A PORTABLE NATURAL LANGUAGE DATABASE QUERY SYSTEM

by

STEVEN JOHN WHITE

B.Ed.(Sec.), University Of British Columbia, 1980

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF COMPUTER SCIENCE

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

October 1985

© Steven John White, 1985
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date Oct 4, 1985
Abstract

With the increased use of computerized databases, the ability to allow users to access information using natural language is becoming more desirable. There are many natural language systems in existence today. The main problem with these systems is the amount of expertise and effort required to adapt them to a new domain.

The design of a portable natural language front-end to a relational database is described in this thesis. It is designed so that a typical Database Administrator can implement a new domain in a reasonable amount of time. Database portability has been achieved by separating the domain dependent natural language definitions from the domain independent definitions. These domain dependent definitions are specified in the database schema, which is structured to extract the semantics contained in the structure of the actual database. A rich supply of standard definitions are available to both aid in the development of the database schema and to help force consistency amongst database domains.
# Table of Contents

Abstract ................................................................. ii
List of Figures ......................................................... v

Chapter I
INTRODUCTION ........................................................... 1

Chapter II
CURRENT NATURAL LANGUAGE QUERY SYSTEMS ......................... 5
  2.1 Early Natural Language Systems .................................. 6
  2.2 Syntactic/Semantic Based Systems ............................... 7
  2.3 Semantic Based Systems ......................................... 10
  2.4 Commercial Systems ............................................. 12
  2.5 Enhancing Syntax With Semantics ............................... 15
  2.6 Summary .......................................................... 22

Chapter III
CONCEPTS IN NATURAL LANGUAGE SYSTEMS ........................... 23
  3.1 Syntactic Analysis Concepts ..................................... 24
    3.1.1 ATN Networks ................................................. 25
    3.1.2 Vocabulary .................................................. 25
    3.1.2.1 Morphing ................................................ 26
    3.1.2.2 Nouns .................................................... 27
    3.1.2.3 Proper Nouns ............................................ 28
    3.1.2.4 Adjectives .............................................. 28
    3.1.2.5 Verbs ................................................... 29
    3.1.2.6 Adverbs ................................................. 30
    3.1.2.7 Prepositions ............................................ 31
    3.1.3 Handling Unknown Words / User Interface .................. 32
    3.1.3.1 Spelling Correction .................................... 32
    3.1.3.2 Abbreviations .......................................... 32
    3.1.3.3 Assumed Database Elements ............................. 33
    3.1.3.4 Ignoring Words ........................................ 34
    3.1.3.5 Knowledge Acquisition ................................ 34
  3.2 Concepts In Semantic Analysis .................................. 35
    3.2.1 Definition Of Semantic Information ....................... 36
    3.2.2 Some Problems In Semantic Interpretation ............... 37
  3.3 Database Interface .............................................. 38
  3.4 Summary ........................................................ 39

Chapter IV
SYSTEM DESIGN I : SYNTACTIC ISSUES ................................ 41
  4.1 Overview Of The System ......................................... 41
  4.2 The Syntactic Processor ........................................ 46
    4.2.1 ATN Parser ................................................ 46
    4.2.2 ATN Grammar .............................................. 46
    4.2.3 ATN Registers ............................................. 48
    4.2.4 The Morpher ............................................... 49
    4.2.5 Parsing A Query ........................................... 51
    4.2.5.1 Noun Phrases ......................................... 52
    4.2.5.2 Conjunction ............................................ 52
    4.2.5.3 Compound Proper Nouns ................................ 53
4.2.5.4 Prepositional Phrases ........................................ 57
4.2.5.5 Relative Clauses ........................................... 58
4.2.5.6 Noise Words ................................................. 59
4.2.5.7 Relaxation Of Grammatical Rules .......................... 59
4.3 Handling Unknown Words ......................................... 60
4.3.1 Abbreviations And Synonyms ................................. 60
4.3.2 The Spelling Corrector ........................................ 61
4.3.3 The User Interface ............................................ 62
4.3.3.1 Spelling Errors .............................................. 62
4.3.3.2 Abbreviations ................................................. 63
4.3.3.3 Synonyms ...................................................... 63
4.3.3.4 Unknown Database Elements ................................. 63
4.3.3.5 Ignoring Words .............................................. 65
4.3.3.6 Cancelling The Query ........................................ 65
4.3.4 Knowledge Aquisition ......................................... 66
4.4 Summary ............................................................ 66

Chapter V
SYSTEM DESIGN II : KNOWLEDGE BASE ............................. 68
5.1 Relational Database Approach .................................. 68
5.2 The Test Databases .............................................. 69
5.3 Relational Database Terminology ................................ 70
5.4 Specification Of Domain Dependent Information .............. 71
5.4.1 The Database Schema ........................................... 71
5.4.1.1 The Database Definition ..................................... 72
5.4.1.1.1 Relation Specification .................................... 72
5.4.1.1.2 Field Specification ....................................... 76
5.4.1.2 Verb Specification ........................................... 80
5.4.1.3 Default Joins ............................................... 82
5.4.2 Database Schema Library ....................................... 84
5.4.3 Inverted Index .................................................. 85
5.5 Specification Of Domain Independent Information ............ 86
5.5.1 General Dictionary ............................................. 86
5.5.2 Global Dictionary ............................................. 88
5.6 Compiling The Information : Active Domain Dictionary ....... 89

Chapter VI
SYSTEM DESIGN III - SEMANTIC INTERPRETATION .................. 91
6.1 Interpreting The Deep Structure ............................... 91
6.1.1 Associating Phrases With Relations And Fields ............. 91
6.1.2 Verb Frames ..................................................... 93
6.1.2.1 Verb Frame Structure ....................................... 94
6.1.2.2 Matching The Deep Structure To A Verb Frame .......... 97
6.1.3 Resolving Multiple Matched Slots ............................ 99
6.1.4 Ignoring Phrases ................................................ 100
6.1.5 Processing Prepositional Phrases .............................. 100
6.1.6 Relative Clauses ............................................... 103
6.2 Building The Standard Query Repesentation ................... 104
6.2.1 SQR Structure ................................................. 104
6.2.1.1 The Select List ............................................ 105
6.2.1.2 The Restriction List ....................................... 105
6.2.1.3 The Relation List ........................................... 105
6.2.1.4 The Join List ............................................... 106
PROGRAM DATABASE ..............................................156
APPENDIX H - SAMPLE PROGRAM DATABASE SESSION .............158
APPENDIX I - DATABASE SCHEMA FOR THE SUPPLIER DATABASE ..174
APPENDIX J - SAMPLE "SUPPLIER/PART/JOB" DATABASE SESSION 179
List of Figures

4.1. Diagram of the System .............................................44
4.2. Sample Queries .......................................................45
4.3. Partial Suffix Table ..................................................51
5.1. Design of the Program Database .................................69
5.2. Semantic CATEGORIES ...............................................75
5.3. TYPES .................................................................79
5.4. L-TYPES ...............................................................80
5.5. Partial Active Domain Dictionary ..............................90
6.1. Verb Frame Slots ....................................................94
6.2. Semantic Classifications for Verb Frame Slots .............95
6.3. Semantic Association Factors .................................102
8.1. Sample Queries to Supplier/Part/Job database ...........128
Acknowledgement

I would to thank Dr. Richard Rosenberg for his support and input into this thesis. I would also like to thank Dr. Micheal McRae for his ideas and for reading the manuscript. I also thank Anne Chou for her help in writing the routines for the Morpher routine. I would especially like to thank my father and my mother for their support.
I. INTRODUCTION

The use of computerized databases is growing at a rapid rate. There are now thousands of databases available to the public in a wide range of areas including business, education, entertainment, and medicine. Even though these databases would benefit many people, very few people actually use them. One of the main obstacles inhibiting their use is the level of skill needed. Before they can become part of our everyday life, there will have to be a suitable user interface which would allow even casual users to access them.

There are currently three popular ways in which users access information from these databases. These are by menu-driven systems, query languages, and customized programs. Menu-driven systems present users with a menu of the information available and allows them to make a selection. This method is useful if there is little information in the database. However, for larger databases containing information in many areas, the user will have to go through many levels of menus in order to obtain the desired information. Query languages are another popular method for extracting information from databases. The problem with this method is it requires a more sophisticated computer user who has some general knowledge of both the query language and the structure of the database being used. Customized programs are appropriate if the user only wants to access information from a specific database, but it is not appropriate if one wants to explore many different databases.

Another method now being used to access information is
natural language (NL). Natural language processing is an area of Artificial Intelligence (AI) which requires the computer to be able to process natural language. In addition to being used for accessing information from databases, NL is also being used for Machine Translation, Expert Systems, and Intelligent Computer Assisted Instruction. Although NL is an extremely complex area with many unresolved problems, research has advanced enough so that techniques can be applied to the successful processing of a high percentage of database queries.

There are currently numerous Natural Language Systems in existence today. Many of them have arisen from research projects (Woods et al. 1972, Waltz 1978) and have been designed for one specific database only. In the last few years some systems have been introduced as commercial products (Intellect, English, Themis). The major problem with most of these systems is that they require a considerable amount of time and effort to adapt them to a new domain or database. This task must be done by a person with considerable knowledge of NL systems. Even after the initial implementation there is a need for such a person to maintain the database. However, most database administrators (DBA's) do not have this knowledge. In order for such systems to become widely used, they must be designed so that an average DBA can easily maintain them with minimal training.

The major concern in the design of a NL system is the ease with which a new domain can be implemented (domain portability) and whether the system can be adapted to other database systems
(database portability). Many NL systems have achieved some degree of domain portability by using procedures which syntactically analyse queries without referencing a specific database. By extending these ideas to semantic interpretation, it is possible to achieve a higher degree of domain portability. In order to achieve database portability, the system should be designed such that the features of a specific database management system (DBMS) are not deeply rooted in the syntactic or semantic parts of the system. In order to achieve domain portability, the domain independent components have been logically separated from the domain dependent information. The domain independent components form the "linguistic core" (Rosenberg 1980), which includes procedures for parsing the query, analyzing each word, prompting users for any additional information, and forming a response. The domain dependent information, called the "database schema", includes a structured representation of the database, as well as semantic definitions of words which describe the database and the contents of the actual database. This information is then compiled together with the global dictionary, which contains syntactic information about individual words. The result of this compilation is the "Active domain dictionary", which is a core component of the system.

In Chapter 2, a number of other Natural Language Systems are examined, with particular emphasis placed on portability issues. Chapter 3 examines which components of a system are domain independent and shows how the domain dependent
information can be supplied to the system in the form of data (not programs). Chapter 4, 5, 6, and 7 describe a prototype system designed to demonstrate the feasibility of such a system. In chapter 8, the issue of domain portability is tested by adapting the system to another domain. The conclusion and areas for future work are presented in chapter 9.
II. CURRENT NATURAL LANGUAGE QUERY SYSTEMS

The first natural language question answering systems were developed in the early 1960's. These systems were based on simple techniques such as keyword analysis and simple pattern matching of sentences. Although these systems were very limited in the types of queries they answered, they did demonstrate that computers could be used to answer questions in specific domains.

Another approach developed in the late sixties followed a linear paradigm (Rosenberg 1980), where the syntactic and semantic processing were separated into two distinct stages. In these systems the query is initially analysed for its syntactic structure and then passed on for semantic interpretation. The problem with this method is that a purely syntactic parse often produced a large number of ambiguous interpretations, which made the semantic interpretation very complex.

Other researchers in the early 1970's developed systems which were entirely based on semantics. These systems examined queries at the semantic level without initially parsing it at the syntactic level. The most popular type of these systems are semantic grammars (Brown et al 1974; Waltz 1977; Hendrix et al 1978). The main problem with these systems is that they are very domain dependent. Implementing a new domain would usually require considerable work.

Recently, researchers have designed systems where both syntactic and semantic processing coexist (Booth 1983; Kaplan 1979). By enhancing the syntactic parser with semantic information, systems can avoid many unlikely interpretations for
ambiguous queries. These systems also tend to have a higher degree of domain portability than most of the earlier systems.

2.1 Early Natural Language Systems

Most of the early natural language question answering systems were based upon ad hoc techniques and answered questions in very limited domains. The most popular of these programs is ELIZA (Weizenbaum 1966), which imitates a dialogue between a patient and a psychologist. The program is based upon scanning the users input for keywords and matching it with a number of sentence patterns. Other systems such as SIR (...) and Student (...) matched queries to a list of sentence patterns. For example, to parse a typical "How many" question, the system would need a sentence pattern as shown below.

\[
\text{HOW MANY * DOES * HAVE ?}
\]

The problem with these systems is that a separate sentence pattern is needed for every type of query the system can answer. The semantic knowledge needed to answer each query is specific to each sentence pattern and is implicit to the given domain. Hence the task of implementing such a system would involve writing sentence patterns with related semantic information for each type of query accepted to the system.
2.2 **Syntactic/Semantic Based Systems**

Syntactic/Semantic based systems are two pass systems which first analyse the query syntactically, and then, upon a successful analysis, continue with the semantic interpretation. The syntactic component of such systems are commonly based upon Transformational Generative Grammars (Chomsky 1965), where there is a mapping between the "surface structure" and the corresponding "deep structure" of a query. The main benefit of this is that the deep structure captures many similarities and differences between similar queries. For example, the following two queries map into the same deep structure.

```
What programs are run by the Apple?  
The Apple runs what programs?
```

Another benefit of this approach is that the deep structure represents the syntactic category of each word in the query. For instance, in the above query the word "programs" is interpreted as a noun and not a verb.

The most common method for producing the deep structure of a query is by using an Augmented Transition Network (ATN) parser (Woods 1970). The ATN was developed from a finite-state transition diagram, which is a set of nodes connected by
directed, labeled arcs. This system has been adapted to Natural Language Processing by having the arcs refer to lexical or syntactic categories and by associating conditions and actions with each arc.

The semantic component is responsible for taking the deep structure (parse) produced by the parser and translating it into a database query to some database. This resulting query, when executed, would produce the answer to the original NL query. Since most of the intelligence of a NL system is at the semantic level, this component is very complex. One of the best known methods for performing semantic analysis is "procedural semantics" used in the LUNAR system (Woods 1967, Woods 1968). This method involves "pattern-matching" the query to predefined templates. Specific templates or rules would match certain parts of the query, such as noun phrases (N-Rules) or the entire sentence (S-Rules). These rules would then specify the appropriate retrieval components called "semantic primitives", which are essentially programmed subroutines. The major problem with this method is that the database designer would literally have to predict every possible query structure that the system could encounter. For example, each of the following queries require a separate S-Rule and associated semantic primitive:

- AA-57 departs from Boston for Rome.
- AA-57 departs from Boston.
- AA-57 departs from Boston at 8:00 am.
- AA-57 departs from Boston on Monday.
- AA-57 departs from Boston for Rome at 8:00 am on Monday.

The main advantage of this Syntactic/Semantic based model
is that the syntactic component is essentially domain independent. The only domain specific information needed is the syntactic information, such as the syntactic category, of each word associated with the domain. This can be easily obtained from a dictionary or even deduced from its position in the query. Hence when a new domain is implemented, the syntactic component of the system requires little change. Another advantage to this approach is that the semantic component does not need to be aware of the various ways of wording the same query since they are often captured by the syntactic component.

Although there are some benefits of separating the syntax from the semantics, there are some serious disadvantages. One problem is that one query may produce more than one parse. The semantic component would then have to select which parse was intended by the user. Many of these ambiguous parses can be avoided by using semantic information during the parse. For instance, the query:

Does the Rose and Crown pub sell cider?

can be interpreted in a number of ways depending if the parser treats "and" as a conjunction and not as part of the name "Rose and Crown". This type of ambiguity can be avoided if the parser is supplied with semantic information about multi-word proper nouns.

To get around this problem, methods were developed to select the most likely parse. One method is to have a semantic component analyse each parse until an appropriate one is found
(Harris 1977a). Once a specific parse is chosen, it is necessary to inform the user which interpretation was picked. One method used is to generate a paraphrase of the query from the chosen parse (Harris 1977a). Other systems required users to determine the interpretation by examining the internal representation of the query (Woods et al 1972). This method is only appropriate for users who have some knowledge of given representation.

The major disadvantage of this syntactic/semantic method lies in the work needed to adapt the semantic component to another domain. Although the syntactic component of these systems is basically domain independent, the larger and more complex semantic component is domain dependent. For example, to adapt a system based on procedural semantics to a new domain would require almost as much work as that expended in explicating the initial domain. It would also require a person with considerable experience with the system to implement and test the new domain.

2.3 Semantic Based Systems

Semantic based systems interpret natural language at the semantic level without completely parsing it at the syntactic level. The most notable works in this area are "semantic grammars". Semantic grammars are based on semantic categories such as "Plane-Type" or "City", rather than the syntactic categories such as Nouns or Prepositional Phrases. These grammars are often implemented by an ATN, where the arcs refer
to semantic categories or specific words.

There are three well-known NL systems based on semantic grammars. SOPHIE (Brown et al. 1974) is an Intelligent Computer Assisted Instruction (ICAI) program which answers questions and evaluates students hypotheses about faults in an electronic circuit. The NL component of the program recognizes word groups for categories such as "Measurement" or "Voltages". PLANES (Waltz 1977) and LIFER (Hendrix et al. 1978) are both natural language front-ends to databases. Both systems translate the query into a database query which will produce the answer.

The PLANES system (Waltz 1977) is a natural language front-end to a database containing information about airplanes and related flight information. The query is parsed by an ATN which employs semantic categories for each "semantic object" (Plante-type, flight-hours) relevant to the domain. The system also parses some parts of queries (eg. quantifiers) based on the syntactic categories of the words. Unlike most semantic grammar systems, PLANES can interpret passive and other variations of a query without separate rules. Once the query has been successfully parsed, the query structure is matched to "Concept Case Frames", which consist of an "act" (usually a verb) and a number of semantic objects. Once a suitable match is found, the query structure is translated into an interim query, which in turn is translated into a formal query expression.

Semantic grammar based systems have a few advantages over syntactic based systems. One advantage is that they don't require the DBA to specify any lexical or syntactic information
about the words used in the domain. These systems can also be more efficient than syntactic based systems since a parse at the syntactic level is not required.

There are two big disadvantages of these systems. First, they require the DBA to specify all possible types of queries that the system will process. Some systems (LIFER & SOPHIE) even require the DBA to specify the passive form of a query in a separate pattern. Besides taking a lot of time, this could cause different databases to be inconsistent in the forms of queries it processes. Hence some queries could be stated in a number of variations (e.g., passive) while others may not. This obviously will cause difficulty amongst users.

Another major disadvantage is the amount of work needed to be done to adapt the system to another domain. The DBA would have to write a new semantic grammar for each domain. Although some tools have been developed to aid in this process (Hendrix et al., 1978), the task is still time consuming. The DBA also has to supply the system with an additional semantic information for each query type. This may involve specifying a concept frame for each query type as in PLANES, or a number of expressions (query templates) as in LIFER.

2.4 Commercial Systems

In recent years a few natural language systems have become commercially available. These include Intellect and English. Unfortunately, due to the commercial interests, there is little published information about the algorithms used in these
programs.

The most successful commercial natural language system is Intellect (Harris 1977a, 1977b, 1977c). Intellect, formerly called ROBOT, acts as a NL front-end to a number of commercial database systems, including IMS, TOTAL, and ADABAS. The system uses an ATN grammar to generate all possible parses of a query based on syntactic and some database information. All words contained in the database are stored in an "inverted index". This allows the parser to recognize database elements as well as determine the database field(s) to which they belong. The system has features to handle pronoun references and spelling correction. It also has a separate ATN to parse sentence fragments, which is invoked if the initial ATN fails to generate a parse.

Intellect's parser will produce more than one parse when lexical or structural ambiguities occur (Harris 1977a). Many of these extra parses occur when a word belongs to more than one field in the database. For instance, the word "GREEN" in the query "Are there any green cars?" could be either a colour or a surname. Intellect will try to determine the intended meaning by executing both of these interpretations on the database. If only one of these returns information, then that interpretation is chosen on the assumption that users only ask questions about information that is actually in the database. The system informs the users of the chosen interpretation by displaying a paraphrase of the query on the screen. If both queries return information, the user will be queried about the intended
meaning. If neither of the queries return information, the system appropriately answers "NO" to the query.

This method for choosing the intended meaning for a query can be very inefficient for databases with many relations. By supplying the system with information about the structure of the database, it is possible for the system to determine the most likely interpretation. If the system knew that colour is an attribute of a car and that the surname is an attribute of an employee, it could make the logical decision (a green coloured car) for the intended interpretation.

Intellect appears to be the most robust system available. It is currently installed at over 200 sites. It is obviously fairly portable since it takes approximately one week to implement a new domain (Johnson 1984). It also is transportable with respect to the database since it is available for many database management systems (DBMS).

Another commercial system, named English, is produced by Mathematica Products Group for their RAMIS II database system (Mathematica Products Group Inc. 1983; Johnson 1984). English answers "yes/no" questions, "how many" questions, and questions which are answered by a report. It uses paraphrases to inform the user of its interpretation when ambiguities occur. The system uses three dictionaries -- a general English dictionary contains the common vocabulary; the RAMASTER dictionary contains information about the active files and their fields; and an optional dictionary contains information unique to a specific user or application.
The major problem with English is that it does not associate any semantic information with verbs other than command verbs such as "print" or "show". Although it is able to parse different types of verbs, they are ignored in the semantic interpretation. For example, the system cannot distinguish between the following two queries:

Who is teaching Math 100?
Who is taking Math 100?

Although the user would be informed of the system's interpretation through the paraphrasing abilities, such a pitfall is not satisfactory. Too much semantic information is contained in verbs to simply ignore them.

2.5 Enhancing Syntax With Semantics

The main problem in using only lexical and syntactic information in parsing a query is the possibility of obtaining more than one interpretation. Many of these interpretations can be avoided if the parser is enhanced with some semantic information. This will help reduce the complexity of the semantic component by eliminating these extra parses at the syntactic stage.

One system that uses semantics effectively to avoid multiple parses is CO-OP (Kaplan 1979, 1984). The system is basically a two-pass (syntax/semantic) system. The syntactic component uses an ATN grammar enhanced with some semantic information. The semantic component is based on a simple case
system, where a list of possible subjects and objects are given for each verb and preposition.

CO-OP reduces the number of parses the ATN produces by using semantic information to determine which head noun a modifier will be attached to. This process is called modifier attachment. For example, in the query:

Which users work on projects in area 3 that are in division 200?

the phrase "that are in division 200" can modify either "users" or "projects". The system uses two heuristics to decide on the most likely interpretation. The first heuristic is the distance between the modifier and each head noun in the query's surface structure. The other heuristic is the "semantic relatedness" of the terms with respect to the semantics contained in the structure of the database. This is derived by examining how close together these terms are in the database. Although this technique is fairly crude, it is very effective if the structure of the database adequately models the semantics of the domain.

In order to be adapted to new domains, CO-OP requires three sources of information - a database schema, a lexicon, and the database itself. The database schema contains semantic information describing the structure of the database. This includes information on how objects are related to each other (ie. one-to-one, one-to-many relationships) and how closely items are related to each other. The lexicon includes morphological, syntactic, and semantic information about all of the words which are known by the system. To adapt CO-OP to a
new domain, the DBA must construct a new database schema and a new lexicon. The major problem here is that both the domain independent and dependent information is stored together in the lexicon. Hence the DBA must go through the entire lexicon, removing all old domain dependent information and modifying the semantics associated with some of the domain independent information. It would make better sense to physically separate the domain dependent from the domain independent information. This could be achieved by simply having two dictionaries -- one for the domain dependent information and the other for the domain independent information.

The DBA is also expected to supply morphological and syntactic information for each new word in the domain. This process is time consuming as well as being susceptible to errors. The lexicon also contains the nouns which identify the fields in the database. It would be more logical to specify these in the database schema, since these words are strictly associated with the fields in the database.

Rather than using an inverted index to determine the database values, CO-OP tries to infer them by examining the context of the query. In the query:

Which projects in oceanography does NASA Headquarters sponsor?

both "oceanography" and "NASA Headquarters" are unknown terms and are assumed to be contents of the database. The system correctly predicts that "oceanography" is an "AREA OF INTEREST" by noting that it modifies a project and is preceded by the
preposition "in". "NASA Headquarters" is determined to be a "SPONSOR NAME" since it acts as the subject to the verb "sponsor". In order for this algorithm to work, the lexicon must contain adequate semantic information about the known words in the query (ie. "projects", "in", and "sponsor"). If there is not enough information, the system is incapable of making any sense of the query. For example, in the query:

Does John Smith have red bolts?

the system would only be able to use the words "does" and "have" to interpret the meaning of the query. Since "Does" and "have" can appear in numerous different contexts, it is very difficult to predict in which database fields the remaining words are contained. The system also would not be able to determine whether "red bolts" correspond to a multi-word noun or to values of two specific fields (say PART and PART-COLOUR). It would have to infer whether "bolts" is stored as "bolt" or "bolts" in the database. Similarly it would have to infer that John Smith is a multi-word proper noun and is stored in the database in the given form and not as "John Smith Limited", or even as values of two distinct fields (say SURNAME and FIRSTNAME).

The system also has a very poor user interface. In the query:

Which F5s carry strut curve radar?

the system must determine whether F5 corresponds to a ship or an airplane since they are both defined as a possible subject for
the verb "carry". In this case CO-OP chooses one at random and answers the query, relaying the interpreted meaning to the user by paraphrasing the query. If the user intended the other interpretation, say airplanes, he/she would have to restate the query as follows:

Which F5 airplanes carry strut curve radar?

This type of assumption is totally unacceptable for any usable system. This ambiguity can be avoided by either examining the database to determine if an "F5" is a ship or an airplane, or prompting the user to choose the intended interpretation.

Another problem caused by not having the contents of the database available to the parser is that all unknown words will be assumed as unknown database elements. However, there may be many other possible sources of unknown words. For example, a word which identifies a field may have been erroneously left out of the lexicon. This method also makes it difficult for the system to do any sort of spelling correction or to be able to process abbreviated terms.

Although CO-OP has some serious faults, it does demonstrate that a system can be made portable by specifying all domain dependent information as data to the system. It also shows how the system can exploit the structure of the database as a rich source of semantic information.

A system that achieves a high degree of portability by accessing semantic information during the syntactic phrase was implemented by Booth (Booth 1983). This system initially parses
queries for noun phrases, verb phrases and prepositional phrases using an ATN parser (Woods 1970). The semantic interpretation part of the system is based on a case grammar (Fillmore 1968). In a case grammar there are a number of "cases" associated with each main verb defined in the system. Each case refers to a phrase in the sentence which fills a specific semantic role associated with the verb. An example of a case frame and the queries it interprets is shown below:

   OPEN ACTION (AG NAME PA MEALS)

   Is the White Spot open for lunch?
   What restaurants are open for dinner?

In the above example the verb "open" has two cases associated with it. They are the "agent" (AG), which will match a phrase in the query corresponding to the name of a restaurant, and "patient" (PA), which will match the phrase corresponding to the meals which the restaurant serves.

Once the system parses a phrase within the query, it tries to determine its case. When a verb phrase is parsed, the system will try and match all noun and prepositional phrases with the case slots designated by the verb. Once all the phrases in the query have been successfully matched with a case slot, the resulting structure is translated into the "standard sentence representation" (SSR), which specifies the fields which are to be retrieved and the restrictions to be placed on the query. This in turn is translated into a query in a database query language.

The system achieves a high degree of portability by clearly
separating the domain dependent information from the domain independent. The domain independent information is called the "linguistic core", which includes the syntactic and semantic components, and the definitions of all words which are normally used in most domains. The domain dependent information, or the "domain definition", consists of words and related syntactic and semantic information which refer to the particular database, a list of additional cases needed in the domain, and an inverted index. Hence when a new domain is implemented, the DBA only needs to supply the system with a new domain definition.

The main disadvantage of this system is that in order to implement a new domain, the DBA needs to be able to determine the cases associated with each new verb. This requires considerable knowledge about linguistics, which most DBA's don't have. It is also possible that the predefined cases in the system may not be satisfactory for all verbs. In this situation additional programming will be required to add new cases to the system. The system also does not have an adequate algorithm for matching the parsed phrases of a query to the cases specified within a verb frame. For instance, if there are prepositional phrases in the query which are not matched to a case, there are no options to try and attach the PP to a suitable NP (modifier attachment).

Another disadvantage to this system is that it is designed to work with databases consisting of one relation or table. Hence it does not address the problems introduced by larger databases with more than one relation.
2.6 Summary

There are currently no natural language query systems available today that are both domain independent and simple enough for an average DBA to use. The systems best designed for portability are Intellect (Harris 1977) and the system implemented by Booth (Booth 1983). Both of these systems use an inverted index and other database information as a source of semantic knowledge. They also permit a new domain to be implemented with little or no additional programming. However, both of these systems require the DBA to have considerable expertise in linguistics. This problem can be resolved by designing a data-driven system which uses a domain independent parser with an extensive dictionary containing lexical, syntactic, and semantic information. Additional knowledge can also be gained by exploiting the structure of the database for semantic information.
The goal in building a natural language query system is to make it both portable and easy to use without sacrificing power. In the past, little attention has been paid to these goals due to the research nature of the projects. Now that NL is being commercialized, more emphasis must be placed on these issues.

The success of previous attempts in designing portable NL systems has been measured by the ease with which an expert can implement a new domain. This criteria is not sufficient if NL systems are to become widely used. Most companies don't want to rely on outside experts or pay their high consulting fees each time a new database is added. If the level of expertise needed to maintain a NL system does not match those of a typical DBA, these systems will have the same fate as many other "difficult-to-use" software packages.

In order to achieve domain portability, it is necessary to clearly separate the domain dependent component from the domain independent part of the system. To minimize the skill needed to implement a new domain, the domain dependent specification should require a minimal amount of linguistic information. All of the domain dependent information should be given in data form. The DBA should not be required to supply any programmed routines.

In the previous chapter we examined the advantages and disadvantages of other NL systems. In this chapter we will closely examine each component of a NL system to determine whether it is domain dependent or independent. I will also
argue for the importance of user friendly components, such as a spelling corrector, and the ability to determine the meaning of unknown words. For the remainder of this thesis I will define a "user" to be a person who enters queries into the system and who has some knowledge of the contents of the database, but has no knowledge of its structure. A DBA (database administrator) is defined as a person who has considerable knowledge of the structure and the contents of the database, but has little knowledge of linguistics and no knowledge of AI programming. A "system designer" is defined as a person who has considerable knowledge of linguistics, AI programming, and database management systems.

3.1 Syntactic Analysis Concepts

Syntactic Analysis involves determining the syntactic category of each word and building a syntactic structure of each query using the rules of a grammar residing in the system. Routines such as a spelling corrector are normally invoked at this stage. The major benefit of initially analyzing a query at the lexical and syntactic level is that it simplifies processing at the semantic level. This is achieved by mapping many similar queries into one syntactic representation called the deep structure.
3.1.1 ATN Networks

ATN's have been used in many NL systems with varying degrees of success (Woods 1970; Harris 1977). One of the main benefits of using an ATN is that the grammar is domain independent. The domain dependent component consists of the vocabulary enhanced with lexical and syntactic information. No additional programming is required when a new domain is implemented since this information is in data form.

There are two disadvantages in using ATN's. The first is that they can produce more than one interpretation of a query. As shown in the previous chapter, many of these can be avoided by enhancing the ATN with some semantic information. The other disadvantage is that the ATN may parse a phrase in the query more than one time. This occurs when more than one arc in the ATN can be taken at a given point. The parse will then continue building a syntactic representation up to a point where no applicable arcs can be taken. At this point the system will back up to the last decision point, discarding all processing done along the incorrect arc. This problem will not be addressed in this thesis, although there are a few features described in chapter 4 which will lessen this problem.

3.1.2 Vocabulary

The first step in designing a portable NL system is to distinguish between the domain independent and the domain dependent vocabulary. Words such as "print", "the", "and",
"of", and "information" are obviously domain independent. The source of domain dependent vocabulary comes from two sources -- words which describe the information and relationships in the database, and the actual contents of the database.

Although words such as "program" and "run" are considered domain dependent, the lexical properties associated with these words is always the same. The past participle of the verb "run" is always "ran" regardless of the semantic meaning within any database. Hence it makes sense to treat the syntactic definition of all common English words as domain independent information. Initially this may seem to be an unrealistic proposition for any NL system, but many microcomputer wordprocessors now have dictionaries which often contain over 100,000 words. Providing a dictionary with syntactic information would produce many benefits to a NL system. It would free the DBA from providing this information when defining a database. It would also enhance the system when dealing with words which are not specified in either the domain dependent or domain independent components. Another advantage is that the syntactic definition of words contained in this dictionary will be consistent from domain to domain, and the chance of any errors caused by incorrect definitions would be reduced.

3.1.2.1 Morphing

One area of Morphology (the study of the internal structure of words) which is often used in NL systems is analyzing a word for prefixes and suffixes. This process is commonly called
Morphing. The benefit of having the system recognize suffixes is that it can recognize various forms of words, such as plural forms of nouns and past forms of verbs, without having each form of the word defined separately. The information needed to perform this function on English words with regular endings is domain independent. Therefore DBA's should not need to specify any morphological information when defining a database, unless an obscure word is used.

It is possible for the system to encounter words (often nouns) where the morphological information is not available. This may happen when a noun is added to the database and is not contained in the dictionary. In this case the system can try to infer the root form of plural nouns in the query by simply attempting to remove possible endings ("s", "es", etc.) and checking to see if the "computed" root exists.

3.1.2.2 Nouns

Although there are a few nouns, such as "report", "information", and "number", which are domain independent, most nouns are domain dependent. Many nouns represent values stored in the database. In the query:

    How much is a bolt?

the noun "bolt" could be stored in a "Supplier/Parts" database. Nouns are also used to identify types of information in a database. In the query:
What programs have good ratings?

the noun "programs" refers to the relation (table) of computer programs, while "ratings" refers to a field (attribute) specifying the rating of the computer program.

3.1.2.3 Proper Nouns

Proper Nouns are usually values stored in the database. The query:

Does John Wilson produce Visicalc?

contains the proper nouns "John Wilson" and "Visicalc", which refer to a software producer and program name respectively. It is also possible that these names may not be contained in the database at all. In this case it is important for the system to recognize such proper nouns and respond appropriately.

3.1.2.4 Adjectives

Adjectives are also domain dependent, although they may appear in many different domains. Some adjectives, such as the set of colours, are likely to be defined similarly in each domain, while other adjectives, such as "good", "terrible", "cheap", or "expensive", may be defined differently each time they are used. For instance, a cheap computer program may be defined to cost less than $100.00, while a cheap house would obviously be defined differently. Since attributes such as colour, rating, and cost are likely to be used in many different
databases, it would be useful to have the needed information defined in a manner such that the DBA can access these definitions in an easy and consistent manner.

3.1.2.5 Verbs

Verbs play many roles in NL systems. Some of these roles are domain dependent, while others are domain independent. One category of verbs which is domain independent are "command verbs". The following two queries:

Print the good programs.
Count the good programs.

contain verbs which could be used in any domain. These verbs instruct the system to perform some task (ie. print a report; count something) and should be defined consistently in every database.

A category of verbs which is largely domain dependent are "relation verbs". In the following queries the verb "run" specifies a semantic relationship between computers and programs, or between an athlete and a race.

What computers run good programs?
Does the Apple II run LOGO?
Which athletes run the 100 metre dash?

Since the required definitions of such verbs are determined by the given database, these verbs should be defined for each database in which they are used.

There are a number of verbs which are similar to relation
verbs but can be used in any domain. The verbs "be" (is, are, etc.) and "have" are two such verbs when they are used as the main verb of a query. In the queries:

Is Logo a good program?
Does LOGO have a good rating?

both "is" and "have" are used to place a constraint (rating = good) on the program Logo.

Another set of verbs which are domain independent are auxiliary verbs. These verbs have many different functions in English. For example, they can be used to specify a "yes-no" question as shown by the verb "does":

Does the Apple have 64 KB?

They can also be used to specify tense as shown by the auxiliary verb "have" in the following query:

What companies have supplied project J3?

3.1.2.6 Adverbs

Adverbs can be either domain dependent or domain independent. Adverbs such as "please" and "immediately", as shown in the following queries, are domain independent and offer little semantic information.
Print the good programs immediately!
Please print the good programs!

Other adverbs could be used to designate the value of a field. The adverbs "quickly" and "slowly" could specify the relative speed of a computer program. In this case they are considered to be domain dependent.

3.1.2.7 Prepositions

Prepositions are very useful in the semantic interpretation of a query. In NL systems with small domains, it is often possible to predict the meaning of the noun following a preposition by analyzing the preposition alone. For example, in the query:

Print the suppliers from Spuzzum.

it is reasonable to predict the name "Spuzzum" is a location (city, town, etc.), especially if each supplier has a location associated with it in the database. In other situations prepositions are used to determine the role a noun phrase plays with respect to the main verb. In the query:

Who flies from Vancouver to Toronto?

the location following "from" signals the origin while "to" signals the destination. Although prepositions are syntactically domain independent, most semantic information associated with them is largely domain dependent.
3.1.3 Handling Unknown Words / User Interface

In designing a system to be used by casual users, it is important for the system to be able to deal with virtually anything a user may type. One downfall with many previous NL systems is that they accepted a limited vocabulary. However, there are numerous reasons why a user may specify a word that is unknown to the system. An unknown word could be caused by a spelling mistake, an abbreviation, a word assumed to be contained in or associated with a database, or even a word which the DBA mistakenly never included in the specification.

3.1.3.1 Spelling Correction

A reasonable spelling correction facility is important in any software system where users are expected to type in natural language. This need is multiplied if the user has little experience with the computer's keyboard. Methods for spelling correction range from simply looking up the word in a dictionary and prompting the user if the word is not found, to complex algorithms of comparing the word to a list of words.

3.1.3.2 Abbreviations

The system must be able to identify both common abbreviations as well as the ones invented by users to save keyboading time. Abbreviations such as "LTD." and "AVE." should obviously be built into the system. It should also allow either the DBA or the user to define abbreviations such as "Kb"
or "K" (for Kilobyte), or "J. W. Software" (for John Wilson Software Limited). Once such an abbreviation is accepted, the system should remember it for future use.

3.1.3.3 Assumed Database Elements

One common source of unknown words is caused when users try to inquire about elements which are not contained in the database. Harris (1977b) assumes that users will only ask questions about elements which are in the database. This may be a reasonable assumption to make if the database is being used in a corporation where the employees are familiar with the data stored. This assumption is totally invalid if users are foreign to the database or if they are inquiring whether something is actually stored in the database. For example, let's assume in the following query that "paper" is not a part (or anything else) contained in a database.

Does Smith supply paper?

One would hope that the system would be able to parse "paper" and appropriately answer:

I don't know. "PAPER" is not a "PART" contained in the database.
3.1.3.4 Ignoring Words

It is impossible to predict every possible word that a user may type. Hence it is important for the system to be able to process queries which contain such words in a reasonable way. One useful technique which can be used is to ignore the word. For example, in the query:

Print the colloid Chemistry programs.

the system may not understand the word "colloid". In this case the user should have the option of having the query answered with "colloid" removed from the query.

3.1.3.5 Knowledge Acquisition

Knowledge Acquisition can range from the system storing new abbreviations or synonyms to learning about new sentence structures and definitions of new verbs. Although the more complex definitions would need to be supplied by a system designer or the DBA, considerable knowledge can be given by the user.

One simple method for learning new words is to allow the user to specify a synonym for a word which is not known to the system. For example, in the following query the system may not have any semantic (or even syntactic) information on the verb "execute".

What computers execute LOGO?
In this case the system may prompt the user for a synonym for the given word. If at this point the user enters a word which has meaning such as "run", the system could correctly answer the query.

When the user supplies the system with information such as an abbreviation or a synonym for an unknown word, it is important that it be stored for future reference. This will prevent the system from asking the user the same question twice, as well as making the system more robust over time.

3.2 Concepts In Semantic Analysis

There has been little work done in constructing a portable semantic processor as opposed to syntactic processors. Methods such as Procedural Semantics (Woods 1967, 1968) require the DBA to do extensive repetative work for each database domain. Other methods (Booth 1983) lessened this work, but still required the DBA to have considerable knowledge of linguistics.

A portable semantic processor should be designed in a similar fashion as the syntactic processor. The domain dependent information should be separate from the domain independent component. The domain dependent component should be in data form and require the DBA to specify as little linguistic information as practically possible.
3.2.1 Definition Of Semantic Information

There are various sources of semantic information, some of which are domain dependent while others are domain independent. The semantic meaning of command verbs such as "PRINT" or "COUNT" is obviously domain independent. Other words such as "A", "THE", "OR", "DOES" and "PLEASE" are also domain independent and should be defined accordingly.

There are a number of types of domain dependent semantic information which must be supplied by the DBA. Obviously the system will need to know which words identify each relation and field in the database. For example, the nouns "program" and "software" could be used to identify the relation program. It is also necessary to specify the relationships between the relations and fields in the database. For example, the verb "RUN" specifies a relationship between the programs and computers.

The structure of the database contains considerable semantic information which can be exploited. For example, in the query:

What has 64 KB of memory?

the system should realize that memory is an attribute (field) of a computer and not of a program. This would allow the system to predict the intended meaning of "What".

Some words appear to be domain independent, but their semantic meaning is actually closely tied to the database. The word "Who" could refer to any human (employee, customer) or a
group of humans (computer manufacturer, program publisher) while the word "What" or "thing" could refer to any physical or abstract object (computer, program, job). This type of semantic information must be supplied for each domain, hopefully in a simple and precise way.

3.2.2 Some Problems In Semantic Interpretation

There are numerous problems in semantically interpreting queries which need to be resolved in such a way that domain portability is maintained. One common problem is that there may be words or phrases in a query which cannot be interpreted. Often it is possible to ignore these words or phrases at the semantic level and still answer the query appropriately. For example, the words "quick" and "quickly" could be ignored in the following query without having a major effect on the main objective of the query.

Print the quick French programs quickly.

The decision on the words which can be ignored is obviously domain dependent since it is possible to have a database where both "quick" and "quickly" are semantically defined.

One classical problem in semantic interpretation is attaching modifiers to the correct noun phrase (modifier attachment). In the query:

What computers run LOGO with 128 KB?

the prepositional phrase "with 128 KB" should modify "computers"
and not "LOGO" since the amount of memory is treated as an attribute of a computer and not a computer program.

Conjunction also presents many problems in semantic interpretation. The query:

Print the Apple and IBM computers.

is ambiguous since the users' intended interpretation may be either:

(1) Print the IBM computers and print the Apple computers.
    (Print the Apple or IBM computers.)

or

(2) Print the computers which are both Apple and IBM.

Using the heuristic that a user will not ask an impossible question and the fact that the database forces each computer to have at most one computer manufacturer, the intended interpretation is obviously (1).

The semantic information needed for modifier attachment and interpreting some types of ambiguous queries is contained in the structure of the database. Hence it is important for the semantic interpreter to be provided with this information.

### 3.3 Database Interface

Many of the earlier NL query systems did not attempt to resolve the problem of transporting the system to other DBMS. Some systems simply implemented the database component in LISP (Woods 1967). This method is suitable for research oriented systems, but is not appropriate for larger commercial databases.
Other systems tightly bound the database component with the semantic interpreter (Hendrix et al. 1978; Sagalowicz 1977). This made it very difficult to adapt the system to a new DBMS.

The most successful approach towards designing a NL system is to design it to act as a front-end to existing database systems. Intellect has taken this approach and now can be interfaced to numerous database systems. In order to make the job of adapting the system to a new DBMS as easy as possible, the database interface component should be separated from the semantic and syntactic components. This can be achieved by having the semantic component pass a structured representation of the query to the database interface routine, which in turn translates it into a query in the database query language of the given DBMS.

3.4 Summary

All of the programmed components, including both the syntactic and semantic processors, discussed in this chapter are domain independent. To implement a new domain, these components only require information in data form. By supplying the system with sufficient information about the structure of the database, it is possible to resolve even some of the more complex problems in NL such as modifier attachment and ambiguous conjunctions.

The system should also be given as much syntactic and semantic information as possible in order to make it more powerful and to make the job of implementing a new domain as easy as possible. Although features such as an inverted index
of the database and a dictionary of the syntactic definitions of common English words will obviously take up considerable storage (primary storage), they are required for a truly intelligent system.

In order for the system to be portable across various database systems, the database interface component should be separated from the semantic and syntactic components. By designing the system as a front end processor, it is possible to adapt it to other applications such as statistical, spreadsheet, or even graphics packages.
IV. SYSTEM DESIGN I: SYNTACTIC ISSUES

A prototype natural language system has been developed to test out the feasibility of a system based on the concepts presented in Chapter 3. The system acts as a natural language front-end to an existing relational database management system. This chapter contains an overview of the system and a detailed description of the syntactic components of the system, including the parser, grammar, and user interface facilities.

4.1 Overview Of The System

The system has been designed so that the domain independent component is separate from the domain dependent information. The domain dependent information is in data format and has been structured so that it is easily maintained by an average DBA.

The system is logically divided into four components - the syntactic processor, the semantic processor, the database interface, and the response generator. Each of these components is domain independent. The database interface and the response generator are the only components which need to be modified if the system is adapted to another DBMS. A diagram of the system is shown in figure 4.1.

The syntactic processor's job is to analyse each word and produce a syntactic representation of the query called the deep structure. The semantic processor takes the deep structure and transforms it into a "Standard Query Representation" (SQR), which contains the information needed to construct a database
query. The database interface then transforms the SQR into a query in a specific database query language. This query is then executed by the DBMS and the results are passed on to the response generator. The response generator's job is to take the information returned from the DBMS and present it to the user in an appropriate format.

The Active Domain Dictionary is the main source of syntactic and semantic knowledge used by the system. This dictionary is compiled from the information contained in both the domain dependent and domain independent components.

The domain independent information is stored in the General Dictionary and the Global Dictionary. The General Dictionary contains syntactic and semantic information about each word commonly used in most domains. The Global Dictionary contains syntactic and some semantic information for most English words. This dictionary serves two main purposes. Its primary use is to supply the syntactic definition of every word specified in the domain dependent component. This relieves the DBA from specifying this information when a word is used. It is also used when a word is not found in the Active Domain Dictionary. This gives the system the ability to process words which are not directly associated with the domain.

The domain dependent information is specified in the "database schema", the "database schema library", and an inverted index. The database schema contains a structured description of the database, which includes the definition of all the relations and their associated fields in the database,
the nouns which identify each relation and field, verbs which specify relationships between relations and/or fields, and information on how relations in the database can be joined together. The database schema may access the database schema library. This library contains definitions of fields and relationships which are commonly used in many different databases. For instance, a standard definition of "CITY" is stored here since cities are contained in many different databases.

The inverted index is a table of all the words contained in the given database. For each of these words, the index contains the database field(s) which the word is associated with, and some syntactic information. The syntactic information for each of these words is supplied by the database schema and the global dictionary. The database schema specifies a default syntactic category for each entry in a field. Additional syntactic information needed to determine various forms of a word (eg. plural) is available for words contained in the Global Dictionary. For example, if the word "phone" is stored in a database field containing parts, the database schema would specify that the word is to be treated as a noun and not as a verb, and the Global Dictionary would supply the information needed to determine the plural and possessive forms.
Figure 4.1 - Diagram of the System
In order for a new domain to be implemented, the DBA needs only to develop a new database schema. If there are any words in the database schema which are not contained in the Global Dictionary, the DBA may optionally specify the syntactic information. The inverted index is automatically constructed by the system.

The system has been tested out using two databases. One database contains information about computer programs and the computers which they run on, and the other contains information about the parts that suppliers supply to various jobs. The system can answer a wide range of questions, including "yes/no" questions, "how-many" questions, and questions which are answered in a report format. Listed below is a sample of the types of questions that the system can answer.

Print the good programs from MECC.
Is LOGO a good computer program?
Tell me if LOGO is an excellent program.
What runs what?
How many chemistry programs run on the Apple II?
How much memory does the IBM PC have?
What programs run on IBM computers with 128 KB?
Print the computers which run software from Microsoft with 128 KB?
Print the good and excellent computer programs.
Does Apple sell food?
Does John Doe sell good programs?
Print the numbers.
Who supplies red, green, or blue bolts?
How much does part P4 weigh?
Does the collator project use bolts from Smith?

Figure 4.2 - Sample Queries

The system is written in LISP (UBC-LISP) and runs on an Amdahl 470-V8. The database management system used is ORACLE
4.0, which uses the query language SQL.

4.2 The Syntactic Processor

The main component of the syntactic processor is the ATN parser. The parser invokes routines such as the morpher, the spelling corrector, and various user interface routines. If the query is successfully parsed, control is passed on to the semantic processor, otherwise an appropriate error message is printed out.

4.2.1 ATN Parser

The job of the ATN parser is to syntactically analyse the query using a specified grammar and produce a structured representation, or deep structure, of it. This deep structure consists of a set of registers, which are passed to the semantic processor.

The ATN parser used in this system is written in LISP and was developed by Dr. R. Reiter (Reiter 1978). This parser is based on the specifications stated by Woods (Woods 1970). A slight modification has been made to this parser to correctly parse relative clauses.

4.2.2 ATN Grammar

A grammar is a set of rules which describe the sentences allowed in a language. The grammar used by the ATN parser is essentially a syntactic grammar enhanced with some semantic
routines. A syntactic grammar analyses a sentence in terms of syntactic structures such as verbs, nouns, noun phrases, and prepositional phrases. Semantic routines have been included to help prevent syntactic ambiguity.

The grammar used in the system is based on the ATN grammar for English developed by Winograd (Winograd 1983). It is capable of parsing many forms of sentences, including passive constructions, relative clauses, some forms of conjunction, and numerous forms of questions. The ATN is divided into five separate components or networks. The main network analyses the complete sentence (S), and invokes the other networks to parse other syntactic components. The other four networks are used to analyse noun phrases (NP), prepositional phrases (PP), proper nouns (NAME), and "noise words" (NOISE). Noise words are groups of words which frequently appear at the start of a question but have no semantic effect on the meaning. For example, the phrase "Can you please" in the query

Can you please print the good wordprocessing programs?

can be ignored without changing the meaning of the query.
Diagrams of the ATNs are contained in Appendix A.

The parser has been designed to limit any syntactic ambiguity which may occur. To avoid the syntactic ambiguity that occurs with proper nouns such as "Wiley and Sons of Canada Limited", a semantic routine has been added to the parser. The system also avoids ambiguous syntactic parses associated with conjunction and modifier attachment by parsing these phrases in
their surface structure. The actual interpretation is determined in the semantic stage for these queries.

4.2.3 ATN Registers

The ATN uses a set of registers to represent the syntactic structure of the query. These registers are also used as the primary data structure during the semantic interpretation. The registers used are similar to the set used by Winograd (Winograd 1983). A sample parse of a query is contained in Appendix B.

In addition to the registers described by Winograd, the registers PRED and VFRAME are used in the sentence (S) network. PRED is used to store the complement (or PREDicate) of copula verbs ("is", "seems", etc.). The register VFRAME is used by the semantic processor and will be explained in Chapter 6.

Each noun phrase in the query has a separate set of registers. These are also described by Winograd with the exception of ANP, POSN, ANPR, and SEMREG. These registers have been added to aid in the semantic interpretation. The surface form of a noun phrase is stored in the register "ANP" (actual noun phrase). This is used to form a prompt to the user when the intention of a noun phrase is ambiguous. The position of the noun phrase within the original query is stored in "POSN". This is used by the semantic interpreter in order to attach modifiers to the correct NP (see section 6.1.5). The register "ANPR" is used to store the full form of a compound proper noun. This is required when users use shortened or abbreviated form of proper nouns. SEMREG is used to store the semantic meaning of
the NP with respect to the database and will be explained in Chapter 6.

4.2.4 The Morpher

The morpher's function is to determine the root form of each word in the query. The advantage of computing the root form of a word is that for words with regular endings, the dictionary only needs to contain the root forms. To accomplish this the dictionary must contain the syntactic category and a "morphological code" for the root of every regular word defined. The morphological code informs the morpher of the allowable suffixes a word may have. For example, the code "IES" in the definition below informs the system how the plural and possessive forms of the word are constructed.

(COMPANY N IES)

In order to find the root of a word, the morpher is supplied with a table containing common suffixes with associated information as shown in figure 4.3. The morpher will then systematically check the input word for each ending in the root table. If the input word has the same ending, then the morpher will add the given ending. For example, for the input word "companies", the ending "ies" will be removed and the ending "y" will be added. At this point the morpher will check for the computed word in the dictionary. If this word exists and the morphological code of the root word matches the ending removed
from the input word, the input word is treated as a valid form of the root word and is assigned the given syntactic category and features. Hence the word "companies" would be defined as follows:

\[(\text{COMPANIES N (COMPANY (PLURAL))})\]

Once the root form of the input word is found, the definition is added to the dictionary for the remainder of the session. This saves the root form of a word from being computed more than once for any session.

It is possible for the dictionary to contain words where the morphological code of the word is not known. This could happen if a word in the database is not contained in the global dictionary. In this case the system will not force the morphological code of the root word to match that of the input word. For example, for the word "collator", the morpher would accept "collators" or "collatores" for the plural form. Although it is possible for the system to make an incorrect assumption, it is highly unlikely.

If the morpher fails to find a root word for the given input word, the morpher will invoke various routines, such as the spelling corrector. These routines will be described later.
4.2.5 Parsing A Query

The sentence (S) ATN is responsible for parsing the main components of the query. These components may include auxiliary verbs (AUX), the main verb (V), noun phrases (subject (SUBJ), direct object (DOBJ), indirect object (IOBJ)) and prepositional phrases (PPOBJS). The S ATN invokes the noun phrase (NP), prepositional phrase (PP), and noise word (NOISE) ATNs to parse specific components of the query. If the query is successfully parsed, the deep structure is produced from the various registers, and the semantic processor is called. If the query cannot be parsed, an error message is printed out. In this case the user is expected to rephrase the query.

The following sections explain some of the techniques and features used in the system. These topics include the handling of conjunction, compound proper nouns, prepositional phrases, relative clauses, and noise words. For a more detailed explanation of the basic ATN grammar, the reader is referred to

<table>
<thead>
<tr>
<th>Ending to Remove</th>
<th>Ending to add</th>
<th>Syntactic Category</th>
<th>Morphological Code</th>
<th>Syntactic features of new word</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>-</td>
<td>N</td>
<td>S</td>
<td>PLURAL</td>
</tr>
<tr>
<td>ES</td>
<td>-</td>
<td>N</td>
<td>ES</td>
<td>PLURAL</td>
</tr>
<tr>
<td>IES</td>
<td>Y</td>
<td>N</td>
<td>IES</td>
<td>PLURAL</td>
</tr>
<tr>
<td>'S</td>
<td>-</td>
<td>N</td>
<td>S</td>
<td>POSSESSIVE</td>
</tr>
<tr>
<td>'S</td>
<td>-</td>
<td>N</td>
<td>IES</td>
<td>POSSESSIVE</td>
</tr>
</tbody>
</table>

Figure 4.3 - Partial Suffix Table
Winograd (Winograd 1983).

4.2.5.1 Noun Phrases

The system uses a separate ATN to parse noun phrases (NP). It is capable of parsing various forms of NP's including relative clauses and some limited forms of conjunction. Some examples are shown below:

computers
the good computer programs
what
how many suppliers
a red part from Smith
the MECC programs which teach chemistry
the Apple and IBM programs

4.2.5.2 Conjunction

Conjunction is a very complex area of natural language. The NP network is capable of parsing some limited forms of conjunction as shown below.

What Apple and IBM computers
the red, green, or blue parts
the good or excellent computer programs

When dealing with conjunction there is a possibility from ambiguity. The parser avoids any possible ambiguity by parsing the query at its surface level. The resolution of any ambiguity is dealt with by the semantic processor (see section 6.2.3.2.3).
4.2.5.3 **Compound Proper Nouns**

One of the main features of the system is its ability to deal with compound proper nouns. The system is capable of recognizing shortened forms of proper nouns. For example, the software supplier:

John Wilson and Sons Limited

could be derived from any of the following names:

J. Wilson Ltd.
Wilson and sons
John

The system is also capable of determining proper nouns which are not contained in the database. For example, in the query:

Does Steve Smith publish good programs?

the system would parse "Steve Smith" as a proper noun, and the semantic processor would later inform the user that "Steve Smith" is not a valid software publisher.

The system uses a separate ATN network augmented with some semantic routines to parse proper nouns. The system will first try to parse as many subsequent words as possible which may be part of a proper noun. These words include proper nouns, abbreviations, initials, and nouns which are commonly parts of names (Mister, company). Words such as "of" and "and" are also treated as proper nouns if they appear in any compound proper nouns. It will then try and match this string of words with proper nouns contained in the database. In the case of an exact
match or an unique match where either a word, an abbreviation, or an initial corresponds to each word in the compound proper noun, the system will use the given NPR without prompting the user. If a possible match is found, the system will prompt the user for verification. For example, the phrase:

J W Ltd

could match either:

John Wilson and Sons Limited
or John W Software of Canada Limited.

In this case the user would select the intended meaning. Once the system determines the intended proper noun, the full version of the name is stored in the ANPR (Actual Noun-Proper) register to aid in the semantic interpretation. If a suitable match is not found, the system will try and match a smaller portion of the input string. For the phrase:

John Wilson and John Brown

the system would first fail on matching the entire phrase with an appropriate proper noun, then it would try the following:

(1) John Wilson and John
(2) John Wilson

At this point (2) a suitable match would be found and the user would be prompted for verification. The system will not try and match "John Wilson and" since such an abbreviation is considered unacceptable.
The system uses two methods to parse compound proper nouns which are not contained in the database. The first method is to check whether the word(s) is stored in the Global Dictionary as a proper noun. If this is the case, the system will infer that the given proper noun(s) is an unknown (compound) proper noun and will then continue to parse the query. In the query:

Does Steve Smith publish good programs?

both "STEVE" and "SMITH" are stored in the global dictionary as proper nouns. If a word is not contained in the global dictionary, the system presently prompts the user for additional information. If the user specifies that the word is an "unknown database element", the system will continue the parse and attempt answer the question appropriately. For example, let's assume that the system does not know about the word "Vavrik" in the following query:

Does John Vavrik sell good programs?

In this case the system will parse "John" as a proper noun and will be informed by the user that "Vavrik" is an unknown database element. In this case the system will assume that "John Vavrik" is a compound proper noun and will produce the following answer:

(JOHN VAVRIK) is not (THE NAME OF THE PUBLISHING COMPANY OF A PROGRAM) contained in this database!

This algorithm can be improved to parse a greater
percentage of compound proper nouns by using a number of heuristics not presently implemented. One simple heuristic would be to attach a semantic marker to each proper noun in the Global dictionary to specify if it commonly used as a first and/or last name. The system could then infer that an unknown word following a known first name is most likely a last name. Similarly, if an unknown word precedes a known last name and possible proceeds a title (Mr., Dr., etc.), the system could infer that it is an unknown first name. Similar techniques could be used to parse addresses. For instance, an unknown word between a number and a word such as "street" is normally considered to be a street name. Another heuristic we use in written English to identify proper nouns is to check for capitalization. This has not been implemented since one of the goals was to explore the identification of proper nouns using semantic and syntactic information.

Once a proper noun is successfully parsed and matched to some database element, the ATN will store this result so that the given string of proper nouns will not have to be parsed again if backtracking occurs. By treating words such as "of" and "and" as possible proper nouns, the ATN can also avoid unnecessary parses. For example, in the query:

Print the John Wilson and Son's computer programs.

the system will not try to parse "and" as a conjuction since it is obviously part of a proper noun. Both of these techniques improve the performance of the ATN parser.
This algorithm for parsing compound proper nouns has worked well for recognizing program publishers in the test database. Although it requires the use of an inverted dictionary and a global dictionary, it allows the system to answer a greater percentage of queries intelligently and correctly.

4.2.5.4 Prepositional Phrases

One of the major sources of syntactic ambiguity is caused by the fact that prepositional phrases (PP) may be physically separated from the head nouns which they modify. For example, in the following query the phrase "from Apple" may either modify "computers" or "Logo".

What computers run Logo from Apple?

The parser avoids the decision of attaching the PP to the correct head noun by parsing prepositional phrases at their surface level. Since this decision requires semantic information, this process is done by the semantic processor.

The only time the parser will attach a PP to a head noun is when there is no possibility for ambiguity. For example, the phrase "from Apple" can only modify "computers" in the following query.

What computers from Apple run Logo?
4.2.5.5 Relative Clauses

The system is capable of parsing relative clauses which begin with a relative pronoun (eg. which, that). These relative clauses act as modifiers of the head noun in the given noun phrase. Some examples are given below:

the programs which teach chemistry
a computer that runs LOGO
any good program which runs on the IBM PC

Relative clauses are initially identified by watching for a relative pronoun in the NP ATN. If one is found, the S ATN is called using a special entry point for relative clauses (S/REL). A copy of the head noun in the noun phrase is passed to the S ATN by the HOLD register. The head noun will then act as the subject of the embedded sentence.

The system does not currently perform modifier attachment for relative clauses. For example, it will not be able to parse the following query correctly:

What programs run on computers which teach French?

It does have the capability of attaching prepositional phrases to noun phrases contained in the relative clause. For example, the system will treat "from Terrapin" as a modifier of "LOGO" and not of "computers".

Print the computers which run LOGO from Terrapin!
4.2.5.6 Noise Words

Often users will place a string of words on the front of a query which supply little semantic meaning. These words, often called noise words, can be ignored. Below are some examples of these phrases:

[Can you inform me if] LOGO is a good program?
[Please let us know whether] LOGO is a good program?
[Tell me] what computers run LOGO?

These phrases are parsed by a separate ATN which is invoked at the start of the parse. A diagram of this ATN (NAME) is contained in Appendix A. This technique has been used successfully in other NL systems (Waltz 1978; Strzalkowski 1983).

4.2.5.7 Relaxation Of Grammatical Rules

The goal of any NL system is to answer as many queries as possible. Since users may type in queries which are ungrammatical, it is important for the system to parse them without forcing strict grammatical rules. For example, even though the following query contains many grammatical errors, it is still possible to answer it.

What computer run a Apple good programs?
4.3 Handling Unknown Words

There are many categories of words which may be unknown to the system. These include mispelled words, abbreviations, words which the user thinks may be contained in the database (but are not), and words which are totally unrelated to the application.

For a user-friendly system, it is important to try to answer a query even if it contains words not defined in the system. In some cases, it is possible for the system to process the word without additional information from the user. For other cases, the user is required to specify some additional information about the unknown word.

4.3.1 Abbreviations And Synonyms

Both the Active Domain Dictionary and the Global Dictionary contain lists of previously defined abbreviations and synonyms. When such a word is found in a query, the system will substitute the appropriate word into the query. In the following query, the system automatically substitutes "KILOBYTE" for "KB" without prompting the user for verification.

Print the computers with 128 KB.

In order to inform the user of this substitution, the following message is printed:

(*** Abbreviation/Synonym *** KB --> KILOBYTE)
4.3.2 The Spelling Corrector

If a word cannot be found in the Active Domain Dictionary or the Global Dictionary, the system tries to determine if the word is misspelled. This is accomplished by comparing the word to a list of frequently misspelled words. If a word from the list substantially matches the unknown word, the system assumes the user misspelled the word and prints out an appropriate message. If the word is not sufficiently similar to any word in the list, the system will try to compare the word with every word in the Active Domain Dictionary. This feature may be turned off if this process takes up too much time.

The Spelling Corrector uses a simple algorithm for determining misspelled words. It compares known words which begin with the same letter to the unknown word and then checks for possible errors such as missing letters, extra letters, incorrect letters, and transposed letters. If the unknown word can be matched to a known word given a reasonable ratio between the number of errors and the number of letters in the word, the word is taken to be the intended spelling. Although this algorithm is fairly ad-hoc, it has proven to be adequate for finding a large number of misspelled words in numerous test sessions.
4.3.3 The User Interface

If the meaning of the word cannot be found in the Active Domain Dictionary or by the spelling corrector, the system prompts the user for additional information. One of two menus is presented to the user depending on whether the syntactic definition of the word is contained in the Global Dictionary or not. If the word is not contained in the Global Dictionary, the user is presented with the following menu.

Unknown Word - "actual word"

What would you like to do?

1. Spelling Error! Enter the correct spelling.
2. Abbreviation! Enter the full word.
3. Synonym! Enter the replacement word.
4. Unknown data element! Continue processing.
5. Ignore the word! Continue processing.
6. Cancel the query.

In this case the user is expected to choose one of the stated options. If the word is contained in the Global Dictionary (but not in the Active Domain Dictionary), the user is presented with a menu containing options 3, 4, 5, and 6.

4.3.3.1 Spelling Errors

The user is given the option of entering the correct spelling when the unknown "word" is not found in the Active Domain Dictionary or the Global Dictionary. This option is necessary since the system's spelling corrector is not capable of finding gross spelling errors or spelling errors which occur
in words not contained in the Active Domain Dictionary.

4.3.3.2 Abbreviations

The user is also given the option of specifying the lengthened form of an abbreviated word. This is required since the dictionaries may not contain all commonly used abbreviations. This option also allows users to define their own abbreviations for brevity purposes.

4.3.3.3 Synonyms

In many situations a user may be able to supply a suitable synonym for the unknown word. For example, the following query contains the word "machines" which may not be defined in the Active Domain Dictionary.

What machines run LOGO?

In this case the user could obtain the desired answer by substituting a word like "computer" for "machines". This feature is very useful if the database schema does not contain a comprehensive list of words which describe the database and its relationships.

4.3.3.4 Unknown Database Elements

One of the most common sources of unknown words are words which users incorrectly assumes to be contained in the database. In this case the system should try to predict the intended
semantic meaning of the word and form an appropriate response. In the following query:

Does IBM sell food?

the word "food" is not defined in the Active Domain Dictionary. Since the syntactic definition is contained in the Global Dictionary, the user is expected to select one of the following options.

1. Synonym! Enter the replacement word.
2. Unknown data element! Continue processing.
3. Ignore the word! Continue processing.
4. Cancel the query.

If the user chooses option #2, the semantic processor will appropriately display the following prompt:

(Food) is not (a brand/type of a computer) contained in the database.

For unknown database elements which are not contained in the Global Dictionary, the system will also ask the user whether the word is a noun, proper noun, adjective, or an adverb. Although this information is not always mandatory for the system answer the query, it is occasionally useful for parsing the query correctly. In the following query, the user will have to specify that "LOTUS" is both an unknown database element and a proper noun.

Does LOTUS run on the IBM PC?

Given this information, the system will correctly parse the
query and print the following response:

(LOTUS) is not (a computer program) contained in the database.

4.3.3.5 Ignoring Words

It is possible to adequately answer some queries by ignoring specific unknown words. For instance, if the word "colloid" is ignored in the following query, the system can correctly parse the query and form a response for the remainder of the query.

Print the colloid chemistry program!

In this case the user will learn that "colloid" is not a valid modifier for the "program" (or "chemistry"). Additionally he/she will be given a list of all the chemistry programs. The system also has the ability to ignore words at the semantic level. This will be discussed in chapter 6.

4.3.3.6 Cancelling The Query

Whenever an unknown word is found in a query, the user is given the option of cancelling the query. In many situations, the user's query may be unintentionally answered when a word cannot be parsed. An example of this is shown in the following query:

Does LOGO run on the Osborne?
If the system is unable to parse "Osborne", the user is essentially informed that "Osborne" is not a computer (or anything else) contained in the database.

4.3.4 Knowledge Aquisition

The system is capable of acquiring knowledge by storing previous spelling (or typing) mistakes, unknown database elements, and newly defined abbreviations and synonyms. This information is presently only stored for the duration of the current session. By saving this information for each user, it is possible to build up a dictionary of the user's individual vocabulary which may be reactivated for following sessions.

This information is also very useful for improving the robustness of the system. By examining this information, the DBA can easily enhance the system as well as finding omissions in the database schema.

4.4 Summary

The Syntactic Processor has been designed to be domain independent. It is completely data driven. No additional programming is required to adapt it to a new database domain.

The system avoids many types of syntactic ambiguity by using semantic information. For example, the Syntactic Processor is able to intelligently interpret ambiguous and abbreviated forms of compound proper nouns by having the parser call some semantic routines. Ambiguity resulting from certain
types of conjunction and modifier attachment is resolved by parsing these phrases at their surface level and resolving any ambiguity during the semantic interpretation.

The system has been enhanced with various user interface routines to aid in the interpretation of a query. For example, if an unknown word is found or if an abbreviated proper noun matches more than one database entry, the system will prompt the user for the required information. It makes more sense to have the user resolve such problems rather than having the system make incorrect assumptions or forcing the user to restate the query in a different form.
V. SYSTEM DESIGN II : KNOWLEDGE BASE

The main objective in the design of this system is to achieve a high degree of portability. This is achieved by clearly separating all domain dependent information from the domain independent information. The domain dependent information consists of the Database Schema, selected definitions from the Database Schema Library, and the inverted index for the given database. The domain independent information is contained in the General Dictionary and the Global Dictionary. These sources of information are compiled together to form the Active Domain Dictionary, which is the primary source of syntactic and semantic knowledge used by the system.

In this chapter we will examine both the domain independent and domain dependent sources of knowledge used by the system. We will also look at the main benefits of using the relational database model as well as the structure of the test databases used by the system.

5.1 Relational Database Approach

The system is designed as a natural language front-end to a relational database system. There are many advantages in using the relational database model as opposed to the other two popular database models -- the hierarchical model and the network model. The main advantage is that the relational database provides a high degree of data independence. This means that
the application program does not need to know how the actual data is stored in secondary (or primary) storage in order to access it (Date 1977). Another major advantage is that it provides a simple view of the data. This allows casual users to easily visualize the structure of the database. Both the network and hierarchical models can become very complex since they often use pointers to model certain types of relationships. Another reason for choosing this model is that it has a sound theoretical foundation based on mathematical set theory (Date 1977). This theoretical foundation has proven to be useful for both the underlying DBMS and this natural language front-end.

5.2 The Test Databases

The system has been tested out using two demonstration databases. One database contains information about computers and programs, and the other database contains information about suppliers, parts, and jobs. Most of the sample queries are taken from the program database. This database consists of the relations PROGRAM, COMPUTER, DESCRIPTOR, and PROGDESC. A diagram of this database is shown in figure 5.1. A complete description of the Supplier/Part/Job database is contained in chapter 8.

| COMPUTER  | [CNO MAKE MODEL RAM] |
| PROGRAM   | [PNO PUBLISHER NAME VERSION CNO RATING COST] |
| PROGDESC  | [PNO DNO] |
| DESCRIPTOR| [DNO DSC] |

Figure 5.1 - Design of the Program Database
5.3 Relational Database Terminology

In order to proceed with this chapter, it is necessary to understand some concepts and terminology in relational database systems. To this point it has been assumed that the reader understands the terms relation and field. A relation is basically a table of information. Each relation contains a number of attributes called fields. For example, the COMPUTER relation contains the fields CNO, MAKE, MODEL, and RAM. The set of all possible values which a field may contain is called a domain. The domain for the field MAKE is the set of all computer makes (or manufacturers). Each relation must also have a field which uniquely identifies an entry in the database. This field is called the primary key. The field CNO is the primary key of the COMPUTER relation.

Another important concept in the relational database model is the "join". A join represents a semantic relationship between two relations. For example, the PROGRAM and COMPUTER relations contain a relationship which determines what computer runs each program. In order for a join to be possible, the two relations must have a domain in common. The join between the COMPUTER and PROGRAM relations is performed through the computer number (CNO) field, which is contained in both relations.

It is also useful to examine the restrictions placed on the relationships between relations. In the program database, a program is restricted to run on one computer, while a computer may run many programs. This is called a "one-to-many" relationship. The other main relationship in the program
database is between the programs and the descriptors. This is termed as a "many-to-many" relationship since a program may have many descriptors, and a descriptor may be associated with many programs. In order to represent a many-to-many relationship in the database, a secondary relation is required. The PROGDESC relation is used to represent this relationship. Note that this relation contains a domain in common with both the PROGRAM relation (PNO) and the DESCRIPTOR relation (DNO). This information is useful for resolving ambiguities arising from conjunction and will be dealt with in Chapter 6.

5.4 Specification Of Domain Dependent Information

The domain dependent information consists of the database schema, the database schema library, and the inverted index. Each of these will be discussed in detail in the following sections.

5.4.1 The Database Schema

The database schema contains a structured definition of the database. It consists of three components -- the database definition, a list of relation verbs which represent relationships in the database, and information on how different relations can be connected, or joined together.
5.4.1.1 The Database Definition

The database definition is designed to model the structure of the database. It defines considerable lexical and semantic information about the domain. The database definition is syntactically similar to the specification of the actual database. Both consist of a number of attributes which contain various types of information about each relation and field in the database. This format makes the job of implementing and maintaining a database easy and concise.

The database definition consists of various attributes which describe each relation and field. Information describing the database itself is contained in the FIELDS, IDENTIFIERS, KEY, and COMMENT attributes. Information about the contents of the database is stored in the CONTENTS and MEM-OF attributes. Linguistic information is contained in the L-NOUN, L-DESC, L-PREP, and L-ABR attributes. Semantic and syntactic information is defined in the CATEG, TYPE, and LTYPE attributes. Each of these attributes will be described in the following sections. The database schema for the program database is in Appendix C.

5.4.1.1.1 Relation Specification

A definition for each relation in the database must be included in the database definition. Although most of the following attributes are optional, the FIELDS and COMMENT attributes must be stated for every relation.

FIELDS contains a list of the fields contained in the given
relation. This is used to inform the system which fields are in
a relation. The KEY attribute specifies the primary key of the
relation. The key is needed in the generation of specific types
of database queries.

The IDENTIFIERS attribute contains a list of database
fields which are commonly used to distinguish one entry from
another. This is helpful in answering questions which do not
explicitly specify the fields to be printed. The IDENTIFIERS
for the program relation are:

(IDENTIFIERS (PNO PUBLISHER NAME))

These three fields will be used in the response for queries
like:

What programs run on the IBM PC?

The L-NOUN attribute lists nouns which are used to identify
the relation. The following definition tells the system that
any of the following nouns may be used to identify the PROGRAM
relation.

(L-NOUN (PROGRAM PACKAGE COURSEWARE SOFTWARE))

L-DESC specifies nouns and adjectives which can be used as
classifiers (or describers) for the nouns in L-NOUN. For
eexample, the following definition specifies that either
"computer" or "educational" may proceed "program" in a query.

(L-DESC (COMPUTER EDUCATIONAL))
These words supply no semantic meaning except when the head noun (program) has more than one possible meaning. In this case the appropriate meaning can be determined by the given describer.

The L-PREP attribute lists the prepositions which may be used in a prepositional phrase where the noun phrase represents the given relation. For example, the following phrases show how the prepositions "on" and "in" are normally used in prepositional phrases containing descriptors.

programs in French
software on chemistry

This knowledge is useful for helping to determine the meaning of unknown words which follow prepositions.

The L-ABR attribute contains abbreviations which are associated with the relation. For example, if one wishes to define "pgm" as an abbreviation for "program", the following statement can be included:

(L-ABR (PGM PROGRAM))

The COMMENT attribute contains a phrase which describes what can be contained in the relation. This is termed the "intention" of the relation in database terminology. This comment is used for many purposes, including forming responses, prompting in ambiguous queries, and even documenting the database schema itself. For example, the COMMENT for the program relation is:
If the system needed to form a response to inform the user that "LOTUS" is not a computer program, it would use this phrase as follows:

(LOTUS) is not (a computer program) contained in the database.

The CATEG attribute specifies the "semantic category" of the entity represented by the relation. A list of the semantic categories is shown in figure 5.2.

<table>
<thead>
<tr>
<th>CATEGORY</th>
<th>DESCRIPTION</th>
<th>EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>HUMAN</td>
<td>any human</td>
<td>student, employee</td>
</tr>
<tr>
<td>GROUP</td>
<td>a group of people</td>
<td>organization, company</td>
</tr>
<tr>
<td>PHYSICAL OBJECT</td>
<td>any concrete object</td>
<td>computers, parts</td>
</tr>
<tr>
<td>ABSTRACT OBJECT</td>
<td>any abstract object</td>
<td>descriptors, ideas</td>
</tr>
<tr>
<td>EVENT</td>
<td>something that takes place</td>
<td>sale, meeting</td>
</tr>
</tbody>
</table>

Figure 5.2 - Semantic CATEGORIES

These semantic categories are derived from McLeod's Semantic Database Model (McLeod 1980). He uses them to model the semantic structure of the data within the database. A natural extension is to apply them to the semantics of natural language. For example, the question noun "who" or the noun "anyone" can represent either a human or a group. CATEG is also used to define relationships which are defined between the relations and fields of the database. This will be fully explained in the Database Schema Library section.
5.4.1.1.2 Field Specification

The database definition also contains a list of attributes for every field in each relation. Most of the attributes used to define a field are similar to those which define a relation. In addition to these attributes, the TYPE, L-TYPE, CONTENTS, and MEM-OF attributes are used in the definition of fields.

Each field will normally have an L-NOUN attribute to define the nouns which can be used to identify the field in a query. The L-DESC, L-PREP and L-ABR attributes can also be used for defining fields.

The CONTENTS attribute describes the actual information contained in the database. It also specifies whether the field needs to be inverted or not. If the field contains integers or real numbers, CONTENTS is set to either INTEGER, REAL, or MONEY. These fields are not inverted since their domain is known. If the field contains characters, the field will normally be inverted. For example, the following statement will cause the given field to be inverted.

(CONTENTS (INVERT NPR))

The "NPR" states that the values of the field will be treated as proper nouns. This will be discussed further in Section ***.

Some fields contain a key to another relation. For example, the PROGRAM relation contains the field CNO which specifies which computer the given program runs on. In this case the CONTENTS attribute will be set to the following:
In some cases the domain of a field may be restricted to a specific set of values. For example, the field PART COLOUR may be restricted to contain either red, blue, green, or yellow. When the domain of the field is known, the set of possible values may be defined in the database schema. This is achieved by using the MEM-OF (member of) attribute as shown below:

MEM-OF ((RED) (BLUE) (GREEN) (YELLOW))

The MEM-OF attribute relieves the need for inverting the given field. It also allows the system to be able to identify each possible value even if it is not contained in the actual database.

Coding of database elements is often used in databases to save file space and make the job of entering data easier. For example, the PROGRAM RATING field is coded by a number from 1 to 5, where 1 represents "TERRIBLE" and 5 represents "EXCELLENT". The CODE-AS attribute can be used in conjunction with the MEM-OF attribute to define coded information. A partial definition of the PROGRAM RATING field is shown below:

MEM-OF ((EXCELLENT CODE-AS (* = 5))
     (GOOD CODE-AS (* > 3))
     .......

Note that in this definition "EXCELLENT" is coded as 5 while "GOOD" is coded as a number greater than 3 (eg. 4 or 5).

One of the most important attributes in the definition of a
field is the TYPE attribute. This attribute is used to call up previously defined definitions contained in the database schema library. For example, many databases may have a field which contains money. By simply stating a field to be of "TYPE MONEY", the associated semantic definitions for words such as "how much", "money", and "amount" will automatically be defined.

The TYPE attribute is not as rigidly defined as the CATEG attribute. Although it is an optional attribute, most fields will normally be given a particular type. Figure 5.3 contains a list of the presently defined TYPES. These TYPES have been developed for use in the two test databases and is not meant to be complete. A commercial system would obviously contain many more definitions.
<table>
<thead>
<tr>
<th>TYPE</th>
<th>DESCRIPTION</th>
<th>EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>an integer</td>
<td>SIN, age</td>
</tr>
<tr>
<td>REAL</td>
<td>a real number</td>
<td>batting average</td>
</tr>
<tr>
<td>MONEY</td>
<td>an amount of money</td>
<td>salary, cost</td>
</tr>
<tr>
<td>GNAME</td>
<td>a general name</td>
<td>program name, part name</td>
</tr>
<tr>
<td>CNAME</td>
<td>a company name</td>
<td>publisher, manufacturer</td>
</tr>
<tr>
<td>Q-MASS</td>
<td>a mass quantity</td>
<td>weight of a part</td>
</tr>
<tr>
<td>Q-QTY</td>
<td>a countable quantity</td>
<td>number of parts</td>
</tr>
<tr>
<td>LOCATION</td>
<td>a location</td>
<td>city, office location</td>
</tr>
<tr>
<td>RATING</td>
<td>any concrete object</td>
<td>rating of a program</td>
</tr>
<tr>
<td>COLOUR</td>
<td>a common colour</td>
<td>colour of a part</td>
</tr>
</tbody>
</table>

Figure 5.3 - TYPES

The L-TYPE, or "Linguistic Type" attribute is also used to extract linguistic definitions of words from the database schema library. This attribute is intended to give more specific definition to fields than the TYPE attribute. For example, a field containing money (TYPE MONEY) may be set to salary (LTYPE SALARY), cost (LTYPE COST), or some other defined L-TYPE representing money. This attribute is optional and is usually used in conjunction with the TYPE attribute. A partial list of L-TYPES is shown in figure 5.4.
Some fields may contain a CATEG attribute. This occurs when a field represents a different entity than the one represented by the relation. For example, the PUBLISHER field in the PROGRAM relation will be defined as a GROUP, even though the PROGRAM relation is defined as a PHYSICAL OBJECT.

5.4.1.2 Verb Specification

The database schema also contains the definition of domain dependent verbs. These verbs usually represent semantic relationships between the relations and fields in the database. For example, the verb "run" represents a relationship between a computer and a program in the program database, and the verb "publish" represents a relationship between a publisher and a program.

Each verb definition consists of a "verb frame" (VFRAME) and possibly some information on how the relations are joined together (JOININFO and JOINTYPE). A verb frame consists of the verb and a number of slots. Each slot represents a syntactic component of the query, such as the subject (SUBJ) or the direct object (DOBJ). The slots are associated with a particular
relation or field in the database. The following is a definition for the verb "run".

(RUN
  VFRAME (((SUBJ (RELATION COMPUTER))
             (DOBJ (RELATION PROGRAM))))
  JOININFO ("COMPUTER.CNO = PROGRAM.CNO")
  JOINTYPE ((COMPUTER 1) (PROGRAM MANY))

When the solution of a query requires that more than one relation needs to be accessed, it is necessary to join the given relations. This is achieved by specifying constraints on fields common to each relation. Since it is possible that two relations may be joined together in more than one way, it is necessary to specify the join which is used by a specific verb. This information is specified by the JOININFO attribute. The previous example shows how the PROGRAM and COMPUTER relations are joined when the verb "run" is used.

It is useful in the semantic interpretation of a query to know about any restrictions on how two relations are related. For instance, in the program database, each program is restricted to run on one computer, while one computer may run many programs. In database terminology this is known as a one-to-many relationship. This information is stored in the JOINTYPE attribute.
5.4.1.3 Default Joins

In some situations two relations will need to be joined together where the information on how to perform the join is not explicitly stated. This is common in noun phrases when a modifier and the head noun refer to different relations. This is shown in the following query, where the modifier "business" represents the DESCRIPTOR relation and "programs" represents the PROGRAM relation.

Print the business programs.

In this case the system uses the default join, which specifies the most likely way two relations are joined together. These joins are stated by the DBA for every likely combination of relations in the database. If a default join is not given, it is assumed that one relation cannot modify the other. In this case the query will not be answered.

Each default join also has a "semantic association" factor associated with it. The semantic closeness of two relations is a factor which represents how closely associated the entities used by the relations are to each other. For example, the relations COMPUTER and PROGRAM are more closely related to each other than the COMPUTER and DESCRIPTOR relations. This factor is used in attaching prepositional phrases and resolving semantic ambiguities. Both of these topics are discussed in Chapter 6.

There has been considerable work done in developing algorithms which automatically determine a complex join between
two relations (Carlson and Kaplan 1976; Sagalowicz 1977). This would allow the DBA to specify only the direct joins between two relations instead of specifying all likely joins with their associated semantic closeness factors. The problem with these algorithms is that they rely heavily on how well the particular database structure models the actual data. They also assume that the semantic closeness of two relations can be determined by the number of links performed to join the relations together. Although these algorithms work for some cases, they can also fail. For example, the phrase:

Dr. Lee's students

may be interpreted as:

(1) the students in the same department as Dr. Lee,

or

(2) the students enrolled in courses taught by Dr. Lee.

If the given database required the system to join the relations STUDENT and PROFESSOR through the DEPARTMENT relation for interpretation (1), and through the STUDENT-COURSE and COURSE relations for interpretation (2), the system would infer that (1) is the intended interpretation. However, the most appropriate interpretation is (2). This example shows the need for the DBA to supply rich information on how various relations can be joined together.
5.4.2 Database Schema Library

The Database Schema Library contains definitions of fields and relationships which are commonly used in many different databases. This information is useful since many databases contain similar types of data. For instance, an entity such as a person may be defined in a database containing employees, students, instructors, customers, criminals, consultants, or athletes. Most entities also have many common properties or attributes associated with them. For example, physical objects (parts, computers, programs, etc.) may have properties such as weight, colour, and cost associated with them.

Since it is likely that such definitions will be used in many different databases by many users, it is worthwhile to include a rich library of syntactic and semantic definitions. This is useful for many reasons. First, it makes the task of implementing standard databases such as inventories and personnel records quick and easy. It also forces standard and more complete definitions for different databases. Hence the definition of an employee would be the same no matter who implemented the database.

The definitions in the Database Schema Library are retrieved by the CATEG, TYPE, and L-TYPE attributes in the Database Schema. For example, the PROGRAM relation contains the field COST. By specifying that the L-TYPE of this field is cost, the nouns "cost", "price", and "value" are automatically defined as identifiers (L-NOUN) for this field. The verb "cost" is also defined as a relation verb where the subject refers to
PROGRAM and the direct object is the COST. A sample portion of the Database Schema Library is contained in Appendix D.

5.4.3 Inverted Index

The Inverted Index is a table of most the words contained in the database. This index enables the system to determine which words belong to each field in the database. The system does not have to invert every field in order to determine the values of each field. For example, fields containing numbers or words from a defined set (MEM-OF) don't need to be inverted since the possible values of the field can be predetermined.

The INVERT option of the CONTENTS attribute in the database schema is used to identify database fields which are to be inverted. For example, the CONTENTS attribute for the PROGRAM PUBLISHER field is:

(CONTENTS (INVERT NPR))

In this case each value in the PROGRAM PUBLISHER field will be treated as a proper noun. For example, the following information will be stored in the Active Domain Dictionary:

(TERRAPIN NPR ?
ELM-OF ((PROGRAM PUBLISHER))))

Since the word "TERRAPIN" is not contained in the Global Dictionary, the morphological code is not known. In this case the code is set to a question mark (?). In addition to this information, a list of the field names is stored under the
attribute "ELM-OF" (element of).

One of the main arguments against the use of an inverted index is the problems of keeping it up-to-date as new data is added into the database. One possible solution is to recreate the inverted index periodically. This can cause some problems since the system will not be able to identify some new entries until a new inverted index is generated. The other alternative is to update the inverted index as the new data is added. Although there is some additional overhead with this method, it will ensure that the inverted index is always up-to-date.

5.5 Specification Of Domain Independent Information

Apart from the program components and the grammar, there are two sources of domain independent information. They are the General Dictionary and the Global dictionary. These will be discussed in the following sections.

5.5.1 General Dictionary

The General Dictionary contains definitions of words commonly used in queries regardless of the particular database domain. These words come from every syntactic category including nouns (information, description), determiners (the, a), question pronouns (who, what), personal pronouns (me, you), relative pronouns (that, which), prepositions (at, on), conjunctions (or, and), verbs (show, is), and adverbs (please, immediately). The dictionary contains a syntactic description
of these words. It also contains semantic information for some
nouns, question pronouns and verbs. For example, the semantic
information associated with the question pronoun "who" is:

\[
\text{SEM } ((\text{CATEG HUMAN}) (\text{CATEG GROUP}))
\]

This information informs the system that "who" can refer to any
human or group of humans (company, etc). The definition of the
noun "information" is:

\[
\text{SEM } ((\text{GENERAL } \ast\text{ANY}\ast))
\]

This tells the system that "information" can refer to any
field(s) or relation(s) contained in the database.

The semantic definition of verbs is more complex. Domain
independent verbs fall into three categories: command verbs
(show, count), relation verbs (be, have), and auxiliary verbs
(do, is). The General Dictionary contains semantic information
for both command verbs and relation verbs. The meaning of
auxiliary verbs is defined within the semantic processor and
will be explained in the next chapter.

Two types of command verbs are presently dealt with by the
system. These are verbs which request information in a report
format (show, print), and verbs which request the system to
perform a function (count). These verbs are identified by the
semantic markers COM-VERB and COUNT-VERB respectively. These
verbs also require verb frames. The verb frame for the verb
"show" is shown below.
This definition allows the system to answer queries like:

You show me the computers.
Show the good programs to me.

A complete description of verb frames and their function is given in chapter 6. A portion of the General Dictionary is listed in Appendix E.

5.5.2 Global Dictionary

The Global Dictionary is a large dictionary which contains the syntactic category and other syntactic features of a large number of words. This dictionary supplies the syntactic definitions for words contained in the Database Schema, the Database Schema Library, the inverted index, and the parser (for unknown words). The main benefit of including such a dictionary in the system is to free the DBA from specifying numerous syntactic definitions of words for each domain. It also ensures these definitions are correct and available for every domain.

In the current system this dictionary only contains the words which are used in the two test domains. A more reasonable dictionary for commercial applications may contain over 100,000 words. Since the majority of these words will not be used in a given domain, the dictionary could be stored on either a disk or even a tape. Only the words contained in the domain definition
are extracted and stored in fast access memory (disk or main). A sample portion of the Global Dictionary is listed in Appendix F.

5.6 Compiling The Information: Active Domain Dictionary

The Active Domain Dictionary is the primary source of syntactic and semantic knowledge used by the system. It is compiled from the domain independent information (General Dictionary and selected words from the Global Dictionary) and the domain dependent information (Database Schema, Database Schema Library, and the inverted index). This syntactic and semantic information is stored on LISP's property lists for each word. Figure 5.5 contains a sample of some of the entries in the Active Domain Dictionary used for the Program database. A larger sample is contained in Appendix G.

<table>
<thead>
<tr>
<th>WORD</th>
<th>PROPERTY</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTER</td>
<td>N</td>
<td>s</td>
</tr>
<tr>
<td></td>
<td>L-NOUN</td>
<td>((RELATION COMPUTER))</td>
</tr>
<tr>
<td></td>
<td>L-DESC</td>
<td>((RELATION PROGRAM))</td>
</tr>
<tr>
<td></td>
<td>NPR</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>PART-OF</td>
<td>((COLOUR COMPUTER))</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>s</td>
</tr>
<tr>
<td></td>
<td>ELM-OF</td>
<td>((FIELD_DESCRIPTOR DSC))</td>
</tr>
<tr>
<td></td>
<td>PART-OF</td>
<td>((FRENCH TEACHER))</td>
</tr>
<tr>
<td></td>
<td>ADJ</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>ADV</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>MEM-OF</td>
<td>((TYPE RATING))</td>
</tr>
<tr>
<td></td>
<td>CODE-AS</td>
<td>(* &gt; 3)</td>
</tr>
<tr>
<td></td>
<td>V</td>
<td>ES</td>
</tr>
<tr>
<td></td>
<td>FEATURES</td>
<td>(REL-VERB TRANS PASSIVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTRANS)</td>
</tr>
<tr>
<td></td>
<td>VFRAME</td>
<td>(((SUBJ ....</td>
</tr>
</tbody>
</table>
Although the Active Domain Dictionary is well suited for use by the system, it is not suitable for use by the DBA. In order to make any changes to the specification of a database, the DBA can make changes to the Database Schema and then recompile the Active Domain Dictionary. Hence the DBA only needs to deal with the structured representation of the database contained in the Database Schema.
VI. SYSTEM DESIGN III - SEMANTIC INTERPRETATION

The job of the semantic processor is to take the deep structure produced by the syntactic processor and transform it into a Standard Query Representation (SQR). The SQR will then be transformed into a database query in the desired query language by the database interface routine.

6.1 Interpreting The Deep Structure

The initial task in interpreting the deep structure is to determine the database relations or fields represented by each phrase in the query. Once this process is done, the system will try to match the deep structure to a verb frame. During this process, various types of semantic ambiguity may be resolved.

6.1.1 Associating Phrases With Relations And Fields

The first step in the semantic interpretation of the deep structure is to determine every possible field and relation that each noun phrase, prepositional phrase, adverb, and predicate may represent with respect to the database. Once these relations and fields are determined, the result is stored in the SEMREG register for later use.

Adverbs and Adjectives which act as predicates (complements) are the simplest cases since they consist of only one word. These words are normally defined as database elements of specific fields. In the query:
What programs are good?

the adjective "good" is defined as a member of (MEM-OF) the field PROGRAM RATING. This field will then be stored in the SEMREG register.

Prepositional phrases and noun phrases are more complicated since they usually contain more than one word. In this case the semantic interpreter checks the syntactic category of the head noun of the given phrase. Using this information, it will determine what relations and fields the word identifies by examining the semantic roles (L-NOUN, ELM-OF, MEM-OF, CATEG, TYPE, LTYPE) of the word. For example, in the phrase:

What programs...

the noun "program" represents the relation PROGRAM since it is defined as a noun which identifies the relation (L-NOUN). The proper noun "Apple" in the phrase:

the Apple

could refer to either the field COMPUTER MAKE or PROGRAM PUBLISHER since it is a database element (ELM-OF) of each of these fields.

Question pronouns, such as "What", are slightly more complex when they appear as head nouns. In this case the system notes that "WHAT" may represent the semantic categories (CATEG) PHYSICAL OBJECT, ABSTRACT OBJECT, and EVENTS. In the program database, "WHAT" may refer to the physical objects PROGRAM or COMPUTER, or the abstact object DESCRIPTOR. Hence the SEMREG
register will then be set to these three relations.

The semantic interpreter also examines the describers of the head noun in order to help resolve any semantic ambiguity. For example, in the phrase:

What IBM things,

the word "thing" may represent any physical object (COMPUTER or PROGRAM) in the database. The system will then find that "IBM" is an element of the field "COMPUTER MAKE", and conclude that the given phrase refers to computers and not programs.

As shown above it is possible for a phrase to refer to more than one field and/or relation. This ambiguity can be resolved by either analysing the phrase's role with respect to the main verb, or by prompting the user for the intended interpretation. Both of these cases will be dealt with in the following sections.

6.1.2 Verb Frames

One of the main processes in the semantic interpretation of a query is determining whether a verb frame suitably matches the deep structure. Every verb which can act as a main verb must have at least one verb frame associated with it or be defined as a synonym for a verb which does.
6.1.2.1 **Verb Frame Structure**

A verb frame consists of a verb followed by a number of slots. Each slot refers to a syntactic component in the query which plays a semantic role with respect to the verb. Figure 6.1 lists the syntactic components which can act as slots.

<table>
<thead>
<tr>
<th>Syntactic Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBJ</td>
<td>Subject of a sentence</td>
</tr>
<tr>
<td>PRED</td>
<td>Predicate (or complement)</td>
</tr>
<tr>
<td>DOBJ</td>
<td>Direct Object</td>
</tr>
<tr>
<td>IOBJ</td>
<td>Indirect Object</td>
</tr>
<tr>
<td>MOD</td>
<td>Modifiers of the main verb (Adverbs)</td>
</tr>
<tr>
<td>POBJS</td>
<td>Prepositional phrases are required by the cooccurrence restrictions of certain verbs (eg. double object verbs -- sell, send).</td>
</tr>
</tbody>
</table>

**Figure 6.1 - Verb Frame Slots**

Each slot has a semantic classification associated with it. This semantic classification determines what each syntactic phrase can represent with respect to the database. Figure 6.2 contains a list of the semantic classifications defined in the system.
<table>
<thead>
<tr>
<th>Semantic Classification</th>
<th>Example Definition</th>
<th>Example Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATION</td>
<td>(RELATION PROGRAM)</td>
<td>What programs run...</td>
</tr>
<tr>
<td>FIELD</td>
<td>(FIELD COMPUTER RAM)</td>
<td>How much memory...</td>
</tr>
<tr>
<td>CATEG</td>
<td>(CATEG HUMAN)</td>
<td>Who is...</td>
</tr>
<tr>
<td>TYPE</td>
<td>(TYPE LOCATION)</td>
<td>Where is...</td>
</tr>
<tr>
<td>L-TYPE</td>
<td>(L-TYPE CITY)</td>
<td>What cities are...</td>
</tr>
<tr>
<td>GENERAL</td>
<td>(GENERAL <em>YOU</em>)</td>
<td>You print...</td>
</tr>
<tr>
<td></td>
<td>(GENERAL <em>ME</em>)</td>
<td>Show me the...</td>
</tr>
<tr>
<td></td>
<td>(GENERAL <em>ANY</em>)</td>
<td>Print anything.</td>
</tr>
</tbody>
</table>

Figure 6.2 - Semantic Classifications for Verb Frame Slots

An example of a verb frame for the verb 'publish' is shown below:

(PUBLISH VFRAME ((SUBJ (FIELD PROGRAM PUBLISHER))
                 (DOBJ (RELATION PROGRAM))))

This definition will allow the system to process queries such as:

Who publishes LOGO?
Does TERRAPIN publish good programs?
Is LOGO published by TERRAPIN?

It is also possible to define verb frames which contain slots which are optional. This allows a specified phrase to be omitted from a query without having to specify a separate verb frame. For example, both the subject and indirect object are defined as optional (OPT) for the verb "show".
(SHOW VFRAME ((SUBJ (GENERAL *YOU*) OPT)
(DOBJ (GENERAL *ANY*))
(IOBJ (GENERAL *ME*) OPT)))

Below are some example queries which will match this verb frame.

Show the good programs to me.
You show me the good programs.
Show the good programs.

When prepositional phrases appear in a verb frame, it is possible to state the prepositions which are acceptable in a query. For example, the following verb frame states that either the preposition "on" or "with" may appear in the stated phrase.

(RUN VFRAME ((SUBJ (RELATION PROGRAM))
(PP (ON WITH) (RELATION COMPUTER))))

This definition will match the following queries:

What runs on the IBM PC?
Does LOGO run with the Macintosh computer?

It is also possible to define a prepositional phrase verb frame slot which will not place any restrictions on which prepositions can be matched. This is done by replacing the preposition definition by an asterisk (*).

This structure for verb frames is similar to Filmore's Case Grammar, except that the slots correspond to syntactic components (Subject, Direct object, etc.) rather than the traditional cases (Recipient, Actor, etc.). The main motivation in applying a case grammar in a case grammar based system is that it is often possible to determine the intended role of a
phrase without knowing the full meaning of the words in the phrase. This ability is useful for processing words which are not defined in the system. It will be shown in this chapter that such processing can also be done by using verb frames. The big advantage of using verb frames over the traditional case system is that they do not require a person with a strong linguistic background to design them.

6.1.2.2 Matching The Deep Structure To A Verb Frame

The process of determining whether a verb frame matches the deep structure is accomplished by systematically comparing the possible contents of each slot in the verb frame with the corresponding phrase in the deep structure. The possible contents for each phrase in the deep structure is stored in the SEMREG register, while the possible contents of each slot in the verb frame is defined by the semantic classification. This matching process is normally done by taking the intersection of the fields defined by SEMREG and the semantic classification. For example, in the query:

Who publishes LOGO?

"who" can refer to either a PROGRAM PUBLISHER or a COMPUTER MAKE. However the verb frame for the verb "publish" states that the subject must be a PROGRAM PUBLISHER. Hence the subject is taken to be a PROGRAM PUBLISHER. "LOGO" is then matched as a PROGRAM since it is contained in the field PROGRAM NAME, which is defined as an IDENTIFIER for the relation PROGRAM.
Once a verb frame has been successfully matched to the deep structure, the given verb frame is stored in the VFRAME register in the deep structure. If any ambiguity resulting from a phrase referring to more than one relation or field can be resolved by the match, the SEMREG register is updated. If the query cannot be matched to a verb frame, the system is unable to answer the query and an appropriate error message is printed out.

It is possible to have more than one verb frame defined for a given verb. This allows a verb to have more than one meaning. In the program database, the verb "run" has the following three definitions:

- COMPUTERS RUN PROGRAMS
- PROGRAMS RUN ON COMPUTERS.
- PROGRAMS RUN RATING. (eg. good)

Although the meanings of these three definitions are similar, it is possible to have completely different meaning such as:

- ATHLETES RUN RACES, or
- TRAIN RUNS FROM LOCATION TO LOCATION.

When more than one verb frame is defined for a verb, there is a possibility that more than one can be applied to a given query. In order to match the query to the most appropriate verb frame, two separate attempts, or passes are made. This first pass requires the deep structure to exactly match the verb frame. In this case every verb frame slot must match with the corresponding register in the deep structure. Phrases which do not correspond to a verb frame slot are not permitted in this
pass.

If a verb frame does not exactly match the query, a second pass is made. This pass allows for phrases referring to optional (OPT) slots to be omitted as well as slots with more than one possible interpretation to be matched. It also allows for some phrases in the deep structure to be left unmatched (Taylor and Rosenberg 1975). In some cases these phrases may be ignored. If a prepositional phrase is not matched to a verb frame slot, it can be treated as a modifier of a noun phrase. Each of these topics will be discussed in the following sections.

6.1.3 Resolving Multiple Matched Slots

It is possible that a phrase may have more than one possible interpretation even after a verb frame has been matched. In this case the user is prompted for the desired interpretation. For example, in the query:

Print the Apples.

"Apples" can refer to either a PROGRAM PUBLISHER or a COMPUTER MAKE. In this case the user is presented with the following prompt:

Does 'APPLE' best refer to
1. a computer manufacturer
2. a computer program publisher

The query will then be answered with the chosen interpretation.
6.1.4 Ignoring Phrases

In some cases there may be a phrase or a word in a query which prevents it from being matched to a verb frame. These words or phrases can often be ignored without effecting the meaning of the query. For example, in the query:

Print the programs quickly.

the adverb "quickly" will prevent the query from being matched to the verb frame for 'print'. In this case the system will display the following prompt to the user:

OK to ignore "quickly"?

If the user chooses to ignore the phrase, the query can be successfully answered. Otherwise an error message will be printed out.

6.1.5 Processing Prepositional Phrases

Prepositional phrases can play two different roles in English. They can either play a role with respect to the main verb, or they can act as a modifier of a head noun. The parser places each prepositional phrase which appears at the end of a query in the PPOBJS register (with the exception of indirect objects and passive constructions). Hence it is the job of the semantic processor to determine the role that the prepositional phrase plays in the query.

During the matching process, the system will try to match
prepositional phrases with any verb frame slots which specify them. For example, the query:

What programs run on the IBM?

will match the following verb frame:

\[
\text{(RUN VFRAME ((SUBJ (RELATION COMPUTER)) (PP (ON WITH) (RELATION PROGRAM)))})
\]

If a prepositional phrase (PP) contained in the PPOBJS register is not matched to a verb frame slot, it will be treated as a modifier to a head noun (HN). As stated in chapter 4, there may be more than one potential HN which a given PP may modify. In order to determine which HN a given PP modifies, the system uses the following three heuristics:

1. The PP must appear to the right of the HN
2. PP's tend to immediately follow the HN which they modify
3. PP's are usually semantically related to the HN
   (eg. PP may refer to a property of the given HN)

In order to determine which HN a given PP may modify, the system uses two factors derived from the above heuristics. They are:

1. the number of head nouns between the given head noun and the preposition phrase (distance), and
2. a defined "semantic association factor" which measures the semantic relationship between the database entities represented by the phrases.

The semantic association factor consists of a number between 1 and 5, and is defined in figure 6.3. The semantic association factors between relations are defined in the database schema,
while the association factor between the fields within relations are implicitly defined.

<table>
<thead>
<tr>
<th>Association Factor</th>
<th>Description and Example</th>
</tr>
</thead>
</table>
| 1                  | Fields within the same relation  
                      (computer model and computer make) |
| 2                  | Relations which refer to the same entity  
                      (computer and computer-subparts) |
| 3                  | Relations which are commonly associated together and are commonly linked together by verbs.  
                      (computers and programs)  
                      (parts and suppliers) |
| 4                  | Relations which are not commonly associated together but have some possible semantic ties.  
                      (descriptors and computers) |
| 5                  | Relations which are not related together at all.  
                      (computer programs and hockey players) |

Figure 6.3 - Semantic Association Factors

These two factors (distance and semantic association) are combined together in the following equation to determine which HN to attach the given PP to. The HN with the lowest calculation will be chosen to be the most appropriate choice.

\[
\text{DISTANCE} + \text{factorial} (\text{ASSOCIATION FACTOR})
\]

Note that the factorial expression in this equation gives more weight to the association factor than the distance factor when the association factors between the given PP and the various HN vary substantially. For example, if the association factor between a PP and one HN is 1, while the association factor between the same PP and another HN is 5, the PP will be attached to the HN where the association factor is 1 regardless of the
distance factor (e.g. \( (D1 + 1!) < (D2 + 5!) \) unless \( D1 > 25 \)).

The following query will demonstrate how this calculation works.

What computers run good programs with 128 KB?

In the above query, the phrase "with 128 KB" could modify either "computers" or "programs". In this case the association factor between "128 KB" and "computers" is 1 and the distance is 1 (1 + 1! = 2), while between "128 KB" and "programs" the association factor is defined as 3 and the distance is 0 (0 + 3! = 6). By applying the above equation, the system will appropriately conclude that "128 KB" modifies "computers".

Although this equation is ad-hoc, it has proven to be adequate in the two sample databases. For a commercial system this equation would probably have to be refined.

6.1.6 Relative Clauses

In order to process queries containing relative clauses, the system must match verb frames to both the outer query and the embedded query. This is implemented by matching the outer query first, and then matching the the embedded query, where the subject of the embedded query is forced to match the interpretation of the phrase which it modifies. This process is shown by the following example:

What computers run programs which teach chemistry?

First the outer query is matched to each verb frame for the verb
"run". Once a successful match is found, the verb frame for "teach" is then matched to the embedded query. In this case the direct object of the verb "run" is treated as the subject for the verb "teach".

6.2 Building The Standard Query Representation

After each phrase in the query has been either matched to a verb frame slot, ignored, or attached to a head noun (PP's only), the system will attempt to build the Standard Query Representation (SQR) from the resulting deep structure. This is done by analyzing each phrase in the deep structure and determining the semantic function of each word in them. Once the SQR has been constructed, the system will invoke the database interface routine, which will transform the SQR into an actual database query.

6.2.1 SQR Structure

The SQR consists of a number of registers, called lists, which contains the needed information to construct a database query in most common database query languages. This information includes the actual database fields which are to be displayed in the response, the restrictions to be placed on the data to be selected, the relations which need to be accessed, information on how the relations are to be joined together, and some additional information on how the question is to be answered.
6.2.1.1 The Select List

The Select List (SLIST) contains a list of fields which are to be printed in the response. For the query:

Print the program publishers.

the SLIST will contain the field PROGRAM PUBLISHER.

6.2.1.2 The Restriction List

The Restriction List (RLIST) contains a list of restrictions which will be placed on the items to be selected from the database. The RLIST for the query:

Print the Macintosh programs.

will be set to:

(COMPUTER.MODEL = "MACINTOSH").

For more complex queries, the RLIST may include many restrictions.

6.2.1.3 The Relation List

The Relation List, or the "from-what-relation" list (FLIST), contains each relation name which needs to be accessed in order for the query to be answered. The FLIST for the query:

What Apple computers run LOGO?

will contain the relations COMPUTER and PROGRAM, since both of
these relations will need to be accessed in order for the query to be answered.

6.2.1.4 The Join List

The Join List (JLIST) consists of any joins needed to join the relations together. For example, the JLIST for the query, What computers run LOGO?

will be set to:

(COMPUTER.CNO = PROGRAM.CNO).

If the solution of the query only requires one relation to be accessed, the JLIST will be empty.

6.2.1.5 The Distinct Flag

The Distinct Flag determines whether the solution of a query will contain duplicate database elements. For example, the query:

Print the distinct program publishers.

will cause each publisher to be printed once, rather than having the publisher printed separately for each program that they publish. The words "unique", "different", and "discrete" are also used to state this option.
6.2.2 Determining The Question Type

The first step in building the SQR is to determine the type of the query. The system can answer four basic types of queries. Each of these query types has some unique features pertaining to how the question is to be answered.

The most common types of query are those which contain a question determiner (or question noun) such as "who", "what", "where", or "when". These are referred to as "WH" queries and are answered in a report format.

Another popular type of query is termed as PRINT query. This type of query has a main verb such as "print", "show", "display", or "list". These verbs are identified by having the semantic marker "COM-VERB" associated with them in the Active Domain Dictionary. These queries are also answered in a report format. The main feature of these queries is that the direct object contains the information requested by the user. Some examples are listed below:

- Print the good programs.
- Can you display the IBM computer types for me?

The system also answers questions where the answer is either "yes" or "no". These queries, termed "YES/NO" queries, have many different constructions. Sentences in declarative form, such as:

- The IBM PC runs LOGO.

is taken to be a YES/NO query. They are also identified by
being in the interrogative mood and either having a copula verb as a main verb, as in:

Is LOGO a good program?

or having either "do", "be", "have" or a modal (e.g. can, will) as an auxiliary verb, as shown be the following queries:

Does LOGO run on the APPLE?
Is Terrapin publishing LOGO?
Can LOGO run on the APPLE?

The other two types of queries answered by the system are ones which request the system to count some selected database items. These are identified by either having "count" as a main verb (COUNT), or having the question determiner "how many" (HOWMANY). Examples of these are shown below:

Count the program publishers.
How many programs run on the IBM PC?

In some cases the question determiner "how many" can be used to simply print out the contents of a field. For example, if the TYPE of the field is set to Q-CNT (a countable quantity), a query would be constructed to request the number contained in this field. This is shown be the following query:

How many bolts were supplied to job J2 by Smith?
6.2.3 Analyzing The Verb Frame

In addition to determining which slots must be matched in a query, the verb frame may also specify how the relations associated with the verb are to be joined together. For example, the verb "run" requires the program and computer relations to be joined together. This information is extracted from the verb definition in the database schema (JOININFO) and added to the JLIST (join list) for use in the actual database query.

The verb frame is also used to select the phrases in the deep structure which are to be processed in the formation of the SQR.

6.2.3.1 Determining The Question Element

In order to determine what information the user wishes to be displayed in queries which are answered in a report format, the system must identify the phrase which is the question element. The question element is usually identified by the parser and is stored in the QE register. This is the case for questions containing words like "who" or "how many". If the query type is set to PRINT (command verbs), the question element is taken to be the direct object of the query.
6.2.3.2 Analyzing Noun And Prepositional Phrases

The major task in building the SQR is determining the semantic role of each word in each noun phrase. Since a prepositional phrase consists of preposition and a noun phrase, prepositional phrases and noun phrases are handled essentially by the same process.

6.2.3.2.1 Head Noun

The head noun of the noun phrase is analyzed first using the semantic interpretation contained in the SEMREG register. Depending on the syntactic category of the head noun, a number of things can happen. If the head noun is either a number, noun, or proper noun, and is defined as a database element (ELM-OF or MEM-OF), then the word and its associated database field will be formed into a restriction and will be added to the RLIST. Also the relation in which the field is contained will be added to the FLIST. For example, the phrase:

"the Macintosh" (SEMREG (FIELD COMPUTER MODEL))

will have the following effects:

RLIST <- (COMPUTER.MODEL = "MACINTOSH")
FLIST <- (COMPUTER)

If the head noun is a compound proper noun contained in the database, the entire phrase is used in the restriction. The following example contains an abbreviated compound proper noun:
In this case the matched database element stored in the ANPR register (actual proper noun) is used to form the following restriction:

(PROGRAM PUBLISHER = JOHN WILSON AND SONS LIMITED)

Another case is when the head noun is defined as word which identifies a relation or a field (L-NOUN). In this case the relation represented by the field or relation will be added to the FLIST. The following examples will both cause the given effect.

"the program" (SEMREG (RELATION PROGRAM))
"a publisher" (SEMREG (FIELD PROGRAM PUBLISHER))

FLIST <- (PROGRAM)

Note that if the phrase happens to be the question element of the query, the fields identified by the head noun will be added to the SLIST. For example, the phrase:

"which publishers" (SEMREG (FIELD PROGRAM PUBLISHER))

will cause the following effect:

SLIST <- (PROGRAM PUBLISHER)

It is also possible that the head noun is not semantically defined with respect to the database. In this case the SEMREG
register is set to "*UNKNOWN". For example, the word "LOTUS" in the following query is not defined.

What computers run LOTUS (SEMREG (*UNKNOWN))

Since the verb frame for "run" specifies that the direct object should represent a computer program, the system will print the following response:

(LOTUS) is not (a computer program) contained in this database.

If the head noun is a question pronoun, the fields represented in SEMREG are added to the SLIST and the relation name is added to the FLIST. Note that if SEMREG is set to a relation, the fields to be selected are determined by the IDENTIFIER attribute in the database schema. The following phrase will cause the given lists to be updated.

"What" (SEMREG (RELATION COMPUTER))
SLIST <- (COMPUTER.CNO COMPUTER.MAKE COMPUTER.MODEL)
FLIST <- (COMPUTER)

The system handles personal pronouns in a limited way. It can only deal with pronouns such as "you", "me", and "us" when they are matched with a verb frame containing the semantic classification "GENERAL *YOU*" or "GENERAL *ME*". For example, the verb frame for "show" will match the following query:

You show me the good programs.

Since these pronouns have no effect on the actual database
query, they are simply ignored.

6.2.3.2.2 Modifiers

The next task is to analyse the adjectives and the prepositional phrases which act as modifiers to head nouns. Often words which are contained in the database (ELM-OF, MEM-OF) act as modifiers. Such words will be used to put restrictions on the data to be selected. Either of the modifiers in the following queries will cause the given restrictions to be added to the SLIST.

the good Terrapin programs
the good programs from Terrapin
SLIST <- ((PROGRAM.RATING > 3)
            (PROGRAM.PUBLISHER = "TERRAPIN"))

A more complex case arises when the modifier refers to a database field in another relation. In the following phrases, the descriptor "chemistry" modifies "programs".

the chemistry programs
the programs on chemistry

In this case the default join between the relations PROGRAM and DESCRIPTOR will be added to the JLIST. The restriction (DESCRIPTOR.DSC = "CHEMISTRY") will be added to the RLIST.

Words which are used to identify relations (L-NOUN) can act as modifiers to the head noun. For example, the word "program" in the following phrases is used to identify the program relation.
the program LOGO (SEMREG (PROGRAM NAME))
the program publisher (SEMREG (PROGRAM PUBLISHER))

The meaning of such words were used to determine the contents of SEMREG and have no other effect in the semantic interpretation. Words which are defined as describers (L-DESC) also have no effect in the construction of the SQR. For example, the word "computer" in the following phrase acts as a describer (classifier) for the relation program.

the computer program (SEMREG (PROGRAM))

Note that the word "computer" is also defined as an identifier (L-NOUN) for the relation computer. In this case the ambiguity is resolved by the context in which the word is used.

As previously stated, words such as "distinct" and "unique" are used to prevent the system from printing out duplicate database items. If these words appear as modifiers, they will simply cause the DISTINCT flag to be set.

A phrase may contain a modifier which does not make any semantic sense with respect to the database, even though it is syntactically correct. In the query:

Print the German programs.

the word "German" is not semantically defined. The system will then respond with the prompt:
The word (GERMAN) in the phrase (the German programs) has no meaning in this query.
Is it OK to ignore it? (Y/N)

If the user answers "YES", the remainder of the query will be answered. Otherwise the following message is printed:

There are no such database items which satisfy the above restriction.

It is possible for prepositional phrases which modify head nouns to contain modifiers as well. The semantic interpretation of these phrases is done in the same way as described above. For example, the PP in the phrase:

the computers with 64 KB memory

will cause the following restriction:

SLIST <- (COMPUTER.RAM = 64)

6.2.3.2.3 Conjunction

As stated in chapter 4, the system is capable of handling some limited forms of conjunction. It allows a number of describers (adjectives, nouns, etc.) to be connected together by "and", "or", or a comma. If the words are database elements, they will be interpreted as a complex restriction. An example of this type of conjunction is shown below:

the red, green, or blue parts
The system will construct the following restriction for this phrase:

\[
\text{RLIST} \leftarrow \left( (\text{P.COLOUR} = "\text{RED}" \text{ OR } \text{P.COLOUR} = "\text{GREEN}" \text{ OR } \text{P.COLOUR} = "\text{BLUE}" ) \right)
\]

In many cases where conjunction is used there may be more than one possible interpretation. For example, the phrase:

the Apple and IBM computers

could be interpreted as either:

(1) the Apple computers and the IBM computers
    (the Apple or IBM computers)

or (2) the Apple and IBM computers
    (computers which are both Apple and IBM)

In order to determine the interpretation of this query, the system uses the heuristic that users will not ask queries which contain impossible restrictions. Since the database schema defines that any computer can only have one computer make associated with it (one to one), the second interpretation is impossible. Hence the system will use the first interpretation.

The above heuristic cannot resolve all ambiguity of this type. For example, the following phrase can also be interpreted in the same two ways:

the French and Chemistry programs

Since a program can have many descriptors, it is not obvious
which interpretation is intended by the user. In this case the user should be asked which interpretation is intended and have the system answer the query accordingly. At present, the system will only print out a message saying that it cannot deal with the given conjunction.

6.2.3.2.4 Relative Clauses

Queries containing relative clauses are dealt with by having the Semantic Processor build a separate SQR structure for the embedded query. This SQR structure will essentially act as a restriction to the head noun in the outer query which it modifies. An example of a database query resulting from a query containing a relative clause is given in section 7.2.

6.2.3.2.5 Determiners

The system does not presently extract any semantic knowledge from determiners. For example, for the query:

Print a good spreadsheet program.

the system will print out information on every "good spreadsheet program" contained in the database.
6.2.3.3 Analyzing Adverbs And Predicates

The task of analyzing verb frame slots which correspond to adverbs and predicates consisting of an adjective is fairly simple since they usually function as a restriction. For example, the word "excellent" in the following query acts as a modifier of "programs".

Which programs are excellent?

Since the SEMREG register for this phrase is set to "FIELD PROGRAM RATING", the following restriction will be added to the SLIST:

\[
\text{SLIST} \leftarrow (\text{PROGRAM.RATING} = 5)
\]

Note that "excellent" is defined as a coded database element (CODE-AS (*=5)).

6.3 Summary Of Semantic Analysis

The Semantic Interpreter has been designed to be domain independent. It is completely data driven. No additional programming is required to adapt it to a new domain.

The system allows words, including nouns and verbs, to have more than one semantic definition. The chosen meaning will be determined by the context in which it appears. If any ambiguity cannot be resolved by the system, the user is prompted for the intended interpretation.

The semantic definitions of most words, including all the
domain dependent ones, are contained in the Active Domain Dictionary. The definition of some domain independent words, such as some auxiliary verbs, is built into the semantic interpreter.
VII. SYSTEM DESIGN IV - DATABASE INTERFACE AND RESPONSE GENERATION

After the Standard Query Representation (SQR) has been produced by the Semantic Processor, it is passed to the Database Interface to be transformed into a database query in a specified query language. This query is then executed by the DBMS. The results obtained by this query are then passed to the Response Generator, which in turn displays the result(s) to the user.

7.1 The Sequel Query Language

The Database Interface currently constructs database queries in the Sequel (SQL) database query language. This language, which was developed at IBM in the early 70's, is used by many relational DBMS, including Oracle (Chamberlin 1977).

The fundamental retrieval command in Sequel is the "SELECT-FROM-WHERE" command. For example, the query:

How much memory does the PC have?

can be expressed in Sequel as:

```
SELECT COMPUTER.RAM
FROM COMPUTER
WHERE COMPUTER.MODEL = 'PC'
```

This will cause the amount of memory to be printed for every entry in the computer relation where the computer model value is set to "PC".

If the query states that duplicate database values are not
to be printed, the DISTINCT operator is used. The query:

Print the unique program publishers.

can be coded as follows:

```
SELECT DISTINCT PROGRAM.PUBLISHER
FROM PROGRAM
```

If the values selected by the query are to be counted, the COUNT function is used. For example, the query:

Count the Microsoft programs.

is translated to the following Sequel query:

```
SELECT COUNT (PROGRAM.PNO)
FROM PROGRAM
WHERE PROGRAM.PUBLISHER = 'MICROSOFT'
```

A complete definition of the Sequel query language can be found in Date (Date 1977).

7.2 Assembling The Sequel Query

The task of assembling the Sequel query is fairly straightforward. The Database Interface is passed the select list (SLIST), relation list (FLIST), restriction list (RLIST), join list (JLIST), DISTINCT flag, and the query type (QTYPE) by the semantic processor. For example, the semantic interpreter will pass the following results for the given query:
What Apple or IBM computers run LOGO?

QTYPE = WH
SLIST = (COMPUTER.CNO COMPUTER.MAKE COMPUTER.MODEL)
FLIST = (COMPUTER PROGRAM)
JLIST = ((COMPUTER.CNO = PROGRAM.CNO))
RLIST = ((COMPUTER.MAKE = "APPLE" OR COMPUTER.MAKE = "IBM")
(PROGRAM.NAME = "LOGO"))
DISTINCT = NIL (FALSE)

In this case the Database Interface will determine what database values are to be printed by the SLIST. These fields are then used with the SELECT clause. The relations contained in FLIST are used with the FROM clause. The joins (JLIST) and restrictions (RLIST) are included with the WHERE clause. Note that these restrictions are connected together by the "AND" operator since they all will apply to the end result. The Sequel query for the previous example is shown below:

SELECT COMPUTER.CNO, COMPUTER.MAKE, COMPUTER.MODEL
FROM COMPUTER, PROGRAM
WHERE
(COMPUTER.CNO = PROGRAM.CNO)
AND (COMPUTER.MAKE = 'APPLE' OR COMPUTER.MAKE = 'IBM'
AND (PROGRAM.NAME = 'LOGO'))

The formation of database queries for queries which contain relative clauses is slightly more complex. Sequel has the feature of allowing an embedded query within a query. The embedded query will simply act as a restriction for the data selected by the outer query. Since this construction closely models the semantic construction of a relative clause, it makes sense to use it. The following example shows how a query containing a relative clause is expressed in Sequel.
What Apple computers run programs which teach chemistry?

```
SELECT COMPUTER.CNO, COMPUTER.MAKE, COMPUTER.MODEL
FROM COMPUTER, PROGRAM
WHERE
  (COMPUTER.CNO = PROGRAM.CNO)
  AND (COMPUTER.MAKE = 'APPLE')
  AND (PROGRAM.PNO IN
    (SELECT PROGRAM.PNO
     FROM PROGRAM, DESCRIPTOR, PROGDESC
     WHERE
       (PROGRAM.PNO = PROGDESC.PNO AND
        PROGDESC.DNO = DESCRIPTOR.DNO)
     AND (DESCRIPTOR.DSC = 'CHEMISTRY')))
```

In this case the embedded query will extract all of the program keys (PROGRAM.PNO) for programs on chemistry. Each program selected in the outer query will have to be contained in this set.

7.3 Response Generation

The Response Generator receives the data extracted by the DBMS and forms a response. Most of the responses are displayed in report format. In this case the system simply displays the data in the format produced by the DBMS. Examples of this format are shown in Appendices H and I.

The only true response generation performed by the Response Generator is for YES/NO queries. In this case the system checks to see if any data has been extracted by the DBMS and replies accordingly. If the query:

```
Does LOGO run on the Apple II?
```

does not produce any results, the following response is printed:
I don't know. There are no records in the database which satisfy the given criteria.

Otherwise a response like the following is printed:

YES. There is 1 record in the database which satisfies the given criteria.
Would you like to see it? (Y/N)

Note that the user is given the option of seeing the extracted information.

Most of the effort in forming responses has been put into error messages and messages for queries which cannot be answered. These responses have been dealt with by the syntactic and semantic processors.
One of the main focuses of the design of the system is portability. In this chapter we will look at both domain portability and database portability.

8.1 Domain Portability

In order to implement a new domain, a Database Schema must first be developed. This is then compiled together with the inverted index, selected information from the Database Schema Library and the Global Dictionary, and the domain independent information to form an Active Domain Dictionary for the given domain.

The easiest way to develop a new domain is to first examine a number of possible queries which may apply to the given database. After the DBA has an idea of the vocabulary which will have to be defined, the Database Schema is developed. As previously mentioned, the Database Schema consists of the database definition, verb frames, and the default joins.

The process of developing the database definition is very similar to developing the database itself. This process requires the DBA to analyze each relation and field, and to supply various types of semantic information to the attributes associated with them (L-NOUN, CONTENTS, etc).

The process of defining new verb frames for domain dependent verbs is the most difficult task since it requires some elementary knowledge of linguistics. To construct a new
verb frame, the DBA must analyze numerous queries for each verb and determine what syntactic phrases (subject, object, etc.) the verb requires, and what each of these syntactic phrases represents with respect to the database. For verbs which access more than one relation, information on how the relations are joined together must also be specified (JOININFO and JOINTYPE).

The DBA must also specify the default joins for every two relations that can be joined together. This should be trivial for DBA's who are familiar with relational database systems. They must also define the semantic closeness factor, which is simple if the DBA is familiar with the domain of the database.

In some cases the DBA may want to add new definitions to the Database Schema Library. This is useful if the definitions are to be used in other databases in the organization. It is also possible that the DBA may want to modify a definition. For example, he/she may want to change the definition of rating from a 5 point scale to a 3 point scale. The DBA may also want to add new words to the Global Dictionary. Obviously the Global Dictionary will not contain every possible word which a user may need.

8.2 The SUPPLIER/PART/JOB Database

In order to test out the feasibility of implementing a new domain, a supplier/part/job database was implemented (Date 1977). This database consists of the following four relations:
SUPPLIER [SNO SNAME STATUS CITY]
PART [PNO PNAME COLOUR WEIGHT]
JOB [JNO JNAME CITY]
SPJ [SNO PNO JNO QTY]

This database was similar enough to the Program database so that many of the same definitions could be used (Physical Object, General Name, etc.). This was useful since it demonstrates that the semantic information associated with these definitions is portable. Since the Database Schema Library only contained definitions which were used in the Program database, the following definitions were added:

```
TYPE Q-MASS (WEIGHT)
TYPE Q-CNT (QUANTITY)
TYPE COLOUR
TYPE LOCATION
L-TYPE CITY
```

Verb frames for the verbs "locate" and "weigh" were also added to the Database Schema Library. Each of these are popular definitions and would be included in a commercially oriented system.

Once the new definitions were added to the Database Schema Library, the construction of the Database Schema was fairly easy. Most of the words which are defined in it were extracted from the sample queries contained in Date (Date 1977). Some sample queries which the system handles are listed in figure 8.1. The actual Database Schema for this database is contained in Appendix I, while a sample session is in Appendix J.
Print the status of the suppliers in Paris.
What suppliers supply job J1 with red nuts in Paris?
Who supplies P3s to jobs in Athens?
Does the collator project use bolts from Smith?
Where is Smith?
How much does part P4 weigh?

Figure 8.1 - Sample Queries to Supplier/Part/Job database

Although the entire database took approximately 3 hours to implement and test, the portion which would normally be done by a DBA in a commercialized system took 1.5 hours. Considering the complexity of the queries the system can answer, this time is minimal. This time could be further reduced by having a specialized editor to create the Database Schema. This would avoid the problem of matching parentheses in the current Database Schema data structure. The amount of time spent on each task during the implementation of this database is listed in figure 8.2.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Time (min.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement Database in Oracle</td>
<td>20</td>
</tr>
<tr>
<td>Load test data into database</td>
<td>20</td>
</tr>
<tr>
<td>Add extensions the Database Schema Library</td>
<td>30</td>
</tr>
<tr>
<td>Design and Implement Database Schema</td>
<td>60*</td>
</tr>
<tr>
<td>Add linguistic information to Global Dictionary</td>
<td>20</td>
</tr>
<tr>
<td>Test queries and make a few corrections</td>
<td>30*</td>
</tr>
</tbody>
</table>

* - Required activities for commercialized system.

Figure 8.2 - Implementation Time for Supplier/Part/Job Database

The only problem which was encountered during the implementation of this database was the selection of duplicate
items. Queries to the supplier/part/job database tended to produce many duplicate database values, which seemed to suggest that the DISTINCT attribute should act as a default. Since the root of this problem lies in the design and semantics of the given database, this problem was not dealt with.

The implementation of this domain has reinforced the initial expectations of the ease with which a new domain can be implemented. It also shows that the DBA does not need to have any knowledge of AI programming or have a strong linguistic background. The DBA is required to have an understanding of relational database design and a limited knowledge of the structure of English.

8.3 Database Portability

Database Portability involves adapting the system to a new relational DBMS. In order to achieve this, the Database Interface routine must be modified. Some changes to the Response Generator may be required, depending on the report generation capabilities of the given DBMS. No changes in either the syntactic or semantic processors are required.

Database portability is not as big an issue as domain portability since the system designer, and not the DBA, is expected to do this task. Due to the lack of another relational DBMS system on MTS, this capability has not been tested out. However, the concept of designing the Standard Query Representation (SQR), which contains all of the needed information to construct a query in a variety of query
languages, will in theory achieve a high degree of database portability.
IX. CONCLUSION

The main achievement of this system is its high degree of domain portability. Domain portability has been approached from the view of having a DBA implement a new domain, instead of the more traditional approach where an AI system designer would implement it. A new domain can be implemented using the skills of a typical DBA in a reasonable amount of time.

Considerable effort has also gone into the design of a friendly user interface. If a problem cannot be adequately resolved by the system, it is better to present the user with an understandable prompt than to make an imprecise assumption. The system handles some types of syntactic and semantic ambiguity, as well as certain unknown words and phrases in this manner.

The system can answer a wide range of queries which are commonly used in database queries. However, many areas of natural language discourse have been untouched or unresolved by this system. By no means are the algorithms and data structures suitable(applicable) for general discourse. However they are suitable for developing complex natural language interfaces to large databases.

9.1 Future Work

There are numerous areas which would require further work before this system could be used in a commercial setting. These topics are briefly presented in the following sections.
9.1.1 Further Explore Domain Portability

In order to fully test out the concept of domain portability, many more domains would have to be implemented. This would require that both the Database Schema Library and the Global Dictionary be expanded to incorporate these domains. It would also be necessary to evaluate the ease in which a DBA without NL experience could implement a new domain. Obviously proper documentation would have to be developed in order to test this out.

9.1.2 Adaptation To Other DBMS

In order to test out the soundness of the Database Interface concept for database portability, the system could be adapted to other relational DBMS using different query languages. It would also be interesting to look into the possibility of adapting the system to work with the network and hierarchical database models.

9.1.3 Extensions To The ATN

The ATN could be extended to answer a wider range of questions. For example, it does not presently parse queries containing relative clauses of the following form:

What jobs using red parts are located in London?

It could also be modified to determine all possible parses. Presently it will only produce the first valid parse found.
9.1.4 Pronoun Reference

Pronouns are very useful in NL systems since they can greatly reduce the number of words needed to state a query. It would be interesting to explore the use of verb frames and semantic definitions of pronouns in order to determine the correct pronoun referent. For example, in the following queries:

What suppliers are located in London?
Do any of them supply bolts?

the pronoun "them" will refer to the suppliers located in London. The system could determine this by noting that the subject of the verb "supply" is defined as a supplier, both "them" and "suppliers" can be defined as groups of individuals (CATEG GROUP), and the question element to the previous query was a group of suppliers.

9.1.5 Ellipses

The ability to process ellipses (sentence fragments) is important for any NL system which is to be used in a commercial setting. This allows users to enter similar queries without having to retype the entire query. For example, after the query:

What spreadsheet programs run on the Apple II?
the user may type:

    on the IBM PC?

In this case the system should substitute "IBM PC" for "APPLE II" to answer the query. This feature has been successfully implemented in PLANES (Waltz 1978). It appears that the verb frame concept would allow this feature to be added to the system fairly easily.

9.1.6 Quantifiers

    The system does not currently process any complicated type of quantifiers, including negation. This is an important area for NL database systems since database queries often contain complex quantification. This topic has previously been researched in numerous NL systems (Woods 1968; Waltz 1978; Booth 1983).

9.1.7 Complex Conjunction

    Conjunction is an extremely complex area in linguistics with many unresolved problems. The current system has successfully dealt with a limited form of conjunction by using the structure of the database to resolve ambiguity. It would be interesting to extend the approaches used to more complex conjunction.
9.1.8 Response Generation

The present system currently only forms responses (other than reports) for "yes/no" questions and for queries which cannot be answered due to problems such as ambiguities or unknown database elements. It would be useful to explore how queries can be answered in complete sentences. In some cases, the generation of a complete response can avoid a misinterpretation. For example, in the following query:

Does the Apple run LOGO?

the user may not be aware that there are more than one "Apple" (or LOGO) in the system. Hence an answer like:

Yes. The Apple II runs Terrapin LOGO.

would prevent any possible misinterpretation which may result from an incomplete response.

Another area in response generation which is not dealt with in this thesis is supplying the user with additional relevant information when the answer of a question may not be satisfactory. For example, the answer "NO" to the following query will not inform the user if Joe Smith has actually taken Math 100 (assume Joe Smith is a student and Math 100 is a course).

Did Joe Smith pass Math 100?
No. Joe Smith did not take Math 100.
No. Joe Smith failed Math 100.
9.2 Summary

A portable natural language front-end to a relational database has been described in this thesis. It consists of 4 components — the syntactic processor, the semantic processor, the database interface, and the response generator. All of these components are domain independent.

Domain portability is achieved by separating the domain dependent from the domain independent information. The domain dependent information is contained in the domain definition. It has been designed so that a new domain definition can be built using the skills of a typical DBA in a reasonable amount of time. A rich supply of standard definitions are available to both aid in the development process and to help force consistency amongst databases.

Database portability is achieved by placing all database dependent components into the database interface. The semantic processor is completely database independent. The system can be adapted to a new DBMS by simply modifying this routine.
BIBLIOGRAPHY


Furukawa, Koichi (1977), A Deductive Question Anwering System on Relational Data Bases, Proceedings of the Fifth Int. Jt. Conf. on Artificial Intelligence, MIT., pp.59-66.

Griffith, Robert L. (1982), Three Principles of Representation


Harris, Larry R. (1977b), Natural Language Database Query: Using the Database Itself as the Definition of World Knowledge and as an Extension of the Dictionary, Technical Report TR77-2, Mathematics Department, N.H.

Harris, Larry R. (1977c), User Oriented Data Base Query with the ROBOT Natural Language Query System, Proc. 3rd. Int. Conf. on Very Large Data Bases, Tokyo, Japan.


Johnson, Jan (1981), Intellect on Demand, Datamation, November, pp. 73-78.

Johnson, Jan (1984), Easy Does It: Natural Language Query Systems have gained new attention in recent months, Datamation, June 15, pp. 48-60.


Information Retrieval, (in Minsky, pp 33-134).


Reiter, Ray (1978), The Woods Augmented Transition Network Parser, Technical Note 78-3, Department of Computer Science, University of British Columbia


Sagalowicz, Daniel (1977), IDA : An Intelligent Data Access Program, Proc. 3rd. Int. Conf. on Very Large Data Bases, Tokyo, Japan.


This section contains diagrams of the ATN's used by the Syntactic Processor.

The NOUN PHRASE Network

NP

JUMP (Initialize)

NSTART

VIR NP

NP/HOLD

PRO QWORD

DET

NP/HOW

WRD "HOW"

NP/DET

MEM ("MANY", "MUCH")

ADV NUM N PUSH NAME CONJ

N PUSH NAME NUM

MEM ("MANY", "MUCH")

NP/N

PUSH NAME

POP

PUSH PP (Subj Only)
PUSH S/REL
The SENTENCE Network
The PREPOSITIONAL PHRASE Network

The NAME (Proper Noun) Network

The NOISE WORD Network
APPENDIX B - SAMPLE PARSE

The following is an example of a deep structure produced by the ATN. Note that the SEMREG and VFRAME registers have been set by the Semantic Processor. The Sequel query produced by this query is listed after the deep structure.

*WHAT SHACK COMPUTERS RUN GOOD OR EXCELLENT PROGRMS ON CHEMISTRY?

> (Assume (RADIO SHACK) for (SHACK) ?)

*OK

> (** Spelling Mistake *** PROGRMS --> PROGRAM)

> S

> MOOD INT ; Interogative

> QE SUBJ ; Question Element

> SUBJ

> NP

> DET WHAT

> DESC

> NPR

> NPR SHACK

> ANPR

> RADIO SHACK ; contains full version of

> HEAD

> N COMPUTER

> QUAL NIL

> QUES YES

> NUM PL

> PER 3

> CASE SUBJ

> POSN 3

> ANP

> WHAT SHACK COMPUTERS

> SEMREG ; set by semantic processor

> RELATION COMPUTER

> AUX NIL

> V RUN

> VFRAME

> SUBJ ; set by semantic processor

> RELATION COMPUTER

> DOBJ

> RELATION PROGRAM

> MOD NIL

> PRED NIL
DOBJ
NP
DET NIL
DESC
ADJ GOOD
CON OR
ADJ EXCELLENT
HEAD
N PROGRAM
QUAL
PP
PREP ON
POBJ
NP
DET NIL
DESC NIL
HEAD
N CHEMISTRY
QUAL NIL
QUES NO
NUM SG
PER 3
CASE PPOBJ
POSN 10
ANP
CHEMISTRY
SEMREG ;also set by Sem. Processor
FIELD DESCRIPTOR DSC
PPPOSN NIL
QUES NO
NUM SG
PER 3
CASE OBJ
POSN 8
ANP
GOOD OR EXCELLENT PROGRAM
SEMREG
RELATION PROGRAM
IOBJ NIL
PPOBJS NIL

; Resulting Sequel Query

SELECT COMPUTER.CNO , COMPUTER.MAKE , COMPUTER.MODEL FROM COMPUTER , PROGRAM , DESCRIPTOR , PROGDESC WHERE (COMPUTER.CNO = PROGRAM.CNO) AND ((PROGRAM.PNO = PROGDESC.PNO) AND (PROGDESC.DNO = DESCRIPTOR.DNO)) AND (COMPUTER.MAKE = 'RADIO SHACK') AND (PROGRAM.RATING > 3 OR PROGRAM.RATING = 5) AND (DESCRIPTOR.DSC = 'CHEMISTRY')
APPENDIX C - DATABASE SCHEMA FOR THE PROGRAM DATABASE

The Database Schema is defined by the DBA for each domain implemented in the system. It consists of 3 components -- the database definition, the verb definition, and the default join information.

Database Definition

(COMPUTER ; ** Computer relation **

((FIELDS (CNO MAKE MODEL RAM))
 (KEY (CNO))
 (IDENTIFIERS (CNO MAKE MODEL))
 (CATEG (PHYSICAL-OBJECT))
 (L-TYPE (PRODUCT))
 (L-NOUN (COMPUTER MICROCOMPUTER MICRO))
 (L-PREP (ON WITH))
 (L-DESC (MICRO PERSONAL))
 (COMMENT (a type/brand of a computer))
)

(CNO
 (TYPE (INTEGER))
 (CONTENTS (INTEGER))
 (L-TYPE (RNUMBER))
 (COMMENT (a unique identifier for each computer))
)

(MAKE
 (TYPE (CNAME))
 (CONTENTS (INVERT NPR ?))
 (CATEG (GROUP))
 (L-NOUN (MAKE TYPE BRAND))
 (COMMENT (the make/manufacturer of the computer))
)

(MODEL
 (TYPE (CHAR 20))
 (CONTENTS (INVERT NPR ?))
 (L-NOUN (MODEL VERSION))
 (COMMENT (the model of the computer))
)

(RAM
 (TYPE (INTEGER))
 (CONTENTS (INTEGER))
 (L-NOUN (MEMORY KILOBYTE RAM))
 (L-ABR ((KB KILOBYTE)))
 (COMMENT (the standard amount of memory on the computer in kilobytes))
)
(PROGRAM ; ** Program relation **
((FIELDS (PNO PUBLISHER NAME VERSION CNO RATING COST))
(KEY (PNO))
(IDENTIFIERS (PNO PUBLISHER NAME))
(CATEG (PHYSICAL-OBJECT))
(L-TYPE (PRODUCT))
(L-NOUN (PROGRAM PACKAGE COURSEWARE SOFTWARE))
(L-PREP (WITH BY))
(L-DESC (COMPUTER EDUCATION (EDUCATIONAL ADJ *)))))
(COMMENT (a computer program))
)
(PNO
(TYPE (INTEGER))
(CONTENTS (INTEGER))
(L-TYPE (RNUMBER ))
(L-DESC (LIBRARY REFERENCE))
(COMMENT (a unique identifier for each program))
)
(PUBLISHER
(TYPE (CNAME ))
(CONTENTS (INVERT NPR ?))
(CATEG (GROUP ))
(L-NOUN (PUBLISHER PRODUCER))
(COMMENT (the name of the publishing company of the program))
)
(NAME
(TYPE (GNAME ))
(CONTENTS (INVERT NPR ?))
(COMMENT (the name of the computer program))
)
(VERSION
(TYPE (REAL))
(CONTENTS (REAL))
(L-NOUN (EDITION VERSION))
(L-DESC (EDITION VERSION))
(COMMENT (the version of the computer program, if any))
)
(CNO
(TYPE (INTEGER))
(CONTENTS (KEY "COMPUTER.CNO")
(COMMENT (the computer number which the program runs on))
)
(RATING
(TYPE (RATING))
(CONTENTS (INTEGER (1 - 5)))
(COMMENT (the rating of a program))
)
)
(COST ; ** Program relation cont. **
 (TYPE (MONEY))
 (CONTENTS (MONEY))
 (L-TYPE (COST))
 (L-TYPE (RETAIL))
 (COMMENT (the original cost of the program))
 )

(DESCRIPTOR ; ** Descriptor relation **
 ((FIELDS (DNO DSC))
 (KEY (DNO))
 (IDENTIFIERS (DNO DSC))
 (CATEG (ABSTRACT-OBJECT))
 (L-NOUN (DESCRIPTOR))
 (L-PREP (ON FOR))
 (COMMENT (Descriptors which describe the program))
 )

(DNO
 (TYPE (INTEGER))
 (CONTENTS (INTEGER))
 (COMMENT (a unique identifier for each descriptor))
 )

(DSC
 (TYPE (GNAME))
 (CONTENTS (INVERT N ?))
 (L-NOUN (DESCRIPTOR))
 (L-ABR ((MATH MATHEMATICS) (ED EDUCATION)))
 (COMMENT (a set of words which describe computer programs))
 )

(PROGDESC ; ** Program-Descriptor relation (for join) **
 ((FIELDS (PNO DNO))
 (CATEG (JOIN))
 (COMMENT (Joins PROGRAM & DESCRIPTOR relations))
 )

(PNO
 (CONTENTS ((KEY PROGRAM PNO)))
 (COMMENT (a unique program identifier for each program))
 )

(DNO
 (CONTENTS ((KEY DESCRIPTOR DNO)))
 (COMMENT (a unique identifier for each descriptor))
 )))
Verb Definition

(RUN ; Note the 3 verb frames for this verb
    VFRAME (((SUBJ (RELATION COMPUTER))
            (DOBJ (RELATION PROGRAM)))
            ((SUBJ (RELATION PROGRAM))
            (MOD (FIELD PROGRAM RATING)))
            ((SUBJ (RELATION PROGRAM))
            (PP (*) (RELATION COMPUTER))))
    JOININFO ("COMPUTER.CNO = PROGRAM.CNO")
    JOINTYPE ((COMPUTER 1)(PROGRAM MANY))
)

(EXECUTE
    SYNONYM RUN
)

(TEACH
    VFRAME (((SUBJ (RELATION PROGRAM))
            (DOBJ (RELATION DESCRIPTOR)))
    JOINREL (PROGDESC)
    JOININFO ("PROGRAM.PNO = PROGDESC.PNO AND
            PROGDESC.DNO = DESCRIPTOR.DNO")
    JOINTYPE ((PROGRAM MANY)(DESCRIPTOR MANY))
)

(PRODUCE
    VFRAME (((SUBJ (FIELD COMPUTER MAKE))
            (DOBJ (RELATION COMPUTER)))
            ((SUBJ (FIELD PROGRAM PUBLISHER))
            (DOBJ (RELATION PROGRAM)))
    JOININFO NIL
    JOINTYPE NIL
)

(SELL
    VFRAME (((SUBJ (FIELD COMPUTER MAKE))
            (DOBJ (RELATION COMPUTER)))
            ((SUBJ (FIELD PROGRAM PUBLISHER))
            (DOBJ (RELATION PROGRAM)))
    JOININFO NIL
    JOINTYPE NIL
)
Default Joins and Association Factors

(DEFAULTJOIN
  'COMPUTER     'PROGRAM ;Relation Names
  '3            ;Association Factor
  '((COMPUTER 1) (PROGRAM MANY)) ;JOIN TYPE
  "((COMPUTER.CNO = PROGRAM.CNO)" ;Join
  NIL) ;Secondary Relation ; (if any)

(DEFAULTJOIN
  'DESCRIPTOR   'PROGRAM
  '2
  '((DESCRIPTOR MANY) (PROGRAM MANY))
  "((PROGRAM.PNO = PROGDESC.PNO) AND
   (PROGDESC.DNO = DESCRIPTOR.DNO))"
  '(PROGDESC))

(DEFAULTJOIN
  'DESCRIPTOR   'COMPUTER
  '4
  '((DESCRIPTOR MANY) (COMPUTER MANY))
  "((COMPUTER.CNO = PROGRAM.CNO) AND
   (PROGRAM.PNO = PROGDESC.PNO) AND
   (PROGDESC.DNO = DESCRIPTOR.DNO))"
  '(PROGRAM PROGDESC))
The Database Schema Library contains linguistic and semantic information that may be used in various domains. For instance, if a domain contains a relation which is in the category of physical objects, \((CATEG PHYSICAL-OBJECT)\), then the needed information is read into the Active Domain Dictionary (the dictionary in core for a particular domain)

```
; ; ****************** CATEG ************************
; 
; \((CATEG PHYSICAL-OBJECT)\)
; \((L-NOUN (OBJECT THING ANYTHING))\)
; \((L-DESC (PHYSICAL))\)
; 
; \((CATEG HUMAN)\)
; \((L-NOUN (SOMEONE ANYONE PERSON HUMAN))\)
; 
; ; ****************** TYPE **********************
; ;
; \((TYPE CNAME)\)
; \((L-NOUN (COMPANY ORGANIZATION))\)
; 
; \((TYPE COLOUR)\)
; \((L-NOUN (COLOUR SHADE))\)
; \((MEM-OF ((BLUE) (GREEN) (RED) (YELLOW)))\)
; 
; \((TYPE MONEY)\)
; \((L-NOUN (VALUE DOLLAR MONEY))\)
; 
; \((TYPE Q-MASS)\)
; \((L-NOUN (WEIGHT MASS))\)
; \((VERB WEIGH FEATURES (RELATION-VERB TRANS PASSIVE)\)
; VFRAME (((SUBJ (GENERAL *ANY*))
; (DOBJ (TYPE Q-MASS)))))
```
(((TYPE RATING)
  (L-NOUN (RATING QUALITY))
  (MEM-OF ((EXCELLENT CODE-AS (* = 5))
            (GOOD CODE-AS (* "">" 3))
            (AVERAGE CODE-AS (* = 3))
            (FAIR CODE-AS (* = 3))
            (POOR CODE-AS (* ""<" 3))
            (BAD CODE-AS (* ""<" 3))
            (TERRIBLE CODE-AS (* = 1)))
)

; ************** L-TYPE **************

; ((L-TYPE RNUMBER)
  (L-NOUN (NUMBER KEY IDENTIFIER))
  (L-DESC (REFERENCE INDEX))
)

; ((L-TYPE COST)
  (L-NOUN (VALUE PRICE)) ;ADD COST LATER

  (VERB COST
    FEATURES (RELATION-VERB TRANS)
    VFRAME (((SUBJ (GENERAL *ANY*))
             (DOBJ (L-TYPE COST))))
    (MEM-OF ((EXPENSIVE CODE-AS (* "">" 100))
             (CHEAP CODE-AS (* ""<" 50))
             (MEM-OF (((SUBJ (GENERAL *ANY*))
                        (DOBJ (L-TYPE COST)))))
             (MEM-OF (((SUBJ (GENERAL *ANY*))
                        (DOBJ (L-TYPE COST)))))
    )
)

; CROSS REFERENCE ***************

(((CATEG PHYSICAL-OBJECT) (L-TYPE COST))
 (VERB COST
  VFRAME (((SUBJ (CATEG PHYSICAL-OBJECT))
           (DOBJ (L-TYPE COST))))
    JOININFO NIL
    JOINTYPE NIL
  )
)

(((CATEG HUMAN) (L-TYPE SALARY))
 (VERB EARN
  VFRAME (((SUBJ (CATEG HUMAN))
             (DOBJ (L-TYPE SALARY)))))
)
)
APPENDIX E - GENERAL DICTIONARY

The General Dictionary contains words that are common to all database domains. All of these words will be loaded into the Active Domain Dictionary.

; *** ADJECTIVES ***
(UNIQUE ADJ * SEM ((UNIQUE)))
(DISTINCT ADJ * SEM ((UNIQUE)))

; *** ABBREVIATIONS ***
(A INT *) (B INT *)
(LTD ABR LIMITED DICT NIL)

; *** ADVERBS ***
(Please ADV *)
(QUICKLY ADV *)

; *** DETERMINERS ***
(THE DET *)

; *** QUANTIFIERS ***
(ONE QTF *
PRO (ONE (OBJ) (SUBJ) (NUMBER SG)) N S)
(ALL QTF *)

; *** CONJUNCTIONS ***
(AND CON *)
(OR CON *)

; *** VERBS ***
(BE V (BE (UNTENSED)))
FEATURES (REL-VERB COPULA TRANS INTRANS AUX)
VFRAME (((SUBJ (GENERAL *ANY*)) (PRED (GENERAL *ANY*)))
)
(ARE V (BE (TNS PRESENT) (PCODE X13SG)))
(IS V (BE (TNS PRESENT) (PCODE 3SG)))
(BEEN V (BE (PASTTART)))
(DO V ES FEATURES (TRANS AUX))
(DID V (DO (TNS PAST)))
.DOES V (DO (TNS PRESENT) (PNCODE 3SG)))
(DOING V (DO (PRESPART)))
(DONE V (DO (PASTPART)))

(PRINT V S-ED FEATURES (COM-VERB TRANS INDOBJ PASSIVE)
VFRAME (((SUBJ (GENERAL *YOU*)) (DOBJ (GENERAL *ANY*))
(IOBJ (GENERAL *ME*) OPT))))

(LIST V S-ED FEATURES (TRANS INDOBJ PASSIVE)
SYNONYM PRINT)

; *** NOUNS ***
(INFORMATION N MASS SEM ((GENERAL *ANY*)))
(DESRIPTION N S)

; *** QUESTION NOUNS ***
(WHO QWORD (WHO (NUMBER SG))
SEM ((CATEG GROUP) (CATEG HUMAN)))
(WHERE QWORD *
SEM ((TYPE LOCATION)))

; *** QUESTION DETERMINERS ***
(WHAT QDET *)
(WHO QDET *)

; *** PREPOSITIONS ***
(AT PREP *)
(ON PREP *)

; *** RELATIVE PRONOUNS ***
(WHICH RELPRO *)
(THAT RELPRO *)

; *** PRONOUNS ***
; DEFAULTS : PERSON 3
(YOU PRO (YOU (SUBJ) (OBJ) (NUMBER SG-PL) (PERSON 2)))
(ME PRO (I (OBJ) (NUMBER SG)))
(I PRO (I (NUMBER SG) (PERSON 1)))

; *** IGNORE WORDS ***
; FOR NOISE ATN
(TELL GRB *)
(INFORM GRB *)
(WHETHER GRB *)
APPENDIX F - GLOBAL DICTIONARY

The Global Dictionary contains words which may be used in any domain or database (i.e. a complete dictionary). Only part of this information will be in main memory at one time. However, the MORPH routine will search through it when an unknown word is found. Information from this file is used during the compile stage by the following sources of information:

- Database Schema
- Database Schema Library
- Inverted Index

<table>
<thead>
<tr>
<th>Word</th>
<th>Part of Speech</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLUE</td>
<td>ADJ</td>
<td>*</td>
</tr>
<tr>
<td>COLOUR</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>COMPANY</td>
<td>N</td>
<td>IES</td>
</tr>
<tr>
<td>COMPUTER</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>COST</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>V</td>
<td>S-ED</td>
<td></td>
</tr>
<tr>
<td>FEATURES</td>
<td>(REL-VERB TRANS PASSIVE)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDUCATION</td>
<td>N</td>
<td>MASS</td>
</tr>
<tr>
<td>EDUCATIONAL</td>
<td>ADJ</td>
<td>*</td>
</tr>
<tr>
<td>GOOD</td>
<td>ADJ</td>
<td>*</td>
</tr>
<tr>
<td>ADV</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>PROGRAM</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>RUN</td>
<td>V</td>
<td>S-IRR</td>
</tr>
<tr>
<td>FEATURES</td>
<td>(REL-VERB TRANS PASSIVE INTRANS)</td>
<td></td>
</tr>
<tr>
<td>IRR</td>
<td>((RAN V (RUN (TNS PAST))))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(RUN V (RUN (PASTPART)))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(RUNS V (RUN (PCODE 3SG))))</td>
<td></td>
</tr>
<tr>
<td>SELL</td>
<td>V</td>
<td>S-IRR</td>
</tr>
<tr>
<td>FEATURES</td>
<td>(REL-VERB TRANS INDOBJ PASSIVE)</td>
<td></td>
</tr>
<tr>
<td>IRR</td>
<td>((SOLD V (SELL (TNS PAST))))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>((SELLS V (SELL (TNS PRESENT) (PCODE 3SG)))))</td>
<td></td>
</tr>
<tr>
<td>SOFTWARE</td>
<td>N</td>
<td>MASS</td>
</tr>
<tr>
<td>STEVE</td>
<td>NPR</td>
<td>*</td>
</tr>
<tr>
<td>SEM</td>
<td>((FNAME)))</td>
<td></td>
</tr>
</tbody>
</table>
(WEIGHT N MASS)

(WEIGH V S-ED FEATURES (REL-VERB TRANS PASSIVE))

(RUN V S-IRR FEATURES (REL-VERB TRANS PASSIVE INTRANS)
IRR ((RAN V (RUN (TNS PAST)))
     (RUN V (RUN (PASTPART)))
     (RUNS V (RUN (PNCODE 3SG))))
APPENDIX G - PARTIAL ACTIVE DOMAIN DICTIONARY FOR THE PROGRAM
DATABASE

Listed below are some sample definitions of words which are used in the Program database. Note that these words are compiled from the General Dictionary, the Database Schema, the Database Schema Library, the Inverted Index, and the Global Dictionary.

*A
> DET * ;Determiner
> INT * ;Initial
> -------------
*AND
> NPR ? ;part of a compound NPR
> PART-OF ((JOHN WILSON AND SONS LIMITED)) ;Conjunction
> CON *
> -------------
*CATEG ;Cross references for Categories
> SEM ((PHYSICAL-OBJECT ((RELATION COMPUTER)
> (RELATION PROGRAM)))
> (GROUP ((FIELD COMPUTER MAKE)
> (FIELD PROGRAM PUBLISHER)))
> (ABSTRACT-OBJECT ((RELATION DESCRIPTOR)))
> (JOIN ((RELATION PROGDESC)))
> -------------
*COMPUTER ;note the 3 semantic definitions
> N S
> L-DESC ((RELATION PROGRAM))
> NPR ?
> PART-OF ((COLOR COMPUTER))
> L- NOUN ((RELATION COMPUTER))
> -------------
*DOES
> V (DO (TNS PRESENT) (PCODE 3SG))
> -------------
*EXPENSIVE ;Coded element of the field
> ADJ * ; PROGRAM COST
> MEM-OF ((L-TYPE COST))
> CODE-AS (* > 100)
> -------------
*GOOD ;Coded element of the field
> ADV * ; PROGRAM RATING
> ADJ *
> MEM-OF ((TYPE RATING))
> CODE-AS (* > 3)
> -------------
;Database Element

*LOGO
> NPR
> ELM-OF ((FIELD PROGRAM NAME))

*ME
> PRO (I (OBJ) (NUMBER SG))

*NUMBER
> L-NOUN ((L-TYPE RNUMBER) (TYPE REAL) (TYPE INTEGER))
> FEATURE (TRANS PASSIVE)
> V S-ED
> N S

*PLEASE
> ADV *

*PRINT
> VFRAME (((SUBJ (GENERAL *YOU*))
> (DOBJ (GENERAL *ANY*))
> (IOBJ (GENERAL *ME* OPT)))
> FEATURES (COM-VERB TRANS INDOBJ PASSIVE)
> V S-ED

*PROGRAM
> N S
> L-NOUN ((RELATION PROGRAM))

*RUN
> FEATURES (REL-VERB TRANS PASSIVE INTRANS)
> V (RUN (PASTPART))
> JOINTYPE ((COMPUTER 1) (PROGRAM MANY))
> JOININFO (COMPUTER.CNO = PROGRAM.CNO)
> VFRAME (((SUBJ (RELATION COMPUTER))
> (DOBJ (RELATION PROGRAM)))
> ((SUBJ (RELATION PROGRAM))
> (MOD (FIELD PROGRAM RATING)))
> ((SUBJ (RELATION PROGRAM))
> (PP (*) (RELATION COMPUTER)))
> VFRAME (**(SUBJ (RELATION COMPUTER))
> (DOBJ (RELATION PROGRAM)))
> ((SUBJ (RELATION PROGRAM))
> (MOD (FIELD PROGRAM RATING)))
> ((SUBJ (RELATION PROGRAM))
> (PP (*) (RELATION COMPUTER)))

*WHAT
> QDET *
> SEM ((CATEG PHYSICAL-OBJECT))
> QWORD (WHAT (NUMBER SG))

*WILSON
> NPR
> PART-OF ((JOHN WILSON AND SONS LIMITED))

*WITH
> L-PREP ((RELATION PROGRAM) (RELATION COMPUTER))
> PREP *
This is a sample session of the system using the Program database. The queries demonstrate some of the features of the system. Lines which start with an asterisk (*) represent the user's input, while lines which start with a less-than sign (<) represent output generated by the system. Comments added in the session are identified by a semi-colon (;). The data in the Program database has been made up in order to demonstrate the ability of the system and is by no means accurate. For some queries, the data retrieved by the system has been deleted to save space.

> Which database would you like to use?
  1. The PROGRAM database
  2. The SUPPLIER/PART/JOB database
> Enter a number between 1 and 2 and press Return
*1
> Welcome to the PROGRAM database.
> To stop type 'STOP!' or 'QUIT'.
>
> Please enter your question.
*<Q1> PRINT THE COLLOID CHEMISTRY PROGRAMS!
>
> Unknown word - COLLOID
> What would you like to do?
>
  1. Spelling Error! Enter the correct spelling.
  2. Abbreviation! Enter the full word.
  3. Synonym! Enter the replacement word.
  4. Unknown data element! Continue processing.
  5. Ignore the word! Continue processing.
  6. Cancel this query.
>
> Enter a number between 1 and 6 and press Return
*5
> SELECT PROGRAM.PNO , PROGRAM.PUBLISHER , PROGRAM.NAME
FROM PROGRAM , DESCRIPTOR , PROGDESC
WHERE
  ((PROGRAM.PNO = PROGDESC.PNO) AND
   (PROGDESC.DNO = DESCRIPTOR.DNO))
AND (DESCRIPTOR.DSC = 'CHEMISTRY')
<table>
<thead>
<tr>
<th>PNO</th>
<th>PUBLISHER</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>JOHN BROWNS SOFTWARE LIMITED</td>
<td>THE ATOM</td>
</tr>
<tr>
<td>13</td>
<td>EDUCOACH</td>
<td>ATOMIC PARTICLES</td>
</tr>
</tbody>
</table>

Please enter your question.

**Q2** DOES LOTUS RUN ON THE IBM PC?

Unknown word - LOTUS

What would you like to do?

1. Spelling Error! Enter the correct spelling.
2. Abbreviation! Enter the full word.
3. Synonym! Enter the replacement word.
4. Unknown data element! Continue processing.
5. Ignore the word! Continue processing.
6. Cancel this query.

Enter a number between 1 and 6 and press Return

**Q2** Which category does LOTUS belong to?

1. a Common Noun (car, computer, pen)
2. a Proper Noun (Joe, Smith, Canada)
3. an Adjective (purple, good, big)

Enter a number between 1 and 3 and press Return

**Q2** NOTE - (LOTUS) is not (a computer program) contained in this database!
The above query cannot be answered.

Please enter your question.

**Q3** WHAT COMPUTERS RUN LOTUS?

**Q3** NOTE - (LOTUS) is not (a computer program) contained in this database!
The above query cannot be answered.

; Note that LOTUS is now defined in the system.

Please enter your question.

**Q4** WHAT CHEMISTRY PROGRAMS ARE RUN BY THE APPLE II?

```
SELECT PROGRAM.PNO, PROGRAM.PUBLISHER, PROGRAM.NAME
FROM COMPUTER, PROGRAM, DESCRIPTOR, PROGDESC
WHERE
    (COMPUTER.CNO = PROGRAM.CNO)
    AND
    ((PROGRAM.PNO = PROGDESC.PNO) AND
    (PROGDESC.DNO = DESCRIPTOR.DNO))
    AND
    (COMPUTER.MODEL = 'II')
    AND
    (COMPUTER.MAKE = 'APPLE')
```
Please enter your question.

*<Q5> WHAT APPLE COMPUTERS RUN PROGRAMS WHICH TEACH CHEMISTRY?

```sql
SELECT COMPUTER.CNO, COMPUTER.MAKE, COMPUTER.MODEL
FROM COMPUTER, PROGRAM
WHERE (COMPUTER.CNO = PROGRAM.CNO)
AND (COMPUTER.MAKE = 'APPLE')
AND (PROGRAM.PNO IN
(SELECT PROGRAM.PNO
FROM PROGRAM, DESCRIPTOR, PROGDESC
WHERE (PROGRAM.PNO = PROGDESC.PNO AND
PROGDESC.DNO = DESCRIPTOR.DNO)
AND (DESCRIPTOR.DSC = 'CHEMISTRY')))```

<table>
<thead>
<tr>
<th>CNO</th>
<th>MAKE</th>
<th>MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>APPLE</td>
<td>II</td>
</tr>
</tbody>
</table>

Please enter your question.

*<Q6> WHAT COMPUTERS RUN LOGO WITH 64 KB?

```sql
(*) Abbreviation/Synonym *** KB --> KILOBYTE)
SELECT COMPUTER.CNO, COMPUTER.MAKE, COMPUTER.MODEL
FROM COMPUTER, PROGRAM
WHERE (COMPUTER.CNO = PROGRAM.CNO)
AND (COMPUTER.RAM = 64)
AND (PROGRAM.NAME = 'LOGO')```

<table>
<thead>
<tr>
<th>CNO</th>
<th>MAKE</th>
<th>MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>APPLE</td>
<td>II</td>
</tr>
</tbody>
</table>

*<Q7> WHAT COMPUTERS WITH 64 KB RUN LOGO?

```sql
(*) Abbreviation/Synonym *** KB --> KILOBYTE)
SELECT COMPUTER.CNO, COMPUTER.MAKE, COMPUTER.MODEL
FROM COMPUTER, PROGRAM
WHERE (COMPUTER.CNO = PROGRAM.CNO)
AND (COMPUTER.RAM = 64)
AND (PROGRAM.NAME = 'LOGO')```

<table>
<thead>
<tr>
<th>CNO</th>
<th>MAKE</th>
<th>MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Please enter your question.

*Q8> SHOW ME THE GOOD CHEMISTRY PROGRAMS!

SELECT PROGRAM.PNO, PROGRAM.PUBLISHER, PROGRAM.NAME
FROM PROGRAM, DESCRIPTOR, PROGDESC
WHERE
  ((PROGRAM.PNO = PROGDESC.PNO) AND
   (PROGDESC.DNO = DESCRIPTOR.DNO))
  AND (PROGRAM.RATING > 3)
  AND (DESCRIPTOR.DSC = 'CHEMISTRY')

no records selected
; This means that no records in the database
; satisfy the above condition.

Please enter your question.

*Q9> HOW MUCH MEMORY DOES THE APPLE HAVE?

Is it OK to assume that (THE APPLE) refers to (the
make of the computer)?

OK
; Note that APPLE is also a program publisher

SELECT COMPUTER.RAM
FROM COMPUTER
WHERE
  (COMPUTER.MAKE = 'APPLE')

RAM
-------------
  64
  128

Please enter your question.

*Q10> WHAT PROGRAMS RUN ON THE IBM PC?

SELECT PROGRAM.PNO, PROGRAM.PUBLISHER, PROGRAM.NAME
FROM PROGRAM, COMPUTER
WHERE
  (COMPUTER.CNO = PROGRAM.CNO)
  AND (COMPUTER.MODEL = 'PC')
  AND (COMPUTER.MAKE = 'IBM')

<table>
<thead>
<tr>
<th>PNO</th>
<th>PUBLISHER</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>VISICORP</td>
<td>VISICALC</td>
</tr>
<tr>
<td>6</td>
<td>MICROSOFT</td>
<td>WORD</td>
</tr>
</tbody>
</table>

Please enter your question.

*Q11> WHAT PROGRAMS RUN ON THE IBM PC FROM MICROSOFT?

; Note that "MICROSOFT" modifies "PROGRAMS"
SELECT PROGRAM.PNO, PROGRAM.PUBLISHER, PROGRAM.NAME FROM PROGRAM, COMPUTER WHERE (COMPUTER.CNO = PROGRAM.CNO) AND (PROGRAM.PUBLISHER = 'MICROSOFT') AND (COMPUTER.MODEL = 'PC') AND (COMPUTER.MAKE = 'IBM')

<table>
<thead>
<tr>
<th>PNO</th>
<th>PUBLISHER</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>MICROSOFT</td>
<td>WORD</td>
</tr>
</tbody>
</table>

Please enter your question.

*<Q12> WHAT COMPUTERS RUN GOOD PROGRAMS ON CHEMISTRY?

SELECT COMPUTER.CNO, COMPUTER.MAKE, COMPUTER.MODEL FROM COMPUTER, PROGRAM, DESCRIPTOR, PROGDESC WHERE (COMPUTER.CNO = PROGRAM.CNO) AND ((PROGRAM.PNO = PROGDESC.PNO) AND (PROGDESC.DNO = DESCRIPTOR.DNO)) AND (PROGRAM.RATING > 3) AND (DESCRIPTOR.DSC = 'CHEMISTRY')

no records selected

Please enter your question.

*<Q13> PRINT THE PROGRAMS QUICKLY!

Is it OK to ignore the word/phrase (QUICKLY)?

*OK

SELECT PROGRAM.PNO, PROGRAM.PUBLISHER, PROGRAM.NAME FROM PROGRAM

<table>
<thead>
<tr>
<th>PNO</th>
<th>PUBLISHER</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VISICORP</td>
<td>VISICALC</td>
</tr>
<tr>
<td>2</td>
<td>VISICORP</td>
<td>VISICALC</td>
</tr>
<tr>
<td>3</td>
<td>APPLE</td>
<td>APPLEWORKS</td>
</tr>
<tr>
<td>4</td>
<td>MICROSOFT</td>
<td>TYPING TUTOR</td>
</tr>
<tr>
<td>5</td>
<td>MICROSOFT</td>
<td>WORD</td>
</tr>
<tr>
<td>6</td>
<td>MICROSOFT</td>
<td>WORD</td>
</tr>
<tr>
<td>7</td>
<td>TERRAPIN</td>
<td>LOGO</td>
</tr>
<tr>
<td>8</td>
<td>MILLIKEN PUBLISHING COMPANY</td>
<td>ADDITION SEQUENCES</td>
</tr>
<tr>
<td>9</td>
<td>EDUCOACH</td>
<td>CALCULUS TUTOR</td>
</tr>
<tr>
<td>10</td>
<td>JOHN WILSON AND SONS LIMITED</td>
<td>GEOMETRY TUTOR</td>
</tr>
<tr>
<td>11</td>
<td>JOHN BROWNS SOFTWARE LIMITED</td>
<td>THE ATOM</td>
</tr>
<tr>
<td>12</td>
<td>JOHN W SOFTWARE OF CANADA LIMITED</td>
<td>CALCULUS COACH</td>
</tr>
<tr>
<td>13</td>
<td>EDUCOACH</td>
<td>ATOMIC PARTICLES</td>
</tr>
<tr>
<td>14</td>
<td>EDUCOACH</td>
<td>FRENCH TEACHER</td>
</tr>
</tbody>
</table>

14 records selected.
> Please enter your question.
*<Q14> PRINT THE EFFICIENT IBM SPREADSHEET PROGRAMS!
>
> Unknown word - EFFICIENT
> What would you like to do?
>
> 1. Spelling Error! Enter the correct spelling.
> 2. Abbreviation! Enter the full word.
> 3. Synonym! Enter the replacement word.
> 4. Unknown data element! Continue processing.
> 5. Ignore the word! Continue processing.
> 6. Cancel this query.
>
> Enter a number between 1 and 6 and press Return
*5
> SELECT PROGRAM.PNO , PROGRAM.PUBLISHER , PROGRAM.NAME
> FROM PROGRAM , COMPUTER , DESCRIPTOR , PROGDESC
> WHERE
> (COMPUTER.CNO = PROGRAM.CNO)
> AND ((PROGRAM.PNO = PROGDESC.PNO) AND
> (PROGDESC.DNO = DESCRIPTOR.DNO))
> AND (COMPUTER.MAKE = 'IBM')
> AND (DESCRIPTOR.DSC = 'SPREADSHEET')
>
> | PNO | PUBLISHER  | NAME     |
> |-----|------------|----------|
> | 2   | VISICORP   | VISICALC |
> |     |            |          |
>
> Please enter your question.
*<Q15> PRINT THE GOOD AND EXCELLENT PROGRAMS!
> SELECT PROGRAM.PNO , PROGRAM.PUBLISHER , PROGRAM.NAME
> FROM PROGRAM
> WHERE
> (PROGRAM.RATING > 3 OR PROGRAM.RATING = 5)
>
> ; Note that the definition of "good" and
> ; "excellent" overlap. This is caused by the
> ; way they are defined in the Database Schema.
>
> | PNO | PUBLISHER            | NAME                    |
> |-----|----------------------|-------------------------|
> | 1   | VISICORP             | VISICALC                |
> | 2   | VISICORP             | VISICALC                |
> | 3   | APPLE                | APPLEWORKS              |
> | 5   | MICROSOFT            | WORD                    |
> | 6   | MICROSOFT            | WORD                    |
> | 7   | TERRAPIN             | LOGO                    |
> | 8   | MILLIKEN PUBLISHING COMPANY | ADDITION SEQUENCES |
> | 9   | EDUCOACH             | CALCULUS TUTOR          |
> | 8   | records selected.    |                         |
>
> Please enter your question.
*<Q16> WHAT APPLE OR IBM COMPUTERS RUN LOGO?
> SELECT COMPUTER.CNO, COMPUTER.MAKE, COMPUTER.MODEL
> FROM COMPUTER, PROGRAM
> WHERE
>     (COMPUTER.CNO = PROGRAM.CNO)
>     AND (COMPUTER.MAKE = 'APPLE' OR COMPUTER.MAKE = 'IBM')
>     AND (PROGRAM.NAME = 'LOGO')
> CNO MAKE MODEL
>     1 APPLE II
>
> Please enter your question.
*<Q17> WHAT IBM AND APPLE COMPUTERS RUN LOGO?
> ; Note that the system interprets the "AND" as
> ; an "OR". This is because one computer can only
> ; have one make associated with it (one-to-one).
> SELECT COMPUTER.CNO, COMPUTER.MAKE, COMPUTER.MODEL
> FROM COMPUTER, PROGRAM
> WHERE
>     (COMPUTER.CNO = PROGRAM.CNO)
>     AND (COMPUTER.MAKE = 'IBM' OR COMPUTER.MAKE = 'APPLE')
>     AND (PROGRAM.NAME = 'LOGO')
> CNO MAKE MODEL
>     1 APPLE II
>
> Please enter your question.
*<Q18> WHAT FRENCH AND CHEMISTRY PROGRAMS RUN ON THE APPLE?
> ERROR — This type of conjunction is not implemented!
> (see Multiple Query)
> Error occurred in:
> (WHAT FRENCH AND CHEMISTRY PROGRAMS)
> The above query cannot be answered.
> ; This is caused by the "MANY-MANY" relationship
> ; between PROGRAMS and DESCRIPTORS.
> ; (see section 6.2.3.2.3)
>
> Please enter your question.
*<Q19> WHAT PHYSICS, MATH OR BUSINESS PROGRAMS RUN ON THE
> APPLES II?
> (** Abbreviation/Synonym ** MATH --> MATHEMATICS)
> SELECT PROGRAM.PNO, PROGRAM.PUBLISHER, PROGRAM.NAME
> FROM PROGRAM, DESCRIPTOR, PROGDESC, COMPUTER
> WHERE
>     (COMPUTER.CNO = PROGRAM.CNO)
>     AND
((PROGRAM.PNO = PROGDESC.PNO) AND
 (PROGDESC.DNO = DESCRIPTOR.DNO))
AND (DESCRIPTOR.DSC = 'PHYSICS' OR DESCRIPTOR.DSC =
 'MATHEMATIC' OR DESCRIPTOR.DSC = 'BUSINESS')
AND (COMPUTER.MODEL = 'II')
AND (COMPUTER.MAKE = 'APPLE')

<table>
<thead>
<tr>
<th>PNO</th>
<th>PUBLISHER</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>TERRAPIN</td>
<td>LOGO</td>
</tr>
<tr>
<td>8</td>
<td>MILLIKEN PUBLISHING COMPANY</td>
<td>ADDITION SEQUENCES</td>
</tr>
<tr>
<td>9</td>
<td>EDUCOACH</td>
<td>CALCULUS TUTOR</td>
</tr>
<tr>
<td>12</td>
<td>JOHN W SOFTWARE OF CANADA LIMITED</td>
<td>CALCULUS COACH</td>
</tr>
<tr>
<td>13</td>
<td>EDUCOACH</td>
<td>ATOMIC PARTICLES</td>
</tr>
</tbody>
</table>

Please enter your question.
*<Q20> WHO MAKES LOGO?
* Is it OK to assume that (WHO) refers to (the name of the publishing company of the program)?
*OK

SELECT PROGRAM.PUBLISHER
FROM PROGRAM
WHERE
 (PROGRAM.NAME = 'LOGO')

PUBLISHER
---------------------
TERRAPIN

Please enter your question.
*<Q21> WHO MAKES THE PC?
* Is it OK to assume that (WHO) refers to (the make of the computer)?
*OK

SELECT COMPUTER.MAKE
FROM COMPUTER
WHERE
 (COMPUTER.MODEL = 'PC')

MAKE
--------
IBM

Please enter your question.
*<Q22> HOW MUCH IS LOGO FOR THE APPLE II?

SELECT PROGRAM.COST
FROM PROGRAM, COMPUTER
WHERE
 (COMPUTER.CNO = PROGRAM.CNO)
AND (PROGRAM.NAME = 'LOGO')
AND (COMPUTER.MODEL = 'II')
AND (COMPUTER.MAKE = 'APPLE')

COST
149

Please enter your question.

HOW MUCH DOES LOGO COST?
SELECT PROGRAM.COST
FROM PROGRAM
WHERE
(PROGRAM.NAME = 'LOGO')

149

Please enter your question.

IS LOGO A GOOD PROGRAM?
SELECT PROGRAM.NAME
FROM PROGRAM
WHERE
(PROGRAM.NAME = 'LOGO')
AND (PROGRAM.RATING > 3)

YES. There is 1 record which satisfies the given criteria.
Would you like to see it?

Tell me if LOGO is a good program!
SELECT PROGRAM.NAME
FROM PROGRAM
WHERE
(PROGRAM.NAME = 'LOGO')
AND (PROGRAM.RATING > 3)

YES. There is 1 record which satisfies the given criteria.
Would you like to see it?

IS LOGO good?
SELECT PROGRAM.NAME
FROM PROGRAM
WHERE
(PROGRAM.NAME = 'LOGO')
AND (PROGRAM.RATING > 3)

Yes. There is 1 record which satisfies the given criteria.
Would you like to see it?

Please enter your question.

*Q27* How much memory does the PC have?
SELECT COMPUTER.RAM
FROM COMPUTER
WHERE (COMPUTER.MODEL = 'PC')

RAM
-------
256

Please enter your question.

*Q28* What programs do we have for the PC?
SELECT PROGRAM.PNO, PROGRAM.PUBLISHER, PROGRAM.NAME
FROM PROGRAM, COMPUTER
WHERE (COMPUTER.CNO = PROGRAM.CNO)
AND (COMPUTER.MODEL = 'PC')

PNO PUBLISHER NAME
------- -------------
2 VISICORP VISICALC
6 MICROSOFT WORD

Please enter your question.

*Q29* What programs have good ratings?
SELECT PROGRAM.PNO, PROGRAM.PUBLISHER, PROGRAM.NAME
FROM PROGRAM
WHERE (PROGRAM.RATING > 3)

PNO PUBLISHER NAME
------- -------------
1 VISICORP VISICALC
2 VISICORP VISICALC
3 APPLE APPLEWORKS
5 MICROSOFT WORD
6 MICROSOFT WORD
7 TERRAPIN LOGO
8 MILLIKEN PUBLISHING COMPANY ADDITION SEQUENCES
9 EDU COACH CALCULUS TUTOR

8 records selected.
Please enter your question.

*<Q30> WHAT TEACHES FRENCH?

SELECT PROGRAM.PNO, PROGRAM.PUBLISHER, PROGRAM.NAME
FROM PROGRAM, DESCRIPTOR, PROGDESC
WHERE
((PROGRAM.PNO = PROGDESC.PNO) AND
(PROGDESC.DNO = DESCRIPTOR.DNO))
AND (DESCRIPTOR.DSC = 'FRENCH')

<table>
<thead>
<tr>
<th>PNO</th>
<th>PUBLISHER</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Educoach</td>
<td>FRENCH TEACHER</td>
</tr>
</tbody>
</table>

Please enter your question.

*<Q31> WHO TEACHES FRENCH?

Unable to match the query to a Verb Frame!

Your query cannot be answered.

; This query cannot be answered since "WHO" will
; not match the "RELATION PROGRAM" (it is defined
; as a PHYSICAL OBJECT and not a HUMAN or GROUP).
; This query could be answered by enhancing the
; verb matching routine to match verb frames to
; phrases which are not semantically correct.
; In this case the system could form a prompt like:

; Is it OK to assume that "WHO" refers to
; "a computer program"?

Please enter your question.

*<Q32> COUNT THE MICROSOFT PROGRAMS!

SELECT COUNT (PROGRAM.PNO)
FROM PROGRAM
WHERE
   (PROGRAM.PUBLISHER = 'MICROSOFT')

------------------
  3

Please enter your question.

*<Q33> HOW MANY MICROSOFT PROGRAMS RUN ON APPLE COMPUTERS?

SELECT COUNT (PROGRAM.PNO)
FROM PROGRAM, COMPUTER
WHERE
   (COMPUTER.CNO = PROGRAM.CNO)
AND (PROGRAM.PUBLISHER = 'MICROSOFT')
AND (COMPUTER.MAKE = 'APPLE')

------------------
  3
Please enter your question.

*<Q34> PRINT THE J WILSON AND SONS AND J W SOFTWARE OF CANADA LTD COMPUTER PROGRAMS!
* (*** Abbreviation/Synonym *** LTD --> LIMITED)
* Assume (JOHN WILSON AND SONS LIMITED) for (J WILSON AND SONS)?
*OK

<table>
<thead>
<tr>
<th>PNO</th>
<th>PUBLISHER</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>JOHN WILSON AND SONS LIMITED</td>
<td>GEOMETRY TUTOR</td>
</tr>
<tr>
<td>12</td>
<td>JOHN W SOFTWARE OF CANADA LIMITED</td>
<td>CALCULUS COACH</td>
</tr>
</tbody>
</table>

Please enter your question.

*<Q35> WHAT APPLE COMPUTERS RUN APPLE COMPUTER PRORAMS?

; Note that both "APPLE" and "COMPUTER" are used twice in this query. Both words have two different semantic meanings here.

(*** Spelling Mistake *** PRORAMS --> PROGRAM)

<table>
<thead>
<tr>
<th>CNO</th>
<th>MAKE</th>
<th>MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>APPLE</td>
<td>II</td>
</tr>
</tbody>
</table>

Please enter your question.

*<Q36> COULD YOU PLEASE PRINT THE UNIQUE PROGRAM PUBLISHERS!

<table>
<thead>
<tr>
<th>PUBLISHER</th>
</tr>
</thead>
<tbody>
<tr>
<td>VISICORP</td>
</tr>
<tr>
<td>APPLE</td>
</tr>
<tr>
<td>MICROSOFT</td>
</tr>
<tr>
<td>TERRAPIN</td>
</tr>
</tbody>
</table>
Please enter your question.

*<Q37> WHAT SHACK OR IBM COMPUTER PROGRAMS TEACH MATH?

Assume (RADIO SHACK) for (SHACK)?

*** Abbreviation/Synonym *** MATH --> MATHEMATICS

SELECT PROGRAM.PNO, PROGRAM.PUBLISHER, PROGRAM.NAME
FROM PROGRAM, COMPUTER, DESCRIPTOR, PROGDESC
WHERE
(PROGRAM.PNO = PROGDESC.PNO AND
PROGDESC.DNO = DESCRIPTOR.DNO)
AND (COMPUTER.CNO = PROGRAM.CNO)
AND (COMPUTER.MAKE = 'RADIO SHACK' OR
COMPUTER.MAKE = 'IBM')
AND (DESCRIPTOR.DSC = 'MATHEMATIC')

<table>
<thead>
<tr>
<th>PNO</th>
<th>PUBLISHER</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>JOHN WILSON AND SONS LIMITED</td>
<td>GEOMETRY TUTOR</td>
</tr>
</tbody>
</table>

Please enter your question.

*<Q38> PRINT THE COMPUTERS WHICH RUN SOFTWARE FROM MICROSOFT WITH 128 KB!

*** Abbreviation/Synonym *** KB --> KILOBYTE

SELECT COMPUTER.CNO, COMPUTER.MAKE, COMPUTER.MODEL
FROM COMPUTER
WHERE
(COMPUTER.RAM = 128)
AND (COMPUTER.CNO IN
(SELECT COMPUTER.CNO
FROM COMPUTER, PROGRAM
WHERE
(COMPUTER.CNO = PROGRAM.CNO)
AND (PROGRAM.PUBLISHER = 'MICROSOFT')))

<table>
<thead>
<tr>
<th>CNO</th>
<th>MAKE</th>
<th>MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>APPLE</td>
<td>MACINTOSH</td>
</tr>
</tbody>
</table>

Please enter your question.

*<Q39> WHAT PROGRAMS DOES JOHN PRODUCE?

Assume (JOHN W SOFTWARE OF CANADA LIMITED) for (JOHN)?

Assume (JOHN BROWNS SOFTWARE LIMITED) for (JOHN)?

*OK
> SELECT  PROGRAM.PNO , PROGRAM.PUBLISHER , PROGRAM.NAME
> FROM    PROGRAM
> WHERE
>  (PROGRAM.PUBLISHER  =
>        'JOHN BROWNS SOFTWARE LIMITED')
>
> PNO PUBLISHER  NAME
> ----- --------------------- ---------------------
> 11  JOHN BROWNS SOFTWARE LIMITED  THE ATOM
>
> Please enter your question.
> *<Q40>  WHAT COMPANIES PRODUCE EXCELLENT SOFTWARE?
> SELECT  PROGRAM.PUBLISHER
> FROM    PROGRAM
> WHERE
>  (PROGRAM.RATING = 5)
>
> PUBLISHER
> ------------------------
> APPLE
> TERRAPIN

> Please enter your question.
> *<Q41>  DOES JOHN LIMITED SELL GOOD SOFTWARE?
> Assume (JOHN W SOFTWARE OF CANADA LIMITED) for
>  (JOHN LIMITED)?
> *N
> Assume (JOHN BROWNS SOFTWARE LIMITED) for (JOHN LIMITED)?
> *N
> Assume (JOHN WILSON AND SONS LIMITED) for (JOHN LIMITED)?
> *OK
> SELECT  PROGRAM.PUBLISHER
> FROM    PROGRAM
> WHERE
>  (PROGRAM.PUBLISHER  =
>        'JOHN WILSON AND SONS LIMITED')
>  AND (PROGRAM.RATING > 3)

> I don't know. There are no records in the database
> which satisfy the given criteria.

> Please enter your question.
> *<Q42>  DOES APPLE SELL FOOD?
>
> Unknown word - FOOD
> What would you like to do?
> 1. Synonym!  Enter the replacement word.
> 2. Unknown data element!  Continue processing.
> 3. Ignore the word!  Continue processing.
> 4. Cancel this query.
Enter a number between 1 and 4 and press Return.

*2

NOTE - (FOOD) is not (a type/brand of a computer) contained in this database!
The above query cannot be answered.

Please enter your question.

*<Q43> DOES STEVE SMITH SELL GOOD PROGRAMS?

NOTE - (STEVE SMITH) is not (the name of the publishing company of the program) contained in this database!
The above query cannot be answered.

Please enter your question.

*<Q44> DOES JOHN VAVRIK SELL GOOD PROGRAMS?

Unknown word - VAVRIK

What would you like to do?

1. Spelling Error! Enter the correct spelling.
2. Abbreviation! Enter the full word.
3. Synonym! Enter the replacement word.
4. Unknown data element! Continue processing.
5. Ignore the word! Continue processing.
6. Cancel this query.

Enter a number between 1 and 6 and press Return.

*4

Which category does VAVRIK belong to?

1. a Common Noun (car, computer, pen)
2. a Proper Noun (Joe, Smith, Canada)
3. an Adjective (purple, good, big)

Enter a number between 1 and 3 and press Return.

*2

NOTE - (JOHN VAVRIK) is not (the name of the publishing company of the program) contained in this database!
The above query cannot be answered.

Please enter your question.

*<Q45> PRINT THE COMPUTER NUMBERS!

What does (THE COMPUTER NUMBERS) best refer to?

1. (a unique identifier for each computer)
2. (the standard amount of memory on the computer in kilobytes)
3. None of the above
Enter a number between 1 and 3 and press Return

SELECT COMPUTER.CNO
FROM COMPUTER

; Note that the retrieved data has been ommitted for
; this example.

Enter a number between 1 and 4 and press Return

Please enter your question.

*<Q46> PRINT THE VERSION NUMBERS!

SELECT PROGRAM.VERSION
FROM PROGRAM

; Note that the retrieved data has been ommitted for
; this example.

Please enter your question.

*<Q47> PRINT THE NUMBERS!

What does (THE NUMBERS) best refer to?

1. (a unique identifier for each computer)
2. (a unique identifier for each program)
3. (the version of the computer program if any)
4. (the standard amount of memory on the computer in kilobytes)
5. (the computer number which the program runs on)
6. (a unique identifier for each descriptor)
7. None of the above

Enter a number between 1 and 7 and press Return

SELECT COMPUTER.CNO
FROM COMPUTER

; Note that the retrieved data has been ommitted for
; this example.
APPENDIX I - DATABASE SCHEMA FOR THE SUPPLIER DATABASE

The following Database Schema is defined for the Supplier/Part/Job database.

Database Definition

\[(S ; ** Supplier Relation **

\[(FIELDS \(SNO \ SNAME \ STATUS \ CITY\))

\(\text{KEY} \ (SNO))

\(\text{IDENTIFIERS} \ (SNO \ SNAME \ CITY))

\(\text{CATEG} \ (\text{HUMAN}))

\(\text{L-NOUN} \ (\text{SUPPLIER}))

\(\text{L-PREP} \ (\text{FROM}))

\(\text{L-DESC} \ (\text{PART}))

\(\text{COMMENT} \ (\text{a supplier of a part used in a job}))

\)\]

\(\text{SNO}

\(\text{TYPE} \ (\text{GNAME}))

\(\text{CONTENTS} \ (\text{INVERT NPR} ?))

\(\text{L-TYPE} \ (\text{RNUMBER}))

\(\text{COMMENT} \ (\text{A unique identifier for each supplier}))

\)\]

\(\text{SNAME}

\(\text{TYPE} \ (\text{GNAME}))

\(\text{CONTENTS} \ (\text{INVERT NPR} ?))

\(\text{COMMENT} \ (\text{the name of a supplier}))

\)\]

\(\text{STATUS}

\(\text{TYPE} \ (\text{INTEGER}))

\(\text{CONTENTS} \ (\text{INTEGER}))

\(\text{L-NOUN} \ (\text{STATUS}))

\(\text{COMMENT} \ (\text{the status of the supplier}))

\)\]

\(\text{CITY}

\(\text{TYPE} \ (\text{LOCATION}))

\(\text{CONTENTS} \ (\text{INVERT NPR} ?))

\(\text{L-TYPE} \ (\text{CITY}))

\(\text{COMMENT} \ (\text{the city where the supplier is located}))

\)\]
(P ; ** The Part relation **

(FIELDS (PNO PNAME COLOUR WEIGHT))
(KEY (PNO))
(IDENTIFIERS (PNO PNAME))
(CATEG (PHYSICAL-OBJECT))
(L-TYPE (PRODUCT))
(L-NOUN (PART PERIPHERAL))
(L-PREP (WITH))
(L-DESC (COMPUTER))
(COMMENT (a part used in a job))
)

(PNO)
(TYPE (GNAME))
(CONTENTS (INVERT NPR ?))
(L-TYPE (RNUMBER ))
(COMMENT (a unique identifier for each part))
)

(PNAME)
(TYPE (GNAME ))
(CONTENTS (INVERT N ?))
(COMMENT (the name of the part))
)

(COLOUR)
(TYPE (COLOUR))
(CONTENTS (CHAR))
(COMMENT (the colour of a part))
)

(WEIGHT)
(TYPE (Q-MASS))
(CONTENTS (INTEGER))
(COMMENT (the weight of the part))
)

(J ; ** The Job relation **

(FIELDS (JNO JNAME CITY))
(KEY (JNO))
(IDENTIFIERS (JNO JNAME))
(CATEG (EVENT))
(L-NOUN (JOB PROJECT))
(L-PREP (ON FOR TO))
(COMMENT (the jobs that the company is involved in))
)
(JNO ; ** The JOB relation (cont.) **
  (TYPE (GNAME))
  (CONTENTS (INVERT NPR ?))
  (L-NOUN (NUMBER))
  (COMMENT (a unique identifier for each job))
)

(JNAME
  (TYPE (GNAME))
  (CONTENTS (INVERT N ?))
  (COMMENT (the name of a job/project))
)

(CITY
  (TYPE (LOCATION))
  (CONTENTS (INVERT NPR ?))
  (L-TYPE (CITY))
  (COMMENT (the city where the job is based in))
)
)

(SPJ ; ** The Supplier/Part/Job relation **
  (FIELDS (SNO PNO JNO QTY))
  (CATEG (EVENT))
  (COMMENT (Specifies the quantity of parts for a job from a supplier))
)

(SNO
  (TYPE (GNAME))
  (CONTENTS ((KEY S SNO)))
  (COMMENT (a unique identifier for each supplier))
)

(PNO
  (TYPE (GNAME))
  (CONTENTS ((KEY P PNO)))
  (COMMENT (a unique identifier for each part))
)

(JNO
  (TYPE (GNAME))
  (CONTENTS ((KEY J JNO)))
  (COMMENT (a unique identifier for each job))
)

(QTY
  (TYPE (INTEGER))
  (CONTENTS (INTEGER))
  (TYPE (Q-CNT))
  (COMMENT (the number of a specified part supplied by a supplier for a job))
))
Verb Definition

(SUPPLY
  VFRAME (((SUBJ (RELATION S)) (DOBJ (RELATION P))
    (IOBJ (RELATION J) OPT))
    ((SUBJ (RELATION S)) (DOBJ (RELATION J))
      (PP (WITH) (RELATION P) OPT ))))
JOINREL (SPJ)
JOININFO ("S.SNO = SPJ.SNO AND P.PNO = SPJ.PNO AND
          J.JNO = SPJ.JNO")
JOINTYPE ((S MANY)(P MANY))
)

(USE
  VFRAME (((SUBJ (RELATION J)) (DOBJ (RELATION P))
    (PP (FROM) (RELATION S) OPT))
JOINREL (SPJ)
JOININFO ("S.SNO = SPJ.SNO AND P.PNO = SPJ.PNO AND
          J.JNO = SPJ.JNO")
JOINTYPE ((J MANY)(P MANY))
)

(OBTAIN
  SYNONYM USE
  )
Default Joins and Association Factors

(DEFAULTJOIN ; Default join between Jobs and Parts
  'J 'P ; Relation names
  '3 ; Association factor
  '((J MANY) (P MANY)) ; JOIN TYPE
  "(J.JNO = SPJ.JNO AND
  P.PNO = SPJ.PNO)"
  '(SPJ)) ; Secondary Relation

(DEFAULTJOIN
  'J 'S
  '3
  '((J MANY) (S MANY))
  "(J.JNO = SPJ.JNO AND S.SNO = SPJ.SNO)"
  '(SPJ))

(DEFAULTJOIN
  'S 'P
  '3
  '((S MANY) (P MANY))
  "(S.SNO = SPJ.SNO AND P.PNO = SPJ.PNO)"
  '(SPJ))

(DEFAULTJOIN
  'J 'SPJ
  '3
  '((J MANY) (SPJ MANY))
  "(J.JNO = SPJ.JNO)"
  '(SPJ))

(DEFAULTJOIN
  'P 'SPJ
  '3
  '((P MANY) (SPJ MANY))
  "(P.PNO = SPJ.PNO)"
  '(SPJ))

(DEFAULTJOIN
  'S 'SPJ
  '3
  '((S MANY) (SPJ MANY))
  "(S.SNO = SPJ.SNO)"
  '(SPJ))

(DEFAULTJOIN
  'S 'SPJ
  '3
  '((S MANY) (SPJ MANY))
  "(S.SNO = SPJ.SNO)"
  '(SPJ))
APPENDIX J - SAMPLE "SUPPLIER/PART/JOB" DATABASE SESSION

> Which database would you like to use?
> 1. The PROGRAM database
> 2. The Supplier/Part/Job database
>
> Enter a number between 1 and 2 and press Return

*2
> THE DICTIONARY HAS BEEN REFERENCED.
> Welcome to the SUPPLIER/PART/JOB database.
> To stop type 'STOP!' or 'QUIT!'.
>
> Please enter your question.

*PRINT THE STATUS OF SUPPLIERS IN PARIS?
>
> SELECT S.STATUS
> FROM S
> WHERE
>     (S.CITY = 'PARIS')

<table>
<thead>
<tr>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
<tr>
<td>30</td>
</tr>
</tbody>
</table>

> Please enter your question.

*WHO SUPPLIES RED, GREEN OR BLUE PARTS?
>
> SELECT UNIQUE S.SNO , S.SNAME , S.CITY
> FROM S , P , J , SPJ
> WHERE
>     (S.SNO = SPJ.SNO AND P.PNO = SPJ.PNO AND
>      J.JNO = SPJ.JNO)
> AND
>     (P.COLOUR = 'RED' OR P.COLOUR = 'GREEN' OR
>      P.COLOUR = 'BLUE')

<table>
<thead>
<tr>
<th>SN</th>
<th>SNAME</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>SMITH</td>
<td>LONDON</td>
</tr>
<tr>
<td>S2</td>
<td>JONES</td>
<td>PARIS</td>
</tr>
<tr>
<td>S3</td>
<td>BLAKE</td>
<td>PARIS</td>
</tr>
<tr>
<td>S4</td>
<td>CLARK</td>
<td>LONDON</td>
</tr>
<tr>
<td>S5</td>
<td>ADAMS</td>
<td>ATHENS</td>
</tr>
</tbody>
</table>
Please enter your question.
*WHAT SUPPLIERS SUPPLY JOB J1 WITH RED NUTS IN PARIS?

```
> SELECT UNIQUE S.SNO, S.SNAME, S.CITY
> FROM S, J, P, SPJ
> WHERE (S.SNO = SPJ.SNO AND P.PNO = SPJ.PNO AND
>         J.JNO = SPJ.JNO)
>        AND (J.JNO = 'J1')
>        AND (J.CITY = 'PARIS')
>        AND (P.PNAME = 'NUT')
>        AND (P.COLOUR = 'RED')
```

<table>
<thead>
<tr>
<th>SN</th>
<th>SNAME</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>SMITH</td>
<td>LONDON</td>
</tr>
</tbody>
</table>

Please enter your question.
*WHO SUPPLIES P3S TO JOBS IN ATHENS?

```
> SELECT S.SNO, S.SNAME, S.CITY
> FROM S, P, J, SPJ
> WHERE (S.SNO = SPJ.SNO AND P.PNO = SPJ.PNO AND
>         J.JNO = SPJ.JNO)
>        AND (P.PNO = 'P3')
>        AND (J.CITY = 'ATHENS')
```

<table>
<thead>
<tr>
<th>SN</th>
<th>SNAME</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>JONES</td>
<td>PARIS</td>
</tr>
<tr>
<td>S5</td>
<td>ADAMS</td>
<td>ATHENS</td>
</tr>
</tbody>
</table>

Please enter your question.
*DOES THE COLATOR PROJECT USE BOLTS FROM SMITH?

```
> (*** Spelling Mistake *** COLATOR --> COLLATOR)
> SELECT J.JNO, J.JNAME
> FROM J, P, S, SPJ
> WHERE (J.JNO = SPJ.JNO AND P.PNO = SPJ.PNO AND
>         S.SNO = SPJ.SNO)
>        AND (J.JNAME = 'COLLATOR')
>        AND (P.PNAME = 'BOLT')
>        AND (S.SNAME = 'SMITH')
```

I don't know. There are no records in the database which satisfy the given criteria.
> Please enter your question.
*WHERE IS SMITH?*
> Is it OK to assume that (WHERE) refers to (the city
> where the supplier is located)?
*OK*
>
> SELECT S.CITY
> FROM S
> WHERE
>     (S.SNAME = 'SMITH')

CITY
--------------
LONDON

> Please enter your question.
*HOW MUCH DOES PART P4 WEIGH?*
>
> SELECT P.WEIGHT
> FROM P
> WHERE
>     (P.PNO = 'P4')

WEIGHT
--------
14

> Please enter your question.
*STOP!*