Information Processing & Management Vol. 24, No. 3, pp. 349-371, 1988
Printed in Great Britain.

0306-4573/88 $3.00 + .00

AN EXTENDED RELATIONAL DOCUMENT RETRIEVAL MODEL

DAVID C. BLAIR
Associate Professor, Computer and Information Systems,
The University of Michigan, Ann Arbor, MI 48109, U.S.A.

Abstract—Relational Data Base Management Systems offer a commercially available tool with which to build effective document retrieval systems. The full potential of the relational model for supporting the kind of *ad hoc* inquiry characteristic of document retrieval has only recently been explored. In addition, commercially available relational DBMS's also provide effective tools for managing document data bases by providing facilities for, *inter alia*, concurrency control, data migration and reorganization routines, authorization mechanisms, enforcement of integrity constraints, dynamic data definition, etc. This article will present a relational logical model to support a sophisticated document retrieval system in which flexible forms of inferential and associative searching can be performed. Examples of *ad hoc* inquiry will be presented in SQL. Several problems of particular importance to document retrieval will be discussed, including the importance of Conjunctive Normal Form in query formulation, unique aspects of document retrieval storage and processing overhead, and techniques for reducing the size of storage without severely impacting retrieval effectiveness.

1. INTRODUCTION

The flexibility of the relational logical model in database system design is well known, so the application of relational design structures to information retrieval is not surprising. A recent spate of papers applying relational logical structures and query languages to document, or information, retrieval systems [1-3] demonstrates the broad applicability of these techniques. These are not entirely new ideas. The first retrieval system to use a relational logical structure, the Relational Data File [4-6], was primarily a document retrieval system, and a 1974 paper showed the efficacy of the early relational data manipulation language SQUARE for querying document data bases [7].

The work to date in relational models of document retrieval has been largely preliminary and has modeled only relatively simple document retrieval situations. This article describes an extended relational model for document retrieval and will discuss some retrieval considerations of particular importance for document retrieval.

2. DOCUMENT RETRIEVAL: THE CENTRAL PROBLEM

One of the principal advantages which relational logical structures offer over the earlier logical structures, hierarchical and network (CODASYL), is the comparative ease with which an inquirer using a relational database can construct *ad hoc*—queries that are not routinely or repeatedly asked of the system. The capacity to be able to answer *ad hoc* inquiries easily is an advantage in database management systems, but not usually a necessity. For document retrieval, on the other hand, the ability to answer *ad hoc* inquiries is a necessity [8]. Document retrieval systems have been implemented using logical database structures and query languages which preceded relational designs [9], but such systems are comparatively difficult to use in *ad hoc* inquiry and, resultingly, will be difficult to use to implement all but the simplest logical models of document retrieval.

Ad hoc inquiry is important for document retrieval systems because of the tremendous variety in the way that people search for needed documents. Inquirers may want documents because they are authored by particular individuals, are published during a particular time frame, appear in one or several journals, concern a particular subject, are written by authors who are affiliated with certain institutions, are of a particular type (e.g.,

article, letter, report, conference proceedings, etc.), or any complex combination of such search categories. Document retrieval systems that do not permit this kind of search flexibility greatly reduce the chances of an inquirer's retrieving useful articles. But the formulation of a wide variety of *ad hoc* queries is not the only important capability of an advanced document retrieval system. Such a system should enable the inquirer to retrieve not just information about individual documents but also information about the aggregate of documents (meta-information, if you will). Information such as, for example, what subject headings are most frequently applied to documents authored by individuals working at institute X. This is not specifically a request for documents, but a request for important tacit information that may be derivable from the document descriptions in the database. The ability to retrieve such tacit information is an important capability of an advanced document retrieval system. The following discussion will show how such tacit information may be retrieved through the use of relational logical structures and Data Manipulation Languages.

3. BASIC RELATIONAL STRUCTURE

The basic (normalized) relational structure of the initial document retrieval facility would look like this (see Appendix D):

Relation name	Attributes
CITATION	DOCUMENT #*, TITLE, DOCUMENT TYPE, PUBLICATION DATE, JOURNAL NAME, VOLUME, NUMBER, PAGES, ACQUISITION DATE
ABSTRACT	DOCUMENT #*, ABSTRACT
AUTHOR	DOCUMENT #*, NAME
DIRECTORY	NAME* (author's name), INSTITUTION
INSTITUTION	INSTITUTION* (name), TYPE, ADDRESS, PHONE
JOURNAL	JOURNAL NAME*, PUBLISHER

Key attributes are indicated by an *.

Such a set of relations would represent a basic document retrieval schema, which could be enhanced in several ways. But before we look at the enhancements we should look at some of the inquiries which the basic logical structure can support. For readability, I will use the SQL query language [10] when describing how actual queries would be constructed, since SQL is relatively understandable even to those with no familiarity with relational query languages.

3.1 *Typical queries*

1. What are the titles of articles written by Raymond Larsen?

```
SELECT  TITLE
FROM    CITATION
WHERE   DOCUMENT # =
        SELECT  DOCUMENT #
        FROM    AUTHOR
        WHERE   NAME = 'Raymond Larsen'
```

2. Retrieve the abstracts of the articles written by Raymond Larsen.

```
SELECT  ABSTRACT
FROM    ABSTRACT
WHERE   DOCUMENT # =
        SELECT  DOCUMENT #
        FROM    AUTHOR
        WHERE   NAME = 'Raymond Larsen'
```

3. Where does Raymond Larsen work?

```
SELECT  INSTITUTION
FROM    DIRECTORY
WHERE   NAME = 'Raymond Larsen'
```

4. What is Raymond Larsen's address?

```
SELECT  ADDRESS
FROM    INSTITUTE
WHERE   INSTITUTION =
        SELECT  INSTITUTION
        FROM    DIRECTORY
        WHERE   NAME = 'Raymond Larsen'
```

5. Which authors of articles in our database are affiliated with the University of California?

```
SELECT  NAME
FROM    DIRECTORY
WHERE   INSTITUTION = 'University of California'
```

A frequent service provided by many information centers is to send individuals a list of the contents of specified journals or magazines. Such a service could be easily provided in the sample database model:

6. What are the titles of the articles appearing in April 1987 issue of *Communications of the ACM*?

```
SELECT  TITLE
FROM    CITATION
WHERE   JOURNAL NAME = 'Communications of the ACM'
AND     PUBLICATION DATE = 04/87
```

This request could be made by specifying the VOLUME and NUMBER of the journal if that information is more readily available than the date. Sometimes, though, the inquirer may not even know the date or volume number of the journal whose contents he or she wants. The inquirer may only want to know the contents of the most recent copy of the desired journal that exists on the database. Such a request can be accommodated by using an arithmetic function that exists in most major relational database management systems:

7. What are the titles of the articles appearing in the most recently published issue of *Communications of the ACM*?

```
SELECT  TITLE
FROM    CITATION
WHERE   JOURNAL NAME = 'Communications of the ACM'
AND     PUBLICATION DATE = MAX
```

Conjunctive queries such as the two above are known in the document retrieval vernacular as Boolean Queries. Such queries are quite frequent in document retrieval.

Another type of current contents request might be made by an inquirer who wants to see the titles of articles appearing in a specific journal (or journals) over a period of time.

Such an information need may occur when an individual is doing a literature search for certain types of articles, is catching up on his or her reading in a journal he or she has not seen in a while, or is looking for a specific article whose title he or she can recognize but whose date of publication he or she has forgotten:

8. Give me the titles of all articles published in *Communication of the ACM* since 1980.

```
SELECT   TITLE
FROM     CITATION
WHERE    JOURNAL NAME = 'Communications of the ACM'
AND      PUBLICATION DATE 12/31/79
ORDER BY PUBLICATION DATE DESCENDING
```

The final statement in the above request ensured that the titles will be in chronological order with the most recent first.

For individuals who want to be informed regularly about the content of journals which are added to the document database, a complex Boolean request could be kept on file for that individual that includes the names of all the journals (and any other document parameters) of which he or she wants to be informed. The request would merely be run at periodic intervals and the results forwarded to the requesting individual. To ensure that only the contents of journals added since the inquirer's last request are returned, the request should be made using the Acquisition Date rather than the Publication Date as a search parameter. For example:

9. Give me the titles of all the articles published in *Communications of the ACM* and *Computer Journal* since my last request (1/31/87).

```
SELECT   TITLE
FROM     CITATION
WHERE    JOURNAL NAME IN ('Communications of the ACM', 'Computer
         Journal')
AND      ACQUISITION DATE > 1/31/87
```

Searching by acquisition date has three advantages. (1) The inquirer is not overloaded with previously retrieved material. (2) The request can be processed very efficiently since only a small part of the database (those articles added since 1/31/87) needs to be accessed. (3) It will retrieve older journal articles that have only recently been added to the database (such as, in our example, a back issue of *Computer Journal*). This is not possible if retrieval is based on Publication Date.

4. THE EXTENDED RELATIONAL MODEL

By adding several relations to the schema described previously, we can greatly increase the kinds of queries that can be answered by the retrieval system. The most important addition to the database scheme at this point would be to introduce some kind of subject-access capability. This can be done through the following relation:

KEYWORDS: DOCUMENT #*, KEYWORD*, WEIGHT

This relation relates subject descriptions (keywords) to specific documents. It also includes a weight (usually between 0 and 1.0) which reflects how applicable a particular subject description is to a given document. For example, a document might be represented as follows:

```
#4357  Special Computers  0.5
#4357  Programming        0.7
#4357  File Organization  0.9
         .
         .
         .
```

This would indicate that document #4357 deals with programming and file organization on special computers, and that it is largely about file organization. With this kind of structure there is no limit to the number of keywords that could be assigned to a particular document, but if we look at existing systems we see that usually 6–10 keywords are typically used to describe the subject content of a document.

Now we can retrieve documents from the system based on subject specifications:

10. Retrieve the documents which concern occupational retraining and have an assigned subject weight of greater than 0.7.

```
SELECT  *
FROM    CITATION
WHERE   DOCUMENT # =
        SELECT  DOCUMENT #
        FROM    KEYWORDS
        WHERE   KEYWORD = 'occupational retraining'
        AND     WEIGHT 0.7
        ORDER BY WEIGHT DESCENDING
```

(The * in the SELECT statement indicates that the entire tuple is to be retrieved.)

It is very useful for a document retrieval system to be able to order retrieved citations according to some priority (such as above, where they are ordered in descending keyword weight). One of the most persistent problems in document retrieval is output overload—the condition where too many documents (i.e., document citations) are retrieved for the inquirer to browse through to find the documents he/she wants [11, 12]. In many modern computerized retrieval systems, a request like number 10 with a single keyword specification may retrieve thousands (or, even tens of thousands) of document citations. All of the retrieved citations will have had, by definition, the specified keyword assigned to them, but many of the citations refer to documents that are only marginally concerned with the subject indicated by the keyword. By ordering the retrieved citations according to keyword weight, the inquirer ensures that even where large numbers of citations are retrieved, the ones that more closely match the request are presented to him/her first. Consequently, even large retrieved sets of documents may not be an impediment to effective retrieval since the citations that are more likely to satisfy the inquirer will be ranked first. The ORDER BY AND GROUP BY commands in SQL give the inquirer wide latitude in prioritizing output.

Because of the sociological nature of research and industry, the name of an individual who works in a relevant area of research can be used as a clue to find other relevant information on the database, by using the KEYWORDS relation in combination with the other relations to retrieve relevant documents. For example:

11. Give me the titles of articles on expert systems that are published by individual who are affiliated with the same institute with which John Murphy is affiliated.

```
SELECT  TITLE
FROM    CITATION FIRST
WHERE   FIRST.DOCUMENT # = ANY
        SELECT  DOCUMENT #
        FROM    CITATION SECOND, KEYWORDS
```

```
WHERE   KEYWORDS.KEYWORD = 'expert systems'
AND     SECOND.CITATION NAME = ANY
        SELECT  NAME
        FROM    DIRECTORY FIRST
        WHERE   FIRST.INSTITUTION =
                SELECT  INSTITUTION
                FROM    DIRECTORY SECOND
                WHERE   SECOND.NAME = 'John
                          Murphy'
```

(The 'WHERE FIRST.DOCUMENT # = ANY command indicates that titles should be retrieved whose document numbers are any of the ones found by the following SELECT commands. The CITATION FIRST and CITATION SECOND specification is a SEQUEL convention that allows the results of one search of the CITATION relation to be used as arguments for a second search of the CITATION relation.)

Sometimes a subject search must be reversed for those inquirers who are familiar with the literature of the database but are not familiar with the exact subject descriptions being used (even a minor spelling error in keyword specification might lead to poor retrieval results). To get into the system an inquirer might retrieve the keywords that are used to index a document in which he or she is interested and that is already on the database. The inquirer would then use those retrieved keywords to formulate a conventional subject request to the system to retrieve other documents on the same subject.

12. Give me the keywords used to describe the document "Process control in shop-floor automation" by Molly Bloom.

```
SELECT  KEYWORD
FROM    KEYWORDS
WHERE   DOCUMENT # =
        SELECT  DOCUMENT #
        FROM    CITATION, AUTHOR
        WHERE   CITATION.DOCUMENT # =
                  AUTHOR.DOCUMENT #
        AND     CITATION.TITLE = 'Process control in shop-floor
                                    automation'
        AND     AUTHOR.NAME = 'Molly Bloom'
```

This search could be combined into one query as follows:

13.
```
SELECT  TITLE
FROM    CITATION
WHERE   DOCUMENT # =
        SELECT  FIRST.DOCUMENT #
        FROM    KEYWORDS FIRST
        WHERE   FIRST.WEIGHT > 0.5
        AND     FIRST.KEYWORD = ANY
                SELECT  SECOND.KEYWORD
                FROM    KEYWORDS SECOND
                WHERE   SECOND.DOCUMENT # =
                        SELECT  DOCUMENT #
                        FROM    CITATION, AUTHOR
                        WHERE   CITATION.DOCUMENT # =
                                  AUTHOR.DOC #
```

```
AND     CITATION.TITLE = 'Process
          control in shop-floor
          automation'
AND     AUTHOR.NAME = 'Molly
          Bloom'
```

The principal disadvantage of Query 13 is that the inquirer loses some of the query-formulation control that he or she would have if the search were conducted as a two-stage process (i.e., if the target document had many keywords assigned to it, the retrieved set of documents which have any of those keywords might be unmanageably large).

5. INFERENTIAL RETRIEVAL IN RELATIONAL DATABASES

A relational document retrieval system does not just contain documents, it also contains a great deal of valuable information of an inferential or tacit nature. For example, an inquirer may want to know what the major journal sources are in the field of flexible manufacturing so he or she will be certain to keep up to date on their articles:

14.
```
SELECT  UNIQUE JOURNAL NAME, COUNT(DOCUMENT #)
FROM    CITATION
WHERE   DOCUMENT # = ANY
        SELECT  DOCUMENT #
        FROM    KEYWORDS
        WHERE   KEYWORD = 'flexible manufacturing'
        ORDER BY COUNT(DOCUMENT #) DESCENDING
```

The command COUNT(DOCUMENT #) keeps a running total of the number of documents that have appeared in a given journal and have been assigned the subject description flexible manufacturing. The ORDER BY . . . command ensures that the output will consist of a rank ordering of journal titles arranged in descending order by how many articles on flexible manufacturing have appeared in them. Often, an inquirer can infer a lot about what kind of research may go on at a particular institution just by looking at the kinds of publications, memos, or reports that are produced by individuals affiliated with that institution. By tabulating the information in certain ways, some interesting relationships may be revealed. For example:

15. Rank the institutions by how many authors they have who publish in "flexible manufacturing."

```
SELECT  UNIQUE INSTITUTION, COUNT (UNIQUE NAME)
FROM    DIRECTORY
WHERE   NAME = ANY
        (SELECT  NAME
        FROM     AUTHOR
        WHERE    DOCUMENT # = ANY
                 SELECT  DOCUMENT #
                 FROM    KEYWORDS
                 WHERE   KEYWORD = 'flexible manufacturing')
                 ORDER BY COUNT(UNIQUE NAME)
                   DESCENDING
```

The results of the above search could be compared to the total number of authors at each institution to get an idea of the percentage of concentration that an institution has in "flexible manufacturing."

The subject terms that have been assigned to documents in the database can also be

used to derive a rough subject profile of the research at a particular institution by asking the following query:

16. List the different keywords which have been assigned to articles produced by individuals affiliated with the General Motors Institute, and count the number of documents to which each of these keywords have been assigned.

```
SELECT  UNIQUE KEYWORD, COUNT(DOCUMENT #)
FROM    KEYWORDS
WHERE   DOCUMENT # = ANY
        (SELECT  DOCUMENT #
        FROM     AUTHOR
        WHERE    NAME = ANY
                 SELECT  NAME
                 FROM    DIRECTORY
                 WHERE   INSTITUTION = 'General Motors Inst.')
                 ORDER BY COUNT (DOCUMENT #) DESCENDING
```

Occasionally, it may be important to use the information on the database to generate a list of institutions that might be interested in receiving information on a particular area. This might be done with the following query:

17. Get the names and addresses of all research groups who have at least one member who has published a recent (1986 or after) paper on "integrated manufacturing."

```
SELECT  INSTITUTION, ADDRESS
FROM    INSTITUTE
WHERE   INSTITUTION =
        SELECT  INSTITUTION
        FROM    DIRECTORY
        WHERE   NAME =
                SELECT  NAME
                FROM    CITATION
                WHERE   DATE > 12/31/85
                AND     NAME =
                        SELECT  NAME
                        FROM    AUTHOR
                        WHERE   DOCUMENT # =
                                SELECT  DOCUMENT #
                                FROM    KEYWORDS
                                WHERE   KEYWORD =
                                          'integrated
                                          manufacturing'
```

6. ASSOCIATIVE SEARCHING USING THE RELATIONAL MODEL

One of the most important facilities of a good document retrieval system is its associative searching capability. This permits the inquirer to discover semantic relationships between the subject index terms that have been assigned to documents on the database. One of the simplest and most useful statistics for inferring semantic relationships between subject terms (keywords) is the percentage of co-occurrence of assignment of these terms. This percentage expresses a probability that if keyword X is assigned to a particular document, then there is a calculatable probability that keyword Y will also be assigned to that document. This probability is merely the percentage of times that Y has been assigned to documents which have keyword X assigned. (Note that the probability of Y being assigned given the assignment of X is not the same as the probability of X being assigned given the

assignment of Y.) The primary use of associative searching is to semantically broaden an inquirer's subject search. For example, if an inquirer exhausts his or her search for documents with the keyword "flexible manufacturing," he or she can retrieve a list of co-occurring subject terms by using the following relation:

THESAURUS KEYWORD*, COOCCURRING TERM*, PERCENT

A typical query might be:

18. Retrieve the keywords which co-occur with the keyword Air Pollution that have a probability of co-occurrence greater than .040. Rank these terms by decreasing probability of co-occurrence.

```
SELECT   COOCCURRING TERM, PERCENT
FROM     THESAURUS
WHERE    KEYWORD = 'Air pollution'
AND      PERCENT > .040
ORDER BY PERCENT DESCENDING
```

The output of such a search might look something like [13]:

Keyword	Cooccuring term	Percent
Air Pollution	Dust	.479
	Waste Disposal	.384
	Water Supply	.231
	Quarrying	.132
	Noise	.132
	Poison	.126
	Environment	.101
	Pesticide	.089
	Occupational Safety	.081
	Gas Industry	.063
	Chemical Industry	.061
	Education	.057
	Natural Resources	.045

Such a list has two principal uses: (1) It can, as mentioned before, be used by inquirers who want to find semantically related keywords (the assumption being that keywords which have a high probability of co-occurring are semantically related). (2) It can be used by indexers to assist them in the indexing process (an indexer would only have to identify the keyword which identifies the main subject of the document to be indexed, and could then select the appropriate secondary subject categories from the list of keywords which co-occur with the principal keyword).

An inquirer could expand his or her search without consulting a co-occurrence list by entering the following formal query:

19. Retrieve all the documents that have any of the keywords that co-occur with "Flexible Manufacturing" at a percentage higher than .40.

```
SELECT   TITLE
FROM     CITATION
WHERE    DOCUMENT # = ANY
         SELECT   DOCUMENT #
         FROM     KEYWORDS
         WHERE    KEYWORD = ANY
                  SELECT   COOCCURRING TERM
                  FROM     THESAURUS
                  WHERE    KEYWORD = 'Flexible Manufacturing'
                  AND      PERCENT > .40
```

7. BOOLEAN SEARCHES

Keyword or subject searches comprise a substantial proportion (often a majority) of the searches that are conducted on a document retrieval system. Such searching is often Boolean in nature and can be accomplished on a relational database system by using the SQL facility of joining a table with itself:

20. Retrieve the titles of all the documents that have been indexed with *both* keywords "shop automation" and "integrated manufacturing."

```
SELECT  TITLE
FROM    CITATION
WHERE   DOCUMENT # = ANY
        SELECT  DOCUMENT#
        FROM    KEYWORDS FIRST, KEYWORDS SECOND
        WHERE   FIRST.KEYWORD = 'shop automation'
        AND     SECOND.KEYWORD = 'integrated manufacturing'
        AND     FIRST.DOCUMENT # = SECOND.DOCUMENT #
```

A simple disjunctive query could be handled as follows:

21. Retrieve the titles of all the documents that have been indexed with *either* of the keywords "shop automation" or "integrated manufacturing."

```
SELECT  TITLE
FROM    CITATION
WHERE   DOCUMENT = ANY
        SELECT  DOCUMENT #
        FROM    KEYWORDS
        WHERE   KEYWORDS IN
                ('shop automation', 'integrated manufacturing')
```

Subject searching is a nondeterministic process in which several topic alternatives often must be described in an inquirer's query. These alternatives are represented by complex conjunctive and disjunctive Boolean combinations of keywords. For example:

22. Retrieve the titles of all the documents that are indexed with either "shop automation," "computerization," or "automation," and either "integrated manufacturing" or "flexible manufacturing." (This query is the conjunction of two disjunction sets of three keywords and two keywords, respectively)

```
SELECT  TITLE
FROM    CITATION
WHERE   DOCUMENT # = ANY
        SELECT  DOCUMENT
        FROM    KEYWORDS FIRST, KEYWORDS SECOND
        WHERE   FIRST.DOCUMENT # = SECOND.DOCUMENT #
        AND     FIRST.KEYWORD IN ('shop automation', 'computer-
                                  ization', 'automation')
        AND     SECOND.KEYWORD IN ('integrated manufacturing',
                                   'flexibility manufacturing')
```

A general formulation is possible for constructing Boolean subject queries, providing a format for even the most complex keyword queries:

```
SELECT   FIRST.DOCUMENT #
FROM     KEYWORDS FIRST
         KEYWORDS SECOND
                  .
                  .
                  .
         KEYWORDS Nth
WHERE    FIRST.DOCUMENT # = SECOND.DOCUMENT #
AND      SECOND.DOCUMENT # = THIRD.DOCUMENT #
  .
  .
  .
AND      N-1.DOCUMENT # = Nth.DOCUMENT #
AND      FIRST.KEYWORD IN ('xxxx', . . . ,'xxxx')
AND      SECOND.KEYWORD IN ('xxxx', . . . ,'xxxx')
  .
  .
  .
AND      Nth.KEYWORD IN ('xxxx', . . . ,'xxxx')
```

The logical format of this kind of query can be represented in the propositional calculus as follows:

$$(K_{a_1} \vee K_{b_1} \vee \ldots \vee K_{n_1}) \cdot (K_{a_2} \vee K_{b_2} \vee \ldots \vee K_{n_2}) \cdot \ldots \cdot (K_{a_r} \vee K_{b_r} \vee \ldots \vee K_{n_r})$$

[where the symbols $\vee$ and $\cdot$ represent disjunction and conjunction, respectively, and K stands for a keyword.]

This particular logical pattern is, of course, conjunctive normal form, and although many Boolean expressions are not in conjunctive normal form, they all can be transformed into conjunctive normal form without loss of meaning or well-formedness. This means that any Boolean retrieval query can be represented in the above format. For example:

$$(K_a \cdot K_b) \vee K_c$$

which is not in conjunctive normal form, can be represented by the equivalent logical construct:

$$(K_a \vee K_c) \cdot (K_b \vee K_c)$$

Or, in another example, the Boolean query:

$$(K_a \cdot K_b) \vee (K_c \cdot K_d)$$

can be represented by the equivalent conjunctive normal form expression:

$$(K_a \vee K_c) \cdot (K_a \vee K_d) \cdot (K_b \vee K_c) \cdot (K_b \vee K_d)$$

Readers familiar with propositional logic will, no doubt, have observed that the expression $(K_a \cdot K_b) \vee (K_c \cdot K_d)$, while not in *conjunctive* normal form, is in *disjunctive* normal form. Since all Boolean expressions can be nonloss transformed into either conjunctive or disjunctive normal form, it appears that the recommendation to convert all complex Boolean SQL queries into conjunctive normal form is somewhat arbitrary. This is not the case. Conjunctive normal form expressions are more easily represented in SQL than disjunctive normal form expressions. The general SQL format for disjunctive normal form expressions looks like:

```
SELECT   UNIQUE FIRST.DOCUMENT
FROM     KEYWORDS FIRST
WHERE    FIRST.DOCUMENT # = ANY
         (SELECT   N1.DOCUMENT #
         FROM      KEYWORDS N1
                   KEYWORDS (N1 + 1)
                        o
                        o
                        o
                   KEYWORDS (N1 + M1)
         WHERE     N1.DOCUMENT # = (N1 + 1).DOCUMENT #
         AND       (N1 + 1).DOCUMENT # = (N1 + 2).DOCUMENT #
           o
           o
           o
         AND       (N1 + M1 - 1).DOCUMENT # =
                     (N1 + M1).DOCUMENT #
         AND       N1.KEYWORD = KA1
         AND       (N1 + 1).KEYWORD = KB1
           o
           o
           o
         AND       (N + M).KEYWORD = KN1
         OR
         SELECT    N2.DOCUMENT #
         FROM      KEYWORDS.N2
                   KEYWORDS.(N2 + 1)
                        o
                        o
                        o
                   KEYWORDS.(N2 + M2)
         WHERE     N2.DOCUMENT # = (N2 + 1).DOCUMENT #
         AND       (N2 + 1).DOCUMENT # = (N2 + 2).DOCUMENT #
           o
           o
           o
         AND       (N1 + M - 1).DOCUMENT # =
                     (N2 + M2).DOCUMENT #
         AND       (N2 + 1).KEYWORD = KA2
         AND       (N2 + 1).KEYWORD = KB2
           o
           o
           o
         AND       (N + M).KEYWORD = KN2
         OR
           o
           o
           o
         OR
         SELECT    Nn.DOCUMENT #
         FROM      KEYWORDS.Nn
                   KEYWORDS (Nn + 1)
                        o
                        o
                        o
         WHERE     Nn.DOCUMENT # = (Nn + 1).DOCUMENT #
```

```
AND     (N_n + 1).DOCUMENT # = (N_n + 2).DOCUMENT #
  o
  o
  o
AND     (N_n + M_n − 1).DOCUMENT # =
          (N_n + M_n).DOCUMENT #
AND     N_n.KEYWORD = K_An
AND     (N_1 + 1).KEYWORD K_Bn
  o
  o
  o
AND     (N_n + M_n).KEYWORD = K_Nn)
```

This is clearly a more complex query format than the one for conjunctive normal form (q.v.). The minor inconvenience of converting an expression from disjunctive normal form to conjunctive normal form should be more than offset by the comparative ease of transforming conjunctive normal form expressions into SQL (or any other relationally complete DMLs) commands.

From this discussion of SQL query formulation we can see that although SQL is a friendly language, some Boolean queries may be translated into SQL commands only with great difficulty. To facilitate query construction complex Boolean queries should be reduced to their simplest form before they are translated into SQL commands. For example, the laws of propositional logic enable us to reduce the Boolean expression:

$$(K_p \cdot K_q) \vee (K_q \cdot K_r) \vee (K_p \cdot K_r) \vee K_r$$

to

$$(K_p \cdot K_q) \vee K_r \text{ (see Appendix A).}$$

This, in turn, is translatable into the conjunctive normal form expression:

$$(K_p \vee K_r) \cdot (K_q \vee K_r).$$

This is a much easier expression to translate into SQL than the original one.

8. SUBSCHEMAS

Although the SQL query language is relatively easy to use compared to most Database Data Manipulation Languages, its syntax still may be a bit forbidding for the occasional or nonprogramming inquirer. Most relationally complete Database Management Systems (e.g., DB2, ORACLE, or INGRES) have the capability to construct virtual relations known as subschemas, which can be used to design a friendlier interface to the system (the recognition of the importance of subschemas in the logical structure of a database is not a relational notion, but goes back to the early recommendations made for Network Databases by the Data Base Task Group of the COnference on DAta SYstems Languages [CODASYL] published in their 1971 report [14]. For example, Query Number 22 (supra) could be thought of as a description of the types of documents which an inquirer would want to see should new ones that satisfy these criteria be added to the database (a kind of current awareness specification). To save the inquirer the trouble of submitting the SQL query as shown (and the query could be much more complex than this one), we could define a virtual, or derived, relation to simplify repeated searches that varied only slightly, if at all, from the original query. In DB2 [10], the relational subschema is called a view and can be defined using the following statement:

```
DEFINE  VIEW CURRENT_DOCS
        AS SELECT *
        FROM   CITATION
        WHERE  DOCUMENT # = ANY
                    SELECT  DOCUMENT #
                    FROM    KEYWORDS FIRST, KEYWORDS SECOND
                    WHERE   FIRST.DOCUMENT # =
                              SECOND.DOCUMENT #
                    AND     FIRST.KEYWORD IN ('shop automation',
                              'computerization', 'automation')
                    AND     SECOND.KEYWORD IN ('integrated
                              manufacturing', 'flexible manufacturing')
```

The inquirer can now do a current awareness search using the keywords specified in Query 22, by simply submitting the following SQL statements:

```
SELECT  TITLE, JOURNAL NAME
FROM    CURRENT_DOCS
WHERE   ACQUISITION DATE > 12/31/85
```

This will retrieve the titles and corresponding journal names for those documents that satisfy the criteria of Query 22 and have been added to the database starting in 1986. This is clearly a much simpler query for an occasional inquirer to submit to the system (the original query might have been constructed by a more frequent database user).

The advantages of using subschemas in this way are clear: the inquirers can personally submit very complicated queries to the system even if they are not experienced SQL programmers; and professional SQL programmers will not have to serve as translators of the same user queries over and over again. In addition, the use of subschemas does not expand the physical size of the database since it does not have the effect of adding a new stored relation to the database. All that is added is the logical definition of the view (as stated before). This is entered into the system dictionary (where logical definitions are kept), but no entry is made to the directory (where links to the stored relations are kept). This is the sense in which the subschemas are considered virtual or derived. They can be added or dropped from the system quite easily.

9. PROCESSING AND STORAGE CONSIDERATIONS IN A DOCUMENT DATABASE

Although no precise measures of database size or processing overhead can be made without considering, *inter alia*, specific commercial DBMSs, the supporting hardware configuration and the operating system under which it would run, we can get a rough idea of some of the unique processing and storage requirements of a document database by estimating the number of tuples or records it contains. The number of tuples in a document database is an important factor in estimating processing and storage overhead for the following reasons: In the first place, the number of tuples required to support a document database is significantly larger than the number of documents being represented. Because so many different attributes of the documents (author, title, date, journal, keywords, etc.) require representation in the database, one document may be represented by 10 to 15 different tuples (assuming a normalized logical structure). These tuples, in turn, will require further supporting information in the form of tuple identifiers and a varying number of indexes (the latter being essential to support the *ad hoc* inquiry necessary in document retrieval; see Section 1). As a result, the size of the document database, as well as the processing overhead necessary to maintain the tuple-at-a-time access typical of relational systems, will increase significantly as documents are added to the collection (although this increase may be slowed by the careful use of data compression facilities and the judicious linking of the physical instantiations of the document attributes). If we assume the above logical structure and a mean subject indexing depth of 6 keywords, then the addition of one document to the collection results in a (theoretical) addition of at least 12 tuples, 12

tuple identifiers, and numerous index entries in our database (the number of tuples that must be added to the physical instantiation of the THESAURUS relation depends on how many of the new document's keywords are new to the vocabulary of the database).

Another reason why the number of tuples is an important component in estimating processing overhead for a document database is that many document retrieval requests involve the joining of two or more relations (a tuple, or record, type defines a relation). This is a direct result of the normalization process that significantly increases the number of tuple or record types over nonnormalized logical structures. Although normalization offers the advantage of avoiding the update and deletion anomalies [see Appendix D] of nonnormalized structures, the trade-off is that normalization causes an increase in the number of join operations required during retrieval. Joins involve some of the most intensive processing of any single operation on a relational database, and the amount of processing required is proportional to the number of tuples in the relations being joined (the size of the individual tuples is an important factor, too, but the process of normalization tends to make most of the tuple types relatively small, so their size becomes a less important factor in a join than the sheer number of tuples involved).

The estimation of the number of tuples in a document database is not an easy or straightforward endeavor. It requires making some assumptions about the statistical properties of indexing languages. Since such assumptions may be unfamiliar to the typical database administrator, it may be instructive to work through an example of how such rough estimations are made.

The largest relation in the database (in terms of the number of tuples) will be the KEYWORDS relation, which needs a tuple for each assignment of a subject term to a document. With a database of 10,000 documents, and a mean indexing depth of 6, we would expect to have a vocabulary of 3400 to 7000 unique subject terms, and a total number of indexing assignments of 59,880 (see Appendix B). The latter figure will be the number of tuples in the KEYWORDS relation.

The total number of tuples in the THESAURUS relation is equal to the number of unique co-occurrences of indexing terms in the database (i.e., the number of distinct pairs of terms that appear together indexing a particular document). Given a database of 10,000 documents and a mean indexing depth of 6, the estimated number of tuples for the THESAURUS relation would be 32,600 (see Appendix C).

In aggregate, then, the total number of tuples needed to build a database of 10,000 documents is estimated to be 129,480 (see Table 1). Clearly, to implement a document retrieval system on a relational database requires a significant commitment of available resources. This may not be a problem on a database management system running on a large mainframe computer or a smaller computer with a backend database machine (such as a VAX with Britten Lee's IDM 600) but to implement a working document retrieval system on a smaller computer (perhaps even a microcomputer) and still maintain the same retrieval capabilities will require some changes in our basic model.

Table 1.

Relation	Tuples
CITATION	10,000
ABSTRACT	10,000
AUTHOR	10,000[1]
DIRECTORY	5,000[2]
INSTITUTE	1,000[3]
JOURNAL	1,000[4]
KEYWORDS	59,880[5]
THESAURUS	32,600
TOTAL	129,480

[1]Assumes only single-author documents.
[2]Assumes only 5000 unique authors in the database.
[3]Assumes many authors will be affiliated with the same institution.
[4]Assumes many documents will be published in the same journal.
[5]Tuples in the KEYWORDS relation will be equal to the total number of index term assignments in the database.

10. REDUCING THE NUMBER OF RECORDS IN THE DATABASE

One of the observed characteristics of document retrieval systems is that retrieval patterns often follow a Pareto distribution. That is, about 20% of the documents on the database will account for approximately 80% of the retrieval activity. In other words, a small core of documents will be retrieved repeatedly. Since the THESAURUS relation contains information about the statistical (and, by inference, semantic) relationships between assigned index terms, these relationships may be accurately modeled by using co-occurrence data from just the core documents rather than all the documents on the database. The core documents can be easily identified by maintaining a count of the number of times each document on the database is retrieved. The core documents are those that have been retrieved, or retrieved a number of times above an established cutoff value. In our example, if we assume that the core documents represent 20% of the database, then the THESAURUS relation can be constructed on data from 2000 documents rather than 10,000. Using the same methods that we used before, we find that we would only need an estimated 1500 index terms to describe the subject content of these core documents, assuming a mean indexing depth of 6 (see Appendix B). The approximate number of unique co-occurrences that are likely to occur for 1500 terms and 2000 documents is 7550 (see Appendix C). This is the number of tuples needed to build a THESAURUS relation using data from only the core documents. The total number of tuples estimated to exist in the reduced database is 104,430—a reduction of 19%.

If greater reductions in the number of tuples is desired, it would probably *not* be wise to base the THESAURUS construction on a subset of the database smaller than the set of core documents. Further reductions in the number of tuples can be effected by reducing the mean indexing depth of keyword indexing (although, naturally, this may not be an easy or desirable policy to implement). If we were able to reduce indexing depth from 6 to 4 then the following changes would occur: (1) The KEYWORDS relation would be reduced from approximately 59,880 tuples to 39,824. (2) The THESAURUS relation would be further reduced from the core document level of 7550 tuples to 3570. In aggregate, the database would now contain an estimated 80,394 tuples as compared with 104,430 tuples (core documents only, mean depth of 6) or with the original database size of 129,480 tuples (all documents, mean depth of 6). This would represent a 38% reduction in the number of tuples from the full database size.

The notion of a core of comparatively highly retrieved documents can also be a useful tool for inquirers. We can add a RETRIEVAL relation to the database defined as follows:

RETRIEVAL DOCUMENT #*, TIMES (retrieved)

An inquirer would greatly speed up the search by limiting requests for documents to those documents that have been retrieved one or more times:

```
SELECT   . . .
FROM     . . .
WHERE    . . .
         .
         .
         .
         AND DOCUMENT # = ANY
         SELECT DOCUMENT #
         FROM RETRIEVAL
         WHERE TIMES > 0
```

This would ensure that the inquirer would see the more highly retrieved (and, by inference, more useful) documents first. This would mitigate the problem of output overload, which we discussed before. If the inquirer did not find all the desired documents, he or she

could then expand the search to the rest of the database by dropping the final SELECT clause.

The RETRIEVAL relation could also be used as the basis for ranking output. The inquirer would merely include a command in the document request to rank the output by the number of times the documents have been retrieved. This assumes that more highly retrieved documents are more likely to be useful to the inquirer.

11. OTHER ADVANTAGES OF DATABASE MANAGEMENT SYSTEMS

The image of Database Management Systems is that they provide better or easier access to databases. But access is only part of what they provide. Database Management Systems also furnish facilities for managing databases. It is just such database management facilities that make the use of DBMSs as a foundation for document retrieval systems even more attractive. These facilities might include:

- Recovery routines (specific to the DBMS)
- Performance measuring facilities (e.g., to tabulate the number of disk accesses needed to answer a database request)
- Database reorganization routines (e.g., to reduce the size of overflow areas in direct access files, or limit the number of extents a physical file may have)
- Data migration routines [to move less frequently used data (citations) down the storage hierarchy to cheaper storage facilities]
- Concurrency control (automatic management of concurrent updates or access and deadlock prevention)
- Elaborate authorization mechanisms (read and write access controlled down to the attribute or element level; access audit logs maintained)
- Logical and physical data independence (to facilitate independent restructuring of the logical and physical databases)
- Data compression and encoding routines
- Automatic enforcement of integrity constrains on data
- Report generators
- Flexible definition of transaction boundaries (e.g., commit and rollback)
- Facility to embed the inquiry language (here, SQL) in a sequential applications language (e.g., COBOL or PL/1)
- Telecommunications interface

12. A COMMENT ON THE PROCESSING SPEED OF RELATIONAL DATABASES

Any discussion of database management system implementation usually must address the controversial issue of processing speed. Traditional beliefs tend to hold that relational systems trade flexibility of query and database structuring for reduced processing speed (when compared with older DBMS models based on hierarchical or network logical structures). Although IBM has made claims that its own relational product, DB2, has delivered 62 transactions per second in a banking environment, more realistic benchmarks have been established at 18 transactions per second [15] and 6 to 9 transactions per second [16]. But most of these numbers are, in reality, relatively soft, since transaction processing rates can vary greatly even on the same database due to variations in tuning the physical structure of the database to support logical access. In general, it is relatively well accepted that, all things being equal, relational systems have a transaction rate approximately one-third to one-half that of older, established hierarchical or network systems. It is also clear that hierarchical and network DBMSs, by virtue of their having been around longer, have benefited from a substantially greater effort at optimization than relational systems have. As a result, some of the major proponents of relational systems have claimed that there is no theoretical reason why relational systems cannot perform as well as, or even better than, older nonrelational systems [16]. Most of this performance controversy has less significance for document databases. Relational systems offer clear advantages in *ad hoc* query process-

ing over nonrelational systems, and it has been the thesis of this discussion that *ad hoc* inquiry is essential to effective information retrieval. The only relevant question, then, is whether or not the transaction processing rate for relational systems is *adequate*, and not whether it is faster than hierarchical or network systems. Transaction rates of 6 to 18 per second should be satisfactory for most information retrieval applications.

One interesting development that may have direct bearing on the performance of document-based DBMSs is the recent development of database machines (or, backend database machines) and associative disk technology. These technologies attempt to utilize specialized hardware, content-addressable memories and limited parallel processing to enhance the performance of relational DBMSs. According to Date [17], these advances may be particularly important for document-based DBMSs:

> We have assumed throughout this book . . . that the database is formatted—that is, that it exhibits a highly regular structure. Such an assumption is appropriate for many applications; but there are also certain text search or information retrieval applications, in which the database contains (for example) scientific abstracts or other textual information, and the overall structure is much less regular. Queries against this kind of database tend to be quite complex . . . Associative disks may prove useful in such applications. [p. 360]

13. CONCLUSION

Recent research has shown that database management systems are effective tools for constructing operational document retrieval systems. This discussion has argued that the retrieval requirements of document retrieval systems can be supported most effectively by the relational model, especially if a system capable of more advanced document retrieval techniques, such as associative or inferential retrieval, is desired. The logical structure for implementing the advanced or extended document retrieval model was discussed at length, and several storage structure issues have been addressed that are of particular importance for the design of document retrieval systems.

APPENDIX A

1. $(K_p \cdot K_q) \vee (K_q \cdot K_r) \vee (K_p \cdot K_r) \vee K_r$
2. $(K_p \cdot K_q) \vee (K_q \cdot K_r) \vee (K_p \cdot K_r) \vee (K_r \cdot 1)$ [identity]
3. $(K_p \cdot K_q) \vee (K_q \cdot K_r) \vee [(K_p \vee 1) \cdot K_r)]$ [distribution]
4. $(K_p \cdot K_q) \vee (K_q \cdot K_r) \vee [(1 \cdot K_r)]$ [identity]
5. $(K_p \cdot K_q) \vee (K_q \cdot K_r) \vee K_r$ [identity]

[repeat steps 2–5]

.
.
.

9. $(K_p \cdot K_q) \vee K_r$
10. $(K_p \vee K_r) \cdot (K_q \vee K_r)$ [distribution]

Another example:

$$F = [(K_b \cdot \bar{K}_c) \vee (\bar{K}_b \cdot K_c)] \cdot [(\bar{K}_a \cdot K_b) \vee (\bar{K}_a \cdot K_c) \vee (K_a \cdot \bar{K}_b \cdot \bar{K}_c)]$$

Set

$$F_1 = (K_b \cdot \bar{K}_c) \vee (\bar{K}_b \cdot K_c)$$

and,

$$F_2 = (\bar{K}_a \cdot K_b) \vee (\bar{K}_a \cdot K_c) \vee (K_a \cdot \bar{K}_b \cdot \bar{K}_c)$$

1. $F_1 = (K_a \cdot K_b \cdot \bar{K}_c) \vee (\bar{K}_a \cdot K_b \cdot \bar{K}_c) \vee (K_a \cdot \bar{K}_b \cdot K_c) \vee (\bar{K}_a \cdot \bar{K}_b \cdot K_c)$ [change to complete disjunctive normal form]
2. $F_1' = (\bar{K}_a \cdot K_b \cdot K_c) \vee (\bar{K}_a \cdot \bar{K}_b \cdot \bar{K}_c) \vee (K_a \cdot K_b \cdot K_c) \vee (K_a \cdot \bar{K}_b \cdot \bar{K}_c)$ [complement of F_1]
3. $F_1 = (F_1')' = (K_a \vee \bar{K_b} \vee \bar{K_c}) \cdot (K_a \vee K_b \vee K_c) \cdot (\bar{K_a} \vee \bar{K_b} \vee \bar{K_c}) \cdot (\bar{K_a} \vee K_b \vee K_c)$ [complete conjunctive NF of F_1]
4. $F_2 = [\bar{K}_1 \cdot K_b \cdot K_c) \vee (\bar{K}_a \cdot K_b \cdot \bar{K}_c) \vee (\bar{K}_a \cdot \bar{K}_b \cdot K_c) \vee (K_a \cdot \bar{K}_b \cdot \bar{K}_c)$ [change to complete disjunctive NF]
5. $F_2' = (\bar{K}_a \cdot \bar{K}_b \cdot \bar{K}_c) \vee (K_a \cdot K_b \cdot K_c) \vee (K_a \cdot \bar{K}_b \cdot K_c) \vee (K_a \cdot K_b \cdot \bar{K}_c)$ [complement of F_2]
6. $F_2 = (F_2')' = (K_a \vee K_b \vee K_c) \cdot (\bar{K}_a \vee \bar{K}_b \vee \bar{K}_c) \cdot (\bar{K}_a \vee K_b \vee \bar{K}_c) \cdot (\bar{K}_a \vee \bar{K}_b \vee K_c)$ [complete conjunctive NF of F_2]
7. $F = F_1 \cdot F_2$
   $= (K_a \vee K_b \vee K_c) \cdot (K_a \vee \bar{K}_b \vee \bar{K}_c) \cdot (\bar{K}_a \vee K_b \vee k_c) \cdot (\bar{K}_a \vee K_b \vee \bar{K}_c) \cdot (\bar{K}_a \vee \bar{K}_b \vee K_c) \cdot (\bar{K}_a \vee \bar{K}_b \vee \bar{K}_c)$ [conjunction of 3 and 6]
8. $F' = (K_a \vee \bar{K}_b \vee K_c) \cdot (K_a \vee K_b \vee \bar{K}_c)$ [complement of F]
9. $F = (F')' = (\bar{K}_a \cdot K_b \cdot \bar{K}_c) \vee (K_a \cdot \bar{K}_b \cdot K_c)$ [complete disjunctive NF of F]
10. $\bar{K}_a \cdot [(K_b \cdot \bar{K}_c) \vee (\bar{K}_b \cdot K_c)]$
11. $\bar{K}_a \cdot [(K_b \cdot \bar{K}_c) \vee \bar{K}_b] \cdot [(K_b \cdot \bar{K}_c) \vee K_c]$
12. $\bar{K}_a \cdot [(K_b \vee \bar{K}_b) \cdot (\bar{K}_b \vee \bar{K}_c)] \cdot [(K_b \vee K_c) \cdot (\bar{K}_c \vee K_c)]$
13. $\bar{K}_a \cdot [1 \cdot (\bar{K}_b \vee \bar{K}_c)] \cdot [K_b \vee K_c) \cdot 1]$
14. $\bar{K}_a \cdot [(\bar{K}_b \vee \bar{K}_c) \cdot (K_b \vee K_c)]$
15. $\bar{K}_a \cdot (\bar{K}_b \vee \bar{K}_c) \cdot (K_b \vee K_c)$ [simplified version of F]

APPENDIX B

Many factors influence the growth of an indexing vocabulary, and although estimates of vocabulary growth are difficult to make they are possible to do if we take into consideration certain observed processes that occur in the development of information retrieval systems.

Each indexing term in the vocabulary is assigned to documents within the database a certain number of times. If we take these individual term assignment frequencies and rank them from the highest to the lowest values, we often find that they conform to a hyperbolic or Zipfian [18] distribution [19, 20]. Thus,

$$N_A = \sum_{i=1}^{N_T} F_i = F_1(\ln(2N_T + 1) - 0.116)$$

where N_A = the total indexing assignments in the database, N_T = the total number of unique terms in the vocabulary, F_i is the assignment frequency of term i, and F_1 is the assignment frequency of the most frequently assigned subject term in the database (i.e., term of frequency rank 1).

In our example, since we know the mean depth of indexing is 6 and the number of documents in the database is 10,000, we can estimate N_A with the following equation [21]:

$$N_A = \sum_{i=1}^{t} t(e^{-m})\left(\frac{m}{i!}\right)(N_D)^*$$

where t = the maximum number of vocabulary terms assigned to any document in the database (with a mean depth of 6 we would expect a maximum depth of about 14 or 15), m = the mean depth (here, 6), and N_D = the number of documents in the collection (here,

*For small values of m, N_A can be estimated more easily as the product $m \times N_D$. This approximation becomes less accurate as m increases.

10,000). Setting $t = 15$, $N_A = 59{,}800$ (this will be the number of tuples in the KEYWORDS relation). Now, by substitution:

$$N_A = 59{,}800 = F_1(\ln(2N_T + 1) - 0.116)$$

Because the rank-frequency distribution is hyperbolic, then if the distribution were perfect, F_1 would be equal to N_T. In empirical studies it has been found that F_1 is somewhat less than a perfect distribution would predict, and N_T somewhat greater. If we solve for a value of N_T slightly greater than F_1 we find that:

$$N_T = 7000$$

$$F_1 = 6350$$

As it turns out, 7000 will be an estimate of the maximum reasonable value for N_T. The Zipfian distribution is an accurate model for the growth of indexing assignments when new subject terms are added to the system vocabulary at a fairly constant rate. This is the case for the early stages of database growth, and for all growth in databases which cover areas like chemical research, pharmacology, or patents. For most databases, however, the term assignment frequency distribution is Zipfian only in the early stages, and the addition of new vocabulary terms falls off as new documents are added to the database [22–24]. This is because most of the new documents deal with the same subjects that older documents on the database deal with. This kind of vocabulary growth is log-normal rather than Zipfian and is best modeled by [25]:

$$N_T = [3000 \log_{10} (N_A + 7100)] - 11{,}000$$

with $N_A = 59{,}800$, then the predicted size of the vocabulary would be 3480. Thus, we can estimate that the likely size of our indexing vocabulary (N_T) would be between 3480 and 7000 terms (where $N_D = 10{,}000$ and mean depth = 6).

APPENDIX C

The number of co-occurring index terms in a database can be estimated in much the same way as index term assignments were estimated (Appendix B). With 10,000 documents and a mean indexing depth of 6, the approximate total number of index term co-occurrences (N_C) can be determined by using the following equation [21]:

$$N_C = \sum_{i=1}^{t} P_2^i(e^{-m}) \left(\frac{m^i}{i!}\right) (N_D)$$

where t = the maximum number of vocabulary terms assigned to any document in the database (here, 15), m = the mean depth (here, 6), and N_D = the number of documents in the collection (here, 10,000). Setting $t = 15$, $N_C = 358{,}411$.

N_C is the total number of co-occurrences that have occurred in the database index term assignments, but the number of tuples estimated to exist in the THESAURUS relation is equal to the number of unique co-occurrences of terms which exist in the database (i.e., no matter how many times (>0) index terms T_i and T_j are both assigned to the same documents it will require only two tuples in the THESAURUS relation to model their relationship).

Some studies have indicated that co-occurrence distributions, like that of term distributions, are basically Zipfian in nature, but other studies have suggested that the distribution of co-occurring term pairs may vary somewhat from the traditional Zipfian model [26]. Our purpose here is to offer some simple heuristics for estimating the number of unique co-occurrences that should occur in our hypothetical database.

N_C is comparable to N_A (Appendix B) and can be substituted for it in the equation we used to represent the distribution of index term assignments:

$$358{,}411 = F_1(\ln(2N_T + 1) - 0.116).$$

We can reinterpret F_1 as the number of co-occurrences of the most frequently co-occurring term pair on the database, and we can reinterpret N_T as the number of uniquely occurring term pairs (or co-occurrences). Solving the above formula for equal values of F_1 and N_T we find that $N_T = F_1 = 32{,}600$. (Unlike our solution for index term assignments we have no evidence that F_1 will be somewhat less than N_T.) N_T, of course is the value that represents the number of tuples estimated to be in the THESAURUS relation.

Unlike index term assignments, we would not expect the number of new unique term co-occurrences to fall off as markedly as the number of new terms added to the vocabulary does during database growth. This is because even if new terms are not added to the vocabulary, new pair combinations can be generated almost indefinitely. With a vocabulary of between 3480 and 7000 terms (Appendix B), the total number of possible unique pair combinations is:

$$P_2^{3480} \rightarrow P_2^{7000} = 12{,}110{,}400 \rightarrow 49{,}000{,}000$$

At most, the 32,600-term co-occurrences estimated to occur in the hypothetical database represent only 0.3% of the possible terms combinations. Clearly, the growth in the number of new unique term co-occurrences is not rigidly dependent on the addition of new terms to the vocabulary, although there is undoubtedly some relationship between the addition of new vocabulary terms and the occurrence of new term pairings.

APPENDIX D

Normalization in relational systems

Normalization is the process by which attributes are grouped into relations in such a way that update and deletion anomalies are avoided. Traditionally, this can be accomplished by decomposition or synthesis. In decomposition, all the attributes in the database are grouped into one relation and this relation is then successively broken down into smaller relations (projections) until, most often, 3rd Normal Form (or, Boyce–Codd Normal Form) is reached and every determinant is a candidate key [10].

Another way to construct relations that satisfy 3rd NF or BCNF is to synthesize them. Basically, this method begins with a list of all the attributes in the database and their functional dependencies, where functional dependency is defined as:

> Given a relation R, attribute Y of R is functionally dependent on attribute X of R if and only if each X-value in R has associated with it precisely one Y-value in R (at any one time). [10]

For example, a person's name is functionally dependent on his or her Social Security number. Thus,

$$\text{SSN} \longrightarrow \text{Name (or, "SSN determines name")}$$

But Social Security numbers are not functionally dependent on people's names, because many individuals have the same name. Thus,

$$\text{NAME} \text{ —X—} \rightarrow \text{SSN}$$

Given all the attributes in the database, and a complete specification of the functional dependencies between them, the synthesizing algorithm for 3rd Normal Form is:

1. (Eliminate extraneous attributes.) Let F be the given set of FDs. Eliminate extraneous attributes from the left side of each FD in F, producing the set G. An attribute is extraneous if its elimination does not alter the closure of the set of FDs.
2. (Find covering.) Find a nonredundant covering H of G.
3. (Partition.) Partition H into groups such that all of the FDs in each group have identical left sides.
4. (Merge equivalent keys.) Let $J = \emptyset$. For each pair of groups, say H_i and H_j, with left sides X and Y, respectively, merge H_1 and H_2 together if there is a bijection $X \leftrightarrow Y$ in H^+. For each such bijection add $X \rightarrow Y$ and $Y \rightarrow X$ to J. For each $A \in Y$, if $X \rightarrow A$ is in H, then delete it from H. Do the same for each $Y \rightarrow B$ in H with $B \in X$.
5. (Eliminate transitive dependencies.) Find $H' \subseteq H$ such that $(H' + J)^+ = (H + J)^+$ and no proper subset of H' has this property. Add each FD of J into its corresponding group of H'.
6. (Construct relations.) For each group, construct a relation consisting of all the attributes appearing in that group. Each set of attributes that appears on the left side of any FD in the group is a key of the relation. (Step 1 guarantees that no such set contains any extra attributes.) All keys found by this algorithm will be called *synthesized.* The set of constructed relations constitutes a schema for the given set of FDs [27, p. 293].

Although the above description of the synthesizing algorithm is quite formal, its application is relatively straightforward. It is especially useful for normalizing databases with large numbers of attributes and functional dependencies, a situation where the decomposition method can be quite cumbersome.

In general, most databases do not require synthesis above 3rd Normal Form, although violations of 4th and 5th Normal Forms are not impossible. It would be possible to construct a relation in a document retrieval database to violate 4th Normal Form (which deals with multivalued dependencies) as follows:

Given:

1. A database with mostly multiple-author articles.
2. A relation such as:

(Document #*, Author, Keyword)

Here, both keyword and author are multidependent on Document # (i.e., each document number determines a *set* of keywords and a set of authors). Also, to violate the 4th NF, the keywords and authors must not be dependent on each other. In the logical structure that we have proposed in this discussion, though, this problem does not occur. This is because we have also included the weight attribute with the keyword. If we included it with the above relation, we would get,

(Document #*, Keyword*, Author, Weight)

Here, Weight depends on both Document # and Keyword, so both attributes must form a concatenated key. But if this happens, Author only depends on the Document # part of the key and this violates 2nd Normal Form. Hence, we don't even have to get to 4th NF to have a problem with the combination of these attributes. It is solved through decomposition, of course, by taking the Author attribute out of the relation.

The Fifth Normal Form deals with the problem of join dependencies and comes into use in the unusual situation where a relation that models a many-to-many-to-many relationship (such as "parts to suppliers to jobs" or "suppliers to parts to assemblies") is nonloss decomposed into three projections of the original relation (where each projection models one of the three possible many-to-many relations). The problem comes when two of the projections are joined to produce the original relation. Sometimes this process results

in the creation of spurious tuples that did not exist in the original relation. Since there are no many-to-many-to-many relations between the attributes in the database we have been discussing, there is no need to consider the solutions to this problem provided by the 5th Normal Form (Projection-Join Normal Form) [10].

Acknowledgment—The author would like to express his appreciation for the helpful comments of Jim Fry, Michael Gordon, Maurita Holland, and Alan Merten of the University of Michigan. This research was conducted in part while the author was a visiting faculty member at the Industrial Technology Institute, and in part under a research grant from The Graduate School of Business at the University of Michigan.

REFERENCES

1. Crawford, R.G. The relational model in information retrieval. Journal of the American Society for Information Science, 32(1): 51–64; 1981.
2. Macleod, I.A. SEQUEL as a language for document retrieval. Journal of the American Society for Information Science, 30(5): 243–249; 1979.
3. Macleod, I.A. The relational model as a basis for document retrieval system design. The Computer Journal, 24(4): 312–335; 1981.
4. Maron, M.E. Relational data file I: Design philosophy. In: Schecter, M.G., editor. Information retrieval: A critical view. Washington, DC: Thompson Book Co.; 1966.
5. Levien, R.E. Relational data file II: Implementation. In: Schecter, M.G., editor. Information retrieval: A critical view. Washington, DC: Thompson Book Co.; 1966.
6. Levin, R.E.; Maron, M.E. A computer system for inference execution and data retrieval. Communications of the ACM, 10(11): 715–721; November 1967.
7. Blair, D.C. SQUARE (Specifying Queries as Relational Expressions) as a document retrieval language. Unpublished working paper, University of California, Berkeley; Spring 1974.
8. Blair, D.C. The data-document distinction in information retrieval. Communications of the ACM, 27(4): 369–374; April 1984.
9. Dattola, R.T. FIRST: Flexible Information Retrieval System for Text. Journal of the American Society for Information Science, 30(1): 10–14; 1979.
10. Date, C.J. An introduction to database systems, vol. 1. Reading, MA: Addison-Wesley, 3rd ed.; 1981.
11. Blair, D.C. Searching biases in large, interactive document retrieval systems. Journal of the American Society for Information Science, 31(4): 271–277; July 1980.
12. Blair, D.C.; Maron, M.E. A test of retrieval effectiveness for a full-text document retrieval system. Communications of the ACM, 28(3): 289–299; March 1985.
13. Jacquesson, A.; Schieber, W. Term association analysis on a large file of bibliographic data, using a highly-controlled indexing vocabulary. Information Storage and Retrieval, 9: 85–94; 1973.
14. Data Base Task Group of CODASYL Programming Language Committee. Report; April 1971.
15. Babcock, C. Users: DB2 chokes on volume test. Computerworld, 21(30): 1; July 27, 1987.
16. Date, C.J. On the performance of relational database systems. Chapter 5 in Date, C.J., Relational database: Selected writings. Reading, MA: Addison Wesley; 1986.
17. Date, C.J. An introduction to database systems, vol. 2. Reading, MA: 1983. Addison Wesley; 1983.
18. Zipf, G.K. Human behavior and the principle of least effort. Cambridge MA: Addison Wesley; 1949.
19. Van Rijsbergen, C.J. Information retrieval, 2nd ed. London: Butterworths; 1979.
20. Arthur D. Little, Inc. Centralization and documentation. Cambridge, MA; 1963.
21. Bird, P.R. The distribution of indexing depth in documentation systems. Journal of Documentation, 30(4): 381–392; December 1974.
22. Lancaster, F.W. Vocabulary control for information retrieval. Washington, DC: Information Resources Press; 1972.
23. McClelland, R.M.A.; Mapleson, W.W. Construction and usage of classified schedules and generic features in coordinated indexing. ASLIB Proceedings, 18: 290–299; 1966.
24. Blagden, J.F. Management information retrieval: A new indexing language. London: British Institute of Management, 1969 (2nd ed.; 1971).
25. Wall, E. Further implications of the distribution of index term usage. Proceedings of the American Documentation Institute, 1: 457–466; 1964.
26. Nelson, M.J.; Tague, J.M. Split size-rank models for the distribution of index terms. Journal of the American Society for Information Science, 36(5): 283–296, September 1985.
27. Bernstein, P.A. Synthesizing third normal form relations from functional dependencies. ACM Transactions on Database Systems, 1(4): 277–298; December 1976.