A DBMS Prototype to Support Extended NF² Relations: An Integrated View on Flat Tables and Hierarchies

P Dadam, K Kuespert, F Andersen, H Blanken, R Erbe,
J Guenauer, V Lum*, P Pistor, G Walch

IBM Heidelberg Scientific Center
Tiergartenstr 15
D-6900 Heidelberg, West Germany

Abstract

Recently, extensions for relational database management systems (DBMS) have been proposed to support also hierarchical structures (complex objects) These extensions have been mainly implemented on top of an existing DBMS Such an approach leads to many disadvantages not only from the conceptual point of view but also from performance aspects This paper reports on a 3-year effort to design and prototype a DBMS to support a generalized relational data model, called extended NF² (*N*on *F*irst *N*ormal *F*orm) data model which treats flat relations, lists, and hierarchical structures in a uniform way The logical data model, a language for this model, and alternatives for storage structures to implement generalized relations are presented and discussed

1 Introduction

Due to growing demands in improved product quality, increased productivity, and faster reaction on market demands, the number of installed systems for computer aided design (*CAD*) and computer aided manufacturing (*CAM*) has grown very fast during the last years To manage their data, these systems generally use only the file system provided by the underlying operating system However, as both the number of installed systems in a company and the number of people in CAD/CAM projects increase and the demand for interaction among systems and among people grows, these systems begin to run into problems like lack of data independence, insufficient recovery, and missing or inadequate concurrency control, problems with which one was faced more than two decades ago in business or commercial data management and related areas Searching for solutions to these prob lems has led to the development of database management systems (DBMS), which today are installed in practically all commercial and business data processing environments Currently there is a strong interest in moving towards computer integrated manufacturing (*CIM*), meaning the integration of CAD, CAM, and business administration CIM requires, however, that all kinds of data - at least all commonly relevant kinds of data - are accessible in an *integrated* fashion within one (not necessarily physically one) system It is rather obvious that this problem can only be solved by applying advanced database technology

For the business administration area, such advanced database technology is - at least partially - already available today, namely *relational database systems* Systems like SQL/DS /IBM1/, DB2 /IBM2/, and INGRES /St76/ provide a high degree of data independence for application programs, can combine the stored data in a very flexible way, and offer query languages which are relatively easy to learn and to use However, the facilities the relational data model offers to describe the logical structure of data and dependencies between data are rather poor, too poor for CAD/CAM/CIM environments Especially CAD objects very often require deeply nested hierarchical structures and a lot of different tuple types (relations) to represent all the necessary information (see /LP83, Ka85/, for example) For semantical (conceptual) as well as for performance reasons (data clustering, avoidance of unnecessary joins) such *complex objects* /HL82/ cannot simply be "flattened" and stored just as ordinary relations That is, a DBMS should provide a facility to directly support *hierarchical structures*

In /HL82, LP83/ it has been reported how one can *extend a relational database system* to support hierarchical structures more efficiently This has been done by adding some new predefined attribute types and system generated keys (to express hierarchical relationships and to speed up joins), and by providing a special interface with appropriate new operators (to insert, retrieve, update, and delete complex objects) The advantage of this approach is that only few parts of the existing DBMS have to be changed in order to support complex objects The disadvantage is, however, that using this approach, complex objects are a "special animal" for the underlying DBMS which cannot be treated in the same way as the usual (flat) objects That is, projections, selections (partial retrieval), and clustering of complex objects are not supported in the same way as for flat objects To avoid this disadvantage and to enable a really *integrated view* on flat and complex objects, a *common data model* which covers (unifies) both types of data is mandatory

An elegant way to integrate flat relations and hierarchical structures within one data model without giving up the elegancy and expressive power of high level relational query languages is to *generalize the relational data model* The key idea is to allow relations to occur as attribute values of tuples in relations As this means to give up the first normal form requirement, we have called relations of that kind *N*on *F*irst *N*ormal *F*orm (NF²) relations /JS82, PHH83, SP82, Sch85/ (sometimes they are also called "relations with relation valued attributes" /Jae85a, Jae85b, SS86/)

(*) Current address Naval Postgraduate School, Dept of Computer Science, Monterey, CA 93943, USA

The main issue of this paper is to explain the concept of NF^2 relations from a user's point of view as well as under implementation aspects We will give an outline of a *generalized relational query language* designed to support NF^2 relations, and we will report on the underlying *DBMS prototype* which has been implemented within the *A*dvanced *I*nformation *M*anagement II (AIM-II) project at the IBM Heidelberg Scientific Center As an extension of the "pure" NF^2 data model, the AIM-II prototype also supports ordered tables (*lists*) which are considered very useful for instance in CAD/CAM applications /Lo82/ We therefore call our data model *extended* NF^2 data model

With the completion of the first running version of our prototype after 3 years of work, the AIM-II project has come to a stage where a retrospection on design and implementation of the system seems to be appropriate Since this paper, however, cannot discuss in detail all aspects of the DBMS prototype, we have included quite a number of references to papers which describe certain topics in the required depth

The remainder of this paper is organized as follows In *Section 2* extended NF^2 relations are explained from a user's point of view A powerful query language for the extended NF^2 data model is described in *Section 3* In *Section 4* aspects of complex object implementation (storage structures, addressing concepts, etc) are discussed *Section 5* provides some final remarks on the main issues of this paper and on future directions in the AIM-II project

2 User's View of NF^2 Tables

Let us first make some remarks on our *terminology* Throughout this paper, the term "table" will be used as a generalization of "relation" (→ unordered table) and "list" (→ ordered table) - In the following, we will only consider complex objects with *hierarchical* structure Therefore, we need not distinguish between "complex", "hierarchical", and "NF^2" tables (Note Our tuple name concept (see Section 4 3) also allows network structures in complex objects) - Finally, the terms "flat" table and "1NF" table (i e table in first normal form) will be used as synonyms

Although the (extended) NF^2 data model has been introduced in Section 1 primarily as a tool to support advanced applications (sometimes called *non-standard applications* /HR83/) in CAD/CAM/CIM environments, this data model can also be extremely useful in the area of *office automation* Assuming that the reader is probably more familiar with office automation than with CAD/CAM/CIM, we decided to take all our examples from this area

Assume now that one wants to model a hierarchical structure, say DEPARTMENTS, with department number (DNO), manager number (MGRNO), projects (PROJECTS), budget (BUDGET), and equipment (EQUIP) at the top level, and with EQUIP telling quantity (QU) and type (TYPE) for every item PROJECTS in turn represents all projects in a department, with project number (PNO), project name (PNAME), and project members (MEMBERS) For every member the employee number (EMPNO) and the function (FUNCTION) shall be shown In an IMS database this could be modelled by defining the segment types and parent child relations as shown in Fig 1 To retrieve an object of this type "navigational" language constructs like "get next" (GN) and "get next within parent" (GNP) etc /Da81/ have usually to be used which are completely different from the high level language constructs used in relational database systems To represent the information of Fig 1 in first normal form (1NF) tables would require at least 4 tables (as an example see Tables 1 to 4)

As already mentioned in the introduction, the NF^2 data model is a generalized relational data model which allows relations to have relations as attribute values Using the NF^2 approach one can model the DEPARTMENTS hierarchy as shown in Table 5 (A very similar graphical representation is also used in /Sh84, SLTC82/) Table 5 reads as follows DEPARTMENTS is an *unordered table* (i e a relation) having 5 "top-level" attributes, namely DNO, MGRNO, PROJECTS, BUDGET, and EQUIP (In our figures unordered tables (relations) are marked by putting their names into curly brackets ({ }) while ordered tables (lists) are marked by < >) The attributes DNO, MGRNO, and BUDGET are atomic while the two other attributes are again relations (i e non-atomic) PROJECTS is an NF^2 relation having 3 top-level attributes PNO, PNAME, and MEMBERS While PNO and PNAME are atomic, MEMBERS is not, etc Note, that the attribute values of BUDGET are bound to a department - not to a project The same holds for EQUIP EQUIP is a flat (1NF) relation describing quantity (QU) and type (TYPE) of equipment used in a specific department PROJECTS, MEMBERS, and EQUIP are all unordered tables (i e relations) There is neither an implicit nor an explicit correspondence between a project or one of its members and some kind of equipment in this table We assume that employee numbers in the DEPARTMENTS table are always unique, whereas project numbers need not be unique (incidentally, in Table 5 they are)

Another example of an NF^2 table, which shows an "inner" table AUTHORS, is Table 6 (REPORTS) AUTHORS is an *ordered table* (i e a list) having just one attribute (NAME) The attribute DESCRIPTORS is also an "inner" table but unordered As a matter of fact, every table can be defined to be either ordered or unordered, depending on the users' needs It is easy to see that "normal" tables, i e tables in first normal form like Tables 1 to 4, are just special cases of NF^2 tables

3. Query Language for NF^2 Tables

In designing an appropriate query language for NF^2 tables we have taken the same approach as with the relations themselves We have used the concept of an existing relational query language (SEQUEL/SQL /Ch76, IBM1/ with its SELECT-FROM-WHERE constructs) and have generalized it analogously to our generalized relations In the case of NF^2 tables where attribute values may again be tables (relations or lists) one needs a mechanism (in the SELECT clause) to describe the *result* structure of a query Another mechanism (in the FROM clause) is needed to describe from which attribute(s) on which nesting level(s) the *source* data shall come from

Due to lack of space we can give here no detailed discussion of our NF^2 query language Instead, we try to illustrate how it works and how it looks like, using some selected examples (a more detailed description of the NF^2 language can be found in /PT85/ and /PA86/)

Example 1

This example shows how to retrieve all tuples of an NF^2 table and to *implicitly* overtake the result structure from the structure of the source table Assuming Table 5 being a stored table, we can simply write

```
SELECT  x DNO, x MGRNO  x PROJECTS, x BUDGET, x EQUIP
FROM    x IN DEPARTMENTS
```

or by using the usual shorthand notation

```
SELECT *
FROM   DEPARTMENTS
```

Example 2

This example shows how to retrieve all tuples of an NF^2 table and to *explicitly* define the result structure Assume again that Table 5 is a stored table and that we want to pose a query which just shows the same structure also for the result table When explicitly defining this

structure our query would look like the one shown in Fig 2 A good mental model to understand the bindings of tuple variables (range variables) is to associate them with a loop which runs over all tuples of the relation they are bound to In the query shown in Fig 2, "x" is bound to DEPARTMENTS That is, "x" is one time bound to the tuple describing department number 314, one time to the tuple describing department number 218, etc For a given binding of "x", the bindings of the tuple variables "y" and "v" are known as well, namely "y" is associated with the attribute value of x PROJECTS (which is again a table), and "v" is associated with x EQUIP (which is also a table) If "x" is bound to the tuple describing department 314 then "y" will be bound at one time to the tuple of the "inner" table which describes project number 17, at another time to the tuple describing project number 23 (Remark As there are only two projects in department 314 the "inner" loop associated with "y" will terminate after having processed projects 17 and 23) For a given binding of "x" (say to department 314) and a given binding of "y" (say to project 23) the binding of the tuple variable "z" is known, too, etc

Example 3

Here we show how to create a complex table structure based on flat source tables (*"nest"* operation /Jae85a, Jae85b/) Assume Tables 1 to 4 are stored tables and that we want to get a result structure like Table 5 The resulting query is shown in Fig 3

Example 4

This example demonstrates how to create a flat result table based on a complex source table (*"unnest"* operation /Jae85a, Jae85b/) It also shows the use of a *projection*, since the attributes BUDGET and EQUIP of the source table are not used for the construction of the result table (cf Table 7)

```
SELECT x DNO  x MGRNO, y PNO, y PNAME, z EMPNO  z FUNCTION
FROM   x IN DEPARTMENTS, y IN x PROJECTS, z IN y MEMBERS
```

For comparison The same query against Tables 1 to 3 would appear as follows

```
SELECT x DNO, x MGRNO, y PNO  y PNAME, z EMPNO  z FUNCTION
FROM   x IN DEPARTMENTS 1NF  y IN PROJECTS 1NF
       z IN MEMBERS-1NF
WHERE  x DNO = y DNO  AND  y PNO = z PNO
       AND y DNO = z DNO
```

As you can see, the "flat table query" is more difficult to formulate than the query against the hierarchical table The example also shows that hierarchical tables can be used to store *pre-computed* (materialized) *joins* /SS81/ as well

Example 5

Consider the following query posed against Table 5 "List DNO, MGRNO, and BUDGET of all departments which use a PC/AT " This query - which shows the use of an *EXISTS* clause - can be expressed as follows

```
SELECT x DNO  x MGRNO, x BUDGET
FROM   x IN DEPARTMENTS
WHERE  EXISTS y  IN x EQUIP  y TYPE =  PC/AT'
```

The output would be a flat table with 3 atomic attributes

Example 6

To demonstrate the use of an *ALL* clause in a query, we look at the following problem "List DNO, MGRNO, and BUDGET of all departments in Table 5 which have *only* consultants as employees " This can be expressed as follows

```
SELECT x DNO  x MGRNO, x BUDGET
FROM   x IN DEPARTMENTS
WHERE  ALL  y  IN x PROJECTS
          ALL  z  IN y MEMBERS  FUNCTION='Consultant'
```

Since MEMBERS is a subtable of PROJECTS, and PROJECTS in turn is a subtable of DEPARTMENTS, two ALL clauses are needed, one for PROJECTS and one for MEMBERS For the contents of the DEPARTMENTS table as shown in Table 5, the result set of this query is empty, since there is no department which fulfills the condition in the WHERE clause

Example 7

Assume now that we have a flat table EMPLOYEES-1NF (cf Table 8) with employee number (EMPNO), last name (LNAME), first name (FNAME), and sex (SEX) as attributes The EMPLOYEES-1NF table shall contain one tuple for each project member and manager stored in Table 5 Using Table 5 and EMPLOYEES-1NF as input tables, the following query shall be formulated "List all employees with employee number (EMPNO), last name (LNAME), first name (FNAME), sex (SEX), and function (FUNCTION) grouped by department On department level show department number (DNO) and manager number (MGRNO) "

To answer this query one has to compute a *join* between MEMBERS (in the DEPARTMENTS table) and EMPLOYEES-1NF The resulting query is shown in Fig 4 (Note This example shows also that join attributes need not be on the same level in the hierarchy of the affected source tables)

One can also express more than one join condition within a query expression For example, one could formulate the above query such that the manager's name and sex are retrieved instead of MGRNO This is shown in Fig 5

Example 8

Consider now Table 6 Assume one wants to see the list of authors, and the titles of all reports where 'Jones' appears as the first author As the AUTHORS attribute is an ordered table (i e a *list*), the query can be expressed as follows

```
SELECT x AUTHORS. x TITLE
FROM   x IN REPORTS
WHERE  x AUTHORS[1] =  Jones'
```

Note that the resulting table is not flat because AUTHORS is a non-atomic attribute

Besides a query language, one also needs facilities to insert, update, and delete (complex) objects as well as DDL support to define the structure of new tables Because of space limitations in this paper we refer to /PT85/ and /PA86/ where these topics are explained in detail

The language presented in this section is just one way to deal with the extended NF^2 data model It is not our intention to claim that it is the best language for this data model one could think of Especially for *end users* one would like to have an easier language, maybe in the style of Query-by-Example (QBE /IBM4, Zl77/) However, the kind of language we have presented here allows to utilize the full power of the extended NF^2 data model and allows also to specify the structure of the result table totally independent from the structure of the source table(s) Other, more simple languages, would have to show this expressive power first if they would claim to be "better" under all aspects

Especially for handling large complex objects in the CAD/CAM/CIM area, a suitable *application programming interface* (API) is required We are currently implementing such an API which is similar to Lorie's approach /Lo84/ It imbeds both DDL and DML statements of the

extended NF^2 data model into a high level programming language A DDL/DML pre-compiler is under construction which translates the imbedded NF^2 statements into subroutine calls These subroutine calls finally invoke the AIM-II run-time system for execution

4. Implementation of NF^2 Tables

4 1 Storage Structures for NF^2 Objects

There are, of course, many different ways to store hierarchically structured objects Some of them, well-known since more than a decade, are used in existing database management systems

- *IMS* storage structures IMS provides four different storage structures for hierarchical database objects, known as HSAM, HISAM, HDAM, and HIDAM /Da81/
- *CODASYL/DBTG* storage structures Since any hierarchical object can be seen as a composition of (possibly many) 1 n relationships, the implementation techniques for COSETs /Sch74/ can be used for NF^2 objects as well Therefore, lists, chains, and pointer arrays together with additional options (attached/detached where appropriate) are also candidates for the implementation of objects in NF^2 tables

In *Lorie's* proposal "a complex object is implemented as a series of tuples logically linked together" /LP83, p 116/ The tuples are stored as part of normal, flat tables with additional attributes not seen by the user These attributes (entirely managed by the system) contain the pointer values used for chaining between tuples inside a complex object Child, sibling, father, and root pointers are used for that purpose The main advantage of this approach is that it can quite easily be implemented on top of an existing DBMS (in Lorie's case System R) without having to do major changes in that system (cf Section 1)

In the *AIM-II* project we had the opportunity to build a totally new DBMS from scratch Therefore, much more emphasis could be put on the *integration* of complex object management into deeper, more appropriate layers of the system This was done in particular to gain *performance* compared to an "on top" solution The following *demands* mainly guided our implementation of hierarchical objects

- Data clustering should be supported on the complex object level Since a complex object is often processed as a whole (for instance deleted, copied, or sent to a workstation) it is rather important that all its data are stored on a relatively small page set and not distributed among too many database pages or even different database segments
- Data on the one hand and structural information (such as pointer lists) on the other hand should be separated If this rule is strictly observed, "navigation" in a complex object (e g to retrieve a certain element of a list) can be done on the structural information without having to access the data at all
- Fast processing (insertion, retrieval, update, and deletion) should not only be supported for complex objects as a whole but for arbitrary parts of these objects as well It should not be necessary, for instance, to scan a complex object more or less entirely if only one piece of data in that object is needed for further processing by the user

Before going into the details of storage structure alternatives and especially explaining the implementation that we have chosen for AIM-II, some more remarks on the terminology are necessary In the extended NF^2 data model we would like to distinguish between

- (NF^2 or 1NF) *tables* (unordered tables → relations, ordered tables → lists),
- (complex or flat) *objects*,
- (NF^2 or 1NF) *subtables*,
- (complex or flat) *subobjects*

If we look at Table 5 again, these terms can be explained by the following examples

- DEPARTMENTS as a whole is an NF^2 *table* (or a relation since there is no ordering of tuples)
- Departments 314, 218, and 417 are *complex objects* (projects, members, and equipment included)
- There are two *subtables* (subrelations) in each of these complex objects PROJECTS, which is hierarchically structured (NF^2), and EQUIP, which is flat (1NF)
- The PROJECTS subtable (subrelation) in department 314 contains two *complex subobjects*, projects 17 and 23 The EQUIP subtable (subrelation) in department 314 contains three *flat subobjects*, items 3278, PC/AT, and PC
- The MEMBERS subtable (subrelation) in project 17 contains three *flat subobjects*, employees 39582, 56019, and 69011

As mentioned above, an essential demand for the implementation of complex objects in AIM-II was to separate structural information from data Therefore, we decided to implement a so-called *Mini Directory* (*MD*) for each complex object The Mini Directory is a *tree* which contains all the structural information of a complex object but not its data The MD layout corresponds exactly to the hierarchical structure of the complex object The MD is composed of *MD subtuples* (nodes in the MD tree) which are linked via pointers A subtuple is the basic storage unit, like a tuple or a record in "traditional" database systems Besides MD subtuples, we also need *data subtuples* to store the data of a complex object In fact, all "first level" atomic attribute values of a complex object/subobject and all atomic attribute values of a flat object/subobject are stored in one data subtuple (For example (cf Table 5) The data subtuple '314 56194 320,000' contains all "first level" atomic attribute values of department 314 (→ complex object), and the data subtuple '17 CGA' contains all "first level" atomic attribute values of project 17 (→ complex subobject) '39582 Leader' and '2 3278 are also data subtuples (of flat subobjects in MEMBERS and EQUIP, respectively)) Obviously, if an object or subobject is flat, it is completely stored in one data subtuple Hence, a flat (1NF) table does not have Mini Directories for its objects at all It is important to see that *data* subtuples do not contain any structural information about the complex objects they belong to

There are several alternatives (see SS1 to SS3 in Fig 6) how to implement the Mini Directories for complex objects Common to all these alternatives is that they always use a "special" MD subtuple (called *root MD subtuple*) as the root of the MD tree Besides pointers, it contains some additional information (see below) about the complex object as a whole For the other levels of the MD tree, one must come to a decision whether

1 to use MD subtuples both for *subtables* and for *complex subobjects* (→ SS1, Fig 6a),
2 to use MD subtuples only for *complex subobjects* (→ SS2, Fig 6b),
3 to use MD subtuples only for *subtables* (→ SS3, Fig 6c)

Note, that we did not mention *flat* subobjects in this list Of course, we need some structural information about flat subobjects, too (e g their length), but this information can easily be stored in the respective data subtuples

In Fig 6, MD subtuples are drawn as rectangles whereas data subtuples are ovals For our examples, we always use department 314 of Table 5

Fig 6a shows storage alternative *SS1* Besides the root MD subtuple, there is one MD subtuple per *subtable* and one MD subtuple per *complex subobject* The "D" and "C" values in the MD subtuples stand

for pointers, with "D" as a *data* pointer (MD subtuple → data subtuple) and "C" as a child pointer (MD subtuple → MD subtuple) At the root level of this MD tree, there is one "D" pointer referring to that data subtuple which contains all "first level" atomic attribute values of department 314 (DNO = 314, MGRNO = 56194, BUDGET = 320,000), and there are two "C" pointers referring to the MD subtuples of subtables PROJECTS and EQUIP, respectively "DCC" is the only entry in the root MD subtuple As a consequence, the root MD subtuple is of fixed length (regarding the number of pointers) as long as there are no structural changes in the NF^2 table (e g addition of new non-atomic attributes) The MD subtuples for PROJECTS and EQUIP at the level below the root are of variable length They contain one pointer for each project and equipment, respectively Clearly, the number of projects in a department as well as the amount of equipment can change over time At the next lower level of the MD tree, the MD subtuples are of fixed length again (under the above assumption concerning the absence of structural changes) Finally, at the lowest level of the MD tree in Fig 6a, variable length MD subtuples represent the instances of the MEMBERS subtable which is in first normal form

Obviously, the layout of storage structure SS1 is really *symmetric*, since neither subtables nor complex subobjects are preferred or discriminated regarding the allocation of MD subtuples One disadvantage in this proposal is, however, that it results in a comparatively *large Mini Directory* tree with many small nodes This is because MD subtuples for complex subobjects are usually short compared to MD subtuples for subtables In real-world applications (in the CAD/CAM area, for instance), a complex object or subobject will usually have just a few non-atomic attributes (say up to 10) whereas a subtable may consist of thousands of tuples In this case, MD subtuples of complex subobjects will contain up to 10 pointers while MD subtuples of subtables will contain up to several thousand pointers To avoid this drawback the two other storage structures shown in Fig 6, SS2 and SS3, *integrate* fixed length and variable length MD subtuples so that all (short) fixed length MD subtuples (except the root MD subtuple in the SS3 proposal, which is still of fixed length, regarding the number of pointers) disappear

Figs 6b and 6c can now be interpreted as follows

- In Fig 6b all (variable length) MD subtuples which belong to *subtables* (in Fig 6a) have been moved upward and integrated into the (formerly fixed length) MD subtuples of complex subobjects On the highest level of the MD tree, this integration has been done into the root MD subtuple As a consequence, *SS2* contains - besides the root MD subtuple - one (variable length) MD subtuple per *complex subobject*
- In Fig 6c all (fixed length) MD subtuples which belong to *complex subobjects* (in Fig 6a) have been moved upward and integrated into the MD subtuples of subtables As a consequence, *SS3* contains - besides the root MD subtuple - one (variable length) MD subtuple per *subtable*

It is quite easy to show that there are always more MD subtuples in SS3 than in SS2 Therefore, an order SS1 > SS3 > SS2 can be established concerning the number of MD subtuples required For the selection of an implementation, however, it cannot be the only goal just to minimize the number of nodes (MD subtuples) in the MD tree since other criteria like storage space, access time, etc have to be considered as well /DGW85/ In AIM-II, storage structure *SS3* has been chosen for the implementation of complex objects since it seems to be a fairly good compromise between SS1 and SS2, not only concerning the size of the MD tree In the current state of the implementation of our system, however, it cannot be said whether SS3 was really the best choice under all possible circumstances (see also /Kue86/) For all three storage structures (SS1 to SS3), the integration of ordered subtables (*lists*) can be done easily just by using the sequence of entries in the MD subtuples to represent the sorting order in a list

The reader should note now that our demands concerning

- separation of structural information and data,
- support for fast processing of arbitrary parts (subtables, subobjects, etc) of a complex object

are fulfilled by all of our proposals The first demand mentioned at the beginning of this section (*data clustering* on the complex object level), however, has not been discussed yet This will be done in the following All what is said below does not only work in the context of storage structure SS3 (chosen for AIM-II), but for the two other alternatives (SS1 and SS2) as well

Each complex object gets its own *local address space* The local address space of a complex object is represented by a *page list* stored in the root MD subtuple (not shown in Fig 6) This page list contains the page numbers of all pages where (data or MD) subtuples of the complex object are currently stored Whenever new subtuples in a complex object have to be stored (or existing subtuples have to be extended), the page list in the root MD subtuple is scanned to find a page with enough free space for the new data Only if this search operation fails (not enough free space available in the local address space), a page outside the complex object's local address space is used to store the data, then, the number of the newly selected page is added to the page list This strategy directly supports *data clustering* in a complex object since new data are usually stored in pages which already contain data of this complex object

The term "*local* address space" will become clearer when we look at the implementation of "D" and "C" pointers Since "D" and "C" pointers need only be valid inside a complex object (*local pointers*), it is quite obvious to use the page list in the root MD subtuple also for addressing purposes So-called *Mini TIDs* are used as "D" and "C" pointers in our implementation Like the well-known TIDs /As76/, Mini TIDs consist of two parts a page number and a slot number In a TID the page number is interpreted relatively to the beginning of the database segment so that arbitrary pages in that segment can be addressed However, this is not necessary for Mini TIDs in a complex object The page number in a Mini TID is always interpreted relatively to the beginning of the complex object's *local address space* Thus, for an access via Mini TID i j (i is the page number, j is the slot number) the "local" page number i must be translated into a "real" ("global") page number i' This number is taken from position i in the page list Subsequently, the respective data or MD subtuple can be accessed via slot j in page i' of the database segment

To keep the pointers in the Mini Directories stable during DB processing, existing Mini TIDs must not be changed when pages are added to or removed from a complex object's local address space When a page number is *removed* from the page list, the "gap" in the list caused by the deletion is not closed immediately When a page number is *added* to the page list, either a "gap" (created by a deletion) is used or - if there is no "gap" - the page list is extended at its end Due to this strategy, other page numbers do not change their *position* in the page list and existing Mini TIDs are not affected at all

Addressing via Mini TIDs instead of TIDs has two *advantages*

- First, Mini TIDs can be somewhat smaller than TIDs This saves storage space in the Mini Directory and thus speeds up complex object processing ("navigation" in the MD tree, etc)
- When a complex object has to be moved to another place in the database or sent to a workstation (checked-out), this can easily be done at the page level, i e without having to look at the subtuples individually No changes are required for "D" and "C"

pointers since Mini TIDs refer to positions in the page list and not in the database segment As a consequence, only the page list must be updated to reflect the complex object's new page set

4 2 Access Paths for NF² Objects

Obviously, *query optimization* and *access path selection* for NF² tables is considerably more complex than for 1NF tables /Se79, JK84, OH85/

- Large hierarchical objects have to be processed instead of small flat ones
- Non-trivial operations have to be performed to transform flat objects into hierarchical ones and vice versa, to evaluate EXISTS and ALL clauses in query statements, etc (cf Section 3)
- Joins between different levels in different NF² tables have to be handled properly (cf Section 3, too)

Altogether, the increased complexity comes from the fact that the extended NF² data model and its query language are far more powerful than "traditional" relational data models and query languages In this paper we will not discuss query optimization and access path selection for NF² tables Instead, we want to show the effects of different kinds of *address information* on query optimization

Conceptually, an index entry is an ordered pair < key, address list > where the components of the address list ($addr_1$, , $addr_n$) refer to those objects which contain the 'key' as an attribute value In System R, for instance, addresses are simply TIDs, i e the structure of an index entry is < key, TID_1, , TID_n > - In the following, we refer again to Table 5 and Fig 6c (storage structure SS3) Let us assume that an index for FUNCTION is required The question is how addresses $addr_1$ in this index should really look like

A *first approach* could be to use *TIDs of data subtuples* as addresses $addr_1$ in the index Then, addresses in the index for FUNCTION are TIDs of data subtuples in the MEMBERS subtable As an example, one index entry is < 'Consultant', TID of data subtuple '56019 Consultant', TID of data subtuple '89921 Consultant', TID of data subtuple '44512 Consultant' > For an assessment of this approach, let us look at the following query

```
SELECT x DNO
FROM   x IN DEPARTMENTS
WHERE  EXISTS y IN x PROJECTS
           EXISTS z IN y MEMBERS  z FUNCTION= Consultant
```

This query retrieves the department numbers of all departments with at least one consultant (two EXISTS clauses are needed since MEMBERS is a subtable of PROJECTS, and PROJECTS in turn is a subtable of DEPARTMENTS) For Table 5, the final result contains DNOs 314 and 218 Using the index for FUNCTION and 'Consultant' as a key value, the TIDs of the data subtuples '56019 Consultant', '89921 Consultant', and '44512 Consultant' can be retrieved With these addresses of data subtuples, however, access to the respective department numbers (314 and 218) cannot be done since - according to Section 4 1 - there is no structural information about the MD tree (root pointers, father pointers, etc) in the data subtuples Nevertheless, would it really solve our problems if we had these pointers in the data subtuples? - Usually, there is more than one consultant in a department (cf department 218 in Table 5) Via the index for FUNCTION, one TID for each consultant can be retrieved From those TIDs it cannot be seen, however, whether they refer to data subtuples ' Consultant' in the same department or in different ones Therefore, some complex objects of the DEPARTMENTS table (department 218 in Table 5) have to be (unnecessarily) accessed more than once during query execution only to find out that their DNO is already known This shows that the straightforward idea of having TIDs of data subtuples as addresses $addr_1$ in indexes is not really sufficient for query optimization

Another approach for address implementation could be to store *TIDs of root MD subtuples* (instead of TIDs of data subtuples) as addresses $addr_1$ in the indexes This, at least, is an appropriate solution for the above query

- Starting from the root MD subtuple of a complex object in the DEPARTMENTS table, the department number DNO can be retrieved easily
- It can be seen from the addresses in the index that department 218 is referenced twice Therefore, multiple access to the same complex object can be avoided

The following query, however, shows that this kind of address information is not always sufficient

```
SELECT y PNO
FROM   x IN DEPARTMENTS, y IN x PROJECTS
WHERE  EXISTS y IN x PROJECTS
           EXISTS z IN y MEMBERS  z FUNCTION='Consultant
```

This query retrieves the project numbers (not the department numbers!) of all projects with at least one consultant (PNOs 17 and 25 in Table 5) From a pointer to the root MD subtuple of department 314 (i e a TID in the index for FUNCTION), for instance, it cannot be seen whether a consultant is working in project 17 or in project 23 Therefore, *all* projects of this department have to be scanned to find the right one (project 17)

In real-world applications complex objects are usually large, and scans in those objects to retrieve certain data should be avoided whenever possible Therefore, addresses in indexes should contain enough information to locate a certain piece of data *directly*

From the above observations it can be concluded that *hierarchical addresses* are needed, since neither data subtuple addressing (first approach) nor root MD subtuple addressing (second approach) alone is really sufficient Therefore, an address $addr_1$ in an index must represent the *path* from a root MD subtuple down to a data subtuple, in our case (index for FUNCTION) from the root MD subtuple of a complex object in the DEPARTMENTS table down to a data subtuple in the MEMBERS subtable - The above statement about hierarchical addresses is still a bit "fuzzy" since it doesn't say how these addresses should really look like Again, we first want to show that a straightforward implementation of hierarchical addresses does not help

Let us assume now that we have indexes for FUNCTION *and* for PNO in the DEPARTMENTS table (Table 5) As an example we look at the following query

```
SELECT x DNO
FROM   x IN DEPARTMENTS
WHERE  EXISTS y IN x PROJECTS
           y PNO= 17 AND
           EXISTS z IN y MEMBERS  z FUNCTION= Consultant'
```

Compared to the first query mentioned above, there is now an additional restriction for the project number (PNO = 17) Fig 7a shows storage structure SS3 for department 314 (cf Fig 6c) P = P1 P2 P3 shall be a hierarchical address for PNO = 17, F = F1 F2 F3 F4 a hierarchical address for FUNCTION = 'Consultant' Each of these addresses represents a path from the root MD subtuple down to a data subtuple Although we have indexes for *both* PNO and FUNCTION, it can still not be seen from the *index information* that P and F refer to the *same* project Obviously they do since there is a consultant in project 17 of department 314 Unfortunately, the fact that P2 and F2 are equal does not help at all since these pointers refer to an MD subtuple of a *subtable* (PROJECTS) and not to an MD subtuple of

a *complex subobject* (in fact, such an MD subtuple does not exist in storage structure SS3 - see Section 4 1) With this kind of index information there are two reasonable ways for query execution

- The index for PNO is used to find all projects with project number 17 (in our example this is just one project but in general more than one will be found since project numbers need not necessarily be unique) In each of these projects the MEMBERS subtable is scanned to check whether there is a consultant or not
- The index for FUNCTION is used to find all consultants For each consultant it is checked whether he/she works in a project with project number 17 or not

In both cases, the index information can only be used to determine a *superset* of the final result set, and this superset must be scanned to filter out those tuples which are not in the final result set It would be much better here, of course, if one could determine the final result set *directly* from the *index information* without having to scan the data

Fig 7b shows once again SS3 for department 314, now with a new implementation of hierarchical addresses The first part of a hierarchical address refers to a root MD subtuple (as usual) The rest, however, refers to *data subtuples* on a path from this root MD subtuple down to a certain data subtuple /Kue86/

The above query can now be executed without having to scan an intermediate result set From P2 = F2 it can already be seen that P and F refer to the same project so that department 314 must be in the final result set

Generally speaking (cf /BM85, Kue86/), the following *rules* must be observed in the implementation of indexes for NF^2 tables

1 Indexes must contain *hierarchical addresses*
2 Address components must identify *complex subobjects* (represented by data subtuples, in our proposal), not subtables

In AIM-II, the first component of an address (e g P1 and F1 in Fig 7b) is always a TID whereas all other components are Mini TIDs

4 3 On the Implementation of Tuple Names

Sometimes system supported references across tables are very useful to either express dependencies between data items or to allow what we call "subtuple or data sharing" between different hierarchical structures In other cases it may be necessary to communicate references to database objects to application programs for later direct access To support such requirements as well, we have extended the NF^2 data model again to support - in addition to user defined (foreign) keys - also *system generated keys*, called *tuple names* Tuple names are not yet implemented in the current version of the AIM-II prototype but, as they are part of our data model and language definition, we intend to start their implementation in the very near future In the following we want to summarize some ideas how this implementation is planned

The implementation of tuple names (shortly *t-names*) will be very similar to the implementation of addresses in index entries (shortly *i-addresses*), i e hierarchical addresses will also be used for that purpose There are at least two reasons for this decision

- Handling of i-addresses and t-names can be done by the same system routines
- Query optimization techniques can be applied, especially when a large set of t-names has to be processed at a time

It will be shown below, however, that there is a (minor) difference between t-names and i-addresses

As explained in the previous section, each i-address refers to a *data subtuple*, i e it represents a path from the root MD subtuple of a complex object down to that data subtuple Therefore, a t-name for a *flat subobject* looks exactly like an i-address for an attribute value in that subobject T = T1 T2 T3 in Fig 8, for instance, is the t-name for the (flat) '56019 Consultant' tuple in the MEMBERS subtable of project 17 (cf F = F1 F2 F3 in Fig 7b) The t-name for a *complex object* as a whole is simply the address of the root MD subtuple of that object In Fig 8, U is the t-name for department 314 as a whole

Another question is how to implement t-names for *complex subobjects*, e g for project 17 in department 314 We plan to use the data subtuple which contains all "first level" atomic attribute values (cf Section 4 1) of a complex subobject for representing the complex subobject as a whole In Fig 8, this means that the '17 CGA' data subtuple represents project 17, and V = V1 V2 is the t-name for that complex subobject (A slightly modified implementation is needed for complex subobjects without atomic attribute values /Kue86/)

Till now, only the implementation of t-names for (complex) objects/subobjects has been explained We also want to have t-names, however, for *subtables* Each subtable corresponds to a MD subtuple which can be used for addressing purposes Therefore, the t-name for the PROJECTS subtable in Fig 8 is W = W1 W2, and X = X1 X2 X3 is the t-name for the MEMBERS subtable in project 17 Obviously, these "special" t-names are not allowed as i-addresses so that there is a difference between i-addresses and t-names

5 Conclusions, Outlook, and Future Plans

The extended NF^2 data model as outlined in this paper is able to integrate flat and hierarchical tables (tables containing "complex objects") in a natural way By doing so, hierarchical tables are an integral part of the data model and need not be treated as "special animals" As a consequence, all operations on flat (1NF) tables like insert, retrieve, update, and delete are applicable to hierarchical (NF^2) tables as well The same holds for projections and selections We have shown how a powerful SQL-like high level query language can be used for retrieval purposes Since our data model is not bound to the implementation of hierarchical structures as described in Section 4, it could, in principle, also be mapped onto an IMS-like system Hence, the extended NF^2 data model could also serve as a possible *migration path* for both *relational* and *hierarchical* databases to join in a *common data model*

We intentionally concentrated on the most important features of our data model and its implementation Thus, we completely neglected the *text support* provided in our system which - optionally based on a *text index* - supports masked search operations in a quite powerful way For instance, to look for all reports (REPNO, AUTHORS, TITLE) in Table 6 which are co-authored by "Jones" and which have words like "computational", "minicomputer", "computer", etc in the title, the following query can be posed (and will be supported by the text index in case that one has been created on TITLE)

```
SELECT x.REPNO, x.AUTHORS, x.TITLE
FROM   x IN REPORTS
WHERE  x.TITLE CONTAINS '*comput*'
       AND EXISTS y IN x.AUTHORS: y.NAME = 'Jones'
```

More details on this text indexing technique can be found in /Sch78, KSW79, KW81/ Maintenance and concurrency control related issues are discussed in /DPS82, DPS83, DLPS85/

Another very important feature of our system is the integrated temporal support, also called *time version support* In fact, we have put a lot of emphasis on performance issues, storage space requirements, and related topics coming along with deriving and maintaining histor-

ical data as an integral - but optional - part of a DBMS (see /DLW84, Lu84/) Currently we are able to support *ASOF (As-of) queries* where one wants to see a (complex or 1NF) table or subtable as it looked like at a fixed point in time in the past If Table 5 had been declared as a "versioned table", the following query would deliver all projects which department 314 has had on January 15th, 1984

```
SELECT y PNO y PNAME
FROM   x IN DEPARTMENTS ASOF January 15th 1984,
       y IN x PROJECTS
WHERE  x DNO = 314
```

Currently we have completed a first version of the DBMS prototype A description of its architecture can be found in /Lu85/ It completely supports the extended NF^2 data model That is, one can insert, retrieve, update, and delete complex tuples either as a whole or only parts of them In addition, all kinds of joins between NF^2 tables are possible Also text support has become an integral part of the DBMS History data support is available but restricted to the above-mentioned ASOF queries "Walk-through-time" queries which work on time intervals are supported at lower system levels (subtuple manager) but have not been brought up to the language interface The reason for this is that there is still some ongoing research on how to support walk-through-time queries on hierarchies in the best way (We will report on this in a separate paper /Bl86/) The current prototype is still a single-user system although it has been designed as a multi-user DBMS One reason is that we first concentrated on getting a stable single-user version and on learning about its applicability and inherent performance issues Another reason is that we are still investigating advanced concurrency control and update processing techniques (see e g /DLPS85/) and we want to finish this first before deciding which technique to use in our system

Currently we are working to make the prototype more complete by adding missing functions and by "streamlining" it where necessary The prototype will be extensively used in a collaboration with partners at the University of Karlsruhe, West Germany, to explore abstract data type concepts in a robotics application area This will also include the issue of *extensible DBMS* Further research issues for the near future will be symbolic query transformation and optimization, workstation support, access path selection, and handling of schema changes

Acknowledgements

We wish to thank H -J Schek (now Tech University of Darmstadt) who initiated and led this project (AIM-I) until the end of 1982 and who is the originator of the NF^2 data model We also want to thank B Hansen, M Hansen, H -D Werner, and J Woodfill who, as visiting scientists, contributed to the development of the AIM-II prototype Thanks are also due to U Deppisch, R Haskin, R Lorie, V Obermeit, and K Shoens for helpful discussions on the design and implementation of our system Last but not least we want to thank our management, especially A Blaser, for their support of our work

References

As76 Astrahan, M M et al System R Relational Approach to Database Management ACM Trans on Database Systems, Vol 1, No 2, June 1976, pp 97-137

Bl86 Blanken, H et al Storage Structures and Query Handling for Time Versions in an Advanced Information Management System (in preparation)

BM85 Bartels, R , Moeller, J Entwurf und Implementierung einer regelbasierenden Planungskomponente fuer die Optimierung von Datenbankanfragen in einer SEQUEL-artigen Sprache (Design and Implementation of a Rule-Based Planning Component for the Optimization of SEQUEL-like Database Queries) Master's Thesis, Tech University of Darmstadt and IBM Heidelberg Scientific Center, Nov 1985 (in German)

Ch76 Chamberlin, D D et al SEQUEL 2 A Unified Approach to Data Definition, Manipulation, and Control IBM Journal of Research and Development, Vol 20, No 6, 1976, pp 560-575

Da81 Date, C J An Introduction to Database Systems (3rd ed) Addison-Wesley Publ Comp , Reading, Mass , March 1981

DGW85 Deppisch, U , Guenauer, J , Walch, G Speicherungsstrukturen und Adressierungstechniken fuer komplexe Objekte des NF^2-Relationenmodells (Storage Structures and Addressing Techniques for Complex Objects of the NF^2 Data Model) Proc Conf "Datenbanksysteme fuer Buero, Technik und Wissenschaft" (A Blaser, P Pistor, eds), Karlsruhe, West Germany, March 1985, Informatik-Fachberichte 94, Springer-Verlag, Berlin Heidelberg New York Tokyo, pp 441-459 (in German)

DLPS85 Dadam, P , Lum, V , Praedel, U , Schlageter, G Selective Deferred Index Maintenance and Concurrency Control in Integrated Information Systems Proc VLDB 85, Stockholm, Sept 1985, pp 142-150

DLW84 Dadam, P , Lum, V , Werner, H -D Integration of Time Versions into a Relational Database System Proc VLDB 84, Singapore, Aug 1984, pp 509-522

DPS82 Dadam, P , Pistor, P , Schek, H -J Praedikat-Sperren mittels Textfragmenten (Predicate Locking Based on Text Fragments) Proc 12 GI-Jahrestagung (J Nehmer, ed), Kaiserslautern, West Germany, Oct 1982, Informatik-Fachberichte 57, Springer-Verlag, Berlin Heidelberg, New York Tokyo, pp 648-668 (in German)

DPS83 Dadam, P , Pistor, P , Schek, H -J A Predicate Oriented Locking Approach for Integrated Information Systems Proc IFIP Congress, Paris, France, Sept 1983, pp 763-768

HL82 Haskin, R L , Lorie, R A On Extending the Functions of a Relational Database System Proc SIGMOD 82, Orlando, Florida, June 1982, pp 207-212

HR83 Haerder, T , Reuter, A Database Systems for Non-Standard Applications Proc Int Computing Symposium (H.J Schneider, ed), Erlangen, West Germany, March 1983, Teubner-Verlag, Stuttgart, pp 452-466

IBM1 SQL/Data System, Concepts and Facilities IBM Corp , GH24-5013

IBM2 IBM Systems Journal (Special Issue on DB2), Vol 23, No 2, 1984

IBM3 IBM Systems Journal (Special Issue on IMS), Vol 16, No 2, 1977

IBM4 Query-by-Example, Terminal User's Guide, IBM Corp , SH20-2078

Jae85a Jaeschke, G Nonrecursive Algebra for Relations with Relation Valued Attributes Technical Report TR 85 03 001, IBM Scientific Center, Heidelberg, West Germany, March 1985

Jae85b Jaeschke, G Recursive Algebra for Relations with Relation Valued Attributes Technical Report TR 85 03 002, IBM Scientific Center, Heidelberg, West Germany, March 1985

JK84 Jarke, M , Koch, J Query Optimization in Database Systems ACM Computing Surveys, Vol 16, No 2, June 1984, pp 111-152

JS82 Jaeschke, G , Schek, H -J Remarks on the Algebra of Non First Normal Form Relations Proc ACM SIGACT-SIGMOD Symp on Principles of Data Base Systems, Los Angeles, Cal , March 1982, pp 124-138

Ka85 Katz, R H Information Management for Engineering Design Springer-Verlag, Berlin Heidelberg New York Tokyo, 1985

KSW79 Kropp, D , Schek, H -J , Walch, G Text Field Indexing Proc Meeting of the German Chapter of the ACM on

Data Base Technology (J Niedereichholz, ed), Bad Nauheim, West Germany, Sept 1979, Teubner-Verlag, Stuttgart, pp 101-115

Kue86 Kuespert, K et al Storage Structures and Addressing Concepts for Complex Objects of the NF^2 Data Model (in preparation)

KW81 Kropp, D , Walch, G A Graph Structured Text Field Index Based on Word Fragments Information Processing and Management, Vol 17, No 6, 1981, pp 363-376

Lo82 Lorie, R A Issues in Databases for Design Applications File Structures and Databases for CAD (J Encarnacao, F L Krause, eds), North-Holland Publ Comp , 1982

Lo84 Lorie, R A et al User Interfaces and Access Techniques for Engineering Databases Research Report RJ4155, IBM Research Lab , San Jose, Cal , Jan 1984

LP83 Lorie, R A , Plouffe, W Complex Objects and Their Use in Design Transactions Proc Annual Meeting - Database Week Engineering Design Applications (IEEE), San Jose, Cal , May 1983, pp 115-121

Lu84 Lum, V et al Designing DBMS Support for the Temporal Dimension Proc SIGMOD 84, Boston, Mass , June 1984, pp 115-130

Lu85 Lum, V et al Design of an Integrated DBMS to Support Advanced Applications Proc Int Conf on Foundations of Data Organization (Invited Talk), Kyoto, Japan, May 1985, pp 21-31 (a very similar version of this paper has also been published in Proc Conf "Datenbanksysteme fuer Buero, Technik und Wissenschaft" (A Blaser, P Pistor, eds), Karlsruhe, West Germany, March 1985, Informatik-Fachberichte 94, Springer-Verlag, Berlin Heidelberg New York Tokyo, pp 362-381)

OH85 Ott, N , Horlaender, K Removing Redundant Join Operations in Queries Involving Views Information Systems, Vol 10, No 3, 1985, pp 279-288

PA86 Pistor, P , Andersen, F Principles for Designing a Generalized NF^2 Data Model with an SQL-type Language Interface IBM Scientific Center, Heidelberg, West Germany, Jan 1986 (submitted for publication)

PHH83 Pistor, P , Hansen, B , Hansen, M Eine sequelartige Sprachschnittstelle fuer das NF^2-Modell (A SEQUEL-like Interface for the NF^2 Model) Proc 13 GI-Jahrestagung, Sprachen fuer Datenbanken (J W Schmidt, ed), Hamburg, West Germany, Oct 1983, Informatik-Fachberichte 72, Springer-Verlag, Berlin Heidelberg New York Tokyo, pp 134-147 (in German)

PT85 Pistor, P , Traunmueller, R A Data Base Language for Sets, Lists, and Tables Technical Report TR 85 10 004, IBM Scientific Center, Heidelberg, West Germany, Oct 1985

Sch74 Schenk, H Implementational Aspects of the CODASYL DBTG Proposal Proc IFIP Working Conf on Data Base Management (J W Klimbie, K L Koffeman, eds), Cargese, Italy, April 1974, North-Holland Publ Comp , pp 399-411

Sch78 Schek, H -J The Reference String Indexing Method Proc Information Systems Methodology (G Bracchi, P C Lockemann, eds), Venice, Italy, 1978, Lecture Notes in Computer Science 65, Springer-Verlag, Berlin Heidelberg New York Tokyo, pp 432-459

Sch85 Schek, H -J Towards a Basic Relational NF^2 Algebra Processor Proc Int Conf on Foundations of Data Organization, Kyoto, Japan, May 1985

Se79 Selinger, P et al Access Path Selection in a Relational Database Management System Proc SIGMOD 79, Boston, Mass , May 1979, pp 23-34

Sh84 Shu, N C A Forms-Oriented and Visual-Directed Application Development System for Non-Programmers Proc IEEE Workshop on Visual Languages, Hiroshima, Japan, Dec 1984, pp 162-170

SLTC82 Shu, N C , Lum, V Y , Tung, F C , Chang, C L Specification of Forms Processing and Business Procedures for Office Automation IEEE Trans on Software Eng , Vol SE-8, No 5, Sept 1982, pp 499-512

SP82 Schek, H -J , Pistor, P Data Structures for an Integrated Data Base Management and Information Retrieval System Proc VLDB 82, Mexico City, Sept 1982, pp 197-207

SS81 Schkolnick, M , Sorenson, P The Effects of Denormalization on Database Performance Research Report RJ3082, IBM Research Lab , San Jose, Cal , March 1981

SS86 Schek, H -J , Scholl, M An Algebra for the Relational Model with Relation-Valued Attributes To appear in Information Systems, Vol 11, No 2, 1986 (also available as Technical Report DVSI-1984-T1, Tech University of Darmstadt, West Germany)

St76 Stonebraker, M et al The Design and Implementation of INGRES ACM Trans on Database Systems, Vol 1, No 3, Sept 1976, pp 189-222

Zl77 Zloof, M M Query-by-Example A Data Base Language IBM Systems Journal, Vol 16, No 4, 1977, pp 324-343

DEPARTMENTS 1NF		
DNO	MGRNO	BUDGET
314	56194	320 000
218	71349	440 000
417	91093	360 000

Table 1 DEPARTMENTS 1NF Table

PROJECTS-1NF		
PNO	PNAME	DNO
17	CGA	314
23	HPAR	314
25	LEXI	218
37	NDBS	417

Table 2 PROJECTS-1NF Table

MEMBERS 1NF			
DNO	PNO	EMPNO	FUNCTION
314	17	39582	Leader
314	17	56019	Consultant
314	17	69011	Secretary
314	23	58912	Staff
314	23	90011	Leader
314	23	78218	Secretary
314	23	98902	Staff
218	25	72227	Staff
218	25	89211	Staff
218	25	92100	Leader
218	25	89921	Consultant
218	25	99025	Secretary
218	25	44512	Consultant
417	37	87710	Secretary
417	37	81193	Leader
417	37	75913	Staff
417	37	96001	Staff

Table 3 MEMBERS 1NF Table

EQUIP 1NF		
DNO	QU	TYPE
314	2	3278
314	3	PC/AT
314	1	PC
218	2	3278
218	2	PC/AT
218	1	3179
218	1	PC/GA
417	1	4361
417	1	PC/XT
417	1	PC/AT
417	2	3278
417	1	3270
417	1	3179
417	1	PC/GA

Table 4 EQUIP-1NF Table

{ DEPARTMENTS }								
DNO	MGRNO	{ PROJECTS }				BUDGET	{ EQUIP }	
		PNO	PNAME	{ MEMBERS }			QU	TYPE
				EMPNO	FUNCTION			
314	56194	17	CGA	39582	Leader	320 000	2	3278
				56019	Consultant		3	PC/AT
				69011	Secretary		1	PC
		23	HPAR	58912	Staff			
				90011	Leader			
				78218	Secretary			
				98902	Staff			
218	71349	25	LEXI	72227	Staff	440 000	2	3278
				89211	Staff		2	PC/AT
				92100	Leader		1	3179
				89921	Consultant		1	PC/GA
				99025	Secretary			
				44512	Consultant			
417	91093	37	NDBS	87710	Secretary	360 000	1	4361
				81193	Leader		1	PC/XT
				75913	Staff		1	PC/AT
				96001	Staff		2	3278
							1	3270
							1	3179
							1	PC/GA

Table 5 Example of an NF^2 Table

{ REPORTS }				
REPNO	< AUTHORS >	TITLE	{ DESCRIPTORS }	
	NAME		KEYWORD	WGT
0179	Jones A	Concurrency and	Concurrency Control	0 6
			Recovery	0 3
			Distribution	0 1
0189	Miller H	Text Editing and	String Search	0 7
	Abraham C		Formatting	0 3
	Meyer F			
0292	Poe A	Branch and Bound	Math Optimization	0 6
	Tacker J		Garbage Collection	0 4

Table 6 Unordered NF^2 Table REPORTS with 3 Ordered "Inner Tables (Lists) AUTHORS

Result Table					
DNO	MGRNO	PNO	PNAME	EMPNO	FUNCTION
314	56194	17	CGA	39582	Leader
314	56194	17	CGA	56019	Consultant
314	56194	17	CGA	69011	Secretary
314	56194	23	HPAR	58912	Staff
314	56194	23	HPAR	90011	Leader
218	71349	25	LEXI	92100	Leader
417	91093	37	NDBS	96001	Staff

Table 7 Result Table of Example 4 (' Unnest ' Operation)

EMPLOYEES-1NF			
EMPNO	LNAME	FNAME	SEX
56194	Schmidt	Horst	male
39582	Mueller	Klaus	male
56019	Bayer	Rolf	male
69011	Maier	Andrea	female
96001	Paulsen	Helga	female

Table 8 Example of a Table in First Normal Form (1NF Table)

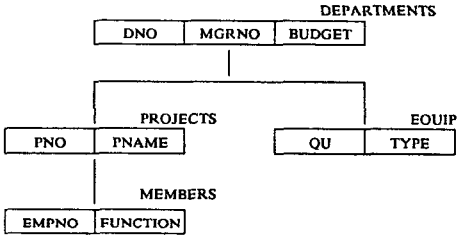


Fig 1 DEPARTMENTS Hierarchy in IMS-Like Representation

```
          first level                second level              ' third level"
          attributes                 attributes                attributes
          ↓                          ↓                         ↓

SELECT    x DNO
          x MGRNO
          PROJECTS =    (SELECT      y PNO
                                     y PNAME
                                     MEMBERS =  (SELECT   z EMPNO
                                                          z FUNCTION
                                                 FROM     z IN y MEMBERS
                                                )
                        FROM         y IN x PROJECTS
                       )
          x BUDGET
          EQUIP =       (SELECT      v QU
                                     v TYPE
                        FROM         v IN x EQUIP
                       )
FROM      x IN DEPARTMENTS
```

Fig 2 Constructing Table 5 from Table 5

```
          first level                second level '            third level
          attributes                 attributes                attributes
          ↓                          ↓                         ↓

SELECT    x DNO
          x MGRNO
          PROJECTS =    (SELECT      y PNO
                                     y PNAME
                                     MEMBERS =  (SELECT   z EMPNO
                                                          z FUNCTION
                                                 FROM     z IN MEMBERS 1NF
                                                 WHERE    z PNO = y PNO
                                                    AND   z DNO = y DNO
                                                )
                        FROM         y IN PROJECTS 1NF
                        WHERE        y DNO = x DNO
                       )
          x BUDGET
          EQUIP =       (SELECT      v QU
                                     v TYPE
                        FROM         v IN EQUIP 1NF
                        WHERE        v DNO = x DNO
                       )
FROM      x IN DEPARTMENTS 1NF
```

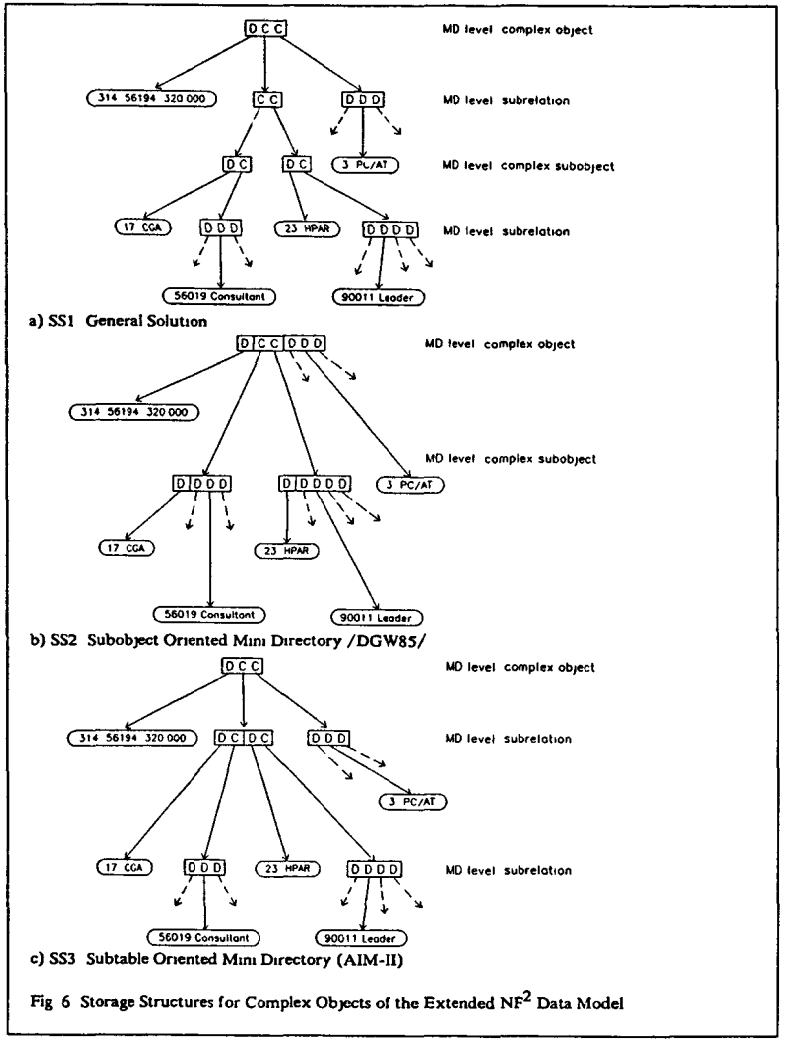
Fig 3 Constructing Table 5 from Tables 1 to 4 (Nest Operation)

```
SELECT    x DNO
          x MGRNO
          EMPLOYEES = (SELECT    z EMPNO
                                 u LNAME
                                 u FNAME
                                 u SEX
                                 z FUNCTION
                       FROM      t  IN x PROJECTS
                                 z  IN t MEMBERS
                                 u  IN EMPLOYEES-1NF
                       WHERE     z EMPNO = u EMPNO
                      )
FROM      x IN DEPARTMENTS
```

Fig 4 Example for a Join (between MEMBERS (in DEPARTMENTS) and EMPLOYEES-1NF)

```
SELECT    x DNO
          y LNAME
          y FNAME
          y SEX
          EMPLOYEES = (SELECT    z EMPNO
                                 u LNAME
                                 u FNAME
                                 u SEX
                                 z FUNCTION
                       FROM      t IN x PROJECTS
                                 z IN t MEMBERS
                                 u IN EMPLOYEES 1NF
                       WHERE     z EMPNO = u EMPNO
                      )
FROM      x IN DEPARTMENTS
          y IN EMPLOYEES 1NF
WHERE     x MGRNO = y EMPNO
```

Fig 5 Query with Two Joins

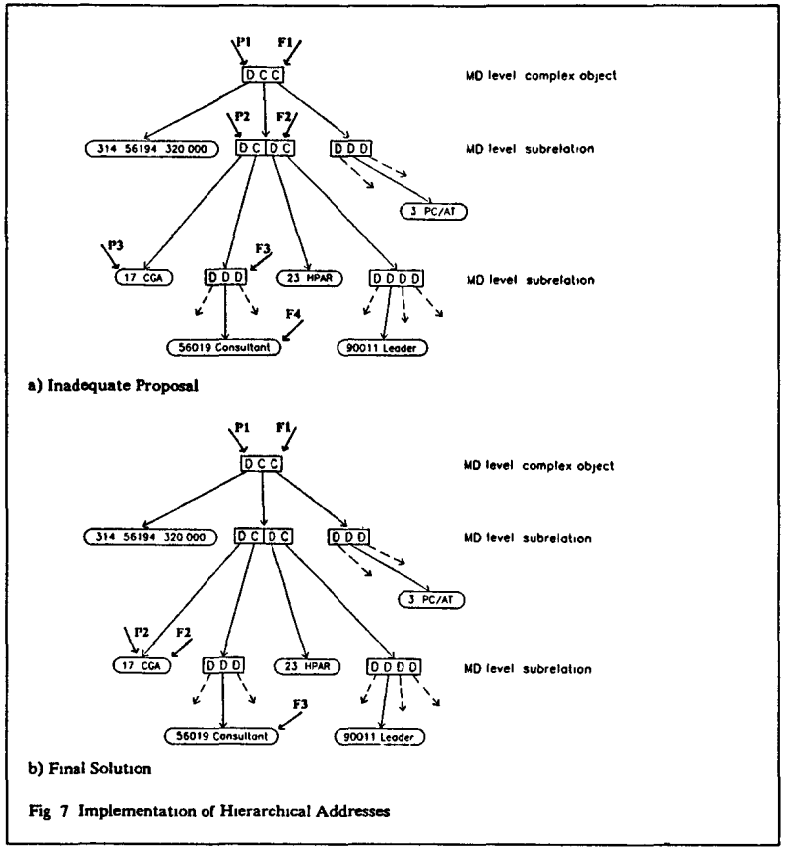


a) SS1 General Solution

b) SS2 Subobject Oriented Mini Directory /DGW85/

c) SS3 Subtable Oriented Mini Directory (AIM-II)

Fig 6 Storage Structures for Complex Objects of the Extended NF^2 Data Model



a) Inadequate Proposal

b) Final Solution

Fig 7 Implementation of Hierarchical Addresses

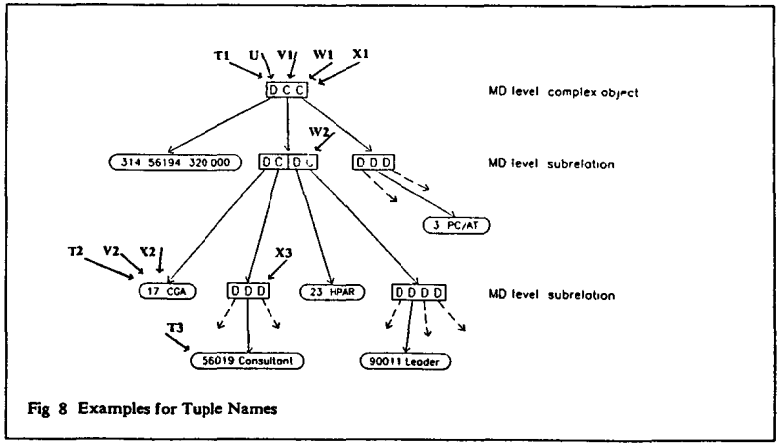


Fig 8 Examples for Tuple Names