M. Sibuya
T. Fujisaki
Y. Takao

Noun-Phrase Model and Natural Query Language

Abstract: Basic considerations in designing a natural data base query language system are discussed. The notion of the noun-phrase data model is elaborated, and its role in making a query system suitable for general use is stressed. An experimental query system, Yachimata, embodying the concept, is described.

Introduction

This paper discusses the design of a query processing system, Yachimata, which accepts natural language queries to a data base and displays a data table satisfying the query. The system was developed as a research prototype to study the feasibility of using a natural language, in particular Japanese, to query a data base.

The key idea in the design is a noun-phrase model, which is a variant of the relational data model but is more suitable for representing data in everyday language. In this paper we elaborate the idea of a noun-phrase model and discuss its appropriateness in our system design. An important benefit of the idea is its contribution to the universality of our system, which has a nucleus that is application-independent and for which the application-dependent part can be produced by feeding the necessary data to a generator.

We note some difficulties peculiar to the Japanese language, which is unique in some respects and with which limited experience in machine processing has been had. Finally, the Yachimata system is outlined and its performance is discussed.

Use of natural language

Natural language is attractive for querying a data base because it permits access to the data by casual end-users without the help of data processing professionals, the studying of manuals, or the memorizing of formal rules. But the problems involved in processing natural language are formidable indeed, and many arguments have been advanced for and against research in this area [e.g., 1-5]. In the case of a data base system, however, designers can anticipate possible queries, the vocabulary is limited, and the meanings of words can be assumed to be well understood by all concerned. Thus, we assumed that some things of value could be learned about processing a natural language in this less demanding environment.

System objectives

To make our system suitable for a variety of applications, and at the same time control development costs, we felt that the query processor ought to be made as independent of the application as possible. The part of the system concerned with syntactic and semantic rules is common to all applications. Conversely, the vocabulary to be processed is heavily dependent on each system application, so the meanings of words are defined specifically in relation to the particular data base. Thus, the application-dependent part was designed to be generated automatically so that it could easily be added to the nucleus.

Another objective for our system is that it provide a more formal data sublanguage for data processing professionals. This would enable, for example, data base updating, a rather routine job that should be done with a formal language. Both languages have to run together under the data base management system.

To satisfy these objectives, we need a common view of the data base for all users. Codd's relational data model [6] is the most suitable view for this purpose and also satisfies the first requirement, as we will see later. In fact, the possibility of using a natural language for querying a

NN	C_2	· · ·	C_n
D_1	D_2	· · ·	D_n
e_{11} ⋮ e_{m1}	e_{12} ⋮ e_{m2}	· · · · · ·	e_{1n} ⋮ e_{mn}

NN: a collective noun or noun phrase

$C_2, \cdots, C_n$: case indicators
$D_1, \cdots, D_n$: domain names
e_{ij}: numbers or primitive nouns

Figure 1 Noun table structure.

stock	C5	C8
number	part	warehouse
50	radiator	WH22
120	bumper	WH24
0	radiator	WH24
50	radio	WH24

C5 = {of}
C8 = {at, in}

(a)

	C8
part	warehouse
radiator	WH22
radio	WH24

(b)

	C5	C8
number	part	warehouse
40	radiator	WH22
70	bumper	WH24
20	radiator	WH24
60	radio	WH24

(c)

Figure 2 Examples of noun tables: (a) stock, (b) good selling parts, (c) recommended minimum.

relational data base has been suggested, and Codd himself is developing a conversational query system, REN-DEZVOUS [7]. To allow natural expression of queries, however, we prefer to modify the relational model and use one called the noun-phrase model, which we discuss next.

The noun-phrase model essentially formalizes the semantic rules for data base retrieval. This function is independent of each natural language (English, Japanese, etc.). However, the model does specify possible structures of a query sentence. This specification is, of course, dependent on each language. In the following text, possible structures are roughly sketched for English and Japanese. The English reader might be interested in learning the structure of Japanese sentences. A typical one is "⟨noun⟩ ⟨case particle⟩ · · · ⟨noun⟩ ⟨case particle⟩ ⟨verb⟩." In noun phrases of the form "⟨verb phrase⟩ ⟨noun⟩," the modifying phrase has almost the same structure as the above sentence. Another type of noun phrase is "⟨noun⟩ ⟨case particle⟩ · · · ⟨noun⟩ ⟨case particle⟩ ⟨noun⟩," where case particles are slightly different from those appearing in the above sentence. Nouns do not inflect, and the ordering of the pairs "⟨noun⟩ ⟨case particle⟩" is not essential.

Noun-phrase model

Because nouns and verbs are the basic components of the query language, and because their usage and meanings usually depend heavily on the application, they are specified to the system by the system administrator in terms of the noun-phrase model. For the nouns whose meanings are considered to be primitive in the data base, the administrator need not give the specifications at all. For example, "Bob," "IBM," "bumper," "50" are nouns whose meanings are primitive in some applications. We call them "primitive nouns" here. Conversely, the meaning of a noun such as "stock," representing a set of associations among parts and numbers, is not so simple. We call these nouns "collective nouns," and their meanings must be specified. In order to represent the meanings of collective nouns and verbs, two types of tables, noun tables for the specification of application-dependent collective nouns and verb tables for the specification of application-dependent verbs, are used. A collection of these two types of tables forms a subschema of the data base, i.e., each user's or user-group's derived view of the common data base.

• *Noun table*

Figure 1 shows the structure and Fig. 2 shows examples of noun tables. The noun table is almost the same as a relation in the relational model except that a noun table consists of two different types of columns: the leftmost column and the other columns. The leftmost column implies a set of values which represents the meaning of the noun directly; for example, the set 50, 120, 0, and 50 is considered to be the meaning of the noun "stock" in Fig. 2(a). The other columns, which are optional, further qualify these values; e.g., "radiator" and "WH22" qualify the value "50," and so on.

The domain of each of the columns denotes a set of possible primitive nouns, just as the domain in the relational model, and specifies comparability among the column entries. Case indicators in columns are somewhat similar to "cases" in the case grammar [8] but are used only to specify the possible qualification forms for this collective noun. In our system, all of the case indicators are associated with a list of case particles, each of which is typically a preposition in English. This mechanism corresponds to that of "role" in the relational model.

The noun table "stock" in Fig. 2(a) provides the meaning of the collective noun "stock." In a similar way, all of the collective nouns that are expected to appear in query sentences must have their corresponding noun tables in the data base. These "collective nouns" can actually consist of more than one word like "good selling part" or "recommended minimum" in Fig. 2. This loose definition is useful in applications and very significant in Japanese sentences, as explained later.

Note that, by defining such noun tables, the collective noun itself and its syntax and semantics are provided to the system. For example, "boy" does not refer to every boy in the world but only to those who are defined by a noun table with columns of, say, their towns and ages. Then phrases like "boys in Tokyo of age over 15" make sense, but those like "boys playing football" or "boys in high school" do not. And the above meaningful phrase corresponds to a list of boys' names or IDs. The noun tables exist virtually. In fact, they are defined in terms of data base retrieval functions.

For the primitive nouns, e.g., "bumper," "WH24," in Fig. 2(a), there is no need to prepare the corresponding tables. Since a primitive noun is its own value (the value of radiator is radiator), the system automatically creates one-entry noun tables temporarily for each of the primitive nouns during the initial scanning of the input query. The need for this will become apparent later.

• *Evaluation of a query*

The noun tables stored in the data base and the noun tables temporarily created provide the meanings of the collective nouns and primitive nouns, respectively. From them the meanings of more complex noun phrases can be determined algorithmically. The system has an operator that can be used to determine the meaning of a noun phrase such as "stock of radiators" from the meanings of "stock" and "radiator." "Stock" is in the form of a noun table in the data base, and "radiator" is a noun table created temporarily. Once the meaning of the phrase "stock of radiators" is determined, another operator can be applied to this result to derive the meaning of a more complex phrase such as "the maximum of the stock of radiators." By introducing more operators, this method can be extended to determine the meaning of general noun phrases, e.g., "the location of the warehouse whose stock of radiators is greater than 25." This is sufficient for the evaluation of the queries in our system because this system restricts the possible syntax of the query to the form "Give me ⟨noun phrase⟩." Note also that because our model is based on the relational data base model, we can construct such operators as extensions of the operators in the relational model, thereby achieving application independence.

• *Data retrieval operation*

The typical data retrieval operation is qualification. Since a noun table has two types of columns, the leftmost and the others, we can have two kinds of qualification: direct (qualification of the leftmost column by the others) and reverse (qualification of one of the other columns by the leftmost).

A direct qualification is applied to a phrase consisting of "⟨noun phrase1⟩ ⟨case particle⟩ ⟨noun phrase2⟩," e.g., "stock of radiators." The number suffixes appended to the noun phrases are included only to distinguish among them. (In Japanese, the structure of this phrase is the same except that the sequence is reversed, i.e., "radiators *no* stock," where "*no*" is a case particle corresponding to "of.") This phrase contains two noun phrases, a qualified noun phrase "stock" and a qualifying noun phrase "radiators," which are represented by two noun tables previous to the application of direct qualification. In this case, "stock" is represented in the noun table shown in Fig. 2(a) and "radiators" in a temporary noun table with only one entry. These two noun tables are passed to the direct qualification operation as the arguments, with the case particle "of" ("*no*" in Japanese), which shows the association with the case indicator "C5." The direct qualification operation then generates a noun table as in Fig. 3(a) that is mainly the result of an equi-join operation, as defined in [7], between the two noun tables. (The result of the equi-join of the two tables with respect to a common domain D is a table in which each row consists of a row of the first table concatenated with a row of the second table which contains the same D-value.) The column used for the equi-join operation is determined by the case particle "of," which implies the case indicator "C5," and by the domain of the qualifying noun phrase "radiators," which implies the domain type "part."

Since the result of direct qualification is itself a noun table, it can be further qualified in other columns. "Stock of radiators in WH22" is an example [Fig. 3(b)]. By the case particle "in" and the primitive noun "WH22," the column used for the qualification is determined in the same way as in the previous example. The order of qualification within a table is not essential and the phrase "stock in WH22 of radiators" is acceptable and has the

	C5	C8
number	part	warehouse
50 0	radiator radiator	WH22 WH24

(a)

	C5	C8
number	part	warehouse
50	radiator	WH22

(b)

	C5	C8
number	part	warehouse
50 50	radiator radio	WH22 WH24

(c)

Figure 3 Examples of direct qualification: (a) stock of radiators, (b) stock of radiators in WH22, (c) stock of good selling parts.

	C8
part	warehouse
bumper	WH24

(a)

	C8
part	warehouse
radiator radio	WH24 WH24

(b)

Figure 4 Examples of reverse qualification: (a) part whose stock is more than 80, (b) part whose stock is less than recommended minimum.

same meaning as above unless a definite order is specified by some grammar rule. When the noun tables corresponding to the qualified and qualifying noun phrases have columns with the same combination of case indicator and domain type, the equi-join operation is applied with respect to columns including these so that the result table is meaningful. For example, given the noun tables "stock" and "good selling parts" of Fig. 2(a) and (b), respectively, then the direct qualification "stock of good selling parts" involves the (C8, warehouse) columns of both the tables and leads to the table of Fig. 3(c).

Reverse qualification is applied to a phrase consisting of "⟨noun phrase1⟩ whose ⟨noun phrase2⟩ ⟨comparing operator⟩ ⟨noun phrase3⟩." The ⟨comparing operator⟩ stands for, for example, "is," "are," "is greater than," or "<." (In Japanese, we have almost the same structure except that the order is reversed, i.e., "⟨noun phrase2⟩ ⟨particle⟩ ⟨noun phrase3⟩ ⟨auxiliary verb⟩ ⟨noun phrase1⟩." In this case, the combination of ⟨particle⟩ and ⟨auxiliary verb⟩ corresponds to the ⟨comparing operator⟩ in the English structure.)

The reverse qualification operation is accomplished by doing an equi-join operation between the leftmost columns of the two noun tables indicated by ⟨noun phrase2⟩ and ⟨noun phrase3⟩. After the join operation, columns are selected from the result of the operation so as to form a new noun table having the same structure as the noun table indicated by ⟨noun phrase3⟩. Figure 4(a) shows an example using the phrase "parts whose stock is greater than 80," and Fig. 4(b) shows an example "parts whose stock is less than the recommended minimum" based on the tables of Fig. 1(a) and (c).

In addition to these two qualification operations, union, intersection, and difference operations are provided to support queries with conjunction and negation such as: "Show me the maker who supplies bumpers or radiators to warehouse WH24 and is located in New York." Note that all of these operations are expressed in terms of relational algebraic operations.

Arithmetic operations such as "+," "×," "number of," "average of," etc., are also supported in this system so as to allow queries such as: "Show me the price × stock of parts in WH24."

Note that the discussion in this section is similar to Heidorn's work [9] on resolution of noun phrases. The setups are different, however, although both are trying to relate phrase structure and formalized meaning.

• *Verb table*

Figure 5 shows the structure and Fig. 6 shows an example of a verb table. It differs from the noun table in that it has no special column in the leftmost position. The roles of the case indicators and the domains are the same as those of noun tables.

The purpose of the verb table is to allow a verb to appear in queries and thus increase the fluency of the query language. Because in Japanese some adjectives are used

in a way similar to verbs, verb tables are also used for introducing adjectives into the query language.

In our system, the use of verbs is restricted to those which are used in the context of a noun phrase construction; for example,

"supplied part from ⟨maker⟩ to ⟨warehouse⟩,"

"supplied warehouse from ⟨maker⟩ with ⟨part⟩,"

"supplying maker of ⟨part⟩ to ⟨warehouse⟩,"

are typical uses of the verb "supply" in this system. These are rather limited expressions in English verb usage, but correspond well to proper Japanese phrases. Since the above phrases have the same structure, verbs in this form are easily handled. That is, the verb table "supply" in Fig. 6 is essentially equivalent to three noun tables "supplied part," "supplied warehouse," and "supplying maker" in Fig. 7. Then a phrase including verbs can be handled by the previously described direct qualification operation just as noun tables are. For example, a phrase "supplying maker of radiators" ("radiators *o* supply *suru* maker" in Japanese) can be treated as the direct qualification of "supplying maker" by "radiator" with "of" as the particle.

Considerations for achieving fluency

To achieve fluency in the query language, which is essential to the user of this system, some additional ideas are incorporated.

Contextual reference After the system produces the result of a query in the form of a noun table, the table is put into memory so that it can be referred to in the next query by using demonstrative pronouns, such as "that," "those," etc. "Give me the price of that," coming after the query "Give me the parts which are supplied from XYZ," can be handled in this way.

It is also possible to refer to the result of previously issued queries by using constructs like "that ⟨noun phrase⟩," and "those ⟨noun phrase⟩." In such cases, from among the query results previously stored, the most recent one that has the same domain type as the noun phrase is substituted as the meaning of the phrase. For example, after the above two queries, "Give me the stock of those parts in WH24" is allowed, and the phrase "those parts" in this sentence means the result of the first query.

Flexibility in tables The system allows arbitrary noun and verb phrases as the names of noun and verb tables, respectively. Therefore, in order to let the system understand the phrase "quantity on order," it is possible to create a noun table whose name is "quantity on order." It is much simpler to use a phrase than to define all of the terms separately: "quantity," "on," and "order."

VV

C_1	C_2	...	C_n
D_1	D_2	...	D_n
e_{11} ⋮ e_{m1}	e_{12} ⋮ e_{m2}	... ...	e_{1n} ⋮ e_{mn}

VV: a verb
$C_1, \cdots, C_n$: case indicators
$D_1, \cdots, D_n$: domain names
e_{ij}: numbers or primitive nouns

Figure 5 Verb table structure.

supply

C3	C7	C10
maker	warehouse	part
AA BB AA CC	WH22 WH24 WH24 WH26	radiator bumper radio radio

C3 = {by}
C7 = {to}
C10 = {∅}

Figure 6 Example of a verb table.

Figure 7 Noun tables generated from example verb table: (a) supplied part, (b) supplied warehouse, (c) supplying maker.

supplied part	C3	C7
part	maker	warehouse

(a)

supplied warehouse	C10	C3
warehouse	part	maker

(b)

supplying maker	C7	C10
maker	warehouse	part

(c)

Moreover, one of the possible ways to allow a noun modifier like "established in 1972" for a noun "company" is to treat the term "established in" just as a case particle. Then the system can handle the phrase "company established in 1971" as though it had the structure of ⟨noun phrase⟩ ⟨case particle⟩ ⟨noun phrase⟩, which can be handled simply by the direct qualification operation.

Redundancy Since the data base of our system is designed on the assumption that all of the tables, noun tables and verb tables, are expressed as subschemata of the common data base, the data base can be logically redundant without physical redundancy, and this fact greatly increases the fluency of the language. For example, a noun table "supplier" and a verb table "supply" might be redundant, because they are just alternative expressions for a single fact. Nonetheless, they can coexist in the data base as two different tables to allow both of the following queries:

"Give me suppliers of radiators to WH24."

"Give me supplying maker of radiators to WH24."

In the former case, the noun table "supplier" gives the answer to the query, and in the latter case, the verb table "supply" gives the answer to the query. Some experimental systems of today realize such functions based on a knowledge base using semantic networks or other structures. The knowledge base of common sense which is independent of applications, however, is very complicated, and it is heavily application-dependent in present systems. Sowa [10] has discussed how much automatic reference will be possible when the relationships among concepts in a data base are formalized as his conceptual graphs. In our system, logically redundant definitions of words meet partially the requirements of inference. For example, a hierarchy of concepts can be defined rigorously. We believe that the meaning of words should be defined by the data base administrator, and that is done more or less during the data base design and redesign process.

Ambiguities The model can accept ambiguous specifications for words. In order to allow both of the queries, "Give me the income of Bob in 1971" and "Give me the income of IBM in 1971," two different noun tables having the name "income" can be present. In spite of this ambiguity, the system can reply to both of the queries correctly. However, a query like "Give me the income in 1971" is ambiguous, and no one can decide which table should be used. To cope with such situations, a "disambiguation" dialogue mechanism is incorporated in our system. For this example, the mechanism would display a menu showing all of the possible interpretations, such as

Are you asking about the "income of a person"?

Are you asking about the "income of a company"?

The mechanism is also invoked when there are more than two possible ways of parsing. Negations and conjunctions often cause this to happen. To resolve such ambiguities, the system first tries to use both contextual information and domain information. If it fails in doing that, the mechanism is invoked to analyze and display all possibilities.

Application system generator

One of the requirements of a practical data base query system is to separate the lexicon and the data base completely. This separation makes it possible for the system to do complicated parsing without looking into the data base; thus the system can respond to queries in reasonable times. However, the separation of the lexicon and the data base raises a problem: someone must prepare the lexicon. To reduce the effort, the application system generator is a program which creates the lexicon from the data base automatically.

Once the noun and verb tables, expressed by mappings to the physical data base, are provided to the system, the generator extracts the following information from each of the tables:

1. A collective noun or verb from the name of the table.
2. Possible qualification forms for the collective noun or verb from the column headers of the table.
3. Primitive nouns and their domains from the body of the current table.

In this way, by scanning all of the tables provided, the generator can create all of the lexicon entries for the verbs, the collective nouns, and the primitive nouns. After that, the generator prompts the administrator for the associations between the case indicators and the case particles. From these associations, the system creates the lexicon entries for the case particles.

• *Generation steps*

The generation of a Yachimata system can be summarized in the following three steps, provided that a relational data base exists.

Step 1 The administrator determines what kinds of verbs and collective nouns are to be allowed in the query language. Also he must decide the possible qualification forms for each of them. This will determine the necessary tables as well as their structures.

Step 2 The administrator describes each of the tables in the following form:

a. Table name: This is the collective noun or the stem of the verb.
b. Column header: This is a list structure consisting of the domain names and the case indicators describing

the possible qualification forms for the collective noun or the verb.

c. Body as an expression: The entries of the table should be defined in terms of a data sublanguage as a mapping from the data base.

Step 3 All of the specifications above are fed to the generator to derive the lexicon.

Processing Japanese sentences

So far we have discussed the use of natural language in general. We now consider some difficulties peculiar to machine processing of the Japanese language. The most rudimentary one is the input of Kanji characters. There is no good device available at present for us to key in thousands of Kanji characters, and we foresee none in the future suitable for inexperienced casual end-users. Thus we are limited to Katakana characters, which causes trouble for the parser: first, the word separation rule in Katakana sentences is flexible, and second, there are a lot of homonyms in Katakana words. The first problem is something like writing both "data base" and "database" for a lot of words. A compound noun can be written as one word or as several words, so that noun tables must be given phrase names.

Ambiguities due to homonyms among nouns are resolved mainly by noun and verb tables, since they restrict possible words in a syntactic structure. For example, in "*Kohchi no kohchi*" (the cultivated area of Kohchi Prefecture), one Kohchi can be a common noun while the other is a proper noun (they are distinct in Kanji, and there is no distinction between upper and lower case letters in Katakana). To allow flexibility in the word separation rule, we have to indicate the position of negligible spaces in the words in the lexicon on the one hand, and match the words with all possible substrings of segmentations of the query sentence on the other hand.

Another fundamental difficulty is due to the fact that Japanese is not an inflectional but an agglutinative language. This means syntactic rules play a less important role and semantic ones a more important role in parsing Japanese sentences. This, as well as the flexibility in separation rules, requires application of rewriting rules more often. Because of the limited universe of discourse, this difficulty is not insurmountable, but it increases parsing time.

It is interesting to observe that Japanese queries fit the noun-phrase model quite well. Since the role of case particles and semantics is stronger and the word order is almost meaningless, a case indicator in a verb table can be just a set of strings to be matched and does not mean the position in verb patterns. Further, matching with the pair (domain, case indicator) of columns of tables plays an important role in syntactic analysis. The syntax of a phrase with a verb in the passive voice is almost the same as that in the active voice. Thus, the passive voice of a verb becomes available if another verb table is added.

Yachimata

Yachimata, named after Haruniwa Motoori's grammar book *Kotoba no Yachimata* (*Maze of Language*), published in 1808, is an experimental system which incorporates the ideas of the previous sections and has been running successfully since the spring of 1976. Yachimata accepts a Japanese-like query sentence keyed with an IBM 3270 Information Display System with the Katakana Character Feature, asks the user questions to resolve ambiguities or determine output format, if necessary, and shows the result in tabular form. It is coded almost entirely in PL/I, runs under IBM VM/370: Conversational Monitor System (CMS), and supports multiple users. Its data base is PRTV [11], and its parser and semantic analyzer are similar to those of REL [1]. Running in a typical IBM System 370/Model 168 CMS environment, the response time is 10–20 seconds for ordinary queries and 30–50 seconds for longer or more complex queries. The virtual CPU times are 2–5 seconds and 10–15 seconds, respectively. The system uses 1.5 megabytes (where 1 megabyte equals 1024^2 bytes) of virtual storage for program, 1.5 megabytes for rules and lexicon, and 0.5–1.5 megabytes for work area.

It is difficult to assess Yachimata's fluency in grammatical terms for non-Japanese languages. We believe that it covers negation, comparatives (greater/smaller only), passives, relative clauses, coordination, and mathematical expressions. It partially covers time and connected sentences.

To demonstrate Yachimata's generality, three small applications have been implemented: 1) inventory of parts in warehouses, a typical example of the relational data model; 2) domestic airline schedule, a typical example of the question answering system; and 3) regional statistical data by prefecture. The details of these applications and sample sentences have been published elsewhere [12, 13].

The last application was demonstrated to thousands of people at the World Environment Exhibition, held in Tokyo in May–June 1976. When people understood what data were stored, they asked reasonable questions and obtained satisfactory answers. A few DP professionals asked essentially difficult questions. For the demonstration three college-educated typists knowing nothing about DP systems were hired. They were shown sample queries and were taught the valid sentence structures and the data base contents for three hours. Then, after two afternoons of self-training with a guide, they could freely ask Yachimata questions.

The Yachimata system is similar to its two sister systems in German and French [14, 15]. In addition to the

language involved, there are other differences. We do not expect the user to alter the grammar rules himself. Since the grammar rules are related to one other, it is difficult to alter some without understanding all of the rules. Our main efforts were directed towards simplifying the system generation, based on the explicit definition of the noun-phrase model. A DP professional can construct it easily with the help of a system programmer to run the application system generator. At present, only the query "⟨noun-phrase⟩ *wa*? (Show me ⟨noun-phrase⟩.)" is implemented. The case particle "*wa*" is a unique one indicating the topic of a sentence. Some improvements to the fluency have been designed but not implemented.

Conclusion

There was widespread doubt that Japanese (especially in Katakana sentences) could be an adequate query language. One of the achievements of Yachimata was to show Japanese to be as good as European languages for querying a data base. Our experiences show that, if a user of an information system within a community knows something about his data base, he can learn quickly how to ask the system what he wants to know.

The main difficulty is the cost: parsing time and work space. This will be justified when the cost becomes sufficiently low and the number of end-users who prefer a natural language to a formal one becomes sufficiently high.

References

1. F. B. Thompson, P. C. Lockeman, B. Dostert, and R. S. Deverill, "REL: A Rapidly Extensible Language System," *Proceedings of the Twenty-fourth National Conference of the ACM*, New York, 1969, p. 399.
2. W. A. Woods, R. M. Kaplan, and B. Nash-Weber, "The Lunar Sciences Natural Language Information System," *Final Report BBN 2378*, Bolt Beranek and Newman Inc., Cambridge, MA, 1972.
3. W. J. Plath, "REQUEST: A Natural Language Question-Answering System," *IBM J. Res. Develop.* **20,** 326 (1976).
4. J. Mylopoulos *et al.*, "TORUS: A Step Towards Bridging the Gap Between Data Bases and the Casual User," *Info. Syst.* **2,** 49 (1976).
5. S. R. Petrick, "On Natural Language Based Computer Systems," *IBM J. Res. Develop.* **20,** 314 (1976).
6. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Commun. ACM* **13,** 377 (1970).
7. E. F. Codd, "Seven Steps to RENDEZVOUS with the Casual User," J. W. Kinbie and K. L. Koffeman, eds., North-Holland Publishing Co., Amsterdam, 1974, p. 179.
8. C. J. Fillmore, "The Case for Case," *Universals in Linguistic Theory*, E. Bach and R. Harms, eds., Holt, Rinehart and Winston, New York, 1968, pp. 1–89.
9. G. Heidorn, "Supporting a Computer-Directed Natural Language Dialogue for Automatic Business Programming," *Research Report RC6041*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1976.
10. J. F. Sowa, "Conceptual Graphs for a Data Base Interface," *IBM J. Res. Develop.* **20,** 336 (1976).
11. S. Todd, "PRTV—An Efficient Implementation for Large Relational Data Bases," *Proceedings of the International Conference on Very Large Data Bases* **1,** No. 1, 554 (1975).
12. T. Fujisaki, M. Sibuya, and Y. Takao, "A System Querying Regional Statistics—Practice of Noun-Phrase Data Model," presented at International Technical Conference on Relational Data Base Systems, June 23–25, 1976, IBM Italy Scientific Center, Bari, Italy.
13. T. Fujisaki *et al.*, "A Data Base Query System Using Japanese-Like Language: Yachimata," *IBM Review*, No. 63, 69 (1976), IBM Japan, Tokyo (in Japanese).
14. H. Lehmann, "Interpretation of Natural Language in an Information System," *IBM J. Res. Develop.* **22,** 560 (1978, this issue).
15. O. Bertrand, "NLS: A Natural-Like Language System," International Conference on Relational Data Base Systems, IBM Bari Scientific Center, Bari, Italy, June 23–25, 1976.

Received September 6, 1977; revised May 3, 1978

The authors are located at the IBM Tokyo Scientific Center, 1-11-32 Nagata-cho, Chiyoda-ku, Tokyo.