Data Structures for an Integrated Data Base Management and Information Retrieval System

H.-J.Schek, P.Pistor

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

New applications like office information systems need interfaces to data bases which integrate classical data manipulation with management and retrieval of textual ("unformatted") data. The relational data model is widely accepted as a high level interface to classical ("formatted") data management. It turns out, however, to be inconvenient for handling even simple data structures as commonly used in information retrieval systems. To attack this shortcoming we propose an extension of the relational model by allowing Non First Normal Form (NF^2) relations. We summarize extensions of the relational algebra, with main emphasis on the new "nest" and "unnest" operations which transform between first normal form relations and the NF^2 ones. A related language is discussed on the basis of a hypothetical SEQUEL language. As a contribution to the problem of efficiently supporting the access to NF^2 tables a novel index support for structured textual data is proposed. It is based on word fragments, words, and word sequences as internal (textual) keys.

1. Introduction

1.1 Motivation of an Integrated System for Formatted and Unformatted Data

In the past, database management systems (DBMS) and information retrieval systems (IRS) were separated in research and development and different products have been developed for either purpose. At present there is a trend towards a single integrated system for data base management and information retrieval - called DBMIRS - because of the following reasons:

Many applications need a DBMIRS. Examples are patients' data within hospital information systems, laboratory document administration, pharmaceutical data bases, library information systems, and - with growing awareness - office informatiom systems. A characteristical feature of these applications is the fact that it is necessary to combine text management and retrieval with usual formatted data manipulation. Therefore a single user interface is necessary.

Another argument for a DBMIRS is related to software architecture. DBMS and IRS share a number of common functions like concurrency control, logging and recovery, and internal indexing techniques. So, it is proposed to have one system instead of duplicating many functions within two different implementations.

In the past, IRS have been regarded as special application programs on top of a DBMS interface, e. g. STAIRS /IBM1/ on DL/1. Now, after having reached a widely accepted DBMS technology it turns out that IRS requirements have been neglected. We will show that an application program solution to a DBMS with a (strict) relational interface is not suitable. We restrict our discussion to relational DBMS /Co82,Da81/ due to its convenience for application programmers and interactive users. Note that IRS are used interactively too.

In the next subsection we will demonstrate why the relational model is inconvenient for IRS data structures and for the most trivial IRS queries. The reason may be seen in the first normal form (1NF) condition. Therefore we propose a relational model (section 2) where Non First Normal Form (NF^2) relations are allowed. The extension encompasses the classical 1NF model and adds two new basic operations "nest" and "unnest" to the relational algebra, described in 2.1. On the basis of the extended relational algebra we propose a query language extension for NF^2 table definition and manipulation.

In section 3 we propose an index component to support predicates against textual (complex) objects in NF^2 tables. We start with a summary discussion on the possible execution of a simple IRS query in a 1NF relational DBMS. It demonstrates again an advantage of the NF^2 model, since it allows to apply techniques known from IRS. We propose a novel technique known as "fragment index" to support basic text retrieval functions. The index keys are word fragments,

words, and word sequences (= descriptor fragments).

Our requirements for the data model of an integrated DBMIRS have been deduced from typical IRS related queries /Sch80a/. It should be noted, however, that similar requirements have been deduced elsewhere /TS81,LY81,SLTC81/ for data models taylored to Office Automation needs, often called "forms data models", or to Engineering applications /Lo81/. The requirement for dropping the 1NF has been stated several times /LY81,Ko81,Ma77/, but there have been no proposals for a general algebra extension to obtain 1NF relations from NF^2 and vice versa, and no investigations on algebraic rules relating the new operations and the conventional ones. An effect similar to dropping the 1NF condition can be obtained by using "quotient relations" as introduced in /FK77/. A language proposal based on a SQL type interface has not yet been made either. However, the more general problem of providing a "universal language" /Da81/ or a language for data conversion and restructuring /SHL75/ has similarities. The index component, originally proposed in /BCLS74/ as word fragment index, has been extended to include word sequences. The index key selection criteria which we use /Sch78,Sch80b/ in order to have a most efficient execution of IRS queries are very similar to the selection criteria for descriptors in automatic indexing investigated mainly by Salton /SW81/.

1.2 An Introductory Example

Figure 1 shows a book inventory table. Due to repeating values in the fields AUTHORS and DESCRIPTORS, this table violates the first normal form condition of relational theory. In order to adapt the table to a classical relational data model, it needs to be restructured as indicated in figure 2.

BOOKS	AUTHORS	TITLE	PRICE	DESCRIPTORS
	A1,A2	T1	P1	D1,D2
	A2	T1	P2	D1,D2
	A1	T1	P1	D1,D2,D3

Fig. 1: Books table as NF^2 relation

BOOK	BNO	TITLE	PRICE
	1	T1	P1
	2	T1	P2
	3	T1	P1

AUTHOR	BNO	AUTHOR
	1	A1
	1	A2
	2	A2
	3	A1

DESCRIPTOR	BNO	DESCRIPTOR
	1	D1
	1	D1
	2	D1
	2	D2
	3	D1
	3	D2
	3	D3

Fig. 2: Normalized relations belonging to the Books table of fig. 1

In comparison to figure 1, the normalized version (figure 2) enforces a rather user-unfriendly view of the book inventory table. In addition, common information retrieval requests exist which are hard to formulate on the basis of the data structure in figure 2, such as

Q1: Display title and price of books described by both descriptors D1 and D2 and written by author A1

Figure 3a gives an SQL like formulation of Q1. Intuitively, a simpler formulation should be possible, as indicated in figure 3b.

```
SELECT TITLE, PRICE
  FROM BOOK, AUTHOR, DESCRIPTOR X, DESCRIPTOR Y
    WHERE AUTHOR = A2
    AND AUTHOR.BNO = BOOK.BNO
    AND BOOK.BNO = X.BNO
    AND X.DESCRIPTOR = D1
    AND BOOK.BNO = Y.BNO
    AND Y.DESCRIPTOR = D2
```

Figure 3a: Formulation of Q1 in SQL referring to fig. 2

```
SELECT TITLE,PRICE
FROM BOOKS
     WHERE DESCRIPTORS ⊇ {D1,D2}
     AND AUTHORS ⊇ {A2}
```

Figure 3b: Q1 in an extended SQL referring to fig. 1

The reason for that seemingly unnecessary complication is the fact that we are not allowed to

keep sequences of values within one attribute. The situation will become even worse if we take into consideration that a single descriptor may in turn be viewed as a composite object (e. g., a word sequence, or even a sequence of sequences of characters).

A relational data base consists of a set of tables (relations). Every table consists of a set of tuples. Every tuple consists of a sequence of attribute values, where every attribute value is atomic. An IRS data base is more deeply structured: It contains one or several document files. Every document file consists of a set of documents. Every document consists of a sequence of fields. Beyond that level of detail, IRS structures become different. Rather than being atomic, a field (e. g., a document abstract or a document table of contents) may contain a sequence of sentences (in the grammatical sense), which in turn are sequences of words. Words, finally, are sequences of atomic values, namely single characters.

Positional information is important when addressing either tuple attributes (DBMS) or document fields (IRS). Inside of DBMS fields or attributes, positional information looses its importance. This is not true with document fields. Here, information is looked for either by mere position, or - more often - by adjacency and order (so-called context information).

2. Non First Normal Form (NF^2)Relational Model

According to the introductory remarks we propose to get rid of the first normal form condition (1NF) and to allow sets and sets of sets as attribute domains. It will allow us to regard the 1NF relational model as a special case but more generally we will be flexible enough for the IRS data structures. It further means that all definitions and all theoretical conclusions of the relational model which are independent from the 1NF remain unchanged and valid. Surprisingly enough, most of the theory can be kept as summarized below:

2.1 Extended Relational Algebra

As usually we start with given sets D1, D2, ..., Dn to serve as atomic domains. However, we allow a k-ary relation $R \subseteq C1 \times C2 \times \ldots \times Ck$ to be a subset of the Cartesian product of k domains where not necessarily every domain Ci is simply some Dj of the given atomic ones. Rather we allow complex domains to be derived from the original ones by constructing powersets of Cartesian products out from the given domains. We call such relations Non First Normal Form (NF^2) relations. As an example we may construct (P(S) denotes the powerset of a set S) $C1 = P(D1 \times D2)$ and $C2 = P(P(D3))$, and take $R \subseteq C1 \times C2$ as binary relation defined on the complex domains C1 and C2. The definition of attributes remains unchanged.

The allowable domains are restricted either to be an atomic domain or a powerset-type domain for any attribute. (We do not allow a Cartesian-product-type domain only, but rather power sets of Cartesian products). Already Kobayashi /Ko81/ discussed these domain extensions and pointed out the importance of powerset-type domains. This restriction allows us to regard every attribute with complex domain as a relation-valued. We will also see that the algebraic operations will generate attribute domains only of this kind.

Now, we discuss the extensions of the operations of the relational algebra using a notation as in /Ul80/. Standard operations as union (U), difference (-), projection Π, and the Cartesian product ($\times$) can be defined for all relations; so they also apply on NF^2-relations in our sense.

For the selection operator σ_F we extend the class of formulae F for which σ_F can be applied. With respect to 1NF relations F is a formula /Ul80/ involving

a) operands that are constants or attributes. A constant is either atomic (for atomic attribute domain) or a constant set of the same type as the attribute domain. A constant set may also contain don't care symbols (see 2.3).

b) arithmetic comparison symbols $=, \neq, <, \leqq, >, \geqq$ and set inclusion symbols $\subset, \subseteq, \supset, \supseteq$.

c) logic symbols $\vee, \wedge, \neg$.

Note that the = and $\neq$ symbols are also applicable for sets.

In our example query Q1 we will be able to do the necessary selection by

$$\sigma_{\text{AUTHORS} \supseteq \{A1\} \ \wedge \ \text{DESCRIPTORS} \supseteq \{D1,D2\}}(\text{BOOKS}).$$

The main extension consists in the introduction of two new operations, called "Nest" and "Unnest". They are used to transform a 1NF relation into a NF^2 relation and vice versa. Given a relation R(A,B,...) we obtain the nested relation along column A of R, abbreviated by $\nu_A(R)$, by forming sets of A values if the tuples in R agree in the remaining components. The following example explains nesting along a single attribute A

R	A	B	C
	a1	b1	c1
	a2	b1	c1
	a3	b1	c2
	a4	b2	c2
	a5	b2	c2

$\nu_A(R)$	A$	B	C
	{a1,a2}	b1	c1
	{a3}	b2	c2
	{a4,a5}	b2	c2

Definition of Nest:
Let R be a n-ary relation with attribute set $\mathbb{A}$. Let $\mathbb{X} = \{X1, X2, \ldots, Xk\}$ be a subset of $\mathbb{A}$ and $\mathbb{Y} = \mathbb{A} - \mathbb{X}$. Let X1X2...Xk$ be a name not in $\mathbb{A}$ to be used as attribute for the resulting relation. For each subtuple $g \in \Pi_{\mathbb{Y}}(R)$ we define an (n-k+1)-tuple wg as

$$wg[\mathbb{Y}] = g$$
$$wg[X1X2...Xk\$] = \{t[\mathbb{X}] \mid t \in R \wedge t[\mathbb{Y}] = g\}$$

The "relation nested along attribute set $\mathbb{X}$" abbreviated as $\nu_{\mathbb{X}}(R)$ is then defined as

$$\nu_{\mathbb{X}}(R) = \{wg \mid g \in \Pi_{\mathbb{Y}}(R)\}.$$

If $\mathbb{X}$ contains a single attribute we call $\nu_{\mathbb{X}}$ a "one-attribute" nest operation, otherwise we have a "multi-attribute nest operation". In our example above we obtain for $\nu_{\{A,B\}}(R)$

$\nu_{\{A,B\}}(R)$ | AB$	C
{[a1,b1],[a2,b1]}	c1
{[a3,b1],[a4,b2],[a5,b2]}	c2

We apply the following naming conventions: if $\mathbb{X} = \{X1,\ldots,Xk\}$ is a set of attributes in R we generate the name 'X1X2...Xk$' as corresponding attribute in $\nu_{\mathbb{X}}(R)$. Especially, if $\mathbb{X} = \{A\}$ we generate A$. The $ sign always indicates a powerset-type attribute and we assume that $ is not allowed for attributes on atomic domains. Occasionally we use also $\mathbb{X}$$ for 'X1X2...Xk$'. Note that a nest operation $\nu_{\mathbb{X}}(R)$ generates an attribute with a powerset-type domain $C(X\$) = P(C(X1) \times C(X2) \times C(Xk))$.

This is a main difference to the partition operator in /FK77/. Whereas we generate new domains for attributes and thereby keeping the notion of tuple or relation, in /FK77/ a new notion of "blocks of tuples" is introduced.

Next we define the unnest operation with the objective in mind to obtain an inverse operation to nest.

Definition of Unnest:
Let R be a n-ary relation with attribute set $\mathbb{A}$ and let $X \in \mathbb{A}$, $\mathbb{Y} = \mathbb{A} - \{X\}$. We distinguish two cases

(a) the domain C(X) is atomic (equal to some Dj), then $\mu_X(R) = R$,

(b) the domain C(X) is a powerset-type domain; i. e. $C(X) = P(C1 \times C2 \times \ldots \times Ck)$ with k (atomic or complex) domains Ci, i=1,...,k. In this case let $\mathbb{X} = \{X1,X2,\ldots,Xk\}$ be a (generated) set of attributes different from those in $\mathbb{A}$. For every tuple $t \in R$ define the set of (n+k-1) tuples w by

$$\mu_X(\{t\}) = \{w \mid w[\mathbb{X}] \in t[X[\wedge w[\mathbb{Y}] = t[\mathbb{Y}]\}$$

and let

$$\mu_X(R) = \bigcup_{t \in R} \mu_X(\{t\}).$$

Note that the case k=1 is included in the above definition. Obviously, if we apply repeated unnest operations on a NF^2 relation we end up with a 1NF relation. Whether we can do that lossless (in the sense that we reobtain the original relation by nesting) will be answered by the following

Theorem:
(1) Unnest is inverse to nest, i. e.
$\mu_{X\$}(\nu_X(R)) = R$.

(2) Nest is not always inverse to unnest but if X is not member of all keys of R
$\rightarrow \nu_{\mathbb{X}}(\mu_X(R)) = R$.

The proof is omitted here. It can be found in /JS81/ for the special case of one-attribute nesting and can be extended easily to multi-attribute nesting. The statement (2) of the theorem is not severe. If X is a key attribute and if there is no other key which does not contain X we may add an (artificial) key attribute to R.

Joins:
In the 1NF relational model we have some convenient operations defined on the basic ones such as joins. A "Θ-join" between two relations R and S on attribute A of R and B of S is defined by
$R \mid\times\mid_{A\Theta B} S = \sigma_{A\Theta B}(R \times S)$

Therefore, after having extended the selection formula by set inclusion symbols we may obtain e. g. a "$\subseteq$-join" between two relations if they have two attributes with the same domain type.

The natural join between two relations can also be applied for powerset type attributes using set equality. The question arises whether we can obtain a natural join between two relations R and S by computing a natural join between $\nu_{\mathbb{X}}(R)$ and $\nu_{\mathbb{X}}(S)$ and then unnest the result. $\mathbb{X}$ is assumed to be set of common attributes. Simple examples show that this is not the case generally. However with the "intersection join" we can find a simple solution:

Definition "Intersection Join":
Let R and S be two relations with attribute sets $\mathbb{A}$ and $\mathbb{B}$ and with $\mathbb{X}$ as common attribute set. The intersection join between $R' = \nu_{\mathbb{X}}R$ and $S' = \nu_{\mathbb{X}}S$, abbreviated by $R' \bar{\times} S'$ is obtained by combining all pairs of tuples $r' \in R'$ and $s' \in S'$ where the intersection $r'[\mathbb{X}\$] \cap s'[\mathbb{X}\$]$, taken over the join attribute $\mathbb{X}$$, is not empty. Precisely,

$$R' \bar{\times} S' = \{t \mid (\exists r')(r' \in R' \wedge (\exists s')(s' \in S' \wedge (t[X\$] = r'[X\$] \cap s'[X\$] \neq \emptyset) \wedge t[A-\{X\$\}] = r'[A-\{X\$\}] \wedge t[B-\{X\$\}] = s'[B-\{X\$\}]))\}.$$

The following is an example for an intersection join.

R' | A	X$	S' | X$	B
a1	{x1,x2,x3}	{x2,x3,x4}	b2
a2	{x4}	{x5,x6}	b1

R'$\bar{\times}$S' | A	X$	B
a1	{x2,x3}	b2
a2	{x4}	b2

The following theorem gives an answer to our original plan:

Theorem: Under the assumptions of the above definition we have
$R|\times|S = \mu_{X\$}((v_X(R)) \bar{\times} (v_X(S)))$.

The proof is obvious and omitted here. The theorem may be useful for a practical implementation when we have to compute a natural join between two nested relations.

Lists

For practical purposes it is convenient to have sets of items with an order. We may ask for an item before or after one specific item or we might ask for two items which are adjacent. As already mentioned typical applications can be found in IRS objects. Sections, sentences in sections, words in sentences can be regarded as (nested) lists. From a relational algebra point of view we regard lists as powersets of pairs. If C is the (complex or atomic) domain for the list elements we use the set of ordinal numbers N as additional domain and define the domain of a list-type attribute as P(C×N). Since we already allowed powersets of Cartesian products of domains we have lists as a special case.

2.2 Dependency Theory

Allowing NF^2 relations does not mean that we exclude 3NF or BCNF. Originally these higher normal forms have been defined on the basis of 1NF /Da81/. Newer definitions do not assume the 1NF /U180/. In /Ma77,Ko80,JS81/ it was pointed out that it has advantages to drop 1NF. It was shown that multivalued dependencies disappear and become simpler functional dependencies. This result may also be regarded as a justification for a NF^2 relational model.

2.3 Query Language Considerations for NF^2 Tables

As will be shown in a forthcoming paper /PiTr82/, the previously discussed extensions of the relational algebra may be introduced into query languages like SQL /Ch76/. In doing so, one is faced with the problem of appropriately accessing non-atomic attributes. Since their values can be viewed as - possibly unnormalized - tables (see section 2.1), they may be handled by queries within queries ("field queries"). Those "queries" may occur in complex query conditions (see formula 6.2)), or in cases where non-atomic attributes require some kind of editing (e. g. projection) before output (see formula (5.1)). These features might be too cumbersome in common tasks like defining less complicated predicates over non-atomic attributes. Therefore the manipulation facilities for non-atomic fields are complemented by features which emphasize the fact of non-atomic objects being lists, sets, or tuples.

The following examples are intended to give an impression of our query language proposal. In order to demonstrate the language providing an appropriate interface for IR applications, all examples have been taken from that environment (cf. fig. 1). A first group of examples is centered around the so-called "Boolean search" and stresses language features taylored to list and set type objects. The following subsection demonstrates the use of "field queries" and is intended to give some evidence for the language to provide the expressive power for supporting so-called "non-Boolean" /Sa81,Ri79/ search conditions. NEST and UNNEST operations are addressed in a final subsection.

In the following subsections we use {...} for sets, <...> for lists, and [...] for tuples. As don't care symbols we introduce:

- `*` don't care either an arbitrary number of set element or a sublist of arbitrary length,
- `_` don't care exactly one set, list, or tuple component.

2.3.1 Predicates Over Non-atomic Attributes

The following query features a so-called Boolean search condition

Q2: Using table BOOKS (see fig. 1), find the titles of all books, the DESCRIPTORS of which contain the word list DECISION TREE OPTIMIZATION.

In our language proposal, this query is expressed as

```
(1.0)   SELECT TITLE FROM BOOKS
(1.1)   WHERE DESCRIPTORS ⊇
            {<'DECISION','TREE','OPTIMIZATION'>}
```

By using the inclusion symbol, the WHERE clause reflects the relational algebra extensions discussed in section 2.1.

Set predicates as in (1.1) are obviously open for joining, thus extending the capabilities of Boolean search into directions not considered so far in IR:

Q3: Display the BNO and TITLE of those books whose DESCRIPTORS are properly contained in the DESCRIPTORS of any book in BOOKS

Q3 will be resolved by

```
(2.0)   SELECT X.BNO, X.TITLE, Y.BNO,
        FROM BOOKS X, BOOKS Y
(2.1)   WHERE X.DESCRIPTORS c Y.DESCRIPTORS
```

It is claimed that queries of this type - when based on a normalized BOOKS table - can hardly be answered without falling back to host language support.

For the formulation of predicates as in (1.1) and (2.1) we offer a probably quite attractive alternative, an extension of the so called template technique featured in SQL/DS /Ch80,IBM2/. With this technique, query Q2 transcribes to

```
(1.0')  SELECT TITLE FROM BOOKS WHERE
(1.1')  DESCRIPTORS =
          {<'DECISION','TREE','OPTIMIZATION'>,*}
```

Advantages of the template technique become more obvious when context conditions are to be expressed in the case of list type objects. Consider e. g.

Q4: Find the titles of all books, the DESCRIPTORS of which contain at least one descriptor with the adjacent words TREE (first position) and OPTIMIZATION.

```
(3.0)   SELECT TITLE FROM BOOKS WHERE
(3.1)   DESCRIPTORS =
                {<*,'TREE','OPTIMIZATION',*>,*}
```

Note that masking is applied here at two different levels. It could be applied even at lower levels, e. g. to catch records with the misspelled descriptor fragment OPTIMISATION:

```
(3.1')  ...DESCRIPTORS =
                {<*,'TREE','OPTIMI_ATION',*>,*}
```

The formulation of predicates as well as the construction of output tuples can greatly be facilitated by an appropriate set of built-in functions. Some useful functions are compiled in fig. 4. While some of them (e. g. SUM, AVG) are fairly application oriented, others (e. g. DCAT, DUNION, ELEMS, MLIST) are more general in that they can be used to restructure non-atomic objects.

2.3.2 Advanced Query Facilities

In contrast to Boolean search, non-Boolean search takes into account not only the presence/absence of specified search terms, but also their relevance for the record under consideration 1). Fig. 5 demonstrates how this information could be stored conceptually. A typical query against this table might be

Function/ Operation		Applicable with fields of type		
		Set	List	Tuple
CARD	Cardinality	x		
LEN	Length		x	
SUM	Sum	x	x	
AVG	Arithmetic Average	x	x	
MAX	Arithmetic Maximum	x	x	
MIN	Arithmetic Minimum	x	x	
DUNION	Union of elements of a set of sets	x		
CAT	Concatenation		x	
DCAT	Concatenation of components of a list of lists		x	
ELEMS	Set of list components		x	
MLIST	Generates a list of a set (any order)	x		
(i)	Index of the i-th component		x	x

Figure 4: Field oriented standard-functions and operators

BOOKSW	BNO	AUTHORS	TITLE	PRICE	WDESCR	
					WGT	DESCR
	1	A1,A2	T1	P1	W1	D1
					W2	D2
	2	A2	T1	P2	W3	D1
					W1	D2
	3	A1	T2	P1	W2	D1
					W1	D2
					W3	D3

Fig. 5: Stylized fictitious book inventory table with weighted descriptors

1) This paper does not address the question of how to extract useful descriptors from a given text (e. g. book titles), nor how to determine the relevance of descriptors for the piece of information containing them (see e. g. /Ri79,Sa81/).

Q5: Using table BOOKSW, find the titles of all books written by A1 and described by descriptor D1, provided its relevance for the book is greater than 0.5.

```
(4.0)   SELECT TITLE FROM BOOKSW WHERE
(4.1)   AUTHORS ⊇ {A1}      AND
(4.2)   (SELECT * FROM WDESCR
           WHERE DESCR = D1
           AND WGT > 0.5) ⊃ {}
```

Obviously, the predicate (4.2) surrogates an existential quantifier expression. It should be noted that the SELECT construct is needed here to extract information from a non-atomic field. The necessity for this technique may also arise outside the WHERE clause, as demonstrated in Q6 and Q7:

Q6: Find authors and descriptors for all books having at least one descriptor, the weight of which is greater than 0.5.

```
(5.0)   SELECT AUTHORS,
(5.1)          (SELECT DESCR FROM WDESCR)
(5.2)   FROM BOOKSW WHERE
(5.3)   (SELECT * FROM WDESCR
           WHERE WGT > 0.5) ⊃ {}
```

Q7: Find the book numbers for all books having at least two descriptors out of D1, D2, D3, D4, provided the cumulative weight of these descriptors exceeds 1. The non-occurrence of a descriptor is considered to reflect a zero weight.

```
(6.0)   SELECT BNO FROM BOOKSW
(6.1)   WHERE SUM (SELECT WGT FROM WDESCR
           WHERE DESCR IN {D1,D2,D3,D4})>1
(6.2)   AND CARD({D1,D2,D3,D4} ∩
           (SELECT DESCR FROM WDESCR)) ≧ 2
```

2.3.3 NEST and UNNEST Mapping Facilities

The NEST and UNNEST operations serve two purposes. First, they provide a means of generating unnormalized views out of first normal form (1NF) tables, or vice versa. Second, they open a further possibility for temporarily changing the structure of tables subjected to queries. This feature will help in cases which cannot be handled by the constructs presented so far. Consider, e. g., the 1NF table AUTHOR (fig. 2). A denormalization as in

D1: Define the view AUTHOR_S on top of AUTHOR, which contains for every book number the set of associated authors

is achieved by the NEST operation:

```
(7.0)   DEFINE VIEW AUTHOR_S (BOOKNO, AUTHORS)
        AS
(7.1)   SELECT BNO,
(7.2)          NEST (AUTHOR)
(7.3)   FROM AUTHOR
```

This statement is supposed to collect the sets of AUTHOR values associated with the individual BNO values.

The mapping from NF^2 tables into 1NF tables, as in:

D2: Define NORM_AUTHOR_S as a normalized view of AUTHOR_S,

is achieved by

```
(8.0)   DEFINE VIEW NORM_AUTHOR_S (BNO,AUTH) AS
(8.1)   SELECT BOOKNO,
(8.2)          UNNEST (AUTHOR$)
(8.3)   FROM AUTHOR_S
```

The resulting table NORM_AUTHOR_S is equivalent to the table AUTHOR used in D1.

When applying UNNEST to list type attributes, the sequence information of LIST objects being normalized may optionally be kept or suppressed. When the inverse operation is used, LIST type objects are collected in a fashion similar to that described along with example D1. This process, however, requires some explicitely or implicitely stored ordering information. The latter case is illustrated by

D3: Define the view AUTHOR_L on top of AUTHOR, which contains for every book-number the list of associated authors in alphabetical order.

This nesting operation is achieved by:

```
(9.0)   DEFINE VIEW
           AUTHOR_L(BOOKNO,AUTHOR_LIST) AS
(9.1)   SELECT BNO,
(9.2)          NEST (AUTH, ORDER BY AUTH)
           FROM NORM_AUTHOR_S
```

For a final example demonstrating multiattribute nesting, assume a weight column WGT in DESCRIPTOR (see fig. 2). By

D4: Define a view DESCRIPTW on DESCRIPTOR, displaying for every book number the associated set of weight/descriptor pairs,

i.e. by

```
(10.0)  DEFINE VIEW
           DESCRIPTW(BNO,WDESCR (WGT,DESCR)) AS
(10.1)  SELECT BNO,
(10.2)         NEST([WGT,DESCRIPTOR])
(10.3)  FROM DESCRIPTOR
```

one obtains the equivalent to a projection of the table in fig. 5.

3. Indices for NF^2 Relations with Textual Attributes

The previous section was devoted to data model and query language considerations. Therefore, the main arguments were centered around convenience. This section adds performance oriented arguments. First we shortly discuss, how a typical (Boolean) information retrieval query could be processed in a relational DBMS like System R /BLAs81/. The assumption would be that we have the IRS data structures (NF^2 relations) only as an external data model but a strict relational one as System R at the conceptual level.

We use again the BOOKS table now stored as the tables B(BNO,T,P), A(BNO,A), D(BNO,DNO), DW(DNO,S,W). The B and A tables correspond to the BOOK and AUTHOR tables, resp., of fig. 2. The DESCRIPTOR table has been split into D(BNO,DNO) and DW(DNO,S,W) where DNO, W, and S denote descriptor number, word, and position of the word, respectively. As usual in IRS, this enables us to support predicates both at word and descriptor level.

Let us consider, the following example query in NF^2 notation:

```
SELECT TITLE FROM BOOKS
WHERE DESCRIPTORS =
          {<*,'DATA','BASE',*>, <*,'QUERY',*>,*}
```

This query searches all book titles whose descriptors contain DATA and BASE adjacently and somewhere else QUERY. Note that this is a quite simple and usual query in an IRS like STAIRS /IBM1/.

A	BNO	A
	1	A1
	1	A2
	2	A2
	3	A1

B	BNO	T	P
	1	T1	P1
	2	T1	P2
	3	T1	P1

D	BNO	DNO
	1	2
	1	2
	2	1
	2	2
	3	1
	3	2
	3	3

DW	DNO	S	W
	1	1	DATA
	1	2	BASE
	2	1	DATA
	2	2	SYSTEM
	3	1	QUERY
	3	2	LANG.

When adapting this query to the normalized tables B, D, and DW, it becomes fairly complicated:

```
SELECT T FROM B, D X, D Y, DW Z1, DW Z2, DW Z3

      WHERE     Z1.W = 'DATA'
        AND Z2.W = 'BASE'
        AND Z1.S = Z2.S-1
        AND Z1.DNO = Z2.DNO
        AND Z3.W = 'QUERY'
        AND X.BNO = Y.BNO
        AND X.DNO = Z1.DNO
        AND Y.DNO = Z3.DNO
        AND X.BNO = B.BNO
```

The reader may have noticed that this represents a join involving 6 tables. Even if we assume that all addressed attributes are supported by indexes, queries of this type are expensive to process. This is especially true with volume of data typically encountered in IRS (e. g. 100.000 documents, 100.000 different descriptor words).

From this example we must conclude that it is not recommendable to implement NF^2 tables on top of a classical relational DBMS. On the contrary, a direct access support is required, as it is offered for textual data by IR systems like STAIRS. This section presents an alternative access support which is more appropriate for textual data with update traffic, and which supports general fragment search.

3.1 Indices for NF^2 Relations With Textual Attributes

In a 1NF relation an index being created to an attribute contains every attribute value as a key. Attribute values are atomic and so are the keys. In a NF^2 relation we have non-atomic attribute values. Therefore we allow to have non-atomic keys also for an index to a non-atomic attribute. Roughly, if the attribute is of type SET or LIST, index keys are of type SET or LIST, respectively. Index keys have the same (complex) domain as the associated attribute and the associated search arguments. An index for textual attributes following these general lines is the Fragment Index. It is a novel text index /BCLS74,SchH76,Sch78/ the keys of which are fragments, words, and word sequences. Compared with a favourite IR indexing technique, namely full word indexing (e. g. STAIRS), this approach has the following advantages:

(i) Small set of index keys: With Fragment Indexing, the number of index keys is typically 2000 - 5000; with full word indexing the number of index keys increases with primary data size. 500 000 index keys are quite common.

(ii) Set of index keys insensitive against changes of primary data content: This property simplifies index maintenance. Primary data changes need be reflected only in record identifier lists.

(iii) It supports general fragment search (masking of search terms at any position) often useful for textual objects in DBMS.

3.1.1 Index Key Selection

The basic ideas of Fragment Indexing are most readily outlined for a file, the records of which contain just one field of the type "word list" such as

```
DATA: <'QUERIES','AGAINST','HYPOTHETICAL',
                    'DATA','BASE','SYSTEMS'>
```

Query arguments exhibit the same structure, e. g.

```
QUERY: <*,'QUERY',*,'DATA','BASE','SYSTEMS',*>
```

Consequently an index for such a non-atomic attribute should reflect just this data structure in its index keys. In our example, keys like

```
INDEX:    <*,'DATA','BASE','SYSTEMS',*>
          <*,'QUERY*',*>
```

might appear. The problem is how to find a "good" set of these "fragments". We have proposed a solution along the following lines: If we built the index such that all possible words occur as index keys, we would have two problematic subsets of keys, namely unselective keys and exotic keys.

In order to make unselective keys more selective (and at the same time shorten the associated pointer list), we make use of the following observation. A non-selective word like "SYSTEMS" will probably be used in context with other words, e. g. "DATA BASE SYSTEMS", in order to increase selectivity. Consequently, word lists (so-called phrases) should be identified as index terms to reflect frequently referenced context information in the index. To return to our example, an object "SYSTEMS AND CYBERNETICS" will be referred to by the key "SYSTEMS", the object "DATA BASE SYSTEMS ANALYSIS", however, by the key "DATA BASE SYSTEMS" rather then by the key "SYSTEMS".

Exotic words, on the other hand, are infrequent both in data and queries. In this case fragments should be identified which also occur in other - possibly exotic - words. E. g. the fragment "HYPO" could be a useful key since it could make dispensable a whole pool of index keys like HYPOTHETICAL, HYPOCHONDRIA, HYPO, HYPOCRITE etc.

Putting the previous discussion into more general terms, index keys are selected on the basis of data and query statistics. These are determined from a frequency analysis of representative samples of data and of queries. Given the data and query statistics and a desired frequency t for every key, an algorithm determines a set of keys where every key has a sum of data and query frequency close to t.

The availability of query statistics is not considered critical. One may start with plausible estimates; at a later point in time, the index can be adapted, if the original assumptions are outdated by observed query statistics.

There is a connection to the Interval Index problem discussed in /Sche80b/. Among other results, it points out that equiprobable keys are cost optimal if no update activities are assumed and if the distributions of values in data and in queries are the same. Since updates are regarded infrequent in IR applications, and, since a correlation between data and query frequencies could be observed, empirically we have a justification of the heuristics applied for textual key selection. It is also interesting to notice that the selection of descriptors in the methods of statistical "Automatic Indexing" is based on the equiprobability approximation /SW81/.

3.1.2 Query Resolution

To resolve a query by using the Text Fragment Index, three processing steps are required, decomposition, pointer list merge, and verification.

NAME-LIST (a)

ID	Name
1	<'CENTRAL','COMPUTER','SERV.'>
2	<'SING','AND','LEASE','ASSOC.'>
3	<'COMPUTER','LEASING','CO.'>

Fragment Index on NAME (b)

NAME FRAGMENTS	POINTER-LISTS
<*,'COMPUTER','LEASING',*>	3
<*,'*LEAS*',*>	2
<*,'*COMP*',*>	1
<*,'*SING*',*>	2

Figure 6: Fictitious table (a) and related fragment index (b)

(Note the non-redundancy of pointer lists. E.g. 'SING' is not associated with record 3 of NAME-LIST, since it does not occur outside of 'COMPUTER LEASING CO.' and note also that pointers do not contain "context information" such as word position).

The decomposition step determines a set of index keys which are compatible with the search argument. For instance, with the data structures of fig. 6 and the predicate

(11) ... WHERE NAME = <*,'COMPUTER','LEASING',*>,

the decomposition recognizes that the word sequence COMPUTER LEASING is the index key which completely reflects the search argument in (11). Things get more involved, when the index keys do not contain the search argument , as in

(12) ... WHERE NAME = <*,'*COMP*','LEASING',*>

The difference is that now the word fragment COMP is used with front and end masking.

In this case the decomposition determines a predicate which defines the least possible superset of all matches to the original predicate. Details are described in /KW81/. In our example, the decomposition of (12) is

```
(13) ... WHERE NAME = <*,'COMPUTER','LEASING',*>
         OR NAME = <*,'*COMP*',*>
            AND NAME = <*,'*LEAS*',*>
            AND NAME = <*,'*SING*',*>
```

Note that the format (13) has been used to explain the decomposition. It is not generated internally in this way.

Subsequent to query decomposition, the pointer lists of the matching index keys have to be merged (set union, intersection) in a way which reflects the query structure. As indicated in (13) the pointer lists of COMP, LEAS, SING must be intersected. The result is set-united with the pointers of COMPUTER LEASING.

The final step is the verification step: The merge result is used to actually access the primary data and to discard those records which do not match the search argument.

Obviously the masking of index keys need not be restricted to two levels of nesting, as in (13). The Fragment Index approach is open for any depth of nesting, be it of type LIST - as in the previous examples - or of type SET (not further to be outlined here).

3.1.3 Results of Prototype Experiments

A quantitative evaluation of performance aspects has been performed, such as the average number of secondary pointer list operations, the number of index keys, the degree of equiprobability, the influence of the index size on the number of IO's, the CPU time needed for the decomposition, the average hit rate (false drops), the influence of adaptation. Results can be found in /Wi81/. In summary, the experiments show, that with a dictionary size of two or three thousand entries the overall number of secondary IO's is between two and four for two word queries with general masking. The hit rate depends on the length of the documents. For short textual objects like names and addresses, the hit rate is between 80 and 100 percent. It was further shown that we obtain a similar performance as obtainable with a full word dictionary.

Conclusions

The NF^2 relational model, if used for the conceptual level would allow to define external views for different applications easily. Especially, where the classical relational model is convenient, it can be provided since it is contained as a special case. Some other applications such as information retrieval would prefer NF^2 relations. In order to support the new predicates for NF^2 relations we propose an index whose keys are also non-atomic. For the special case of textual attributes we could show the feasibility of this approach. Many questions still need more investigations, e. g. embedding NF^2 relations into a host language, and general index support for complex attributes beyond the textual case.

References

/BCLS74/ I. J. Barton, S. E. Creasey, M. F. Lynch, M. J. Snell: An Information Theoretic Approach to Text Searching in Direct Access Systems, CACM 17 (1974), pp. 345 - 350.

/BlAs81/ M. W. Blasgen, M. M. Astrahan et al.: System R: An Architectural Overview, IBM Systems Journal 20 (1981), No. 1, pp. 41 - 62.

/Ch76/ D. D. Chamberlin et al.: SEQUEL2: A Unified Approach to Data Definition, Manipulation and Control, IBM Journal of Research and Development 20 (1976), pp. 560 - 575.

/Ch80/ D. D. Chamberlin: A Summary of User Experience with the SQL Data Sublanguage Proceedings of the Int. Conf. on Data Bases (ed. S. M. Deen), Heyden & Son, London 1980.

/Co82/ E. F. Codd: Relational Database: A Practical Foundation for Productivity, CACM 25 (1982) No. 2, pp. 109 - 118.

/Da81/ C. J. Date: An Introduction to Database Systems, 3rd ed., Addison-Wesley Publ. Co., Reading, Mas., 1981.

/FK77/ A. Furtado, L. Kerschberg: An Algebra of Quotient Relations, in ACM SIGMOD (ed. D. Smith), August 1977.

/JS81/ G. Jaeschke, H.-J. Schek: Remarks on the Algebra of Non First Normal Form Relations, in Principles of Database Systems, ACM SIGACT-SIGMOD Proceedings, Los Angeles, March 1982, ACM Order No. 475820.

/Ko80/ I. Kobayashi: An Overview of the Database Management Technology, Techn. Report TRCS-4-1, Sanno College, 1753 Kamikasuya, Isehara, Kanagawa 259-11, Japan, Juni 1980.

This is a worst-case figure. In general, only the maximum hot point needs to be computed.

6. REFERENCES

[ASTR76] Astrahan, M. M., et al. "System R: A Relational Approach to Database Management," ACM Trans. on Database Systems, 1, 2, June 1976 (97-137).

[BLAS79] Blasgen, M. W., et al. "System R: An Architectural Update," IBM Research Report: RJ2581, July 1979.

[DENN68] Denning, P. J. "The Working Set Model for Program Behavior," Comm. of the ACM, 11, 5, May 1968 (323-333).

[OBER80] Obermark, R. "Global Deadlock Detection Algorithm," IBM Research Report RJ2845, June 1980.

[SACC82] Sacco, G.M. and Schkolnick, M. "Buffer Management in Relational Database Management Systems," in preparation.

[SELI79] Selinger, P. G., et al. "Access Path Selection in a Relational Database Management System". Proc. of the 1979 SIGMOD Conference.

[LANG77] Lang, T., et al. "Database Buffer Paging in Virtual Storage Systems," ACM Trans. on Database Systems, 2, 4, Dec. 1977 (339-351)

[LORI77] Lorie, R. "Physical Integrity in a Large Segmented Database," ACM Trans. on Database Systems, 2, 1, Mar. 1977 (91-104)

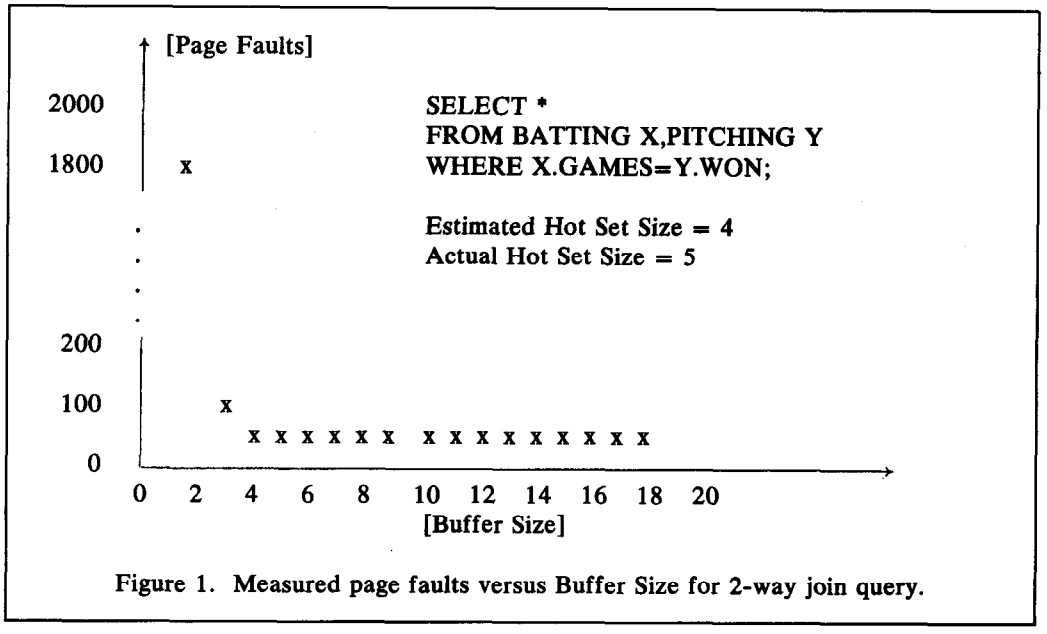

Figure 1. Measured page faults versus Buffer Size for 2-way join query.

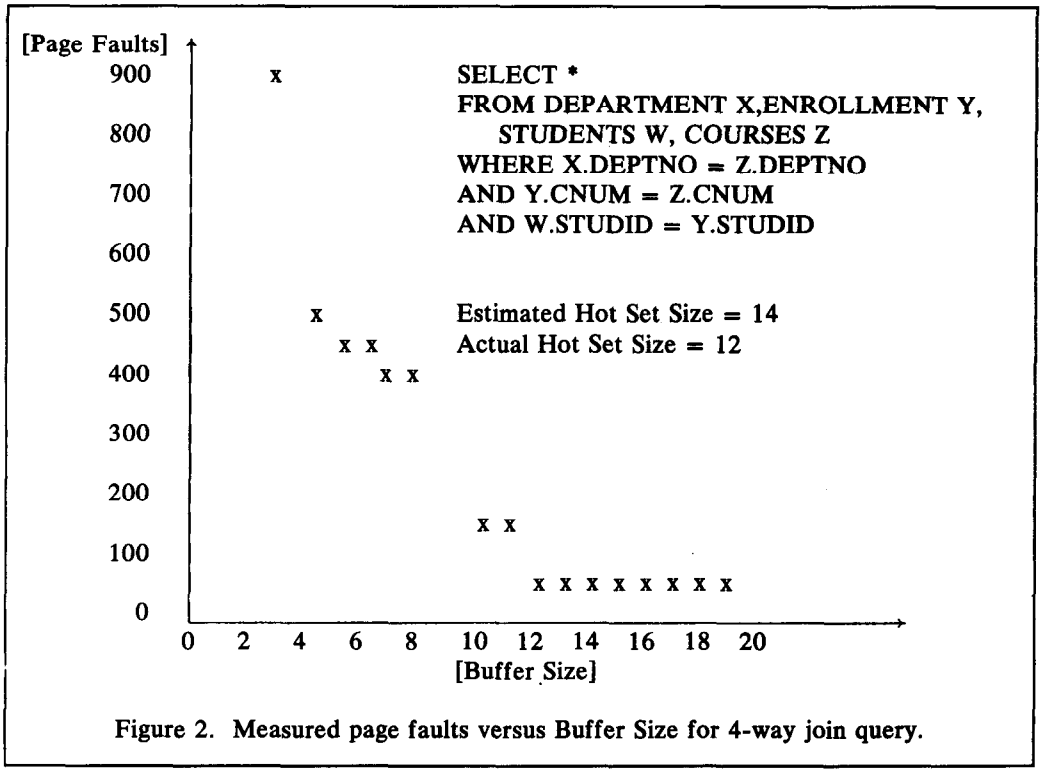

Figure 2. Measured page faults versus Buffer Size for 4-way join query.