A SCHEME-DRIVEN NATURAL LANGUAGE QUERY TRANSLATOR

David W. Embley** ++
Brigham Young University
Provo, Utah 84602

Roy E. Kimbrell**
Planning Research Corporation
Bellevue, Nebraska 68005

Abstract

An approach to natural language query translation is presented that is driven mainly by the semantics contained in an extended database scheme. This approach has the advantage of ease of implementation and thus portability since the scheme can easily be extended to interface with the translation system's natural language understanding modules. The required extensions consist of adding domain specific routines to recognize and classify literals and a lexicon to recognize context keywords. The results from these recognizers are then presented to a domain independent translator for further analysis. A prototype system has been implemented and some initial experimentation has been done. Observations about the effectiveness of the translator and its efficiency are reported.

1. Introduction

Processing database queries is an important application for natural language understanding systems. There is a recognized need for natural language communication especially for database users who are noncomputer specialists [Blanning 84, Kelly 84]. This need, coupled with the increased likelihood of successfully interpreting database queries because of the limitations imposed on the domain of discourse, has encouraged several researchers to develop natural language query processors [e.g., Woods et al. 72, Waltz 76, Harris 77, Hendrix et al. 78, Codd 78, Ballard et al. 84]. With varying degrees of success, these systems apply techniques of natural language processing [Tennant 81] to analyze requests and generate formal database access queries.

Like these systems, Mnemos* generates database access queries from natural language input. Mnemos differs from these systems in its direct use of the database scheme as its knowledge base for natural language understanding. This approach has the advantage of ease of implementation and thus portability since the scheme can easily be extended to interface with the natural language understanding modules of Mnemos. In this regard Mnemos is similar to CO-OP [Kaplan 84] and INTELLECT [Harris 77, AIC 82], but differs from these systems in the details of its approach.

This paper reports our investigation of the Mnemos approach for interpreting natural language queries. In Section 2. specifics about the scheme requirements are stated and the interpretation generation routines are explained. A prototype Mnemos interpreter has been implemented, and the extent to which it understands queries posed by naive database users and the speed at which it executes have been assessed. The implementation and observations about its use and its potential are discussed in Section 3.

** Part of this work was done while we were at the University of Nebraska-Lincoln.

++ This material is based in part on work supported by the National Science Foundation under Grant No. MCS-8219941.

*Mnemosyne, from which the name Mnemos ('ne mos) was taken, was the goddess of memory in Greek mythology. One theory of how human memory is used to understand natural written communication corresponds to how Mnemos uses computer memory to interpret database queries [Anderson and Bower 73].

2. Mnemos

Traditional database schemes have been augmented in various ways to capture more information about the organization or system being modeled [Smith and Smith 77, Codd 79, Borkin 80]. The augmented scheme for Mnemos is comparatively simple. To a large extent, the scheme is a reorganization of information already part of the scheme or database support system.

Mnemos assumes that the database scheme is relational and is derived from an entity-relationship model [Chen, 76]. As an addition to the entity-relationship model, each attribute, entity, and relationship is described by a data frame [Embley 80, Khan et al. 82]. A data frame encapsulates knowledge about the appearance, behavior, and context of a data element or collection of data elements. Specific information about the written appearance of data-element literals, about applicable operations that can be performed on the data elements, and about words that commonly refer to or are found in context with data elements, collections of data elements, or applicable operations are all contained in data frames.

For Mnemos the role of domain descriptions for the data elements in the scheme is substantially increased. Instead of merely specifying the type generally as one of integer, real, or character string, data elements are defined with more restrictive types such as dollar amount, social security number, account number, department name, and date. Narrowly defined input routines recognize and classify literals so that $21.43, for example, is associated with the dollar amount data frame and 630-75-4485 with the social security number data frame. Context keyword recognizers properly associate words and phrases with data frames, for example amount, cost, price, subtotal, and total with the dollar amount data frame. Data frames for entities and relationships also contain context keyword recognizers that, for example, would associate teacher, instructor, and professor with a data frame for the entity faculty member.

It is the ability to recognize and classify literals, words, and phrases and to make sense of their meaning in the context of an entity-relationship model that allows Mnemos to interpret natural language queries. Once Mnemos has made an initial classification of unit words and phrases in a query, an attempt is made to combine these interpreted low-level units into high-level units. This is done in one of two ways:

(1) match operators with operands, and

(2) embed low-level interpretations in a graph of the entity-relationship diagram.

Unit words and phrases such as "greater than", "total", and "how many" refer to operators. Each operator expects certain operands: "greater than" demands two quantities that can be compared, "total" requires two or more quantities that can be summed, and "how many" needs a column in a relation so that the distinct values in the column can be counted. When a query includes an operation, the operands should appear either explicitly as literals or implicitly as references to attributes in the database where values can be found. The data frame in which an operator resides contains information about the operator's operands and thus enables operator-operand matching.

As an example of matching operators with operands to formulate a database request, consider the query

> Give me the names of employees in the accounting department whose salary is greater than $50,000

in a database with a relation

> EMPLOYEE(ID#, NAME, ADDRESS, DEPARTMENT, SALARY).

Figure 1 shows a low-level interpretation that would be generated by Mnemos. Given this low-level interpretation, Mnemos

Word or Phrase	Association	Properties
Give		Common Word
me		Common Word
the		Common Word
names	NAME	Data Frame: Name Context Keyword Attribute of EMPLOYEE
of		Common Word
employees	EMPLOYEE	Relation Context Keyword
in		Common Word
the		Common Word
accounting	DEPARTMENT	Data Frame: Department Literal Attribute of EMPLOYEE
department	DEPARTMENT	Data Frame: Department Context Keyword Attribute of EMPLOYEE
whose		Common Word
salary	SALARY	Data Frame: Dollar_Amt Context Keyword Attribute of EMPLOYEE
is		Common Word
greater than	>	Data Frame: Dollar Amt Context Keyword Operator
$50,000	DOLLAR_AMT	Data Frame: Dollar_Amt Literal Attribute of EMPLOYEE

Figure 1. A Sample Low-level Interpretation Involving an Operator.

would combine SALARY, >, and $50,000 into the expression

```
SALARY > 50000
```

by matching interpreted fragments from the sentence with expected operands for the greater-than operation. This interpretation is now ready to be converted into the expression

```
SELECT    NAME

FROM      EMPLOYEE

WHERE     SALARY > 50000 AND
          DEPARTMENT = 'ACCOUNTING'
```

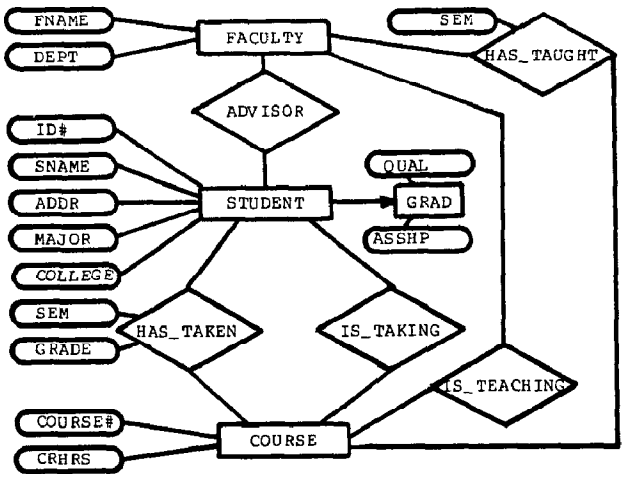
which can be passed to a database management system to retrieve the results.

The second method of creating high-level interpretations from low-level interpretations is by an embedding in an entity-relationship diagram. By considering entities and relationships as nodes in a graph and by marking nodes that are referenced in low-level interpretations, contiguous paths linking marked entities and relationships can be observed.

For example, consider the query

> How many students who have Dr. Jones for an advisor are taking CS352?

in the context of the entity-relationship diagram and derived relational scheme shown in Figure 2. A low-level interpretation generated by Mnemos for this query is given in Figure 3. Observe that the query covers the nodes FACULTY, ADVISOR, STUDENT, IS_TAKING, and COURSE. Hence, these relations are joined to form the relation from which the results of the query can be obtained. The request sent to the database management system for this query is



FACULTY(FNAME, DEPT)
STUDENT(ID#, SNAME, ADDR, MAJOR, COLLEGE)
COURSE(COURSE#, CRHRS)
ADVISOR(FNAME, ID#)
HAS_TAKEN(ID#, COURSE#, SEM, GRADE)
IS_TAKING(ID#, COURSE#)
HAS_TAUGHT(FNAME, COURSE#, SEM)
IS_TEACHING(FNAME, COURSE#)

Figure 2. Entity-Relationship Diagram and Derived Scheme for a Sample Database.

Word or Phrase	Association	Properties
How many	COUNT	Operator
students	STUDENT	Relation Context Keyword
who		Common Word
have		Common Word
Dr. Jones	FNAME	Data Frame: Name Literal Attribute of FACULTY
for		Common Word
an		Common Word
advisor	ADVISOR	Relation Context Keyword
are taking	IS_TAKING	Relation Context Keyword
CS352	COURSE#	Data Frame: Course_Nr Literal Attribute of COURSE

Figure 3. A Sample Low-level Interpretation Involving Several Relations.

```
SELECT COUNT(ID#)

FROM   FACULTY, ADVISOR, STUDENT, IS_TAKING,
       COURSE

WHERE  FNAME = 'JONES' AND
       COURSE# = 'CS352' AND
       FACULTY.FNAME = ADVISOR.FNAME AND
       ADVISOR.ID# = STUDENT.ID# AND
       STUDENT.ID# = IS_TAKING.ID# AND
       IS_TAKING.COURSE# = COURSE.COURSE#
```

In general, SQL-like queries [Chamberlin et al. 76] are generated from high-level interpretations as follows:

> The WHERE clause contains a conjunction of (1) any explicit boolean functions (e.g., SALARY > 50000 in the first example above), (2) boolean equivalence terms that equate each literal not already combined into a high-level operation and the attribute with which it is associated in the low-level interpretation (e.g., COURSE# = CS352 in the second example above), and (3) the join terms for the relations in the FROM clause.
>
> The FROM clause is a list of all stored relations constituting the path (which may be degenerate if only one node is covered) in the graph of the entity-relationship diagram.

The SELECT clause consists of (1) any nonboolean functions (e.g., COUNT(ID#) in the second example above) and (2) the names of all attributes referenced and on the path in the FROM clause that are not combined into high-level units or boolean equivalence terms in the WHERE clause (e.g., NAME in the first example above).

Although not yet stated, it should be clear that low-level and even high-level interpretation routines can assign more than one meaning to a word or phrase and thus create multiple interpretations and possible ambiguities. Whenever Mnemos recognizes that more than one interpretation can be given to a word or phrase, it creates two or more interpretations, each with one of the meanings. Since this may occur repeatedly, a tree of possible interpretations is generated.

Normally most branches of the interpretation tree can be pruned by an application of domain independent heuristics. These heuristics are similar to those of NFQL [Embley 82] and include maximal involvement of lexical units in the query, minimal path length in the entity-relationship diagram, and preference for high-level over low-level interpretations. The greater than operator in Figure 1, for example, would be found in almost every data frame describing an ordered value set. Because of the possible operands in the context, however, only the greater than operator in the dollar amount data frame would be recognized in a high-level interpretation. Thus, many possible branches would be pruned from the interpretation tree.

Sometimes the request is ambiguous even after the heuristics are applied. For the query "List Dr. Jones' courses", which courses are wanted - those Dr. Jones currently teaches, has taught, or both? When there are several equally valid interpretations, Mnemos can find them all and thus interact intelligently with a user to resolve ambiguity.

There are several advantages of the Mnemos approach to generating interpretations. (1) Ungrammatical sentences are easily interpreted. The query "Students taking CS352, how many Dr. Jones advisor for?", for example, is interpreted in the same way and just as easily as the corresponding query discussed above. (2) Local changes to data frames such as the addition of context keywords or operators do not affect the Mnemos interpretation modules. There are no restrictions about what keyword associations may be created. This allows data frames to be tuned locally without concern for their larger context and enhances portability. (3) Since literal and context keyword recognition routines can operate independently, there is a high degree of natural parallelism than can potentially be exploited by advanced-computer architectures.

There are also disadvantages; these are pointed out in the next section. These advantages were enough, however, to encourage us to build and experiment with a prototype Mnemos system.

3. Prototype Implementation and Initial Experimentation

Mnemos was implemented in Pascal on a Cyber 170/730 [Kimbrell 82]. Most of the intelligence of Mnemos is contained in routines that operate on a file of data frames. For this implementation these routines are a collection of deterministic finite automata that recognize lexical patterns representing the literals, context keywords, and token patterns of the data frames. This approach increases portability since lexical analyzer generators and compiler-compliers such as LEX [Lesk 75] and YACC [Johnson 78] can be used to help augment the database scheme for use with Mnemos. A variation of LEX was used in the implementation.

In the implementation patterns are run sequentially against all partial interpretations. Initially, the only partial interpretation is the text of the input query. After all low-level patterns are exhausted, high-level patterns attempt to embed partial interpretations into the given entity-relationship diagram. For this implementation operator-operand patterns are not recognized.

For our initial experimentation the Student-Instructor-Course database shown in Figure 2 was used. Fourteen students in an undergraduate database systems class were asked to write English language queries for this database, which was described only as one containing information on faculty, students, and courses at a university. The queries were to be phrased as questions the students might ask of someone who had information on faculty, students, and courses. Because little was specified about the exact contents of the database, many of the queries asked for information not in the database (e.g., meeting places for courses).

Before submitting the queries for analysis by Mnemos, the literals of many queries were modified to fit the limited domains recognized by the initial prototype version. For example, class identifiers were changed to two letters followed by three digits, department names were limited to Computer Science, Anthropology, and English, all faculty names were prefaced by Dr. or Prof., and only three student names were used, Tom, Dick, and Harry.

Except for names, these modifications were not significant either because the domain size is small so that literals can be exhaustively listed in a data frame (e.g., department name) or the pattern has only a few standard variations (e.g., student id#'s and course identifiers). If names are prefaced by titles, they are easily recognized and can often be further classified (e.g., as faculty member). If the number of names is expected to be small, a list can be stored in the name recognizer of a data frame, but large lists would need to be stored in the database resulting in obvious inefficiencies. As a compromise, a heuristic that yields good results is to classify a word as a name if it is not a common word (standard lists are available) and not classified by some other data frame.**

Of the 139 queries submitted, 134 were processed (5 were not processed because of a failure at the operating system level). Of these, 32 were outside the scope of the information contained in the database (e.g., "How old is the oldest student?"). Of the remaining 102, Mnemos generated correct responses for 51%. Examples of queries not correctly interpreted include "Does every student have an advisor?" and "What students are enrolled in CS400?". Of these incorrectly interpreted queries, 78% (39 of 50) would have been properly interpreted had the context keywords in the data frames been more complete. For example, "What students are enrolled in CS400?" would have been properly interpreted if "enrolled" would have appeared as a context keyword associated with the relation IS_TAKING. Thus, the total correct responses generated after some initial tuning would have been 89%.

Understanding the remainder of these improperly interpreted queries would have required knowledge beyond that of Mnemos. For example, Mnemos thinks it should return a list of students who have advisors for the query "Does every student have an advisor?". It does not recognize that the question concerns the existence of the complement of this list.

** "My name is Alice, but -- "

"It's a stupid name enough!" Humpty Dumpty interrupted impatiently. "What does it mean?"

"Must a name mean something?" Alice asked doubtfully.

"Of course it must," Humpty Dumpty said with a short laugh: "my name means the shape I am -- and a good handsome shape it is too. With a name like yours, you might be any shape, almost."

-- From Lewis Carroll's Through the Looking Glass

In addition to correctly interpreting natural language queries, the time required is also important. Mnemos was designed to run on a machine capable of supporting many routines executing in parallel. Implemented serially, it interprets requests slowly, but the pattern matching routines can and should all run independently.

To obtain some idea of how long it might take to interpret a query on an advanced-architecture, parallel machine, the code was instrumented to count calls and time procedure execution. About 90% of the calls and 93% of the run time were attributed to only three procedures. These three procedures are precisely those that would normally execute in parallel. If these routines were run in parallel, the run time could be reduced considerably. Using the timing values obtained, it was estimated that the average query submitted to Mnemos in the initial experiment would take about 10 CPU seconds to interpret. Further reductions would be likely in an efficiency-conscious implementation, and it is expected that respectable response times for interactive operation can be achieved.

4. Concluding Remarks

Experience to date is encouraging. Mnemos performs somewhat satisfactorily even in its current, rudimentary state. More significantly, it has been shown that a natural language translation system driven mainly by the semantics of a database scheme augmented by domain specific data frames is worthy of serious consideration and can serve as the basic framework for interpreting natural language queries. Although much remains to be done, there are positive indications that the Mnemos approach to natural language query understanding may prove to be acceptable.

References

Anderson, J.R. and Bower, G.H., Human Associative Memory, V. H. Winston & Sons, Washington D.C., 1973.

Artificial Intelligence Corporation, INTELLECT Query System: User's Guide, Waltham, Massachusetts, January 1982.

Ballard, B.W., Lusth, J.C., and Tinkham, N.L., LDC-1: a transportable, knowledge-based natural language processor for office environments, ACM Transactions on Office Information Systems, Vol. 2, No. 1, January 1984, 1-25.

Blanning, R.W., Conversing with management information systems in natural language, Communications of the ACM, Vol. 27, No. 3, March 1984, 201-207.

Borkin, S.A., Data Models: A Semantic Approach for Database Systems, MIT Press, Cambridge, Massachusetts, 1980.

Chamberlin, D.D., Astrahan, M.M., Eswaran, K.P., Griffiths, P.P., Lorie, R.A., Mehl, J.W., Reisner, P., and Wade, B.W., SEQUEL 2: a unified approach to data definition, manipulation, and control, IBM Journal of Research and Development, Vol. 20, No. 6, November 1979, 560-575.

Chen, P.P., The entity-relationship model: toward a unified view of data, ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, 9-36.

Codd, E.F., Extending the data base relational model to capture more meaning, ACM Transactions on Database Systems, Vol. 4, No. 4, December 1979, 397-434.

Codd, E.F., How about recently?, in Database: Improving Usability and Responsiveness, B. Shneiderman (ed.), Academic Press, 1978, 3-28.

Embley, D.W., A natural forms query language - an introduction to basic retrieval and update operations, in Improving Database Usability and Responsiveness, P. Scheuermann (ed.), Academic Press, Inc., 1982, 121-145.

Embley, D.W., Programming with data frames for everyday data items, Proceedings NCC 80, Vol. 49, Anaheim, California, May 1980, 301-305.

Harris, L.R., User oriented data base query with the ROBOT natural language query system, International Journal of Man-Machine Studies, Vol. 9, 1977, 697-713.

Hendrix, G., Sacerdoti, E.D., Sagalowicz, D., and Slocum, J., Developing a natural language interface to complex data, ACM Transactions on Database Systems, Vol. 3, No. 2, June 1978, 105-147.

Johnson, S.C., Yacc: yet another compiler-compiler, UNIX(tm) Time-Sharing System: UNIX Programmer's Manual, Vol. 2B, Bell Laboratories, Murray Hill, New Jersey, July 1978.

Kaplan, S.J., Designing a portable natural language database query system, ACM Transactions on Database Systems, Vol. 9, No. 1, March 1984, 1-19.

Kelly, J.F., An iterative design methodology for user-friendly natural language office information applications, ACM Transactions on Office Information Systems, Vol. 2, No. 1, March 1984, 26-41.

Khan, S.A., Paige, M.R., and Embley, D.W., Reading data items without constraints on form, format or completeness, Proceedings Trends and Applications 1982, Gaithersburg, Maryland, May 1982, 74-82.

Kimbrell, R.E., A database schema driven natural language query translator, Master's Thesis, University of Nebraska, Lincoln, 240 pages, December 1982.

Lesk, M.E., LEX - a lexical analyzer generator, Bell Laboratories Technical Report #39, October 1975.

Smith, J.M. and Smith, D.C.P., Database abstractions: aggregation and generalization, ACM Transactions on Database Systems, Vol. 2, No. 2, June 1977, 105-133.

Tennant, H. Natural Language Processing, PBI-Petrocelli Books, Inc., Princeton, New Jersey, 1981.

Waltz, D.L., An English language question answering system for a large relational database, Communications of the ACM, Vol. 1, No. 3, September 1976, 526-539.

Woods, W.A., Kaplan, R.M., and Nash-Webber, B., The lunar sciences natural language information system, BBN Report #2373, Bolt, Beranek, and Newman, Inc., Cambridge, Massachusetts, June 1972.