvalues consist of simple values only. For example, in Fig. 2, it may be that

(keyvalues TUPLE-N11)=(VALUE-A21)
and (keyvalues TUPLE-N22)=(VALUE-A32 VALUE-A42).

(Assume that A1 functionally depends on A2.)

Note that the schema and instance information (the list objects in the middle of Fig. 2) are specified as a model, not as an implementation structure. However, we may store the list objects directly in the database. For example, each object (list) can be a variable-length record that is stored or retrieved by hashing its identifier.

3.2 NEST and UNNEST Operations

NEST and UNNEST operators are widely accepted for transformation between 1NF relations and NF2 relations (or between NF2 relations) [5–7, 14]. They are defined here as functions. First, UNNEST is defined as follows:

(unnest RELATION-N SCHEMA-Na)
=RELATION-U

Here, SCHEMA-Na is the schema to be unnested, and SCHEMA-N (corresponding to RELATION-N) is an ancestor of SCHEMA-Na. SCHEMA-Na is replaced by its members in SCHEMA-U. INSTANCE-U has tuples that are defined by the function "unnesttup" described below.

Assume that SCHEMA-Ni, SCHEMA-Ui (i=1, 2, . . .) are schemas in SCHEMA-N and SCHEMA-U, respectively (including SCHEMA-N and SCHEMA-U), and that SCHEMA-Nb is the owner of SCHEMA-Na. Assume also that TNij and TUik are the tuples for SCHEMA-Ni and SCHEMA-Ui (j, k=1, 2, . . .), where (compid TNaj)=(Ch Ch-1 . . . C2 C1) (C1, C2, . . . , Ch-4 may be null if SCHEMA-Nb=SCHEMA-N). ASSET denotes a sublist of SCHEMA-Ub, corresponding to SCHEMA-Na. ASSETX denotes an attributeset, or an attributeset and a schemaset, that satisfies (append ASSETX ASSET)=SCHEMA-Ub

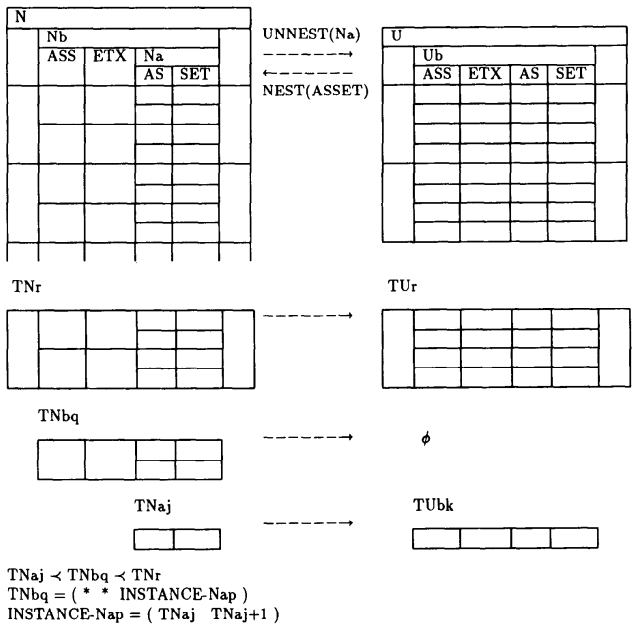


Fig. 3 UNNEST/NEST Operation.

(see Fig. 3). Then,
(unnesttup TNij)=TUij, if $i \neq a$ and $i \neq b$, where (compid TUij)=(compid TNij) and TNij and TUij have the same component set:
(unnesttup TNaj)=TUbk, where (compid TUbk) =(Cm Ch-3 . . . C2 C1), Cm is the member-sequence-number of TUbk within its owner, (val TUbk ASSETX)=(val (owner (owner TNaj)) ASSETX), and (val TUbk ASSET)=(val TNaj ASSET);
(unnesttup TNbj)=null.

Notice that unnesttup is a one-to-one and onto mapping from INSTANCE-Ni to INSTANCE-Ui ($i=1, 2, \ldots; i \neq a, i \neq b$), and from INSTANCE-Na to INSTANCE-Ub. Additionally, the keyattributes for SCHEMA-Ub are the concatenation of the keyattributes for SCHEMA-Nb and SCHEMA-Na.

Next, NEST is defined by the function
(nest RELATION-U ASSET NAME)=RELATION-N.
Here, ASSET is a sublist of a SCHEMA-Ub, which should include at least one attribute. The components of ASSET are replaced by a schema NAME in SCHEMA-Nb. TNij is constructed by the reverse mapping of unnesttup described above.

4. Extended Tuple Relational Calculus

This section describes simple extended tuple relational calculus (ETRC), which will be the basis for the following discussions. It is motivated by the fact that the nested queries proposed so far are not very easy to understand, nor can they be easily implemented. The ETRC proposed here does not have any nested syntax, which makes querying simple.

4.1 Definitions

The ETRC is defined as the following select function:

(select RELATION-N VARDEF VARDSC COOND) =RELATION-S;
VARDEF=((QUANTIFIER TNij SCHEMA-Ni) . . .), $i=1, 2, \ldots, n$; $j=1, 2, \ldots, mi$;
VARDSC=((dsc TNij TNkl) . . .), $1 <= i$, $k <= n$; $1 <= j <= mi$; $1 <= l <= mk$.

RELATION-N (source) and RELATION-S (target) have the same schema. VARDEF contains expressions for tuple variable definitions, where QUANTIFIER is here restricted to either the existential quantifier ∃ or null. TNij is a tuple variable for the SCHEMA-Ni. SCHEMA-Ni is a schema contained in SCHEMA-N or SCHEMA-N itself. VARDSC contains expressions for "descendant-ancestor relationships" between tuple variables, which is the extension to TRC for 1NF relations. The expression above means that tuple TNij is a descendant of tuple TNkl (see Fig. 3). COND contains characters for conditional expressions. It includes logical and comparative operators, constants, and expressions of the form "(val TNij ATTRIBUTESET)". Tuple variables TNij are used only in this form in COND expressions. That is, the condition expressions are the same as the ones for normal TRC. This makes the extension as small as possible, and makes the syntax simple.

Notice that the targets to be selected by the condition are tuples of the highest-level instance (of RELATION-N), so that the TUPLE-N structure does not change. This restriction makes the meaning of a query easy to understand (in some studies, the targets may be tuples of multiple different instances, which makes the query syntax complicated). Notice also that the select function defined above does not have any nested syntax, which makes the expression easier to understand.

4.2 Characteristics

When NF2 queries are implemented on 1NF relations, it is necessary to interpret the query and transform it into a 1NF query. At first, two NF2 relations, RELATION-N and RELATION-U, are considered where the latter is obtained by unnesting the former. That is, (unnest RELATION-N SCHEMA-Na)=RELATION-U. Additionally, assume that (owner SCHEMA-Na)=SCHEMA-Nb. The first goal is to transform a select for RELATION-N into a select for RELATION-U. The procedure is as follows:

Transform-Query
Source: (SELEXPR-N SCHEMA-Na)
SELEXPR-N=(select RELATION-N VARDEF-N VARDSC-N COND-N), where SCHEMA-Na is the schema to be unnested, and notations, defined in 4.1, are used.
Target: SELEXPR-U
SELEXPR-U=(select RELATION-U VARDEF-U VARDSC-U COND-U), which is in a sense equivalent to Source.
Procedure:
1. Replace schema names SCHEMA-Ni with SCHEMA-Ui ($i=1, 2, \ldots, n$), except $i=a$.
2. Replace schema name SCHEMA-Na with SCHEMA-Ub.
3. Replace tuple variables TNij with TUij ($i=1, 2, \ldots, n$; $j=1, 2, \ldots, mi$), except $i=a$.
4. Replace tuple variable TNaj with TUb,$mb+j$ ($j=1, 2, \ldots, ma$).
5. Delete the expression (dsc TUb,$mb+j$ TUbl) in VARDSC-U (resulting from the procedures 3. and 4. above), and add a new expression (val TUb,$mb+j$ KEYATR-B)=(val TUbl KEYATR-B) to COND-U ($1 <= j <= ma$, $1 <= l <= mb$) with the logical operator "and." KEYATR-B is the keyattributes for SCHEMA-Nb.

As a result, SELEXPR-U becomes like the one specified below.
VARDEF-U=((QUANTIFIER TUij SCHEMA-Ui) . . .) $i=1, 2, \ldots, n$; $i \neq a$; $j=1, 2, \ldots, mi$ for $i \neq b$; $j=1, 2, \ldots, mb+ma$ for $i=b$;
VARDSC-U=((dsc TUij TUkl) . . .) $1 <= i$, $k <= n$; i,

$k \neq a$; $1 <= j <= mi$ for $i \neq b$; $1 <= 1 <= mk$ for $k \neq b$; $1 <= j$, $1 <= mb + na$ for i, $k = b$;
COND-U=((conditions for TUij) and ((val TUb, $mb+j$ KEYATR-B)=(val TUbl KEYATR-B)) . . .) $1 <= j <= ma$; $1 <= 1 <= mb$;
The above procedure has the following properties:

Proposition 1.

Assume that a tuple TNf is in INSTANCE-N, and that TUf=(unnesttup TNf) is in INSTANCE-U (if SCHEMA-Nb≠SCHEMA-N), or {TUe}(e=1, 2, . . .) are tuple sets in INSTANCE-U where TUe=(unnesttup TNae) and (owner (owner TNae))=TNf (if SCHEMA-Nb=SCHEMA-N). If and only if TNf satisfies SELEXPR-N, then TUf satisfies SELEXPR-U (when SCHEMA-Nb≠SCHEMA-N), or ∀TUe in {TUe} satisfies SELEXPR-U (when SCHEMA-Nb=SCHEMA-N).

Proof.

The notations given in Sections 3.2, 4.1, and 4.2 are used. Additionally, suppose (select RELATION-N VARDEF-N VARDSC-N COND-N)=RELATION-S, (unnest RELATION-S SCHEMA-Sa)=RELATION-SU, (unnest RELATION-N SCHEMA-Na)=RELATION-U, (select RELATION-U VARDEF-U VARDSC-U COND-U)=RELATION-US. Moreover, suppose that the schemas included in RELATION-X are named SCHEMA-Xi (i=1, 2, . . . , n) and that they contain SCHEMA-X.

Assume that TNf satisfies SELEXPR-N; then in INSTANCE-Ni, there are typles TNij (i=1, 2, . . . , n; j=1, 2, . . . , mi), which are instantiations for the tuple variables in SELEXPR-N. Note that INSTANCE-Ni may be INSTANCE-N, and that {TNij} includes TNf. Then ∃{TUij} is in the INSTANCE-Ui (including INSTANCE-U) (i=1, 2, . . . , n; $i \neq a$; j=1, 2, . . . , mi if $i \neq b$; j=1, 2, . . . , mb, $mb+1$, . . . , $mb+ma$ if $i=b$) such that

TUij=(unnesttup TNij) if $i \neq b$;
TUbj=(unnesttup TNad) if $1 <= j <= mb$, where (owner (owner TNad))=TNbj; or
TUbj=(unnesttup TNa, $j-mb$) if $mb < j <= mb+ma$.

This mapping completely matches the tuple variable replacement of "Transform-Query," and {TUij} can be the instantiations for the tuple variables in SELEXPR-U, because

1. VARDEF-U holds, since there are the same number of corresponding tuples.
2. Assume that (dsc TNij TNkl) in VARDSC-N holds within INSTANCE-N. This is transformed into

(dsc TUij TUkl) if $i \neq a$ and $k \neq a$; or
(dsc TUb, $mb+j$ TUkl) if $i=a$ and $k \neq b$; or
(dsc TUij TUb, $mb+1$) if $k=a$.

(The case in which $i=a$ and $k=b$ will be handled in Item 3.)
These expressions in VARDSC-U hold within INSTANCE-U, since the descendant-ancestor relationship between tuples remains unchanged unless $i=a$ and $k=b$. Therefore, VARDSC-U holds for {TUij}.
3. The conditions for TUij in COND-U hold for {TUij}, since a tuple's attribute value for TNij is the same as the one for TUkl if the former maps to the latter. In addition, (val TUb, $mb+j$ KEYATR-B)=(val TUbl KEYATR-B) holds if (dsc TNaj TNbl) holds in COND-N. Thus, COND-U holds for {TUij}.

From Items 1 to 3 above, {TUij} satisfies SELEXPR-U. Since TNf is in {TNij}, TUf is in {TUij}, or any TUe (in {TUe}) is in {TUij}. Therefore, TUf or ∀TUe in {TUe} satisfies SELEXPR-U. The reverse condition can be proved in almost the same way. (Note that the proof depends on the fact that (dsc TNaj TNbl) is equivalent to (val TUb, $mb+j$ KEYATR-B)=(val TUbl KEYATR-B). Therefore, the concept of keyattributes is essential.) Q.E.D.

Proposition 2.

Given any SELEXPR-N and SELEXPR-U, which is transformed from SELEXPR-N by "Transform-Query," the equation UNNEST (SELECTN(N))=SELECTU (UNNEST(N)), or more precisely, (unnest (select RELATION-N VARDEF-N VARDSC-N COND-N) SCHEMA-Sa))=(select (unnest RELATION-N SCHEMA-Na) VARDEF-U VARDSC-U COND-U) holds. Note that SCHEMA-Sa is equivalent to SCHEMA-Na.

Proof.

The same notation as in Proposition 1 is assumed. First, SCHEMA-SU is equal to SCHEMA-US. This is trivial, since "select" does not affect schemas.

Next, it is shown that INSTANCE-SU is equal to INSTANCE-US. The relation instance equality is shown, if there is a one-to-one and onto mapping between each object on each level. For the present purpose, one-to-one and onto tuple mapping (for each instance) is sufficient.

Assume that ∀TSUij is in INSTANCE-SUi (note that INSTANCE-SUi may be INSTANCE-SU), and that ∃TSUk, which is an ancestor of TSUij (or itself), is in INSTANCE-SU, where $i \neq a$, since SCHEMA-Sa is unnested. Then ∃TSgh is in the INSTANCE-Sg such that

(unnesttup TSgh)=TSUij (*) ($g=i$ if $i \neq b$; $g=a$ if $i=b$),

and ∃TSf, which is an ancestor of TSgh (or itself), is in INSTANCE-S. TSf is in INSTANCE-N, and TSgh is in INSTANCE-Ng, their relationship being unchanged, since "select" extracts tuples in the highest level with all their descendants. Here, TSf satisfies SELEXPR-N. ∃TUde is in the INSTANCE-Ud such that

(unnesttup TSgh)=TUde (**)
since $g \neq b$ ($d=g$ if $g \neq a$; $d=b$ if $g=a$).

From (*) and (**),

TSUij=TUde (***)

since unnesttup is a one-to-one and onto mapping from INSTANCE-Sg (=INSTANCE-Ng). In addition, ∃TUc is an ancestor of TUde (or itself) in INSTANCE-U. Here, either TUc=(unnesttup TSf), or TUc is in {TUp} ($p=1, 2, \ldots$), where TUp=(unnesttup TNap) and (owner (owner TNap))=TSf (if SCHEMA-Nb=SCHEMA-N). Then, from Proposition 1, TUc satisfies SELEXPR-U. Therefore, TUde is in USd, since TUc is in US. Note that $i=d$, since $i=g=d$ if $i\neq b$ (since $i\neq a$), or $i=b=d$ otherwise. Therefore, from (***), TSUij is in USi, which shows that INSTANCE-SUi is a subset of INSTANCE-USi ($i=1, 2, \ldots$). The fact that INSTANCE-USi is a subset of INSTANCE-SUi is proved in almost the same way, which results in INSTANCE-SU being equal to INSTANCE-US. Q.E.D.

The discussions for "Transform-Query" and "Proposition 2" can be extended to allow consideration of the "UNNEST*" (unnest all) [5] operation. The next procedure transforms a select for an NF2 relation into a select for a 1NF relation:

Transform-Query*

Source: SELEXPR-N

SELEXPR-N=(select RELATION-N VARDEF-N VARDSC-N COND-N) where source relation RELATION-N is in NF2.

Target: SELEXPR-U*

SELEXPR-U*=(select RELATION-U* VARDEF-U* COND-U*), which is a (normal) TRC, where RELATION-U* is in 1NF.

Procedure:

1. Replace schema names SCHEMA-Ni with SCHEMA-U* ($i=1, 2, \ldots, n$).
2. Replace tuple variables TNij with TUp ($i=1, 2, \ldots, n$; $j=1, 2, \ldots, mi$), where $p=m1+m2+\ldots+mi-1+j$.
3. Delete the expression (dsc TUp TUq) in VARDSC-N (resulting from (dsc TNij TNkl) by step 2), and add a new expression (val TUp KEYATR-K)=(val TUq KEYATR-K) to COND-U* with the logical operator "and". KEYATR-K is the set of keyattributes of SCHEMA-Nk.

As a result, SELEXPR-U* becomes like the one specified below.

SELEXPR-U*=(select RELATION-U* VARDEF-U* COND-U*),
VARDEF-U*=((QUANTIFIER TUp SCHEMA-U*) . . .) $p=1, 2, \ldots, u$; $u=m1+m2+\ldots+mn$;
COND-U*=((conditions for TUp) and ((val TUq KEYATR-K)=(val TUr KEYATR-K)) . . .).

Next, a definition is given for the function unnest*, which transforms an NF2 relation into a 1NF relation:

(unnest* RELATION-N)=RELATION-U*.

This is defined by an unnest sequence:

(unnest* RELATION-N)::=
(unnest . . . (unnest (unnest RELATION-N SCHEMA-N1) SCHEMA-N2) . . . SCHEMA-Nn) where SCHEMA-Ni ($i=1, 2, \ldots, n$) are schemas in RELATION-N other than SCHEMA-N.

The procedure "Transform-Query*" has the following property:

Proposition 3.

Given any SELEXPR-N and SELEXPR-U*, which is transformed from SELEXPR-N by "Transform-Query*," the equation UNNEST* (SELECTN(N)) =SELECTU* (UNNEST*(N)), or more precisely, (unnest* (select RELATION-N VARDEF-N VARDSC-N COND-N))=(select (unnest* RELATION-N) VARDEF-U* COND-U*) holds.

Proof.

Proposition 3 is proved by successive applications of Proposition 2. The details are omitted here.

Proposition 3 ensures that, instead of executing an NF2 query, we can execute an equivalent 1NF query for a totally unnested (1NF) relation. In an actual system, the query processing proceeds as follows:

1. A user issues a nested query for a nested external/conceptual relation.
2. The system transforms the query into an unnested query, applies it to a 1NF relation, and gets the result as a 1NF relation.
3. The system transforms the resulting relation into a nested relation (conforming to the external/conceptual schema) by successive nest operations.
4. The user gets the result as a nested relation.

The procedure above shows that "Transform-Query*" provides a safe basis for implementing an NF2 relation over 1NF relations.

The next problem is to determine whether there is a "reverse-transform-query" procedure that satisfies the equation NEST (SELECTU(U))=SELECTN (NEST(U)). The answer here is negative.

Proposition 4.

For a general TRC query (SELECTU), there is no corresponding ETRC query (SELECTN) that satisfies the equation
(nest (select RELATION-U VARDEF-U VARDSC-U COND-U) ASSET NAME)=(select (nest RELATION-U ASSET NAME) VARDEF-N VARDSC-N COND-N).

Counter-Example.

Assume that RELATION-U is the unnested Employee in Fig. 1. Let SELEXPR-U include only expressions such as "TUi; TUi(Cage)=2" (only the second tuple is extracted). It cannot be transformed into a corresponding ETRC expression, since it is only possible to extract the highest-level tuple from the nested Employee (similar discussions are found in Fischer and Thomas [5]).

Proposition 4 depends on the fact that, though it is possible to transform a tuple-group condition for an unnested relation into one for a nested relation, it is not possible to transform every tuple condition. A tuple-

group denotes tuples in an unnested relation that will make a higher-level tuple in a nested relation.

5. Conclusion

This paper gives the following results for NF2 relation processing:

1. A simple extended tuple relational calculus is proposed, which is easy to understand because of its flat (rather than nested) syntax.
2. An algorithm is proposed that transforms a query for a nested relation into one for an unnested relation.
3. The equivalence of queries transformed by the proposed algorithm is proved. The equivalence means that UNNEST (SELECTN(N))=SELECTU (UNNEST(N)) is satisfied.
4. A LISP-like notation is proposed for NF2 relation modeling, which can be a basis for fully computable formal NF2 relation processing.

The following considerations show the significance and effectiveness of the above results:

1. Using extended tuple relational calculus, we can easily construct an NF2 query interface, which is more friendly to database users than nested algebra because of its flat syntax.
2. The query transformation algorithm with its equivalence proof ensures the safe implementation of NF2 relations by 1NF relations. That is, we can easily implement NF2 query processing by using current 1NF relational databases. Current database users get the benefit of the NF2 external view without their stored data being affected.
3. The nested list notation that has been proposed handles nested schemas, nested instances, and nested tuples uniformly. It captures the structural aspect of NF2 relations more accurately, and is more convenient for formal NF2 processing, than current NF2 notations.

I have implemented an intelligent office form generation system called "FORMATION" [18], in which the logical representation of an office form is based on NF2 relations.

Acknowledgment

I would like to thank Mr. Kazuo Watabe for valuable discussions on form management systems and form-processing languages.

References

1. ABITEBOUL, S. and BIDOIT, N. Non First Normal Form Relations to Represent Hierarchically Organized Data, *Proc. PODS* (1984), 191–200.
2. ARISAWA, H., MORIYA, K. and MIURA, T. Operations and the Properties on Non-First-Normal-Form Relational Databases, *Proc. VLDB* (1983), 197–204.
3. COLBY, L. S. A Recursive Algebra and Query Optimization for Nested Relations, *Proc. SIGMOD* (1989), 273–283.
4. DADAM, P. et al. A DBMS Prototype to Support Extended NF2 Relations: An Integrated View on Flat Tables and Hierarchies, *Proc. SIGMOD* (1986), 356–367.
5. FISCHER, P. C. and THOMAS, S. J. Operators for Non-First-Normal-Form Relations, *Proc. COMPSAC* (1983), 464–475.
6. JAESCHKE, G. and SCHEK, H.-J. Remarks on the Algebra of Non-First-Normal-Form Relations, *Proc. PODS* (1982), 124–138.
7. KITAGAWA, H. and KUNII, T. L. Form Transformer—Formal Aspects of Table Nests Manipulation, *Proc. Hawaii Intl. Conf. System Sciences* (1982), 132–141.
8. KIYAMA, M. and NAKANO, R. Reduction of NF2 Operation Based on NF2 View to Relational Calculus (in Japanese), *IPSJ SIG-DBS Reports*, **58**-2 (1987).
9. LUM, V. Y. et al. Design of an Integrated DBMS to Support Advanced Applications, *Proc. Foundations of Data Organization* (1985), 21–31.
10. MAKINOUCHI, A. A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model, *Proc. VLDB* (1977), 447–453.
11. PISTOR, P. and ANDERSEN, F. Designing a Generalized NF2 Model with an SQL-Type Language Interface, *Proc. VLDB* (1986), 278–285.
12. ROTH, M. et al. SQL/NF—A Query Language for Non-1NF Relational Databases, *Information Systems*, **12**, 1 (1987), 99–114.
13. SCHEK, H.-J. Towards a Basic Relational NF2 Algebra Processor, *Proc. Foundations of Data Organization* (1985), 173–182.
14. SCHEK, H.-J. and PISTOR, P. Data Structures for an Integrated Data Base Management and Information Retrieval System, *Proc. VLDB* (1982), 197–207.
15. SCHEK, H.-J. and SCHOLL, M. The Relational Model with Relation-Valued Attributes, *Information Systems*, **11**, 2 (1986), 137–147.
16. TSICHRITZIS, D. Form Management, *CACM*, **25**, 7 (July 1982), 453–478.
17. TSURUOKA, K. et al. PALET: A Flexible Office Form Management System, *Journal of Information Processing*, **8**, 4 (Mar. 1985), 280–287.
18. WATABE, K. and TSURUOKA, K. A Form Generation Method through Form Title Analysis, *Office Knowledge: Representation, Management, and Utilization*, North-Holland (1988), 143–161.
19. YAO, S. B. et al. FORMMANAGER: An Office Forms Management System, *ACM TOOIS*, **2**, 3 (July 1984), 235–262.

(Received August 12, 1988; revised July 11, 1991)