USING A RELATIONAL DATA BASE SYSTEM TO STORE TEXT

by

Gordana M. Pavlovic

Memorandum No. UCB/ERL M83/44

21 July 1983

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Using a relational data base system to store text

by

Gordana M. Pavlovic

Abstract

One way of handling texts in relational data base systems is a lexical- based text storing. Lexical processing of texts stored in such a way is described. Experiments in automatic indexing, abstracting and text search as well as some editing facilities are described.

1. Introduction

It was proposed [WONG 82] that relational data base system should be enhanced to support text processing by supporting lexical data types to be used in domains.

The main characteristics of this approach to handling documents is that a token (word), as a lexical and semantic unit, is a basic unit of a text, as opposed to an arbitrary segmentation as strings or lines [STON 82] are. Nevertheless, all basic concepts investigated in the context of string based handling of texts, especially variable width and ordered relations, are also useful in this approach. Word-based fragmentation of texts makes it easy to apply the whole group of lexical operations on texts (ex. automatic indexing, abstracting, bibliographic retrieval, fact retrieval).

This paper is an attempt to use existing capabilities of relational data base systems, INGRES [INGR 81] in particular, to handle lexical data types, and to add lexical processing capabilities to such systems. This was to be done

in two ways: by defining and implementing as many operations as possible that concern texts, and by investigating different storage structures for storing lexical data types.

We have concentrated on lexical operations -automatic indexing, abstracting, retrieval - for which this approach is especially suitable, as well as INGRES itself because of its aggregation functions like count and its storage structures, both primary and secondary; we also treat, in some extent, editing operations, in order to examine whether they are compatible with this approach.

The global chart of text flow directed by the operations that are presented in this paper is depicted in Figure 1.

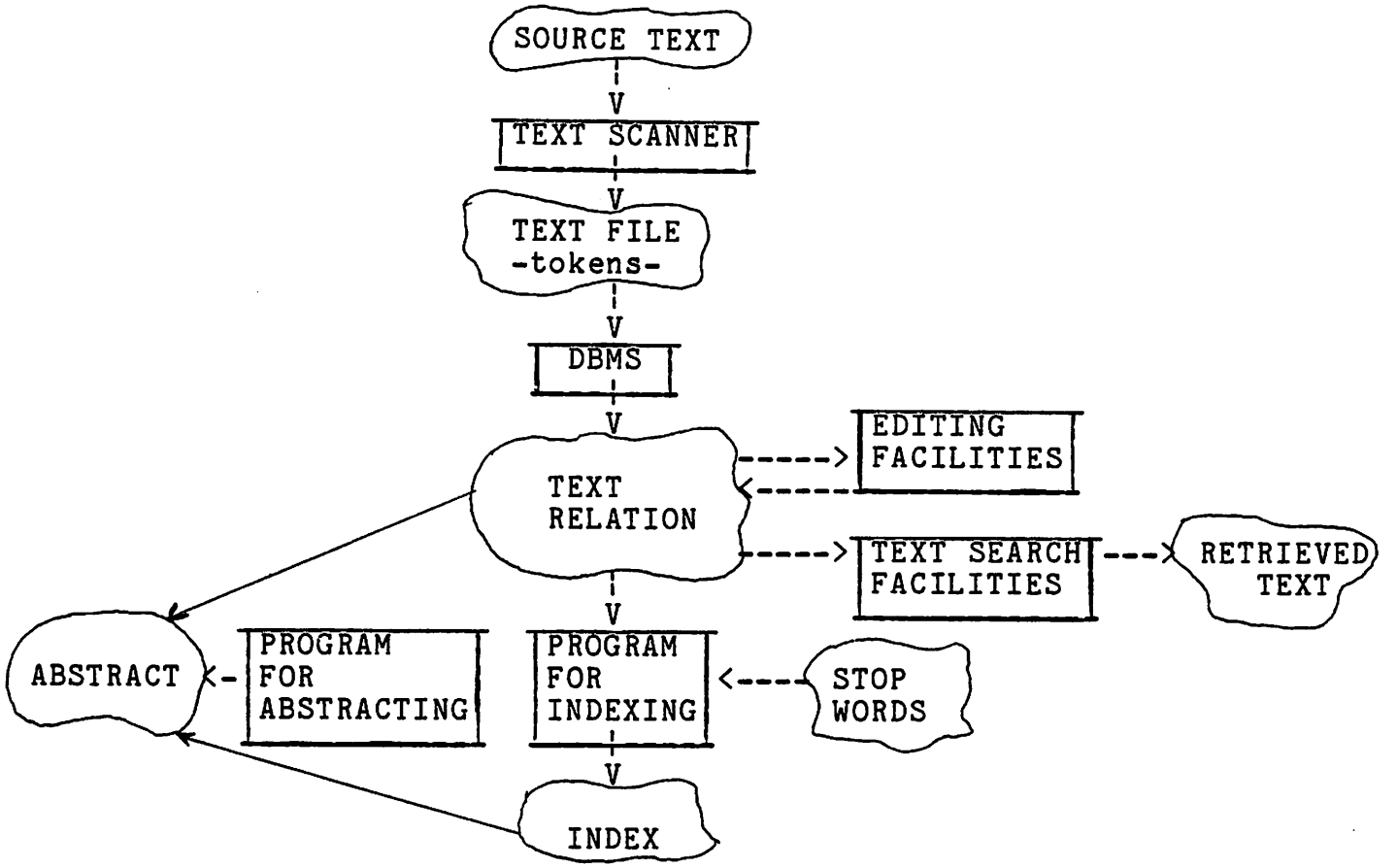

Figure 1.

2. Text scan program

Program for text scanning takes, as an input, a source text and produces, as an output, a file containing tokens (sequence of characters that has a meaning) and two kinds of informations about each token in the text: lexical type of informations concerning paragraph, sentence and sequence in which the token is found, and source text format informations concerning the line in which the token is found. Output records that are load into the data base are of the form

t, type, len, l#, tl#, p#, s#, sq#, tsq#

where the meaning of individual fields are as follows:

t - token,

type - type of the token (word, numeral, etc.),

len - length of the token in characters,

l# - line number,

tl# - token number inside a line,

p# - paragraph number,

s# - sentence number,

sq# - sequence number,

tsq# - token number inside a sequence.

It is obvious that some information is duplicated. Since lexical informations can be always deduced from the source text and the source text can be completely regenerated from the source text format informations, lexical informations can be deduced from source text format informations. We choose this model because it is easier to perform different kinds of operations if we want to execute them on the same file (i.e. relation).

If we are interested only in one special kind of operations (only lexical or only editing operations), less information (and fewer fields) is sufficient.

Our text scan program, as an example of such programs, assumes that source text uses NROFF for formating and, consequently, that type of a token can be as follows (notation is a regular_expression-like notation):

word: $l(l|d)^{*} \mid l$.

l and d stand for letter and digit;

numeral: $(+|-|e)(d^{+}|d^{*}.d^{+})$

e is a symbol for empty string;

command: $.w^{+}$

where w stands for "whatever", provided a dot is the first character in a line and this is not a numeral; all characters in a line constitute a command, except for section heading commands for which a command is a sequence of characters until the name of a section is reached;

punctuation: p^{+}

where p is a punctuation character;

heading: a word or a numeral in a line beginning with a section command;

footnote: a word or a numeral between commands for opening and closing a footnote;

bibliographic: a word or a numeral after the command for the bibliographic portion of a text.

Also, paragraph number, starting with 1 (except for title and author that are assigned paragraph number 0), is incremented whenever any of new paragraph commands is reached. Sentence (sequence) number, starting with 1 in each paragraph (sentence), is incremented whenever sentence (sequence) terminator is reached, where sentence (sequence) terminator is any of the symbols

.|?|! (,|;|:) - maybe followed by right parenthesis.

Other fields have the obvious meanings.

For example, for an input text beginning with the lines in Figure 2, a portion of the output is depicted in Figure 3.

```
.tp
.sp 2i
.(1C
DEPENDENCIES AND NORMAL FORMS
IN THE RELATIONAL MODEL OF DATA
WITH TWO TYPES OF NULL VALUES
.sp
by
Gordana M. Pavlovic
.)1
.bp
.he ''%''
.ls 2
.in 0
.sh 1 "Introduction"
.pp
In this paper we first briefly describe the extended rela-
tional...
```

Figure 2

```
type  len   l#   tl#   p#   s#   sq#   tsq#   t
-------------------------------------------------------
c,     3,    1,   1,    0,   0,   0,    0,     .tp
c,     6,    2,   1,    0,   0,   0,    0,     .sp 2i
c,     4,    3,   1,    0,   0,   0,    0,     .(1C
w,    12,    4,   1,    0,   0,   0,    1,     DEPENDENCIES
w,     3,    4,   2,    0,   0,   0,    2,     AND
w,     6,    4,   3,    0,   0,   0,    3,     NORMAL
                          .
                          .
w,     2,    8,   1,    0,   0,   0,   17,     by
c,     3,    9,   1,    0,   0,   0,    0,     .sp
w,     7,   10,   1,    0,   0,   0,   18,     Gordana
w,     2,   10,   2,    0,   0,   0,   19,     M.
                          .
                          .
c,     9,   13,   1,    0,   0,   0,    0,     .he ''%''
                          .
                          .
c,     5,   16,   1,    0,   0,   0,    0,     .sh 1
p,     1,   16,   2,    1,   0,   0,    1,     "
h,    12,   16,   3,    1,   0,   0,    2,     Introduction
p,     1,   16,   4,    1,   0,   0,    3,     "
c,     3,   17,   1,    0,   0,   0,    0,     .pp
w,     2,   18,   1,    1,   1,   1,    1,     In
w,     4,   18,   2,    1,   1,   1,    2,     this
                          .
                          .
```

Figure 3

This output is a load into a data base.

More general text scanner can be written, that takes as an input not only source text, but also the following items:

-possible types of tokens that should be considered, defined by regular expressions along with additional conditions that should be tested;

-actions implied by each found token (setting conditions),

-the order in which different types of tokens should be examined.

Policy of setting paragraph, sentence, sequence and token numbers can be included in actions implied by some token types, and thus contained in input definitions rather then in scan program itself. Scan program is, then, an extension of finite automaton that recognizes tokens and produces actions defined in input by corresponding token types; program also keeps track of line and token numbers.

In this way the same scan program would work for different users environments -independently on formatting tool used in a source text (e.g. NROFF) or on characteristics of the source text itself, as far as the source text has the predefined syntax structure (paragraph, sentence, sequence, token).

Our example of scan program that assumes NROFF as formatting tool of source text, would be, then, just an instance of such a general scanner, whose input, except the source text, can be some form of the description of token types (types in boxes), conditions, actions, paragraphs, sentences and sequences (Figure 4).

```
condition indicators: indsh /*section header*/,
                      indft /*footnote*/,
                      indbb /*beginning bibliographic*/;
                      /*all set to zero*/
```

token_type	condition	token	action
number	---	digit$^{+}_{*}$ ¦	---
		digit * .digit^{+}	---
sign	---	+¦-¦e	---
nh	---	digit¦.¦space	---
nn	---	letter¦digit¦punct¦space	---
lpn	---	letter¦punct	---
¦heading¦	indsh=1	word¦numeral	---
¦footnote¦	indft=1	word¦numeral	---
¦bibliogr¦	indbb=1	word¦numeral	---
¦command¦	tl#=1	.pp¦.lp¦	indsh=0, p#=p#+1, s#=q#=1, tsq#=0;
		(.sh¦.uh¦.$p¦.$0) nh* ¦	indsh=1;
		.(f¦	indft=1;
		.)f¦	indft=0;
		.++B¦	indbb=1;
		.lpn nn* ¦	---
		.digit letter^{+} nn*	---
¦word¦	---	letter (letter¦digit)* ¦	---
		letter	---
¦numeral¦	---	sign number	---
¦punctuation¦	---	.¦?¦!¦ (.¦?¦!))¦	s#=s#+1, sq#=1,tsq#=0;
		,¦;¦:¦ (,¦;¦:))¦	sq#=sq#+1, tsq#=0;
		punct^{+}	---

Letter, digit and punct are single characters with their usual meaning; space is blank or tab.

Figure 4

3. Lexical processing

First group of operations that we wanted to implement were

-automatic indexing

-keywords extracting

-automatic abstracting

-retrieval

These operations are the most common operations found in references concerning text processing, and they are based just on words as text units, so that our approach in handling texts is especially suitable for their implementation.

Automatic indexing consists of extracting from (or assigning to) the text significant words, whatever it means, in an automated process.

Keywords extracting consists of deriving from the text the most relevant terms, ones that can be descriptors of a text, whatever the notion of "relevancy" means.

Automatic abstracting is an automated process of extracting the most significant sentences from a given text, where significance of a sentence is a function of significance of words in it.

Retrieval has more than one meaning. Bibliographic retrieval is a process in which, for a given request describing a topic, system responds by a list of references (maybe ordered by a level of significance). Text search is

a search for particular paragraphs or sentences from a given text, that satisfy given request.

There are many different techniques and methods for automatic keywords extracting, indexing, abstracting and retrieval, based on how one defines notions of significant words, the most relevant terms, the most significant sentences, significance of a text [STEV 70, SPAR 76, LUHN 63, SALT 81, SWAN 63]. Research in that field has been being carried for last twenty years. Main objective of those investigations, (since indexing and abstracting influence retrieval) is to increase the so-called recall and precision of the documents retrieved. These notions are defined as follows:

$$\text{recall} = \frac{\text{number of relevant documents retrieved}}{\text{number of relevant documents in a collection}}$$

$$\text{precision} = \frac{\text{number of relevant documents retrieved}}{\text{number of documents retrieved}}$$

We will briefly review some of the techniques and approaches for indexing, abstracting and retrieval.

In automatic indexing there are two basically different methods [STEV 70]:

-derived indices

-assigned indices

Derived indices are words from the text. They can include all words from a text except for "stop" words (articles, prepositions, auxiliary verbs, pronouns), or words

from titles, or cited references, each accompanied by a list of source documents which cite it.

Derivative indices can also be modified in a sense that not all of non-stop words from a text (even from title) are indices. In modified derivative indexing there are two basic approaches: syntactic and semantic indexing. As syntactic indices usually some special syntactic word groups are chosen -ex. nouns, or adjective-noun combinations [BAXE 62]. Semantic approach attempts to determine significance of a term on the basis of how meaningful for a given text that term is. The measure of "meaningfulness" is chiefly based on statistical informations like frequency of a term in a given text, or relative frequency with respect to the whole collection [BARH 59], or frequency of a term relatively to the size of a text [SPAR 74], or position of the first occurrence of a term in a text [LESK 62], or the frequency in "relevant" texts [SALT 81], etc.

There is a special method - classification - devoted to expanding the index with terms that tend to co-occur with index terms in a given collection [SPAR 76].

On the basis of all the above mentioned criteria, terms may be included in the index with different "weights". In retrieval, weights of index terms that appear in a request will be used to determine the overall "weight" (significance) of a text.

Assigned indices are words assigned to a text in the following way: collection of documents is investigated in order to obtain index terms for special categories of texts (by statistical methods as correlation matrix, for instance [BORK 64]), than a new text is appended to some of those categories if it contains more than some number of its index terms, and all of the index terms for the specific collection become the index terms for that text (ex. [MARO 61]).

Both assigned and derivative indices can be weighted on the basis of the criteria mentioned for derivative indices.

In automatic abstracting, sentences from a given text are extracted that contain the highest concentration of the most important words, where importance of a word is its weight, whichever criterion of those mentioned before is chosen for terms weighing [LUHN 58, CARR 81].

Text search is defined by a given request; only those paragraphs (or sentences) might be chosen that contain given terms, or paragraphs and sentences might be weighted on the basis of previous criteria and a given request.

In bibliographic retrieval, texts can be chosen by boolean search - query terms matching with indices of texts - whichever method for indexing has been chosen, or by weighing texts on the basis of weights of index terms appearing in the request or in the request extended by terms that tend to co-occur, in a given collection, with request terms (ex. [STIL 62]).

Bibliographic retrieval can be expanded to complex search where not only texts retrieved by previous methods are chosen, but also texts containing some significant amount of index terms of previously retrieved texts [ROBE 81], or to interactive on-line retrieval where some of the previously mentioned methods is applied first, then user is consulted; if he is not satisfied, some other weighing scheme is applied and the list of texts retrieved is given; if the user is not satisfied with any of them, he should be asked to reformulate his request, maybe using terms assigned to different categories of texts (if assigned indices are applied) [SPAR 74].

Our goal was to implement the simplest techniques for lexical processing as examples of how efficiently relational system can be used for doing the whole class of techniques. All other techniques can be implemented using the same tools of the relational system and lexical-based approach to text storing; the two we wanted to validate instead of validating techniques themselves. All programs are written in EQUEL [INGR 81].

As index terms our program for indexing accepts all words except STOP-words, ignoring the difference between upper and lower case letters (STOP-word relation is explicitly created). Index term relation also contains frequences of those terms.

Our program for abstracting then chooses keywords - terms from the index term relation - as index terms, in dependence on users input, in some special range of frequences (the least, the most frequent words, or in the middle range of frequences), and extracts sentences that contain any of the chosen terms.

Since it is obvious that some different index terms are the same in their meaning (one might be in singular, another in plural), and that it affects the decision of which word is more frequent, we create another relation containing truncated (up to 5 characters) index terms and their frequences, and that relation (instead of index term relation) is actually used in the program for abstracting. The program also takes as an input the maximum number of sentences (or words) that the user wants to have in an abstract, and sentences are then selected (if there are more than required) as those contained more than one (or two, or three) truncated keywords or their occurrences.

Since we did not have a text collection, nothing concerning bibliographic retrieval has been done. However, some text search operations can be easily implemented in QUEL.

We give two examples of such operations and their QUEL-formulations.

Example 1.

Find all sentences that contain word-phrase "null values"

```
range of e is text
range of s is text
range of t is text
retrieve (e.all) where e.p#=s.p# and e.s#=s.s# and
     s.p#=t.p# and s.s#=t.s# and s.sq#=t.sq# and
     t.tl#=s.tl#+1 and s.t="null" and t.l="values"
```

Example 2.

Find all sentences that contain a word with a root "val" ($root^{-1}$(val))

```
range of e is text
range of s is text
retrieve (e.all) where
     e.p#=s.p# and e.s#=s.s# and s.t="val*"
```

Conclusion that we have come up with, doing experiments on lexical processing, is that relational systems, INGRES in particular, and our lexical-based approach to handling texts, are very suitable for lexical processing.

4. Editing

Implementing efficient editing operations on texts stored in a described way was not our primary goal. Nevertheless, one may wish to have that possibility on the same structure. Hence, we have run examples that cover nearly all operations of a standard editor. Our intention is not to show the superiority of this approach for editing operations, but only that they are possible.

The conclusion is still that this approach is not suitable for editing operations; a lot of resorting should be done, although most of it can be postponed until the end of an editing session (done off-line). Editing would be better done before source text scanning.

In editing operations we can address either line number and token number inside a line, (illustrated by examples 3 and 4) or we can address lexical unit numbers - token, sequence, sentence, paragraph numbers (example 5).

Example 3.

Substitute first occurrences of the word "values" by the word "value" in lines 5 to 40.

In editor "ed" it is expressed as: 5,40 s/values/value/

In QUEL it is expressed as:

```
range of e is text
replace e(t="value") where e.l#>=5 and e.l#<=40 and
        e.tl#=min(e.tl# by e.l# where e.t="values")
```

Example 4.

Move lines 33 to 35 after the line 47.

In "ed" it is expresses as: 33,35 m 47.

In order to formulate this operation in QUEL efficiently, first some restructuring of the corresponding relation is required (multiplying all line numbers by some factor, ex. 100). Now, query in QUEL has a form:

```
range of e is text
replace l(l#=e.l#/100+4700-32) where
        e.l#>=3300 and e.l#<=3500.
```

Then, we need some kind of reordering of line numbers so that the distance of 100 is preserved between the adjacent lines, and, even more, lexical units resorting - paragraph, sentence sequence and token numbers. Both can be done after the editing session because all informations about texts are contained in the source text that is reflected on the order of lines, tokens inside lines and tokens themselves. So, after the editing session, we can again scan the text that is now stored in a relation, and, according to the informations found there (new paragraph commands, sentence and sequence terminators), to resort paragraph, sentence, sequence and token_in_sequence numbers.

Example 5.

Move third sentence from the first paragraph into third paragraph after its second sentence.

Similar multiplication as in the example 4 has to be done, but on paragraph and sentence numbers. In QUEL, query has the form:

```
range of e is text
replace e(p#=300, s#=201) where
        e.p#=100 and e.s#=300
```

Again, after the editing session we have to reorder p#, s# (instead of 1#, t1# as in example 4), as well as to resort 1# and t1# (instead of resorting lexical informations in example 4).

5. Storage structures

It is obvious that the efficiency of all operations mentioned above is highly influenced by the storage structure.

We have run examples of editing operations on all existing INGRES storage structures. Although we do not have yet reliable timing results, ordered relations seem to be superior, at least in formulating queries. Multi-dimensional ordered relations seem to be the most suitable structure.

Also, both primary and secondary structures (secondary indices) of INGRES are very useful in dealing with lexical processing.

6. Conclusion

This paper presents some initial results on performing lexical text processing in INGRES. Two main areas covered are lexical-based text storing and lexical operations on texts - automatic indexing, abstracting and text search. Experiments are carried out on editing facilities on text

stored in such a way, as well as on storage structures, and results show that, while this approach is especially suitable for lexical operations, editing operations can be done but in less efficient way. Future plans include:

- efficiency investigations in a sense of looking for the most appropriate, new storage structure and some data compression methods;

-bibliographic retrieval;

- design of a structured dictionary containing linguistic informations about words and providing for fact retrieval from the textual database.

References

[BARH 59] Y.Bar-Hillel,The mechanization of literature searching, in National Physical Laboratory,"Mechanization of thought processes", Symposium No.10,vol11,1959,p.791-807.

[BAXE 62] P.B.Baxedale,An empirical model for computer indexing, in Machine indexing, American U., 1962, p.207-218

[BORK 64] H.Borko,M.D.Bernick,Toward the establishment of a computer based classification system for scientific documentation, Rept.no.TM-1763,System development corp.,Santa Monica, Cal, Feb 1964, 47.p

[CARR 81] J.M.Carroll, Content analysis as a word-processing option, in Proc. of the fourth intern. conf. on information storage and retrieval, Oakland, Cal, 1981, ACMSIGIR, vol.XVI, no1, 1981, p.126-131

[INGR 81] INGRES - VERSION 7 Reference Manual, Electronic

Research Lab, University of California, Berkeley, Memorandum No. UCB/ERL M81/61, 1981

[LESK 62] M.Lesk,E.Storm, A computer experiment for sentence extraction, Section I, in G.Salton, Information Storage and Retrieval, Rept.no.ISR-2, Sept 1962, p.I-1 to I-31

[LUHN 58] H.P.Luhn, The automatic creation of literature abstracts (Auto-abstracts), IBM J. Research and Development 2, 1958, p.129-165

[LUHN 60] H.P.Luhn, Keyword_in_context index for technical literature (KWIC Index), In American Documentation 11, 1960, p.288-295

[LUHN 63] H.P.Luhn, ed. Automation and scientific communication, Short papers, Pt.1, American Documentation Institute, Washington, D.C, 1963, p.1-128; also -Pt.2, p.129-384

[MARO 61] M.E.Maron, Automatic indexing: an experimental inquiry, in J. ACM 8, 1961, p.404-417

[ROBE 81] S.E.Robertson, Term frequency and term value, In Proc. of the fourth intern. conf. on information storage and retrieval, Oakland, Cal, 1981, ACMSIGIR, vol.XVI, no1, 1981, p.22-29

[SALT 81] G.Salton, H.Wu, The measurement of term importance in automatic indexing, in Journal of the American Society for Information Science, May 1981, 175-186

[SPAR 74] J.K.Sparck, Automatic indexing, in Journal of Documentation, 30, 1974, p.393-432

[SPAR 76] J.K.Sparck, R.G.Bates, Research on automatic indexing 1974-1976, Computer Lab. University of Cambridge,

England, 1976

[STEV 70] M.E.Stevens, Automatic indexing: A state_of_the_art report, National Bureau of Standards Monograph 91, Washington,D.C, 1970

[STIL 62] H.E.Stiles, Machine retrieval using the association factor, in Machine indexing, American U., 1962, p.192-206

[STON 82] M.Stonebraker, et.al., Document processing in a relational data base system, Electronic Research Lab, University of California, Berkeley, Memorandum No. M82/32, 1982

[SWAN 63] D.R.Swanson, Automatic indexing and classification, NATO Advanced Study Institute on Automatic Document Analysis, Venice, 1963, 4p

[WONG 82] E.Wong, EXQUEL: A semantic extension to QUEL, Electronic Research Lab, University of California, Berkeley, Memorandum No. M82/44, 1982