The relational model has received increasing attention during the past decade. Its advantages include simplicity, consistency, and a sound theoretical basis. In this article, the naturalness of viewing information retrieval relationally is demonstrated. The relational model is presented, and the relational organization of a bibliographical database is shown. The notion of normalization is introduced and first, second, third, and fourth normal forms are demonstrated. Relational languages are discussed, including the relational calculus, relational algebra, and SEQUEL. Numerous examples pertinent to information retrieval are presented in these relational languages. Advantages of the relational approach to information retrieval are noted.

Introduction

There are several important data models which have been used as a basis for developing database systems. Principally, these are the hierarchic model, the network model, and the relational model. These are conceptual models, describing how a user views and manipulates the data, independent of the structure and physical storage of the data. Atkinson [1] surveys this area, providing useful evaluation related to applications in the field of documentation, though most specifically with regard to library automation. Van Rijsbergen [2] usefully distinguishes between such library automation (e.g., automating catalogs and circulation systems) and information retrieval (document retrieval, bibliographic retrieval), and notes the greater complexity of file structures in information retrieval.

While the ideas presented here underlie a particular ongoing research effort, the MISTRAL project [3, 4], it is not intended to describe that project. Rather, the aim is the more general one of describing the relational view as applied to information retrieval. To this end discussion covers the relational organization, the development of a relational bibliographic database including normalization, relational query languages, and retrieval examples.

The Relational View

We introduce briefly the concepts and terminology of the relational data model. A concise introduction and good bibliography are found in Chamberlin [5].

Received March 3, 1980; accepted April 10, 1980.
© 1981 by John Wiley & Sons, Inc.

Mathematically, given sets $D_1, D_2, \ldots, D_n$ (not necessarily distinct), a relation $R$ is a set of $n$-tuples each of which has its first element from $D_1$, second element from $D_2$, etc. The sets $D_i$ are called domains.

Intuitively, a relation is an unordered two-dimensional table in which each row represents a tuple and no two rows are identical. The columns of the table are called attributes. Figure 1 shows the relation AUTHOR-HISTORY which is formed by the attributes AUTHOR (author's name), BORN (year of birth), FIRST_PUBLISHED (year of first publication), and #PUBLICATIONS (number of publications). We write this as AUTHOR-HISTORY <AUTHOR,BORN, FIRST_PUBLISHED,#PUBLICATIONS>.

It is useful to note the difference between attributes and domains. As seen in the example, the second and third columns are based on identical domains: the set of possible dates. However, each column has a different attribute name to describe its meaning in this particular relation: BORN and FIRST_PUBLISHED.

An attribute or set of attributes whose values uniquely identify the tuples in a relation is called a candidate key. A relation may have more than one candidate key, and if so, it is customary to designate one as the primary key.

For the relation AUTHOR-HISTORY, the only candidate key is the single attribute AUTHOR which is therefore the primary key. It is important to note that keys are determined by the database designer, not by the current values stored. That is, in Figure 1 each row may be uniquely identified by the attribute pair <BORN,FIRST_PUBLISHED>. However, this is intuitively unsatisfactory as this uniqueness could easily disappear upon addition of a new tuple to the relation (e.g., <SMITH, 1956, 1978>). Thus we decide that AUTHOR will be unique in our example (recognizing the naivety of that decision in real life, particularly with using only last names).

It has been observed that certain collections of relations have better properties in updating than do other collections.
Fundamental to this normalization is the notion of functional dependency. For a relation \( R \), attribute \( B \) is said to be functionally dependent on attribute \( A \) if, at every instant of time, each \( A \) value in \( R \) has no more than one \( B \) value associated with it in \( R \). We express this by the notation \( A \rightarrow B \); \( A \) determines \( B \). This applies to \( A \) and \( B \) as sets of attributes as well as single attributes.

In our example, each of the nonkey attributes is functionally dependent on the key attribute, AUTHOR. Thus, \( AUTHOR \rightarrow+BORN \), \( AUTHOR \rightarrow+FIRST-PUBLISHED \), and \( AUTHOR \rightarrow+PUBLICATIONS \). There are no other functional dependencies in the relation AUTHOR-HISTORY.

In the next section, these notions will be further developed as a bibliographic relational database is derived.

A Relational Bibliographic Database

While the relational organization to be derived is quite natural and might be produced directly, to facilitate understanding of the concepts involved we develop this relational organization step by step. Assume that we have available and wish to maintain, for each document in a bibliographic database, the information shown in Figure 2. How can we organize this relationally? Were we to treat our data as a single relation directly, as shown in Figure 2 (with the terms listed being attribute or column names), it would be an unnormalized relation. Since most references to relations implicitly deal with normalized relations, that is, relations in first normal form, we look first at normalization.

First Normal Form

A relation in first normal form is a relation in which each component of each tuple is nondecomposable, i.e., the component is not a list or relation. Relations in first normal form are sometimes called “flat tables.” Clearly, the organization in Figure 2 does not represent a first normal form relation. Treating the items as potential attributes we note the following. The attribute pair \(<author, author_address>\) will be repeated for papers with multiple authorship, and

\[
\begin{array}{ll}
\text{title} & \} \ast \\
\text{author} & \\
\text{author_address} & \\
\text{date_of_publication} & \\
\text{publication_name} & \\
\text{publisher} & \\
\text{volume} & \\
\text{number} & \\
\text{pages} & \\
\text{keyword} & \ast \\
\text{abstract} & \ast \rightarrow \text{repeating group}
\end{array}
\]

FIG. 2. Information kept for each document in a bibliographic database.
<table>
<thead>
<tr>
<th>Relation</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCUMENTS</td>
<td>&lt;DOC#, TITLE, DATE, PUBLICATION, PAGES, ABSTRACT&gt;</td>
</tr>
<tr>
<td>AUTHORS</td>
<td>&lt;DOC#, NAME, ADDRESS&gt;</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>&lt;DOC#, KEY&gt;</td>
</tr>
</tbody>
</table>

FIG. 3. Bibliographic database in first normal form.

From our example in Figure 3, we see that while DOCUMENTS (with a single attribute key) and KEYWORDS (being "all key") are in second normal form, AUTHORS is not. <DOC#, NAME> is a candidate key and ADDRESS is a nonprime attribute. However, ADDRESS is functionally dependent on the attribute NAME by itself, not on the whole of the candidate key. To obtain second normal form, we split this relation into two relations. Our data organized as relations in second normal form is shown in Figure 4.

**Anomalies**

The concern for normalization arises as a result of undesired side effects or anomalies which occur in a dynamic database environment. The three anomalies are illustrated from the first and second normal form relations in Figures 3 and 4. The seriousness of the anomalies might better be shown on other databases, but our database is sufficient to show the types of effects that can occur.

1. **Update anomalies**: note Figure 4, second normal form.

Suppose that publication "abc" is bought out by a new publisher. Since "abc" may appear in many tuples in the relation DOCUMENTS, the publisher may be represented many times. All tuples in DOCUMENTS with a value of "abc" for attribute PUBLICATION must be searched out and their publisher values updated.

2. **Insertion anomalies**: note Figure 3, first normal form.

Suppose that we obtain the address of a researcher of whose work we have just learned. If we have no single document for this person, this information cannot be entered in the database. The information reasonably belongs in the relation AUTHORS, but we cannot put it there without generating a fictitious DOC # (null values not being permitted in keys). Notice that this anomaly is removed by the further normalization to second normal form (see Fig. 4, DIRECTORY relation).

3. **Deletion anomalies**: note Figure 4, second normal form.

Suppose that we delete all documents as their publication date falls beyond a certain time in the past. At some point we may delete the only remaining document in our database which was from a particular PUBLICATION. At that point we also lose the further information of the name of the publisher of that publication. It may be that that information is not needed in our database, but the point remains that it was lost as a side effect of another (intended) deletion.

Consideration of these anomalies leads to a very simple principle which is fundamental to good database design: do not represent more than one "concept" or "entity" in a single relation. Thus in our first normal form relation AUTHORS (Fig. 3) we had two quite distinct notions: who is the writer of a particular document and what is a particular author's address. These concepts were separated in the second normal form relations AUTHORS and DIRECTORY (Fig. 4). Likewise, in our second normal form relation DOCUMENTS we include the extraneous notion of a publisher's name with other information which is largely citation related. This problem can be corrected by use of third normal form relations. However, it is important to recognize that a careful initial design which insures that each relation describe a single "concept" will be in third normal form. Thus, marching through the steps of first and second normal forms is not a necessary requirement of the design process.

**Third Normal Form**

Intuitively, the third normal form definition simply expressed the idea, stated previously, that each relation should describe a single concept or entity. Stated more formally:

A relation R is in third normal form if it is in first normal form and, for every attribute collection C of R, if any attribute not in C is functionally dependent on C, then all attributes in R are functionally dependent on C. [7]

Considering the DOCUMENTS relation in Figure 4 in terms of the above definition, note that for the (single) attribute collection <PUBLICATION> of DOCUMENTS, the attribute <PUBLISHER> is functionally dependent on <PUBLICATION>, but all other attributes in DOCUMENTS are not functionally dependent on <PUBLICATION>. Therefore the relational organization shown in Figure 4 is in second normal form but not in third normal form.

The relations shown in Figure 5 are in third normal form. In addition to the required removal of the publisher's address from the DOCUMENTS relation we have chosen to divide that relation further, separating the abstract from what is essentially citation information and naming the resulting relations appropriately.

Determine whether a database is in third normal form depends on knowledge of the functional dependencies among the attributes of the data. This knowledge cannot be

<table>
<thead>
<tr>
<th>Relation</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCUMENTS</td>
<td>&lt;DOC#, TITLE, DATE, PUBLICATION, PAGES, ABSTRACT&gt;</td>
</tr>
<tr>
<td>AUTHORS</td>
<td>&lt;DOC#, NAME, ADDRESS&gt;</td>
</tr>
<tr>
<td>DIRECTORY</td>
<td>&lt;NAME, ADDRESS&gt;</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>&lt;DOC#, KEY&gt;</td>
</tr>
</tbody>
</table>

FIG. 4. Bibliographic database in second normal form.
discovered automatically by a system, but must be furnished by a database designer who understands the semantics of the information. Finally, note that there is not a unique third normal form representation for a given database.

### Fourth Normal Form

We will introduce and illustrate this notion only briefly since our proposed relational organization (Fig. 5) is already in fourth normal form. Third normal form relations may still have anomalies, and some of these may be removed in fourth normal form. The relation WRITERS <AUTHOR, TITLE, ADDRESS, DATE> in Figure 6 specifies the publications and location history (where he lived beginning in what year) of each author. This relation has no functional dependencies; it is "all key," and it is in third normal form. However, there are multivalued dependencies [8] in WRITERS. These are intuitively clear: an author "has" a set of TITLES just as he "has" a current address (DIRECTORY relation in Fig. 5). In our example we have the multivalued dependencies (notation \( \rightarrow \rightarrow \)) AUTHOR \( \rightarrow \rightarrow \) ADDRESS. Note that it does not follow that AUTHOR \( \rightarrow \) ADDRESS.

Roughly speaking, we say that a relation is in fourth normal form if all dependencies (functional and multivalued) are the result of keys [8]. Since AUTHOR is not a key in our example, WRITERS is not a fourth normal form relation. We may decompose into two projections without loss of information by using our knowledge of multivalued dependencies. WRITERS can be decomposed into the relations AUTHORSHIP <AUTHOR, TITLE> and HOMES <AUTHOR, ADDRESS, DATE>. Probably the most significant thing for us to note at this point is that, to provide an appropriate example (Fig. 6) we had to include more than one concept in our relation. A thoughtful design would not include repeating the author's job history for every one of his titles.

### Extending the Bibliographic Database

Figure 5 showed a possible organization of our bibliographic data in third normal form. However, to provide the complete facilities that we expect in an information retrieval system, further relations are required. In the first place, we would like to be able to retrieve documents based on the terms used in titles and abstracts, not just on assigned keywords. To do this, we index the desired fields (say TITLE and ABSTRACT) to form the relation INDEX <TERM, DOC #, WEIGHT>, which specifies for each term the number of each document in which it occurs and the total number of occurrences within the document. Furthermore, we include the capability of examining or retrieving related terms through use of the relation THESAURUS <TERM 1, TERM 2> which specifies for a given word (TERM 1) one of its synonyms (TERM 2). Figure 7 summarizes the relations developed for the bibliographic database.

In an operational information retrieval system, further information than is shown in Figure 7 is required. This information, such as stop words, suffix lists, etc., should all be stored and manipulated relationally.

#### Size Considerations

While we are not considering implementation, it is of interest to note the total amount of information specified by our relational organization. In terms of the original data in Figure 2, the relations in Figure 7 show certain duplications as well as certain consolidations. To cite one example, PUBLICATION appears not only as an attribute in the CITATION relation, but also as the key attribute to the SOURCE relation. However, PUBLISHER now appears only once for every publication, rather than for every document from that publication. Clearly, if we have thousands of documents from only a dozen publications, we are saving considerably. Nevertheless, even extensive (apparent) duplication in the relational view need not result in actual duplication when the storage structures are implemented [9].

#### Relational Languages

One of the advantages of the relational model is that it is readily compatible with high-level languages. High-level query languages are easy to learn and use, making databases available to casual users. Additionally, high-level languages permit a system to optimize the execution of a given request. Finally, these languages are generally nonprocedural, permitting a unified treatment of data definition, manipulation, and control. We distinguish between a query language and a data sublanguage as follows. Query language refers to a stand-alone language in which an end user interacts directly with the system, where data sublanguage denotes a set of database operations intended to be embedded in a host programming language. The same set of operators may serve both as a data sublanguage and as a query language.

Several classes of languages will be considered [5]: relational calculus, relational algebra, mapping-oriented lan-

---

**FIG. 5. Bibliographic database in third normal form.**

**FIG. 6. Relation in third normal form but not in fourth normal form.**
TABLE 1. Relational calculus.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Relation name</td>
</tr>
<tr>
<td>a</td>
<td>attribute name</td>
</tr>
<tr>
<td>X(R₁,a₁,R₂,a₂,...)</td>
<td>A relation named X composed of the attributes R₁,a₁,R₂,a₂,...</td>
</tr>
<tr>
<td>∃</td>
<td>Existential quantifier: &quot;there exists&quot;</td>
</tr>
<tr>
<td>∀</td>
<td>Universal quantifier: &quot;for all&quot;</td>
</tr>
<tr>
<td>∧, ∨, ¬</td>
<td>The standard logical operators AND, OR, NOT to form qualifiers.</td>
</tr>
<tr>
<td>=, ≠, &lt;, ≤, &gt;, ≥</td>
<td>The standard operators equal, not equal, less than, less than or equal, greater than, greater than or equal.</td>
</tr>
</tbody>
</table>

which specifies as its result a relation X with attribute DOC#. It includes all values such that there is a tuple in INDEX with such a DOC# value and whose TERM attribute has the value "pollution." Thus, the left side of the colon states what is to be retrieved from the database and the expression to the right of the colon is the predicate or qualifier. In the calculus, it is what you want to obtain that is specified; details of how it is to be obtained are not indicated.

Relational Algebra

The relational algebra [10] is a collection of operators that deal with whole relations, yielding new relations as a result. The major operators are presented in Table 2 in a form which is simplified but sufficient for our purposes. The complete algebra, as summarized recently by Codd [13], includes several different JOIN operators as well as extensions which handle null values. Use of the RESTRICT, PROJECT, and JOIN operators is illustrated and described in Table 3, using relations from Figure 7. Note that we avoid a particularly rigorous syntax and insert English particles for readability.

Mapping-Oriented Languages

Mapping-oriented languages offer power equivalent to that of the relational calculus or algebra while avoiding certain mathematical concepts. Thus, these are largely directed at the nonprogramming professional. We illustrate this class of languages by SEQUEL [14], a nonprocedural structured language with English keywords.

Following the notion of "mapping," a known attribute or set of attributes is mapped into a desired attribute or set of attributes by means of some relation. The general form of a simple SEQUEL query is:

```
SELECT desired-attributes
FROM relation-name
WHERE known-condition-on-attributes
```

For example, to obtain a list of titles of all papers in our database which were published in TODS we write:

```
SELECT TITLE
FROM CITATION
WHERE PUBLICATION = 'TODS'
```

The use of SEQUEL for retrieval of bibliographic information has been explored by Macleod [15], who proposes extensions to the language for document retrieval.

Graphics-Oriented Languages

In graphics-oriented languages, a user states his query by making choices or filling in blanks on a graphic display. In one such language, Query by Example [16], the user is presented with a blank relation on his display. He fills in one or more rows with an example of the desired result, directly specifying known values, and representing unknown values by underscoring arbitrarily chosen example values. A query may span more than one relation.

TABLE 2. Relational algebra.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET OPERATORS</td>
<td>Union, Intersection, Set Difference, Cartesian Product</td>
</tr>
<tr>
<td>RESTRICT</td>
<td>Sometimes called SELECT. Selects only those tuples of a relation which satisfy a given condition. We permit arbitrary logical expressions based on the attributes of the tuple and constants.</td>
</tr>
<tr>
<td>PROJECT</td>
<td>Returns the specified columns of a given relation; duplicates eliminated.</td>
</tr>
<tr>
<td>JOIN</td>
<td>Given two relations, say A and B, returns a new relation formed by concatenating a tuple of A with a tuple of B whenever a given relation holds between them.</td>
</tr>
<tr>
<td>DIVISION</td>
<td>This is the algebraic counterpart of the universal quantifier (∀). It may be expressed in terms of the other algebraic operators.</td>
</tr>
</tbody>
</table>
TABLE 3. Examples of relational algebraic operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTRICT AUTHORS by DOC # = 827</td>
<td>produces a relation of the same form as AUTHORS, but restricted to only those tuples with a DOC # value of 827.</td>
</tr>
<tr>
<td>PROJECT DIRECTORY on ADDRESS</td>
<td>produces a relation with the single attribute ADDRESS. Since duplicates are removed, this provides a list of unique author addresses.</td>
</tr>
<tr>
<td>JOIN CITATION, AUTHORS by DOC # (precisely by CITATION.DOC # = AUTHORS.DOC #)</td>
<td>produces a relation with all the attributes of CITATION plus the attributes of AUTHORS. Since we specified JOIN by DOC #, we are combining the information for a given document from two different relations. Note however that papers with multiple authorship are concatenated with each matching author, forming distinct tuples.</td>
</tr>
</tbody>
</table>

Additional Language Considerations

Because of the naturalness of viewing bibliographic data relationally, it is possible to use existing bibliographic query languages, requiring the system to translate the queries into the equivalent relational language statements. As explained by Macleod [15], whereas in many existing languages we require different commands to operate on different data structures, in the relational languages we use a consistent set of operators along with the explicit naming of structures.

Information Retrieval

Examples of bibliographic retrieval are shown in the relational calculus, the relational algebra, and SEQUEL. These examples are based on the relational organization shown in Figure 7 and illustrated in Figure 8. Since the idea is to illustrate the concepts involved rather than to teach the relational languages explicitly, rigorous syntax is not adhered to. Rather, sufficient explanation is given in notes following each example to overcome any problems with the notation.

Simple Retrieval

Here we specify the operation of looking up a value in a relation and listing some or all of the associated attribute names.

**Query 1a.** Find the numbers of all documents indexed by 'interface'.

**Calculus**

\[ X(INDEX.DOC #) : INDEX.TERM = 'interface' \]

**Algebra**

RESTRICT INDEX to TERM = 'interface'

PROJECT ON DOC #

**SEQUEL**

SELECT DOC #

FROM INDEX

WHERE TERM = 'interface'

Notes

In the calculus, from the relation INDEX we want the value of the DOC # attribute for every tuple in which the value of the TERM attribute is "interface." We need only qualify an attribute (i.e., specify its relation name as well) if it is ambiguous; however, we use qualified names here to ensure clarity. In the algebra, when we have a sequence of operations we will assume that each operation acts on the relation resulting from the previous operation, unless otherwise specified. Thus, here we mean "PROJECT the result on DOC #."

The capability illustrated here may be used in accessing any of the relations in the database. Thus, each of the following questions would be structured in the relational languages in the same way as Query 1a.

**Query 1b.** Find the address of the author whose name is John Smith.

**Query 1c.** Specify the Publication and Volume in which document 472 appeared.

**Query 1d.** Find the terms in the thesaurus that are related to 'interface'.

**Query 1e.** List the publications that are published by Wiley.

Simple Retrieval—Multiple Relations

It is often the case that the information we seek is in a relation other than the one on which the specified condition must be examined. Consider the following query which is similar to Query 1a.

**Query 2a.** Find the Titles of all documents indexed by 'interface'.

**Calculus**

\[ X(CITATION.TITLE) : \exists INDEX (CITATION.DOC # = INDEX.DOC # \land INDEX.TERM = 'interface') \]

**Algebra**

RESTRICT INDEX to TERM = 'interface'

JOIN with CITATION by DOC #

PROJECT on TITLE

**SEQUEL**

SELECT TITLE

FROM CITATION, INDEX

WHERE CITATION.DOC # = INDEX.DOC #

AND INDEX.TERM = 'interface'

FIG. 7. Extended bibliographic database (in fourth normal form).
### ABSTRACTS

#### DOC #
1423

#### ABSTRACTS
The essential concepts of the relational data model are defined, and normalization, relational languages based on the model, as well as advantages and implementations of relational systems are discussed.

#### DOC #
1596

#### ABSTRACTS
A multiuser database system...
The importance of knowing which relations hold which information can be seen. The calculus query states: obtain a list of TITLES from the relation CITATION which are such that there exists an INDEX tuple whose DOC# value is the same as that in the CITATION tuple and whose TERM value is pollution. In the algebra we begin with the relation which has the information we need to check and work our way through to the desired result relation. In SEQUEL, the FROM clause in this case specifies a JOIN of the two given relations; this join to be based on the criterion in the WHERE clause. In an earlier version of SEQUEL we would have written:

```sql
SELECT TITLE
FROM CITATION
WHERE DOC# = SELECT DOC#
FROM INDEX
WHERE TERM = 'interface'
```

For our purposes this form is just as clear, but it has disadvantages as described in ref. 17.

We may extend this type of query to any number of relations; for example:

**Query 2b.** List all authors who wrote papers which appeared in journals published by IEEE.

To respond to this query, from the SOURCE relation we determine all publications whose publisher is IEEE. Then from the CITATION relation we determine the numbers of all documents in these publications. Finally, from the AUTHORS relation we determine the names of the authors of the documents with these numbers.

**Calculus**

\[
X (AUTHORS.NAME) : \exists \text{SOURCE} \left( \text{SOURCE.PUBLISHER} = 'IEEE' \land \exists \text{CITATION} \left( \text{SOURCE.PUBLICATION} = \text{CITATION.PUBLICATION} \land \text{CITATION.DOC#} = \text{AUTHORS.DOC#} \right) \right)
\]

OR

\[
X (AUTHORS.NAME) : \exists \text{CITATION(\exists \text{SOURCE} \left( \text{SOURCE.PUBLISHER} = 'IEEE' \land \text{SOURCE.PUBLICATION} = \text{CITATION.PUBLICATION} \land \text{CITATION.DOC#} = \text{AUTHORS.DOC#}) \right))
\]

**Algebra**

```sql
RESTRICT SOURCE to PUBLISHER = 'IEEE'
JOIN with CITATION by PUBLICATION
JOIN with AUTHORS by DOC#
PROJECT on NAME
```

OR

```sql
RESTRICT SOURCE to PUBLISHER = 'IEEE'
PROJECT on PUBLICATION
```

**SEQUEL**

```sql
SELECT NAME
FROM AUTHORS
WHERE DOC# = SELECT DOC#
FROM CITATION
WHERE PUBLICATION = SELECT PUBLICATION
FROM SOURCE
WHERE PUBLISHER = 'IEEE'
```

**Notes**

In each case two examples shown, all producing an equivalent result. In the calculus, the second case shows the quantifiers and their variables shifted all the way to the left in the predicate. This is prenex normal form. For a particular NAME 'abc' to be accepted there must be tuples such that the following is true:

\[
\text{AUTHORS} \text{ CITATION} \text{ SOURCE}
\]

Thus there are two quantifiers used; there must exist an appropriate tuple in CITATION and there must exist an appropriate tuple in SOURCE for the predicate to be true for a particular AUTHORS tuple. In the algebra, the first version is clear and concise, but a lot of unrequired information is retained until the final PROJECT. For example, after the JOIN with CITATION we are interested only in the resulting DOC# values. This is reflected in the second version in which we introduce sufficient PROJECT operations to maintain only required information. In SEQUEL, the two versions shown are similar to those for Query 2a. In the first case, with three relations specified in the FROM clause, two joins are necessary, and the conditions for these are specified in the two AND clauses. It may be noted that the first SEQUEL version roughly follows the calculus whereas the second (read backwards) is roughly similar to the algebra.

**Qualified Retrieval**

Here we are interested in selecting information which must satisfy several criteria. In the first instance, the specified conditions are all on the same relation.

**Query 3a.** Find the Titles of all documents published in TODS in 1977 and 1978.
Calculus

\[ X(\text{CITATION.TITLE}) : \text{CITATION.PUBLICATION} = 'TODS' \land (\text{CITATION.DATE} = 1977 \lor \text{CITATION.DATE} = 1978) \]

Algebra

RESTRICT CITATION for PUBLICATION = 'TODS' \land (DATE = 1977 \lor DATE = 1978)

SEQUEL

SELECT TITLE
FROM CITATION
WHERE PUBLICATION = 'TODS'
AND (DATE = 1977 OR DATE = 1978)

We may have queries of this form which specify conditions involving more than one relation.


Calculus

\[ X(\text{CITATION.TITLE}) : \exists \text{AUTHORS} (\text{AUTHORS.DOC} = \text{CITATION.DOC} \land \text{AUTHORS.NAME} = 'MACLEOD, I.A.') \]

Algebra

JOIN AUTHORS with CITATION by DOC#
RESTRICT for DATE > 1977 AND NAME = 'MACLEOD, I.A.'

OR

RESTRICT AUTHORS for NAME = 'MACLEOD, I.A.'
PROJECT over DOC#
JOIN with CITATION
RESTRICT for DATE > 1977
PROJECT on TITLE

SEQUEL

SELECT TITLE
FROM CITATION, AUTHORS
WHERE CITATION.DOC = AUTHORS.DOC
AND CITATION.DATE > 1977
AND AUTHORS.NAME = 'MACLEOD, I.A.'

Notes

In the calculus, the existential quantifier \( \exists \) is again used. The query says: obtain a list of values for the relation CITATION which are such that (the DATE value is > 1977 and) there exists an AUTHORS tuple whose DOC# value is the same as that in the CITATION tuple and whose NAME value equals 'MACLEOD, I.A.' In the algebra, a query may usually be expressed in more than one way. In the first example the query is concise, but at some cost. Were this to be executed, the JOIN (the first operation) would produce an immense result (the whole database!). Thus, in the second example restrictions are introduced as early as possible to keep the result only as large as necessary at each step.

Retrieval Using a Thesaurus

We may examine the thesaurus by means of a simple query (e.g., Query 1d). However, we may wish to retrieve through the thesaurus without specifying all the related terms in our query.

Query 4. List the numbers for all documents indexed by terms related to 'interface'.

Calculus

\[ X(\text{INDEX.DOC#}) : \exists \text{ THESAURUS} (\text{THESAURUS.TERM1} = 'interface') \]

Algebra

RESTRICT THESAURUS by TERM1 = 'interface'
JOIN with INDEX by TERM2 = INDEX.TERM
PROJECT on DOC#

SEQUEL

SELECT DOC#
FROM INDEX, THESAURUS
WHERE INDEX.TERM = THESAURUS.TERM2
AND THESAURUS.TERM1 = 'interface'

The form of these queries is identical to those for Query 2a. It should become apparent that the effort involved in using these relational languages is in being aware of where information is in the relational database, rather than knowing a specialized syntax for handling special cases.

Boolean Retrieval

We have seen the use of the AND and OR operators already in expressions to qualify retrieval (e.g., Query 3a). Here, by Boolean retrieval we mean the use of the usual Boolean operators specifically with index terms. It is important to recognize the difference between this and our previous examples. Here we are specifying a condition which requires more than one tuple from a single relation to produce each resulting tuple. Thus, if we want to list the numbers of all documents indexed by both 'interface' and 'language' a given DOC# which satisfies that condition must have an entry in the INDEX relation with a TERM value of 'interface' and another entry with a TERM value of 'language'. Conditions specified thus far have been on a single tuple in any one relation and to attempt to use the same notation (getting something like: TERM = 'interface' AND TERM = 'language') is clearly unsatisfactory. However, by treating the query terms as tuples in a query relation, we may express the desired query quite simply in the relational calculus and algebra, at least in the case of the Boolean AND.

Query 5. Find the document numbers for all documents indexed by 'interface', 'language', and 'high-level'.

Language

Relation Q5

<table>
<thead>
<tr>
<th>TERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface</td>
</tr>
<tr>
<td>language</td>
</tr>
<tr>
<td>high-level</td>
</tr>
</tbody>
</table>

Calculus

\[ X(\text{CITATION.DOCS}) : \forall Q5 \exists INDEX(\text{INDEX.TERM} = Q5.\text{TERM} \land \text{INDEX.DOCS} = \text{CITATION.DOCS}) \]

Algebra

DIVIDE INDEX by Q5 over TERM
PROJECT on DOC #

SEQUEL

SELECT DOC #
FROM INDEX
WHERE TERM = 'interface'
INTERSECT
SELECT DOC #
FROM INDEX
WHERE TERM = 'language'
INTERSECT
SELECT DOC #
FROM INDEX
WHERE TERM = 'high-level'

Notes

In the calculus, the query says: obtain a list of DOC # values which are such that for all tuples in Q5 there exists a tuple in INDEX with that DOC # value and with TERM value matching Q5.TERM. In the algebra, the DIVIDE operation mentioned earlier is explained by this example. The result of the DIVIDE is those sets of tuples in INDEX which have, for a given DOC # value, a tuple with a TERM value matching each entry in Q5. In SEQUEL, the expression of the query has become awkward. We have had to enter essentially three distinct queries and INTERSECT the results. Clearly we do not want to write that much to express a relatively simple idea. Macleod [15] proposes a macrocapability to permit simplification of queries of this form.

Weighted Retrieval

The use of statistical or weighted retrieval may occur, for example, in systems permitting natural language queries. Here natural language processing techniques are used to determine those terms in a query which are useful for retrieval purposes. Then the correlation between the set of query terms and the set of terms identifying a document is determined [18]. Documents are ranked according to their correlation with the query.

In the relational model, we face difficulty in determining the correlations because of the necessity of performing computations based on several values from more than one relation. Thus, we assume the existence of a function, CORR, appropriate to the particular query language, which provides the correlation between a query and a given document, based on the weights of the terms. We assume the use of some indexing method which isolates the important terms in the query.

Query 6. List the numbers of documents correlating higher than 0.60 with 'high-level language interface'.

Relation Q6

<table>
<thead>
<tr>
<th>TERM</th>
<th>QWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>high-level</td>
<td>1</td>
</tr>
<tr>
<td>language</td>
<td>1</td>
</tr>
<tr>
<td>interface</td>
<td>1</td>
</tr>
</tbody>
</table>

Calculus

\[ X(\text{CITATION.DOCS}) : \text{CORR}(Q6,INDEX) > 0.60 \]

Algebra

JOIN Q6 with INDEX by TERM
CORR ( )

SEQUEL

SELECT DOC #
FROM INDEX, Q6
WHERE INDEX.TERM = Q6.TERM
GROUP BY DOC #
HAVING CORR( ) > 0.60

In each case we have utilized a function CORR to accomplish the required task. In neither the algebra nor the calculus was this defined as part of our language, and we will discuss this momentarily. However, it is useful to examine the SEQUEL query first.

The SEQUEL query is processed in the following order:

1. Based on the relations specified in the FROM clause, tuples are selected by the WHERE clause. This yields all tuples from INDEX having one of the query terms as TERM value (i.e., a JOIN of Q6 with INDEX).
2. Groups are formed by the GROUP BY clause. This gathers into one group all tuples which relate to a single document.
3. Groups are selected which satisfy the HAVING clause. This is determined by invocation of the CORR function which is applied to each group.

SEQUEL also permits the DOC # values to be ranked by correlation through use of a clause of the form: "ORDER BY CORR( )."

The problem with handling Query 6 in the calculus may be that we think of this problem procedurally. The query as stated is not procedural. However, as we attempt to handle the clause "correlating higher than 0.60" we begin to think procedurally; our target language does not have a nonprocedural equivalent of that clause. The need to extend the selective capability of the calculus is recognized. As Codd states [11], the basic relational calculus "represents a very basic selective power; which in most practical environments would need to be enhanced." For instance, ALPHA not only includes such library functions as COUNT, TOTAL, and MAX but is also designed to be embedded in a (procedural) host language. Thus, for our purposes, we write a calculus query which reads: list all document numbers such
that the CORR function, which uses the query and INDEX relations, is greater than 0.60.

The actual approach taken in the algebra is much simpler. Here, by means of a join we produce a relation of the form

\[ R < \text{DOC}#, \text{TERM}, \text{QWT}, \text{WEIGHT} > \]

which specifies, for each query term, those documents in which the term occurs and the appropriate weights. For a given document number in \( R \) we have 1, 2, or 3 tuples with that \( \text{DOC}# \) value, each with a different \( \text{TERM} \) value. Note that some difficulty is caused by the fact that a particular document may be of interest even though it is not indexed by all three query terms. Thus the required CORR function must take this into account.

We can take an alternative approach in the algebra. We produce a relation of the form

\[ S < \text{QWT}_1, \text{DWT}_1, \text{QWT}_2, \text{DWT}_2, \text{QWT}_3, \text{DWT}_3 >, \]

where \( \text{QWT}_i \) and \( \text{DWT}_i \) are the weights in the query and document, respectively, for the \( i \)th query term. Desired tuples are selected from \( S \) by means of

\[ \text{RESTRICT } S \text{ to } \text{CORR}(\cdot) > 0.60 \]

where \( \text{CORR}(\cdot) \) is calculated based on the value of weight in a single tuple. There are two drawbacks to this approach. First, producing the relation \( S \) is complex, requiring the use of such algebraic operators as OUTER THETA JOIN [13], because of the fact that certain \( \text{DWT}_i \) values may be NULL. Second, this idea does not extend well to queries with an arbitrary number of terms. What we really have is a repeating group of the form \( (\text{QWT}, \text{DWT}) \) and this is best resolved by use of the relation \( R \) shown above, despite the necessity of a more complex CORR function for that relation than for relation \( S \).

Feedback Searches

Feedback searches are those which involve the results of a previous search. For example, a user may examine the results of a search and decide that he wants "more documents like documents numbered 1423 and 657 but not like document numbered 1596." The system then forms a statistical query consisting of (optionally) the original query terms supplemented by the terms occurring in the documents indicated as being relevant, but not including terms occurring in the document judged not relevant. For our example, we will assume that Query 2a ("interface") has been entered and the results examined. A new query is to be formed based on the assessment of these results. Only the formation of the new query is shown. From that point the query would proceed as in a statistical search (Query 6).

Query 7. (Form a query to) Retrieve the numbers of documents which are like documents 1423 and 657 but not like document 1596.

**Calculus**

- RANGE INDEX \( S \)
- RANGE INDEX \( T \)
- \( Q7X (S.\text{TERM}) : (S.\text{DOC}# = 1423 \land S.\text{DOC}# = 657) \land \neg S.\text{TERM} \land (T.\text{TERM} = S.\text{TERM} \land T.\text{DOC}# = 1596) \)

**Algebra**

- RESTRICT INDEX to \( \text{DOC}# = 1423 \lor \text{DOC}# = 657 \)
- \( TPLUS \leftarrow \text{PROJECT ON TERM} \text{ RESTRICT INDEX to } \text{DOC}# = 1596 \)
- \( TMINUS \leftarrow \text{PROJECT ON TERM} \)
- \( Q7X \leftarrow \text{SET DIFFERENCE} TPLUS - TMINUS \)

**SEQUEL**

- ASSIGN \( Q7X (\text{TERM}) \)
- SELECT \( \text{TERM} \)
- FROM INDEX
- WHERE \( \text{DOC}# = 1423 \lor \text{DOC}# = 657 \) MINUS
- SELECT \( \text{TERM} \)
- FROM INDEX
- WHERE \( \text{DOC}# = 1596 \)

**Notes**

In the calculus we use notation from the language ALPHA to specify a range declaration. We need to specify a condition in our predicate based on more than one tuple from the relation INDEX. Thus, we state that \( S \) and \( T \) are to designate typical tuples from this relation. The conditions on different tuples are linked by means of a join term, in this instance \( T.\text{TERM} = S.\text{TERM} \). In the algebra we form the temporary relations \( TPLUS \), consisting of terms from the relevant documents, and \( TMINUS \), consisting of terms from the nonrelevant document. The desired query, \( Q7X \), is the set of terms which are in \( TPLUS \) but not in \( TMINUS \).

In SEQUEL, the approach is as in the algebra, the set difference operator in this case being MINUS. The use of ASSIGN indicates that the relation resulting from the following operations is to be named \( Q7X \).

In each case we may proceed with a statistical search using the query relation \( Q7X \). In these examples we dropped the original query which gave rise to the evaluation of the documents. In fact we may include the initial query term(s) if we wish. Also, we did not include weights for the terms. To do so requires more than just specifying this additional attribute; it requires additional sophistication in the query derivation. If, for example, a particular term occurs in several of the documents for which judgments are given, we need a composite weight based on the various weights assigned to the term. In general, we may wish our feedback query to be a more complex function of terms and weights [19], requiring far more than just the manipulation of sets of terms. However, our query is sufficient to illustrate the general approach of feedback.

**Obtaining Secondary Information**

All the examples thus far have had the retrieval of documents as a goal of the query, this being the predominant use of a bibliographic retrieval system. However, given the existence of an automated retrieval system, users find other possibilities for using these data. For example, in the area of bibliographics, a researcher may have reason to request
the number of authors from a particular institution, the average number of pages in articles published in JASIS, or other composite information. Tasks such as these are in a large part accomplished by means of auxiliary functions to perform counts, averages, etc.

Here we show two queries which do not require auxiliary functions, but illustrate additional features useful in obtaining secondary information.

Recall Query 2b which requested the names of authors whose works appeared in journals published by IEEE. Here we augment that query with the word "only" as follows:

Query 8a. List all authors whose works appeared only in journals published by IEEE.

Calculus

\[ X(DIRECTORY.NAME) : \forall AUTHORS (AUTHORS.NAME \neq DIRECTORY.NAME \land \exists CITATION (AUTHORS.DOC \# = CITATION.DOC \# \land \exists SOURCE (CITATION.PUBLICATION = SOURCE.PUBLICATION \land SOURCE.PUBLISHER = 'IEEE'))) \]

Algebra

RESTRICT SOURCE to PUBLISHER = 'IEEE'
PROJECT on PUBLICATION
JOIN with CITATION by PUBLICATION
PROJECT on DOC#
JOIN with AUTHORS by DOC#
T1 ← PROJECT on NAME, DOC#
T2 ← PROJECT on NAME
JOIN with AUTHORS on NAME
T3 ← PROJECT on NAME, DOC#
SET DIFFERENCE T3 - T1
T4 ← PROJECT on NAME
SET DIFFERENCE T2 - T4

SEQUEL

Create a relation with all NAME, DOC# pairs that involve IEEE publications.

\[ IEEE\_A\_D (NAME,DOC\#) \]
SELECT NAME, DOC#
FROM AUTHORS
WHERE DOC# =
SELECT DOC#
FROM CITATION
WHERE PUBLISHER =
SELECT PUBLISHER
FROM SOURCE
WHERE PUBLICATION = 'IEEE'

Delete from the list of all IEEE authors those with other publications.

SELECT NAME
FROM IEEE\_A\_D
MINUS
SELECT NAME
FROM AUTHORS

WHERE NAME in IEEE\_A\_D
AND DOC# NOT in IEEE\_A\_D

Notes

In the calculus we consider each entry in the DIRECTORY relation, this relation providing a convenient list of author’s names. The predicate says: for all AUTHORS tuples, either the name does not match the one now being considered OR the associated DOC# is published by IEEE. This latter fact is determined by the existence of appropriate tuples in the CITATION and SOURCE relations to demonstrate the fact. In the algebra solution, the first five lines are as for Query 2b. Relation T1 has all IEEE authors and their IEEE document numbers. T2 has all IEEE authors (as Query 2b). T3 contains all the IEEE authors and the DOC# values for all the documents they wrote (IEEE or otherwise). The DIFFERENCE between T3 and T1 gives all IEEE authors with their non-IEEE documents, and T4 is a list of these authors. Thus the DIFFERENCE of T2 (all IEEE authors) and T4 (IEEE authors with non-IEEE publications as well) is the desired result (the authors who are “purely” IEEE). In SEQUEL we proceed through two steps. First, we form a temporary relation IEEE\_A\_D. This is formed identically to Query 2b except that the DOC# value is retained as well as author NAME. Second, we take all IEEE authors MINUS those IEEE authors with additional publication. This latter list is determined from the relation AUTHORS by locating entries for which the author did publish in IEEE but for which the particular DOC# value is not for an IEEE publication.

Another area in which we may wish to examine the database relates to indexing and keyword selection. For example, we may wish to check the relationship between keyword selection and the frequency (weight) of terms in the text.

Query 8b. For document 1234 list all index terms which are keywords, along with their weight.

Calculus

\[ X(INDEX.TERM,INDEX.WEIGHT) : DOC\# = 1234 \land \exists KEYWORDS(KEYWORDS.KEY = INDEX.TERM \land KEYWORDS.DOC\# = INDEX.DOC\#) \]

Algebra

RESTRICT INDEX to DOC\# = 1234
JOIN with KEYWORDS by <DOC\#, TERM>
PROJECT on TERM, WEIGHT

SEQUEL

SELECT (TERM, WEIGHT)
FROM INDEX, KEYWORDS
WHERE DOC\# = 1234
AND INDEX.TERM = KEYWORDS.KEY
AND INDEX.DOC\# = KEYWORDS.DOC\#

Notes

What is of interest here is illustrated most clearly in the
algebra. There the condition of the JOIN requires that two attributes match. This same requirement is stated in the compound conditions in the calculus and in SEQUEL. It should also be noted that the relation KEYWORDS is only required to determine the existence of the particular <DOC #, KEY > tuple.

Retrieval Summary

Many more examples to illustrate the relational languages could be given. However, most needs in information retrieval can be met by the few general forms of query already shown, this being one of the advantages of the relational approach. One of the apparent difficulties of the approach is considered below.

In general, for bibliographic retrieval, several special-purpose auxiliary functions would be required in the relational language used. This was seen in the use of CORR in Query 6. Another area in which such a function may be necessary bears more detailed discussion.

In a previous section we discussed the problem of multiple authorship of documents insofar as the relational organization was concerned. However, we have not solved the multiple-authorship problem completely. Note that, even though a document has authors in the same way it has a title, we cannot treat them in the same way; the title is in a single tuple whereas the authors’ names are dispersed over arbitrarily many tuples for a given document. In particular, consider the operation of listing the citation information, including authors’ names, for a given document. We are essentially asking to display an unnormalized relation, that is, a relation with a repeating group. Codd [20] presents an operation called factoring, which converts a normalized relation to unnormalized form for presentation purposes. Such an operation would need to be easily, even automatically, invoked for an attribute as common as the author’s name in an information retrieval system.

In concluding this section it must be emphasized that while comparison of the three languages was unavoidable in the examples, such comparison was not with a view to choosing one as the best. For such a choice to be made, at the very least a more intuitive language based on the calculus would need to be shown rather than one using mathematical notation. The purpose of the examples is to give a feeling for the style, consistency, and scope of the relational languages in an information retrieval context.

Conclusions

The advantages of the relational model for information retrieval may be summarized as follows. First, the simplicity of the relational model is compelling. The user is able to view all aspects of the system in a clear, simple, coherent way. Second, there is a strong theoretical basis for the relational model. It is based on the mathematical theory of relations and on the first-order predicate calculus. Third, there is consistency of access to all information from the user point of view. Under other organizations, certain questions may be awkward to ask or require special language constructs. Fourth, the relational model permits a high degree of data independence. It is possible to eliminate the details of data storage and access methods from the user interface.

A further advantage relates particularly to the relational model as it applies to document retrieval. It is natural to view such structures as document collections, indexes, and dictionaries as two dimensional and therefore as relations.

The question of implementation of the relational model is currently receiving a lot of attention. These efforts include defining of underlying data structures to implement relations efficiently, optimizing the execution of relational systems, and designing of associative hardware to support relational databases. A few large-scale prototype relational database management systems have been implemented. Results to date indicate the present suitability of the relational model for bibliographic databases of small to medium size. It is expected that future developments will result in ever wider applicability of the model.

References

Study of Engineering Information Needs

The Committee on Engineering Information of the World Federation of Engineering Organizations /WFEO/CEI/ has prepared, under contract with Unesco, a study on “Engineer’s Need for Scientific and Technical Information.” The study, which summarizes the long-standing experience of WFEO/CEI, aims at formulating a global and comprehensive approach to the multiple aspects of engineers’ information needs.

Copies of the study are available in limited number and can be obtained from

Division of the General Information Programme
Unesco
F, Place de Fontenoy
75 700 Paris, France