A Natural Language Database Interface For SQL-Tutor

5th November 1999

Honours Project by Seymour Knowles
Supervisor: Tanja Mitrovic

Abstract

An investigation into integrating a database Natural Language Processing (NLP) component into the SQL-Tutor Intelligent Tutoring System (ITS) is presented. Tailor-made NLP systems created by a programmer, and NLP systems created automatically by a general database NLP system, are considered with respect to the requirements of SQL-Tutor. A tailor-made system is created for the MOVIES database using a semantic grammar, and its strengths and weaknesses are demonstrated. Three 'levels' of information are identified in the tailor-made system; database independent information, database structure and database semantics. These levels are used to assess a commercial system for automatic database NLP generation. Some weaknesses are identified in this system and a proposal of an 'authoring tool' to remedy these weaknesses is presented.

Contents

1. Introduction

"Why aren't computers easier to use?" inquired the unsuspecting beginner computer user. Such an easily posed question has many long and complicated answers.

'Why don't you understand!" demanded the frustrated computer abuser. This is a situation that computer programmers and researchers spend their lives studying and trying to prevent.

One potential solution to these issues plaguing computer interface design is Natural Language Processing (NLP). The main goal of NLP is for an English sentence (or a sentence in any spoken language) to be interpreted by the computer and appropriate action taken. The sentence could be typed into the computer or obtained from a speech recognition program. Then the difficulty is to work out what the sentence means, whether or not some action should be taken, and what the appropriate action is.

The area of NLP research is still very experimental and systems so far have been limited to small domains, where only certain types of sentences can be used. When systems are scaled-up to cover larger domains, NLP becomes very difficult due to the natural ambiguity in spoken sentences, and the vast amount of information that needs to be incorporated in order to disambiguate such sentences. For example, the sentence: "The woman saw the man on the hill with the telescope." could have many different meanings. To understand what the intended meaning is, we have to take into account the current context, such as the woman is a witness, and any background information, such as there is a hill nearby with a telescope on it. Alternatively the man could be on the hill, and the woman may be looking through the telescope. All this information is very difficult to represent in the computer, so restricting the domain of an NLP system is the only practical way to get a manageable subset of English to work with.

One area in which NLP systems are powerful enough to be effective is database query systems. Databases usually cover a small enough domain so that an English question about the data within it can be easily analysed by an NLP system. The database can be consulted and an appropriate response can be generated.

Much research is still going on in NLP database interfaces. Areas such as error reporting, when a sentence cannot be analysed successfully, new approaches to NLP systems, and automatic generation of database NLP systems are still important unresolved issues. However, the standard approach to database NLP systems is well established. This approach creates a 'semantic grammar' for each database and uses this to parse the English question. The semantic grammar creates a representation of the semantics, or meaning, of the sentence. After some analysis of the semantic representation, a database query can be generated in SQL

(Structured Query Language) or any other database language. Then, if necessary, a response can be given or appropriate action taken.

The drawback of this approach is that the grammar must be tailor-made for each database. Some systems allow automatic generation of an NLP system for each database, but in almost all cases there is insufficient information in the database to create a reliable NLP system. Additional information about what the data in the database represents must be provided to create NLP systems that can handle all possible questions, rather than just some questions.

One of many possible applications for such systems is an Intelligent Tutoring System (ITS) developed by Tanja Mitrovic (1998), named SQL-Tutor, which tutors students in the database language SQL. SQL-Tutor guides students through questions from four different databases, and helps the student to create an SQL query to answer the question. The first goal of this paper is to explore how grammar systems are designed, what they can be used for, and whether there are any better ways to create them. A grammar for the MOVIES database (a database in SQL-Tutor) is created and analysed. Such a system could be used to automatically add new problems into SQL-Tutor, allowing a lecturer to customize SQL-Tutor by producing a new list of questions. Students could even ask the tutor new questions, simply by typing in their question, and SQL-Tutor would be able to assist them as usual.

Some commercial systems are already available to automatically generate a database NLP system, based on the existing database structure. An evaluation of one of these systems is carried out, and deficiencies are identified when advanced queries are attempted. SQL-Tutor trains students in advanced queries, which may involve difficult concepts such as HAVING clauses, sub-select statements or difficult join conditions. For this reason an appropriate database NLP system will have to be able to handle advanced English questions. Simply creating an NLP system based on the MOVIES database alone will not be sufficient for these advanced queries; extra information will be needed from someone who understands the database.

Using the results from this attempt at creating a database NLP system, the second goal of this research is to investigate the feasibility of an authoring tool to help automate the creation of database NLP systems so that more advanced queries can be used successfully. Such an authoring tool would enable anyone who is familiar with a database to create an NLP system for it. This sort of authoring tool would allow lecturers, or even students, to input their own databases, as well as their own questions, into SQL-Tutor. The same approach could be used as an extension to commercial general-purpose database NLP systems, separate to SQL-Tutor.

2. Background

There are three areas of research associated with this study; NLP in general, NLP as an interface to databases and ITS. This section provides a brief overview of these three areas.

2.1 Natural Language Processing

There has been much work on NLP recently, but the area has been around for a relatively long time in the computing world. The main aim of NLP research is to create a better interface to the computer. Spoken language is the most natural interface available for humans to use, but computers are still unable to come close to the rich communication humans can achieve with each other. Science fiction has created many robots or computers that are able to understand and carry out tasks based on spoken orders or communication. 'Data', an android from the *Star Trek the Next Generation* movies and series can communicate as well as any human in English. The 'HAL' computer from the book and movie *2001 a Space Odyssey* converses verbally with the members of the space ship. Even a toaster from the book and television series *Red Dwarf* manages to hold an intelligent conversation. As these authors have been imagining computers that communicate with humans through natural language, computer scientists have been attempting to make it a reality, but success has so far been limited to specific domains. Here are some examples of early NLP systems:

- ELIZA – by Joseph Weisenbaum (1966). This program is a natural language interface to a psychiatrist. It used pattern-matching rules that were triggered based on key words found in user's dialog. ELIZA used literal text form within users dialog to reformulate questions. There was no 'understanding' of what was being said, ELIZA just gave back questions that seemed most relevant according to the last user input. Weisenbaum reported that some subjects were convinced that ELIZA was a real person. He notes "The human speaker will contribute much to clothe ELIZA's responses in vestments of plausibility."
- SHRDLU – by Terry Winnograd (1973). This is one of the first programs that could carry out tasks and provide responses in natural language well. It was bound within an artificial blocks world of coloured bricks and pyramids. SHRDLU was able to perform tasks like moving objects around within the limited world, when directed to do so in English. The program used a procedural representation for semantics. This means that each English predicate or term was associated with a procedure that conveyed the meaning (or semantics) of the term. The problem with procedural semantics is that they do not scale up into large domains.

Today there is much more demand for better interfaces to computers and much has been achieved in areas of Graphical User Interface (GUI) development. The Windows and Macintosh operating systems are both based primarily on a GUI environment. However, NLP systems have not yet become widely used in the computer world. The main reason for this is that good NLP systems are very difficult to make, due to the scale-up problems encountered in large domains. ELIZA got around this problem by reusing the users input to the system to formulate new questions, and SHRDLU was limited to blocks world, so that the domain would be small enough to handle. Ambiguity in English sentences becomes a more important problem when larger domains are considered and no method to resolve this ambiguity correctly has yet been discovered (but many people are trying).

2.2 Natural Language Interfaces to Databases

The very first attempts at NLP database interfaces are just as old as any other NLP research. In fact database NLP may be one of the most important successes in NLP since it began. Asking questions to databases in natural language is a very convenient and easy method of data access, especially for casual users who do not understand complicated database query languages such as SQL. The success in this area is partly because of the real-world benefits that can come from database NLP systems, and partly because NLP works very well in a single-database domain. Databases usually provide small enough domains that ambiguity problems in natural language can be resolved successfully.

Here are some examples of database NLP systems:

- LUNAR (Woods, 1973) involved a system that answered questions about rock samples brought back from the moon. Two databases were used, the chemical analyses and the literature references. The program used an Augmented Transition Network (ATN) parser and Woods' Procedural Semantics. The system was informally demonstrated at the Second Annual Lunar Science Conference in 1971. Its performance was quite impressive: it managed to handle 78% of requests without error, a figure that rose to 90% when dictionary errors were corrected. This figure is misleading because the system was not subject to intensive use. A scientist who used it to extract information for everyday work would soon have found that he wanted to make requests beyond the linguistic ability of the system. ATN parsers are useful because they are very efficient, even for large grammars; however, ungrammatical sentences are not handled well and they are not very flexible.

- LIFER/LADDER was one of the first good database NLP systems. It was designed as a natural language interface to a database of information about US Navy ships. This system, as described in a paper by Hendrix (1978), used a semantic grammar to parse questions and query a distributed database.

  The LIFER/LADDER system could only support simple one-table queries or multiple table queries with easy join conditions. Hendrix demonstrated the capabilities of LIFER/LADDER by giving these examples in his paper:

What are the length, width, and draft of the Kitty Hawk?

When will Reeves achieve readiness rating C2?

What is the nearest ship to Naples with a doctor on board?

What ships are carrying cargo for the United States?

Where are they going?

Print the American cruisers' current positions and states of readiness?

These are all relatively simple queries, as were the rest of the demonstration queries. Note that the fifth question "Where are they going?" refers to the ships from the answer to the previous question. This gives a very powerful dialogue system that may compensate for the lack of advanced queries.

The LIFER/LADDER system used a semantic grammar (that is, it used labels such as "SHIP" and "ATTRIBUTE" rather than syntactic labels such as *noun* and *verb*). This NLP systems using semantic grammars are closely tied to the domains for which they were designed, and they can be easily adapted to suit new terms or phrases.

Even today the same general method is still being used; semantic grammars are now widely used in most NLP systems, but there are many variations and new approaches are continually being developed. Akama (1997) describes some variations on semantic grammars, including Montague semantics and operational semantics, which can support different forms of logic, and reasoning with incomplete information.

- More recently there have been ambitious projects investigating information represented in different ways, and over larger domains. For example Wenhua (1992) designed an NLP system for Computer Integrated Manufacturing (CIM) databases, which are large, diverse and represent completely different concepts, from accounting to computer-aided-design and schedule planning. Four different databases were used, and one NLP system was designed to access all of them, as necessary, to answer the user's questions. This system uses a Definite Clause Grammar (DCG) and a semantic interpreter to process the English question into a database query. This system is much larger than the earlier attempts at database NLP systems and advanced queries are possible across the different databases and different data representations.
- Commercial products have also emerged to take advantage of this new technology. Systems such as ELF (ELF Software Co. 1999), English Query and English Wizard attempt to generate NLP systems 'on the fly' for any database, so that new or user-made databases can be queried with a newly made NLP system. Various techniques are used to extract information from a given database so that an NLP system can be generated. Ideally the process of creating a new NLP system is almost as simple as choosing the database and clicking 'go'. Unfortunately this ideal situation is almost never the case because extra information is needed from outside the database. These systems sound very promising, but there is still a large amount of work to be done to

make them easy to use and more reliable. The ELF system's capabilities will be reviewed in Section 4.1.

2.3 Intelligent Tutoring Systems

ITSs are a relatively new field in computer science; they are computer programs that attempt to tutor students in some skill or task. They are referred to as 'Intelligent' because they attempt to identify how to best assist the student, just as a human tutor would.

For example, a student may have a problem with matrix multiplication. The tutor is asked for some assistance, the problem now is what sort of assistance to give – if the student has never attempted matrix multiplication before, then a full explanation is probably needed. Otherwise there may just be a small mistake in the students working – just pointing out the mistake may be enough in this situation. An ITS attempts to estimate the level of help that is required as well, by keeping a model of each student's performance and/or monitoring their work. Various methods can be used to decide what level or form of assistance would be appropriate for each student.

Because ITSs attempt to be almost as good as human tutors, a logical step in their design is to integrate them with NLP so that students may interact more naturally with the ITS. Several ITS have successfully used NLP to enhance their interfaces and allow richer interaction between computer and student.

One such ITS named SHERLOCK (Moore, 1995) had much success in the area of NLP. SHERLOCK teaches avionics technicians how to troubleshoot complex electronic equipment. Students can interact with a virtual test environment and SHERLOCK can give advice if they do something wrong. If a student is still confused or wants further confirmation, they may ask SHERLOCK a question in English, and SHERLOCK answers in English. A question and answer style discourse can then ensue between the student and SHERLOCK until the student is ready to move on.

Another ITS which as yet has no natural language interface is SQL-Tutor (Mitrovic, 1998). This teaches students the database query language SQL. The SQL-Tutor guides students through a number of database query problems from several databases. Because ITSs are a natural application area for NLP systems, and NLP systems work very well in a database domain, SQL-Tutor seems like a perfect candidate to merge the two fields together, by adding a NLP component onto the SQL-Tutor interface.

3. An NLP Interface to SQL-Tutor

This section gives an introduction to SQL-Tutor and the MOVIES database, followed by a discussion on how a database NLP system could be integrated with the SQL-Tutor ITS and what benefits this would bring. Following this, an NLP system designed for the MOVIES database is outlined and a demonstration of the system's strengths and weaknesses is given.

3.1 SQL-Tutor

SQL-Tutor is an ITS designed by Tanja Mitrovic (1998). SQL can be very difficult for beginner users to understand. The SQL-Tutor program tutors students by assisting the students through a number of database questions form four different databases. A student model is kept for each student based on query constraints (each constraint represents a part of the query that is necessary to answer the question). Each time a particular query constraint is used, SQL-Tutor records whether it was used successfully or unsuccessfully. In this way a model of a student's strengths and weaknesses is generated and SQL-Tutor can select questions which re-enforce problem areas or introduce new query concepts.

Figure 1 shows the windows interface to SQL-Tutor (an Internet based version is also now available). At the top of the window the current problem is shown as English text, 'List all directors born in or after 1920'. Below this is a section where the student can work on the SQL query that will answer the problem. A text box is provided for each SQL clause for the student to fill in. In the figure the question has already been solved correctly. The bottom section of the window displays the structure of the MOVIES database. Students can use this section to help fill in the SQL question or get further information on the current database.

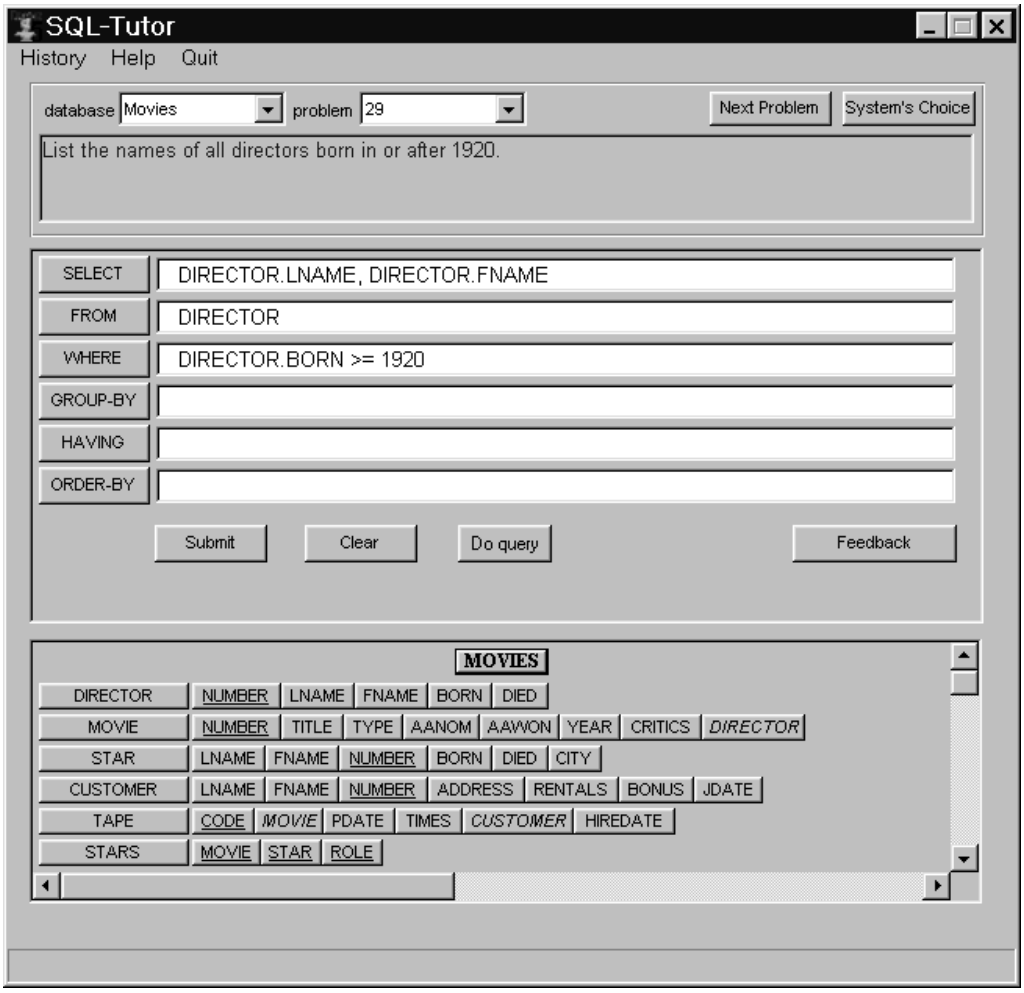

Figure 1. The SQL-Tutor interface – Students could type their own new problems into the top text box, and have SQL-Tutor help them solve the problem.

3.2 The MOVIES Database

One of the databases within SQL-Tutor is the movies database. It contains information for a video hire store. The structure of the database is shown in Figure 2; it consists of six tables, each of which contains several attributes. Customers (or members) of the store are recorded in the database, along with their details such as name, address and join date. The videotapes that the store owns are also recorded in the database, and information such as the purchase date, the customer who is renting it (if any), and the date it was hired. There is also information on the movies recorded on the videotapes, like title, the movie type (or genre), the number academy awards that each movie has been nominated for and has won, the year it was made, and a critics rating. Directors and stars (or actors) are also listed in the database along with some of their details. Each movie records the number of the director that directed it and a separate table (the 'stars' table) records the stars that have acted in each movie, along with the roles they played.

Figure 2 shows how the tables are joined up with connecting lines matching pairs of attributes. The primary keys of each table are shown in bold.

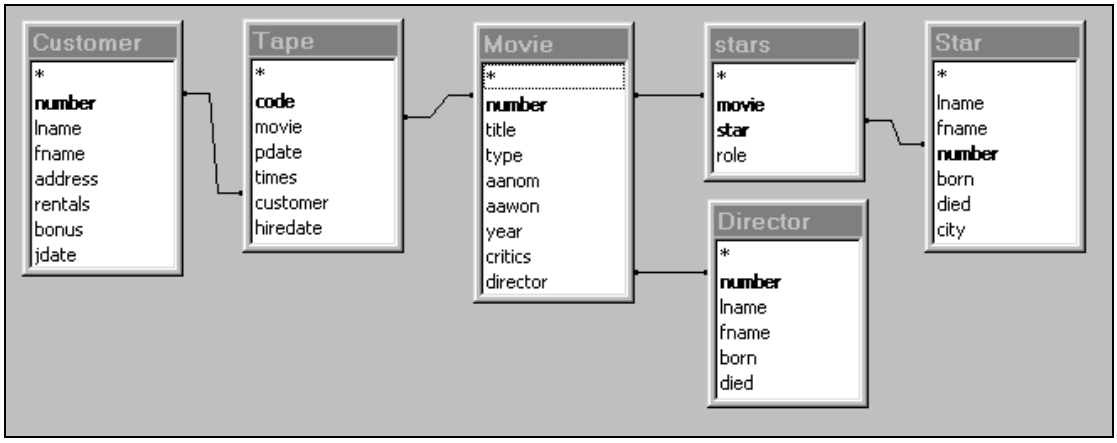

Figure 2. The MOVIES database

3.3 Requirements and Goals

Integrating an NLP component into SQL-Tutor could take many different forms. Conversational systems, that enable the student and ITS to carry out a discussion, can be very powerful and useful to students. Also, there is already a well-established body of research in the area of database NLP that would be useful. The simplest and most useful NLP system to integrate into SQL-Tutor is essentially a database NLP system that could be used to input a new problem in English. A programmer usually creates these systems by hand and much fine-tuning is required to create a robust NLP system that can handle advanced queries. These NLP systems will be referred to as tailor-made systems in this paper. This type of system could be used in two different ways:

- When students encounter a new problem that they cannot understand, the student could input the question as English text into the SQL-Tutor interface, and then SQL-Tutor can help the student to solve it.

- Lecturers or tutors that use SQL-Tutor to help teach students could add their own new questions simply by typing in the English text.

Another possibility in database NLP is a **general** NLP system. That is a system that will analyse a database and come up with a new NLP system for that particular database. Systems such as ELF (ELF Software Co. 1999) and other similar commercially available applications claim to be able to achieve this at the moment. These systems will be investigated in Section 4.1 to see whether they would be suitable for SQL-Tutor.

A general database NLP system would be much more useful; lecturers, tutors or possibly even students, could easily add new databases and new questions. SQL-Tutor could then be used with almost any database with ease.

SQL-Tutor is used to teach all aspects of the SQL language. Therefore the NLP system needed to support SQL-Tutor will also be required to handle all aspects of the SQL language. For example, the following advanced queries may cause

problems for NLP interfaces (all questions are for the MOVIES database, as shown in Figure 2).

- For each director list the director's number and total number of awards won by comedies he or she directed if that number is greater than one.

  This query requires a GROUP BY and HAVING clause combination, which can be very difficult for an NLP system to interpret correctly.

- Find the list of any pairs of stars who have the same last name.

  This query involves a complicated join of the STAR table with itself.

- List the movies in which the director has played a part in the movie.

  This query requires a complicated WHERE condition, matching the names of the DIRECTOR table and the STAR table. An NLP system would probably not know to match up the director and star names in this situation.

Unfortunately these types of questions are very important, as they demonstrate the full power of databases, and therefore these are the most useful types of queries. Simple questions are relatively easy to deal with, but part of the reason for having a database in the first place is so that complicated and powerful queries can be performed on the data. Having an NLP query system that can only handle simple queries is not sufficient; advanced queries are just as important, or even more important. More robust NLP systems are required to deal with the ambiguities that may occur with advanced queries.

Tailor-made systems have only recently become advanced enough to achieve the required level of performance for SQL-Tutor, but each new database NLP system takes a large amount of work to get the initial configuration right. Recently developed general systems aim to generate a new database NLP system, based on an existing database. This would be a much more useful addition to the SQL-Tutor, as it would allow users to add new databases and queries as they wish.

However it was found that general systems such as ELF have limited use. The new databases supplied to these systems must contain logical and well-named tables and attributes, and only certain pre-programmed simple types of queries are supported. Another option provided by the ELF system allows queries to be learnt by example. When using this option the user provides a set of example queries for the ELF system to analyse. Then the system reuses, or copies these queries to answer any new questions given from the user. This can be quite difficult for any user, because many different variations will be needed to achieve a comprehensive NLP system. Also, the user will have to formulate the answer to their questions before they ask them. This seems to defeat the purpose of having an NLP system.

Therefore, it was concluded that it is necessary to construct a tailor-made system able to support advanced queries for SQL-Tutor. Even though tailor made systems take a long time to make and a lot of initial configuration to get right, this is the best method available at the present time. Alternatively, we need to find a better method of creating general database NLP systems, so that more advanced and useful queries can be made, with as little user input as possible.

In order to research database NLP systems and their suitability for use within SQL-Tutor, an NLP system was created for the MOVIES database. The goals in creating this system were to investigate the workings of database NLP systems in general, and create a working NLP system for the MOVIES database, with the possibility of later being used in SQL-Tutor.

In Section 3.4 the NLP interface to the MOVIES database will be introduced and discussed and then in then evaluates in Section 3.5. In section 4 general systems will be analysed and some improvements suggested.

3.4 Semantic Grammars

Most small-scale NLP systems use a semantic grammar. These are different to normal English grammar parsers in the way they classify words and phrases. For example, in normal English the word 'movie' is just a *noun*, but in a semantic grammar it can be classified differently depending on its meaning, or semantics. For the MOVIES database, 'movie' would represent the MOVIE table in the database. In a similar way, words can represent any tables or attributes in the database, and other words or phrases can represent aspects of the SQL query. For example, the phrase 'greater than or equal to' is a *prepositional phrase* in normal English representation, but a semantic grammar may simply represent it as '>='. All these semantic representations can then be put together using the grammar rules to form an SQL query for the given sentence.

The MOVIES NLP system was built on top of an existing grammar parser designed by Russell (1995), written in Common Lisp. This parser supports semantic grammars for use in any type of NLP system. Using this system, a semantic grammar was created for the MOVES database, based on English questions from the COSC313 course, which teaches advanced use of SQL, and the COSC110 course, which teaches beginners database skills, using the Microsoft Access program. I also used some of my own questions, and my knowledge of English, as well as some informal testing with fellow students and friends to enhance the grammar for different styles of questions. A general approach to the grammar was taken, trying to reflect the structure of the database wherever possible, and grouping other sections of the grammar as logically as possible.

There are two main parts of a semantic grammar. The first is a **lexicon**, that stores all the possible words that the grammar is aware of. A simple entry in the lexicon might look like this:

```
(customer   -> customer patron member)
(customers   -> customers patrons members)
```

On the left-hand side of the '->' the word 'customer' defines a symbol that can be used in the grammar. When 'customer' is used in the grammar, it refers to the English words on the right-hand side of the '->', that is 'customer', 'patron' or 'member'. Similarly, the plurals of these words are shown in the next line. Only single English words, or terminal symbols, can appear in the lexicon.

The other part of the semantic grammar involves **rules** to combine the terminal symbols in the lexicon to form phrases or sentences in a specific way. For example, the rule

```
(ATT_TAPE_CUSTOMER -> RENTED by customer ATT_NUMBER)
```

Demonstrates how the TAPE.CUSTOMER attribute can be referred to in English. On the left-hand side is the non-terminal representation of the phrase. This can be used in other rules to refer to this phrase. On the right hand side is a combination of non-terminal symbols from other rules (in UPPER CASE) and from the lexicon (in lower case). RENTED represents another rule for synonyms of the verb 'rented', such as taken out, borrowed etc. Note that RENTED is not in the lexicon because 'taken out' is a two-word phrase, and so must be defined as a rule. The words 'by' and 'customer' appear in the lexicon, and the ATT_NUMBER symbol refers to a number attribute. Now the above rule can be used to parse the phrase "borrowed by member number 23", or "taken out by customer 4" etc.

At this point the semantics of the phrase have not been considered. If desired, a rule may have some meaning or semantics associated with it. The following rules represent several phrases by the semantics '>=':

```
((INTERVAL >=) -> greater than or equal to)
((INTERVAL >=) -> at or above)
((INTERVAL >=) -> at least)
((INTERVAL >=) -> no less than)
```

The left-hand side again shows the symbol, INTERVAL, but it is combined within brackets with a >= symbol to represent the semantics. If a rule with semantics appears on the right-hand side of a rule, the semantics may be re-used or duplicated on the left-hand side. For example:

```
((WHERE_PART (RENTALS $I $V)) ->
              (INTERVAL $I) (prop-number $V) ATT_RENTALS)
```

This rule represents part of a where clause, the semantic representation on the left-hand side is (RENTALS $I $V). RENTALS represents the rentals attribute (in the customer table), $I represents an interval condition, such as '>=', and $V represents a number, as parsed by the prop-number rule (a special rule to recognise numbers). This rule can represent phrases like "more than 15 rentals" as (RENTALS > 15).

At present the semantic grammar for the MOVIES NLP system consists of a lexicon of 300 words (or terminals) and 500 grammar rules (or non-terminals).

3.5 Three Levels of Information

While creating the MOVIES database NLP system, three main categories of information were identified within the semantic grammar.

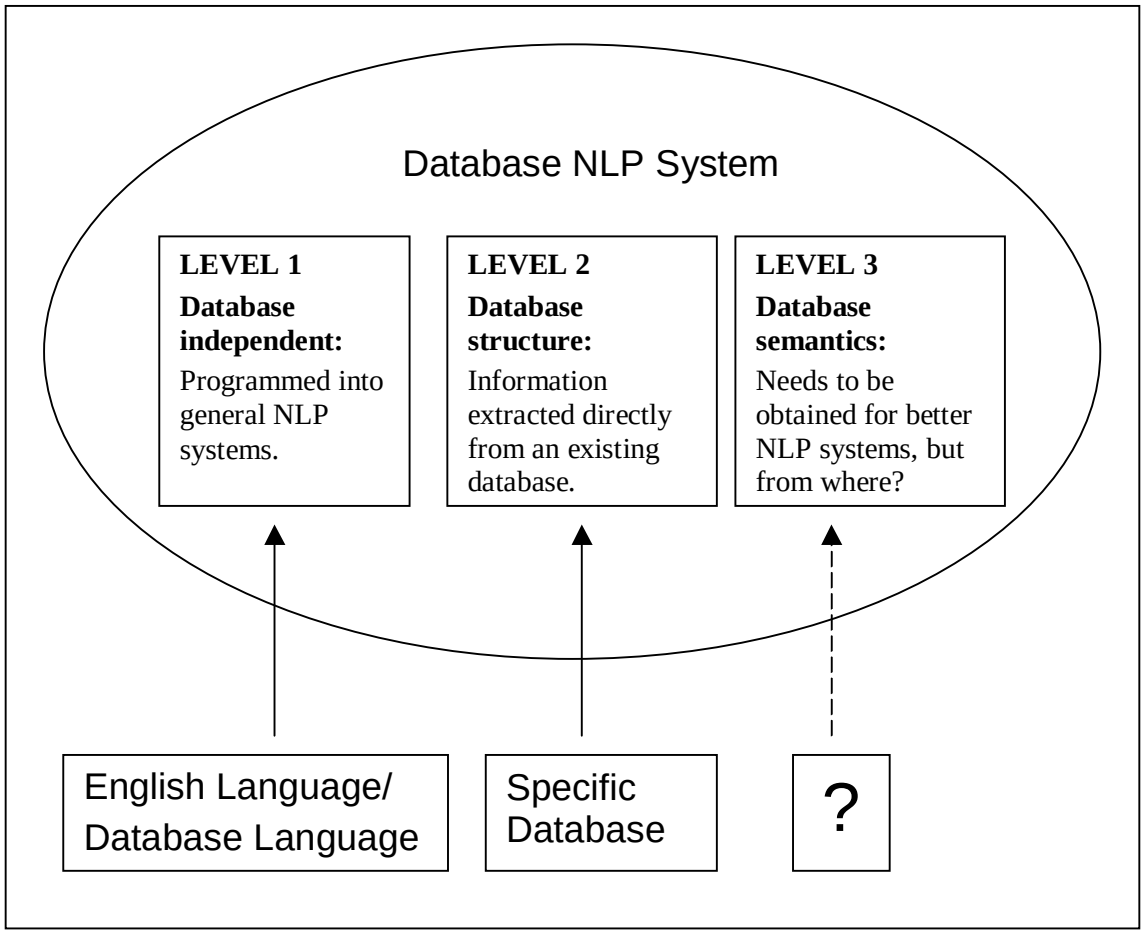

Figure 3. Three levels of information in a database NLP system

Two things that are common to any database NLP system are the spoken language (in this case English), and the database language (in this case SQL). As the first level of information relates these two languages as much as possible, this level of information will be named 'database independent'. The second level of information uses the database itself to add on information to the NLP system. Only information that can be taken directly from the database is in this level, so it will be called the 'database structure' level. The third level of information cannot be found from the database; it is information about what the database represents, and how the data is referred to in English sentences. This level will be referred to as 'database semantics'. A more in depth analysis of each level follows.

3.5.1 Database Independent

The first, lowest-level type of information is a combination of the two languages involved, English and SQL. These languages will be common to any similar NLP system. At this level we can represent the basics of any queries, no matter which database we will eventually use. For example the 'greater than or

equal to' in English is a representation of '>=' in SQL. This sort of information can be represented in the same way for any different type of database because it is relative to the types of query that can be made, rather than to any specific data.

There are two main ways to ask a database question; by giving a description of the information you want, such as "Retrieve a list of all the stars names." or by asking a question you want the information to answer, such as "What are all the stars names?" This distinction can be represented in the following way:

```
((QUERY $SQL) -> (DESCRIPTION $SQL))
((QUERY $SQL) -> (QUESTION $SQL))
```

For the MOVIES NLP system QUERY is the top-level symbol, or start symbol. Sentences are parsed successfully if they can be produced from the QUERY symbol. In that case, the resulting semantics for the sentence are shown as $SQL. On the right hand side two alternatives are shown, a DESCRIPTION or a QUESTION. Whatever semantic representation is produced by DESCRIPTION or QUESTION is the same as the $SQL that will end up as the final semantic representation

The following examples illustrate rules from the database independent level.

These are the rules for the 'less than' interval:

```
((INTERV <) -> less than)
((INTERV <) -> smaller than)
((INTERV <) -> under)
```

Rules for the aggregate function MAX():

```
(AGG_MAX   -> most)
(AGG_MAX   -> maximum)
(AGG_MAX   -> highest)
(AGG_MAX   -> biggest)
(AGG_MAX   -> maximum number of)
(AGG_MAX   -> highest number of)
(AGG_MAX   -> biggest number of)
```

Rules indicating several different ways to represent an 'and' or 'as well as' concept:

```
(AND_S  -> and)
(AND_S  -> as well as)
(AND_S  -> and also)
(AND_S  -> also)
(AND_S  -> in addition)
(AND_S  -> followed by)
(AND_S  -> next to)
(AND_S  -> with)
(AND_S  -> along with)
```

Note that _S after some rules indicates that synonyms are added. In the example above the 'AND_S' represents 'and' and some synonyms. This is so that the 'and' and 'AND_S' symbols are not confused (LISP is not case sensitive).

At present about 230 of the 500 rules in the MOVIES NLP system can be attributed to the database independent information.

3.5.2 Database Structure

The next 'level' of information in the NLP system is the database-specific information. This includes information such as the database structure – how database tables can be joined together, what type of information is stored in the different attributes, as well as table and attribute names and the database information content. This is very useful in the database NLP system and much information can be inferred from it. For example, in the MOVIES database there is a table named DIRECTOR, so we can refer to a director and know that this must be the relevant table. Of course this sort of rule assumes that the database has been logically named and structured. However, this is not always the case. For example, the STAR table contains information about stars or actors, but the STARS table lists movie numbers and the star (actor) numbers that star in them. This is a potential source of ambiguity in the NLP system, and the problem arises because 'stars' can be a noun or a verb in English. To remedy this problem we must introduce third level information about how the tables will be referred to in English.

In the database structure level we can represent the database within the semantic grammar. The word 'director' represents the DIRECTOR table; the word 'title' represents the TITLE attribute in the MOVIE table, etc. The following examples are from the MOVIES system:

Table names (semantics indicate which table is referred to):

```
((TABLE_S DIRECTOR) -> DIRECTOR_S)
((TABLE_S MOVIE) -> MOVIE_S)
((TABLE_S STAR) -> STAR_S)
((TABLE_S CUSTOMER) -> CUSTOMER_S)
((TABLE_S TAPE) -> TAPE_S)
((TABLE_S STARS) -> STARS_S)

(DIRECTOR_S -> director)
(MOVIE_S -> movie)
(STAR_S -> star)
(CUSTOMER_S -> customer)
(TAPE_S -> tape)
(STARS_S -> stars)
```

At this stage the tables are only referred to by their name, no extra synonyms are added, and the STARS table can be referred to as 'stars'. In practice this word should be reserved for the STAR table.

Attribute names:

```
(ATT_CITY  -> city)
(ATT_YEAR  -> year)
(ATT_ROLE    -> role)
(ATT_ADDRESS  -> address)
(ATT_RENTALS  -> rentals)
```

Another important source of information for the NLP system is the information content of the database. In a question such as “When was Stanley Kubrick born?” it is important to know if Stanley Kubrick is listed as a director or a star (or a movie title!). This information is more difficult to extract from the database, and if the data in the database changes, then the NLP system will have to be updated to reflect these changes.

Finally, the second level can be used to obtain information about how to join tables together when the final query is being constructed.

About 80 of the 500 rules in the MOVIES NLP system can be attributed to the database structure level of information.

3.5.3 Database Semantics

The third level of information can extend, or alter the other layers, depending on how the database is referred to in English. Sometimes simple assumptions may have to be made in order to represent the database in English; for example the FNAME attribute is not referred to as ‘fname’, but rather ‘first name’. Other more difficult assumptions are needed to resolve problems such as the STARS table, which has no physical representation in the real world – ideally the word ‘star’ or ‘stars’ will always refer to the STAR table unless a specific reference to the ‘stars table’ is made. This sort of information is not always obvious from the database itself, so some other source of information must be found. For the MOVIES NLP system and other tailor-made systems the human author serves as this source of information. We shall see in Section 4.1 that this is the main weakness of general NLP systems – there is no extra source of third level information.

Here are some examples of this information as applied in the MOVIES NLP system:

Attribute names:

```
(ATT_LNAME  -> last name)
(ATT_LNAME  -> surname)
(ATT_LNAME  -> maiden name)
(ATT_LNAME  -> family name)

(ATT_FNAME  -> first name)
(ATT_FNAME  -> given name)

(ATT_NAME   -> name) ;; Both first and last names
```

The AANOM attribute represents the number of academy award nominations:

```
(ATT_AANOM  -> number of AA_S nominations)
(ATT_AANOM  -> AA_S nominations)

(AA_S       -> academy award)
(AA_S       -> AA)
(AA_S       -> award)
```

The JDATE attribute represents the joining date for a customer.

```
;; Note: THEY_S gives some alternatives to they (he or she
etc).
(ATT_JDATE  -> join date)
(ATT_JDATE  -> date THEY_S joined)
(ATT_JDATE  -> date of joining)
(ATT_JDATE  -> when THEY_S joined)
```

Some extra table names:

```
(MOVIE_S -> movie)
(MOVIE_S -> film)

(STAR_S  -> star)
(STAR_S  -> celebrity)
```

This small extract shows what a wide range of information is contained in the database semantics level. In natural English sentences people are use this sort of information so that their sentences make proper sense. Therefore it is essential that the NLP system is aware of this information and can make use of it well.

The database semantics level can add to or even replace the information obtained from the database structure level. In the MOVIES NLP system about 190 of the 500 rules have been added due to the database semantics level.

Alternatives to the semantic grammar approach, such as Montague semantics and operational semantics (Akama 1977) offer advantages in the different forms of logic and reasoning that can be performed on resulting grammar parses. However, these approaches still require the three levels of information stored in the grammar to work properly. The database semantics level is still needed for a properly robust NLP system not matter what approach you use, so any general database NLP system should provide a way to obtain this information.

3.6 The MOVIES NLP System

The first goal in this project was to create a tailor-made NLP system for the MOVIES database; a semantic grammar was used to implement this system. The MOVIES NLP system currently accepts almost all basic questions (involving only SELECT, FROM and WHERE clauses) about the MOVIES database, and produces the corresponding SQL query. Aggregate functions (such as SUM(), COUNT(), AVERAGE(), etc) are also accepted as well as some questions requiring a GROUP BY clause, but questions requiring the HAVING clause are not yet accepted by the grammar.

The general layout of the NLP system structure is shown in Figure 4.

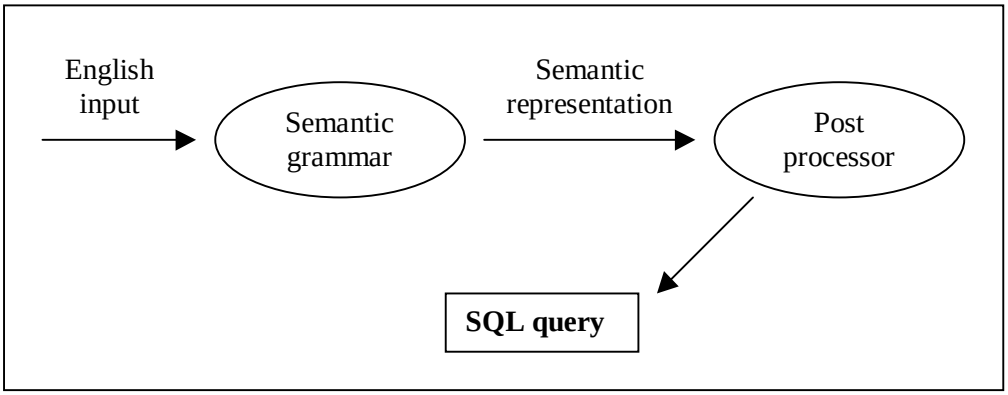

Figure 4. MOVIES NLP system structure

Firstly, the English input (in the form of a list) is parsed by the semantic grammar, then a post-processor matches table and attribute names and joins up tables if the query involves more than one table. After that the post-processor can construct the resulting SQL query and output it.

The question shown in Figure 1, being solved in the SQL-Tutor interface, asks the student to "List the names of all directors born in or after 1920". A demonstration of how this question is processed by the MOVIES NLP system is given below.

The English test is converted into a list, so that the Lisp-based grammar system can process it. The sentence becomes:

```
(LIST THE NAMES OF ALL DIRECTORS BORN IN OR AFTER 1920)
```

Now this list is parsed by the semantic grammar. The parser attempts to find all possible parses of the sentence, using the MOVIES semantic grammar. This sentence only has one possible parse, so only one semantic representation is returned:

```
((SELECT NAME! FROM (DIRECTOR (WHERE (BORN >= 1920)))))
```

The semantics already look similar to an SQL query. The brackets in the semantics indicate how the phrases from the sentence relate to each other. For example, the `(DIRECTOR (WHERE (BORN >= 1920)))` section shows that the 'born in or after 1920' condition relates to the director. The 'NAME!' symbol, after SELECT, is a special representation for both the first and last names.

This semantic representation of the sentence is now taken by a post-processor to transform it into an SQL query. The 'NAME!' symbol is expanded to FNAME and LNAME and these are identified as attributes in the DIRECTOR table because director is the next table name in the semantic representation (rather than the CUSTOMER or STAR tables). The BORN attribute is also matched to the DIRECTOR table because the WHERE clause directly refers to the DIRECTOR table. After processing the semantics, there is only one table needed for this query,

so only the DIRECTOR table is included in the FROM clause, and there is no need to add join conditions. Now the following SQL query is produced:

```
SELECT DIRECTOR.FNAME, DIRECTOR.LNAME
FROM DIRECTOR
WHERE DIRECTOR.BORN >= 1920
```

This SQL query correctly represents the English question; we could run this query on the MOVIES database to find the answer.

Sometimes several possible parses of the sentence are obtained, indicating that the English sentence has several possible meanings. This inherent ambiguity is difficult to handle in NLP systems. Luckily, most of the ambiguities that arise in database NLP systems can be resolved because of the small domain (one database). If an ambiguous sentence does occur the MOVIES NLP system converts all parses into SQL queries, resulting in several possible answers. In such cases it may be desirable to have some way to confirm which meaning the user has in mind by asking them a question, showing them the possible alternatives, or re-iterating in English how the question will be interpreted. Shu (1994) studied the effects of a system echo – where each question is restated to show what the system intends to do with the query. The results showed that the system echo produces no improvement in error recovery or confidence for novice users.

Another problem can arise when the English sentence is not parsed by the grammar at all. Joseph (1991) designed a system that identifies badly posed queries and generates a meaningful explanation of what went wrong, so that the user can re-word the question. This feedback is especially useful when the NLP system would otherwise give an uninformative error message. In this situation the MOVIES NLP system simply returns the error message “Sorry, can’t help”.

3.7 Evaluation

To give an indication of the current performance of the MOVIES NLP system, this section will demonstrate several examples, showing both successful and unsuccessful queries to the system.

QUESTION: List the names of directors over 30.

ANSWER:
```
SELECT DIRECTOR.FNAME, DIRECTOR.LNAME
    FROM DIRECTOR
    WHERE NOW() - DIRECTOR.BORN > 30
```

For this question the MOVIES NLP system correctly assumes that the ‘over 30’ at the end of the question refers to the director’s age. In the WHERE clause the age of the director is calculated using a NOW() function to give the current year. This function could be altered to suit the SQL query engine that is being used.

QUESTION: List the number of directors over the age of 30.

ANSWER: SELECT DIRECTOR.NUMBER
FROM DIRECTOR
WHERE NOW() - DIRECTOR.BORN > 30

SELECT COUNT(*)
FROM DIRECTOR
WHERE NOW() - DIRECTOR.BORN > 30

This question has resulted in two possible parses through the grammar, resulting in two candidate SQL queries. The first query lists the DIRECTOR.NUMBER attribute for each of the directors over the age of thirty, and the second query counts how many directors are over 30. The second query is probably the most appropriate answer for this question, but both interpretations could be correct. This is a difficult ambiguity to resolve; in this situation the user should be able to choose which meaning they had intended when asking the question.

QUESTION: How many movies have won more than 3 academy awards.

ANSWER: SELECT COUNT(*)
FROM MOVIE
WHERE AAWON > 3

This question is worded more carefully – instead of 'list the number of', the phrase 'how many' is used to ask for a count operation, so there is no ambiguity. The NLP system has also identified the correct attribute (AAWON) from the phrase 'won more than 3 academy awards', by using rules from the database semantics level programmed into the grammar.

QUESTION: How many movies in each category have won more than 2 academy awards.

ANSWER: SELECT: COUNT(*)
FROM: MOVIE
WHERE: AAWON > 2
GROUP BY: TYPE

This query uses a GROUP BY clause to split the movies up by their TYPE attribute, and then count how many are in each 'category' or 'type'.

QUESTION: List customer names and the codes of tapes they are renting.

ANSWER: SELECT CUSTOMER.FNAME, CUSTOMER.LNAME, TAPE.CODE
FROM CUSTOMER, TAPE
WHERE TAPE.CUSTOMER = CUSTOMER.NUMBER

This query joins two tables to answer the question; the join condition is worked out automatically.

Not all questions are parsed correctly by the MOVIES NLP system; here are some common examples of legitimate questions that cannot be parsed correctly by the current grammar:

QUESTION: Which movies have won the most academy awards?

This question requires a complicated Sub-select statement, which is not yet supported by the MOVIES NLP system.

QUESTION: List all customer names and the names of stars that appear in the movies they are currently renting.

This question would require several join conditions to find the appropriate star's names; however, this is not the problem. The phrase 'stars that appear in the movies they are currently renting' is not entered into the grammar to join the CUSTOMER and STAR tables. This is a 'nested' type of reference, going from stars to movies to customers. This demonstrates a gap in the database semantics level of the MOVIES NLP system. Introducing some more rules in the grammar would remedy this problem.

QUESTION: For each director list the director's name and the total number of awards won by comedies he or she directed, if that number is greater than 1.

This question cannot be parsed successfully, because it requires a HAVING clause to make sure that the number of academy awards won is greater than 1, and the HAVING clause is not yet supported by the MOVIES NLP system.

Despite these limitations, the MOVIES NLP system is able to successfully produce an SQL query to answer most basic questions. The database semantics layer is used to enhance the NLP system so that terms such as 'age' are recognised and can be used in a question. In Section 4.1 we shall see that 'age' is often misinterpreted by the ELF system. Therefore in this small domain the MOVIES NLP system outperforms the NLP system automatically generated by ELF.

A small test on the MOVIES NLP system was performed at the same time as the COSC110 lab test on databases. Students in the COSC110 Computer Science course were given a lab test on the MOVIES database. The three different test sheets were used as test material for the MOVIES NLP system. The initial results for the system were 5/10 for all three tests. For each test paper the third question (worth 5 marks) required a HAVING clause which could not be interpreted by the MOVIES NLP system. If the text relating to the HAVING clause was removed from the questions, then two of the results improved to 9/10 (1 mark off for having no HAVING clause). The third was not interpreted successfully, and the mark remained at 5/10. Most marks for students were between 5 and 10 out of 10, it would appear that the MOVIES NLP system is comparable to a beginner database user.

	Test 1	Test 2	Test 3
Unaltered text	5/10	5/10	5/10
Having requirement removed	9/10	5/10	9/10

Table 1. MOVIE NLP system results for COSC110 lab test.

While developing and testing the MOVIES NLP system several senior Computer Science students and non-Computer Science students assisted by trying to ask the system different types of questions. Based on these informal evaluations of the system-in-progress, some estimates of the current system's success rates, shown in Table 2, have been made for four types of queries. Each type of question was tested by providing example SQL code for, or explaining what information was desired and having subjects try to obtain a similar result from the MOVIES NLP system. Subjects made between one and five attempts to have the question answered correctly. Overall between 30 and 40 attempts were made for each type of question. Simple one-table queries are handled very well (about 95% success); errors only occur when spelling mistakes or grammatical errors are made. Queries that require a WHERE clause are also handled very well (90% success), with few unexpected errors. Queries with join conditions were not well handled (50% success); many 'nested' references (such as "starring in the movie she is renting") were not in the grammar. GROUP BY queries were slightly better (70% success), although there were still some problems with ungrammatical sentences, and some people had trouble deciding how to word GROUP BY questions.

	Simple, one table queries.	Queries with WHERE clause	Queries with join conditions	Queries with GROUP BY clause.
Success Estimate	95%	90%	50%	70%
Number of Trials	30	35	40	40

Table 2. Estimates of success rates for the MOVIES NLP system.

Note that these estimates can be misleading, as subjects asking the system questions knew that a computer would interpret their words, and saw the results from their last attempt before asking another question. This can make a big difference. Hendrix (1978) found that humans very quickly adapt to the new style of asking questions, once they know what the system expects.

4. Improving General Database NLP Systems

This section begins by reviewing a general database NLP system and identifying some problems encountered by these systems. Then a proposal for an authoring tool to provide a solution to these problems is outlined in Section 4.2 and then an example of the interface to such a system is shown in Section 4.3.

4.1 The ELF System

ELF is one of the commercial systems available for generating database NLP systems for any database. As ELF currently performs better than its other competitors (such as English Query, developed by Microsoft, and English Wizard) this review is primarily on the ELF system.

The ELF system is broadly analysed in terms of the three levels of information discussed in Section 3.5, and is based on information on the ELF web page, ELF Software Co. (1999).

- **Database independent**

  The first level involves English and SQL. ELF performs very well at this level; aggregate functions work well and custom made functions are available to calculate times, dates or lengths of time if necessary.

- **Database structure**

  The second level is information that comes directly from the database. ELF is very good at extracting this information. In ideal situations (where the database elements are well named), setting up an NLP system for a new database involves little more than selecting the database and clicking go. The ELF system operates within the *Microsoft Access* database program, and so can access the database directly. The user may choose whether the ELF system uses the information contents of the database or not (so that "Stanley Kubrick" is recognised as a director rather than a star or movie title). This is an important choice, as any changes to the database will not be recorded by the NLP system, so if this extra source of information is used, then the NLP system may need to be periodically updated. Otherwise the information is not used and the NLP system is less useful.

- **Database semantics**

  The third level of information is the database semantics not stored in the database. ELF provides several different ways to access this information. Some query types are already programmed into the ELF system, and if the database table or attribute names match the examples ELF knows about, then one of these pre-defined query types can be used. For example, some query types that ELF knows about for an order/supplier database are 'order subtotals' (for a sum over various orders), 'ten most expensive products',

'invoices' and 'order details extended' (for calculating sales information). In this way human-entered information can be used in the new database. The only problem is that only some common query types are available – names, numbers, prices, dates, etc. When an unseen or strangely named attribute or table is seen then no database semantics can be guessed. In this situation the user who is setting up the system can record a new type of query. The user can provide example database queries for the NLP system to copy whenever there is a question. This approach is much more difficult for the user because they have to provide the query in the first place. This copy-the-user approach is an advanced form of the approach used by the ELIZA psychiatrist NLP system to enlarge system domain. Unfortunately this results in a very limited system; if there are no pre-defined queries that can be applied, then only the provided types of queries will be used by the NLP system (along with the basic queries generated from the first two levels of information). This almost defeats the purpose of an NLP system, as the user has to enter the answers to questions before they can ask them! The target users for the ELF system are "users that are not familiar with databases" and these people probably will not be able to choose the appropriate queries for the NLP system to work with. Section 4.3 presents an argument for taking advantage of the user's knowledge of English and their knowledge of what is in the database rather than relying on their database and programming skills.

Some additional changes to the ELF lexicon may also have to be made by hand to correct any wrong assumptions that have been made, or to customise terms (such as 'join date' = JDATE). This method causes a problem because users need to know which words they will be using, and customise the lexicon accordingly, before asking any questions. In reality users will only identify words that need to be added when an attempted query does not work properly. For this reason there should be some method to obtain this information from the user before they attempt to use the NLP system. A demonstration that such a system is feasible follows in Section 4.4.

The ELF system combines the three levels of information very well, but not well enough to provide the robust NLP system that would be required for use in SQL-Tutor. A demonstration system provided by ELF on their Internet site will be used to illustrate how the ELF system does not meet the requirements. Three demonstration systems are provided on the Internet site, two are for very small databases and the third is a large, realistic database named "Northwind Specialty Foods" (shown in Figure 5). The Northwind database involves eight different tables containing information about a small food supplier. Information about the company's employees, suppliers, products, shippers, customers and orders are included in the database. Microsoft uses this database to demonstrate the capabilities of the Microsoft Access database program.

There are three main types of problems that can occur in queries to a database NLP system; the following examples are from the ELF demonstration system on the "Northwind Specialty Foods" database, shown in Figure 5.

The easiest form of error to deal with arises when the user spells something wrong, or uses a word that the NLP system is not aware of. The system may simply be able to report which word it did not understand, so that the question can be re-worded.

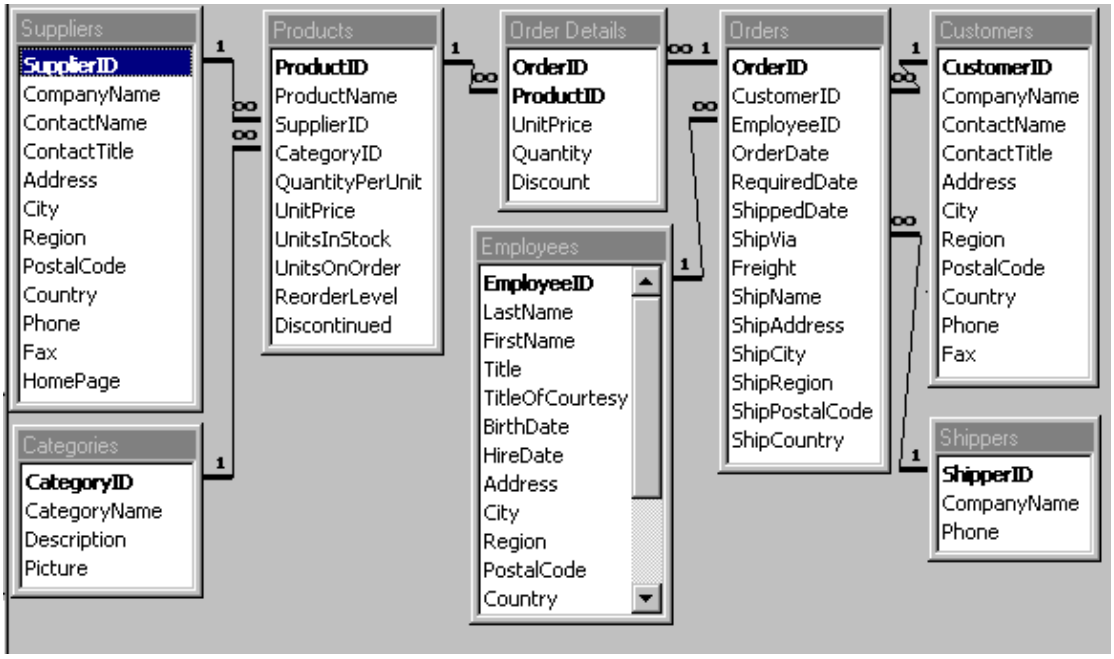

Figure 5. The Northwind Specialty Foods database.

QUESTION: Whar are the employee names and dates of birt?

ELF: SQL = Unable to interpret the question.
Unknown word(s): whar, birt

It is easy to see the mistakes in the question and correct the errors.

QUESTION: What are the employee names and dates of birth?

ELF: SQL = SELECT DISTINCT Employees.LastName, Employees.BirthDate
FROM Employees;

LastName	BirthDate
Buchanan	3/4/55
Callahan	1/9/58
Davolio	12/8/48
Dodsworth	1/27/66
Fuller	2/19/52
King	5/29/60
Leverling	8/30/63
Peacock	9/19/37
Suyama	7/2/63

A more serious form of error arises when the NLP system cannot correctly interpret the question, even though everything is spelt correctly. When this type of error occurs, the NLP system should inform the user that it was unsuccessful and possibly give an explanation, as outlined by Joseph (1991). The ELF system does not do this; it simply gives an error message informing the user that it cannot understand.

QUESTION: For each customer, list how many orders they have and the customer's name.

ELF: SQL = Unable to interpret the question.

This type of error is very frustrating for users, especially if there is no explanation given. The user will probably try several times to reword the question and may never get an answer.

The most serious, and potentially confusing, form of error arises when the NLP system misinterprets the question and gives the wrong answer. This can be very dangerous if the database contains important data or is used to make important decisions, especially when the answer to the question is similar to what the expected outcome was.

QUESTION: What are the ages of the employees?

ELF: SQL = SELECT DISTINCT Employees.LastName, Employees.EmployeeID FROM Employees ORDER by Employees.EmployeeID;

LastName	EmployeeID
Davolio	1
Fuller	2
Leverling	3
Peacock	4
Buchanan	5
Suyama	6
King	7
Callahan	8
Dodsworth	9

The ELF system has incorrectly assumed that 'age' means EmployeeID. This type of error can be confusing, or even go unnoticed. If the Employee numbers were around 20-50 the list may be accepted as the correct ages! If we ask the wrong sort of question we may even get an answer completely opposite to the correct one:

QUESTION: How many employees are over 30?

ELF: SQL = SELECT DISTINCT Employees.EmployeeID, Employees.LastName FROM Employees WHERE (Employees.EmployeeID > 30);

SELECT DISTINCT [elfQ1].EmployeeID FROM [elfQ1];

SELECT [elfQ1].EmployeeID FROM [elfQ1];

SELECT (SELECT count ([elfQ2].EmployeeID) FROM [elfQ2]) AS [Count_Of EmployeeID (Distinct/All)] FROM elfRow in 'C:\PROGRA~1\VBELF\vbelf.mda' UNION SELECT DISTINCT (SELECT count ([elfQ3].EmployeeID) FROM [elfQ3]) FROM elfRow in 'C:\PROGRA~1\VBELF\vbelf.mda';

Count_Of EmployeeID (Distinct/All)
0

ELF responds to this question with a large amount of confusing SQL code, and a completely wrong answer. ELF has again confused age with EmployeeID. As we can see from the first answer that ELF gave us, all employees are over 30. If the question is re-worded we might get a correct answer, but we would only know this needed to be done if the answer is identified as a wrong answer.

4.2 A Proposal For an NLP Authoring Tool

The second goal for this project was to investigate the feasibility of an authoring tool to help automate the creation of database NLP systems and improve their performance. To enhance general NLP systems such as ELF, more ‘database semantics’ information (information about what the database represents) is required in the NLP system. The problem is how to get it, particularly from novice users. ELF provides a method of specifying particular queries which may be useful, and some phrases can be added to customise an existing system, but this is still not sufficient to provide an advanced, robust database NLP system.

The first problem to consider is that SQL can be very difficult to understand for most people, especially if they are unfamiliar with computers and computer programming. This is why SQL-Tutor was designed, and why database NLP is becoming a commercially available product. For this reason we cannot rely on the users of the system knowing how to program SQL related information into the NLP system. This is an option in ELF, and it can be very useful for advanced users, but most will not be able to utilise this feature.

Most users will fortunately be experts on the English language (they probably use it every day!) and they should have a good idea of what is in the database they are using. This all comes very naturally to most people; understanding and talking about the real world is an almost trivial task, communicating with computers is

much more difficult. However this same natural exchange of information, is exactly what a computer system needs to generate a better, more robust NLP system.

Therefore we should try to rely on the user's English skills rather than programming ability, and provide an interface for the user to input this information into the NLP system.

4.3 Authoring Tool Interface

To build an appropriate interface for inputting database semantics information we need to develop a framework for this information. This will allow us to identify the different areas needed and easily obtain the information from the user. The database semantics level is divided into three sections to make classification easier.

4.3.1 Table/Attribute Names

The first thing we need to know about the database is what the tables and attributes represent in the real world. Someone who knows what the database represents, and has a good grasp of English, should find this no problem at all; the task is similar to an easy crossword puzzle. The computer interface, as shown in Figure 6, may be able to help by suggesting synonyms from a thesaurus, or offering pre-defined definitions for some frequently used types. Some tables may not have any real world name. The STARS table, for example, does not represent anything in the real world; rather it shows the relationship between 'stars' and 'movies'.

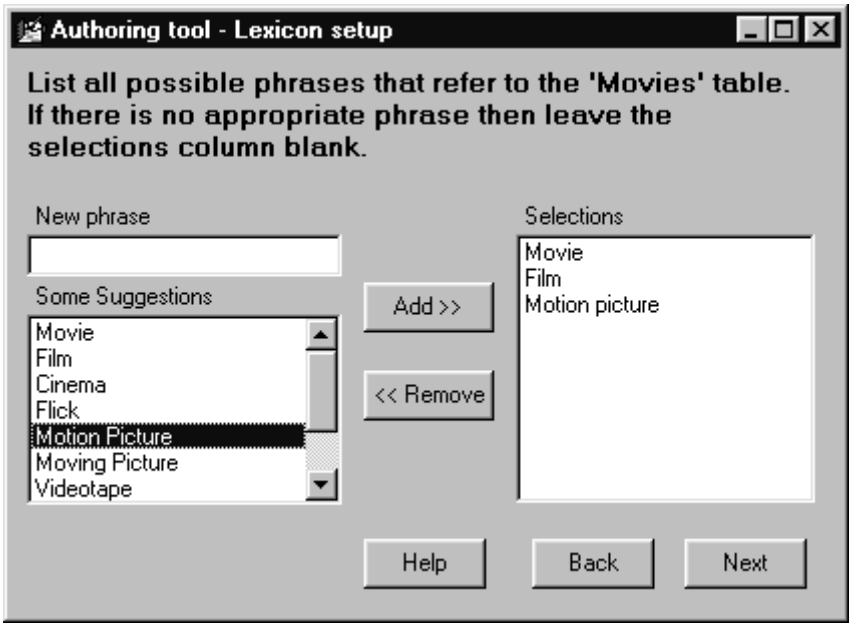

Figure 6. Authoring tool phase one.

In this phase there would be one such window to fill in for each table and each attribute. Entering this information in the grammar would involve converting the selected phrases to their singular and plural forms, and adding them to the grammar. For example the information entered in Figure 6 would be stored in the grammar as follows:

Rules for singular:

```
(MOVIE_S -> movie)
(MOVIE_S -> film)
(MOVIE_S -> motion picture)
```

Rules for plural:

```
(MOVIES_S -> movies)
(MOVIES_S -> films)
(MOVIES_S -> motion pictures)
```

Note that `MOVIE_S` indicates a reference to the MOVIE table, and possible synonyms.

The automatic thesaurus look up may even help to make the NLP system more thorough than a tailor made system. For example, the MOVIES NLP system does not yet include the synonym 'motion picture', but this simple assistance by the authoring tool could help users include all synonyms the first time the database NLP system is created.

4.3.2 Table-Table Relationships

The next piece of information required is how the tables (or the entities they represent) relate to each other. For example, 'the director *directs* movies' 'the movie is *directed by* the director' 'the customer *rents* tapes', etc. There can even be relationships across more than one link in the database; these phrases link tables that are linked by an intermediate table; 'the movie *is being rented by* the customer', 'the star *stars in* the movie'.

As there are many connections that can be made, this process could take a long time. However, most pairs of tables will have no direct relationship, and this can be detected from the database structure level of information, making this section easier to fill in. Tables that have no real-world name (such as the STARS table) can be identified in the first phase and should not be considered here. They may simply be used as a link between two other tables.

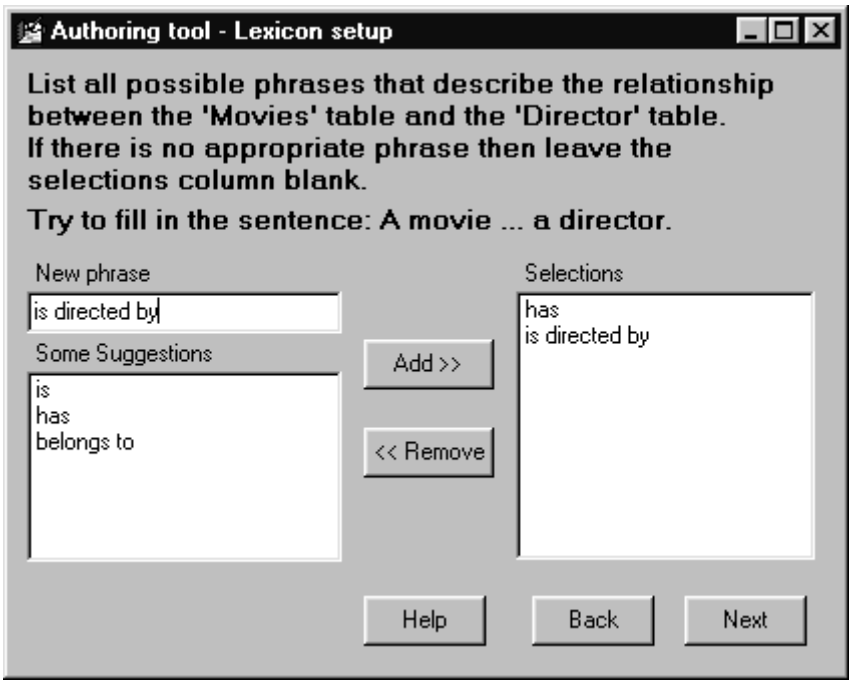

Figure 7. Authoring tool phase three.

After the phrases have been selected, the authoring tool program will enter all variations of the verb forms into the lexicon, the 'is directed by' phrase will become be transformed into the following.

Rules added:

```
(MOVIE-DIRECTOR -> is directed by)
(MOVIE-DIRECTOR -> is being directed by)
(MOVIE-DIRECTOR -> was directed by)
```

A phrase like 'has' can create special entries in the grammar, for example, if 'A movie has a director' is valid then the phrase 'the movie's director' is also valid. The authoring tool will have to be able to do this sort of linguistic analysis on phrases.

4.3.3 Table-Attribute Relationships

There are also relationships from attributes to the tables in which they are contained. For example, 'the movie *receives* academy award nominations' or 'the star *lives in* the city'. Attributes might even relate to other tables, e.g. 'the star *plays* the role'. Attributes that are foreign keys should not be considered in this phase, because they link tables to other tables, and the appropriate phrases should have been covered by the table-table relationships.

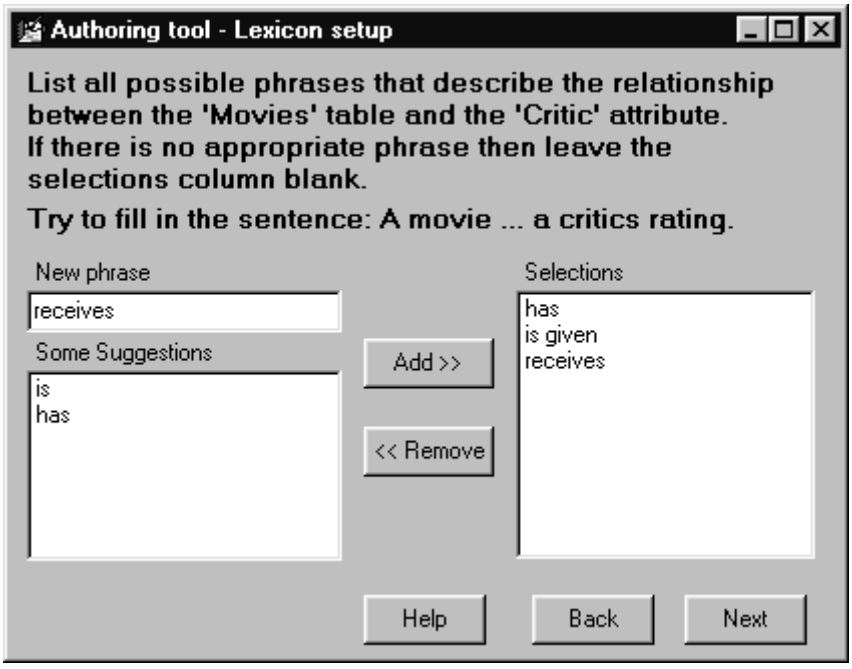

Figure 8. Authoring tool phase three.

Note that ‘Movie’ and ‘critics rating’ are the phrases that were identified for tables and attributes in the first phase. This step will require one set of phrases for each attribute.

In a similar way to the table-table relationships the authoring tool will do a conversion of these phrases to different verb forms and fill in the grammar rules accordingly. The phrase ‘receives’ would be altered to create the following rules:

Rules added:

```
(MOVIE-ATT_RATING -> receives)
(MOVIE-ATT_RATING -> is receiving)
(MOVIE-ATT_RATING -> receiving)
(MOVIE-ATT_RATING -> received)
```

The entries for ‘is receiving’ and ‘receiving’ may not make much sense, as the rating was received in the past, but it is important that questions such as “list any movie receiving a rating higher than 3” can be accepted.

These three phases require the user to perform a large amount of work. Even with the assistance that can be provided by the program, in the form of thesaurus look-ups, suggestions from pre-defined sets of phrases, and referring back to previously defined terms, setting up a database NLP system in this way could take a long time. For large databases (with many tables and attributes) using such an authoring tool to set up an NLP system may take several hours. However there is no easy way to avoid this. The information has to come from somewhere, as the computer program cannot guess it all correctly, and the only other source of information is the program user. Fortunately the process only has to be done once to create the database NLP system.

For some databases there may still be some database semantics information that is not covered by these three phases, but most of the third level information can be obtained in this way, resulting in a much more robust and useable NLP system. Because the three sections of the database semantics information are covered in their entirety, this method may even improve on the tailor-made approach to NLP design by identifying some missing information.

The implementation of such an authoring tool would require a lexical database, so that plural and verb conversion can be performed on the phrases entered by the user. A database such as LAD (Language access database) designed by Zickus (1995) would contain the relevant lexical information. The database semantics obtained from this process could be stored for later integration into a system such as ELF, or an NLP system could be generated directly from the authoring tool. The ELF system performs very well at the ‘database independent’ and ‘database structure’ levels so integration into a system like this would save much of the initial system implementation costs.

5. Conclusion

User interfaces are a very important part of Computer Science research. They define the way computer programs are perceived by users, and contribute to the overall worth of the program. An interface enhancement for SQL-Tutor has been proposed to allow new problems to be added, simply by typing in the English question by using a database NLP system. Using a general database NLP system would allow both new databases and new questions to be added to the SQL-Tutor with minimal user effort.

Natural Language Processing can bring powerful enhancements to virtually any computer program interface, because language is so natural and easy to use for humans. The Intelligent Tutoring System, SQL-Tutor is no exception. Alternatives for integrating a database NLP component into the SQL-Tutor ITS were considered and assessed. The goal of this project was twofold. The first goal was to create a tailor-made NLP system for the MOVIES database. The second goal was to consider the possibility of developing an authoring tool for creating database NLP systems. Both goals were investigated with respect to the requirements of SQL-Tutor. Because of the advanced nature of the problems in SQL-Tutor, advanced questions not usually handled by NLP systems would have to be successfully interpreted.

SQL-Tutor could be enhanced with four database NLP systems (one for each existing database). These NLP systems would have to be able to handle advanced forms of questions. At present this means that the database NLP systems would have to be tailor-made for each particular database. For the four databases currently in SQL-Tutor this would require a significant amount of work, and additional databases would require another NLP system to be created. General database NLP systems offer a much more flexible solution, allowing databases to be added relatively easily, yet still offering the same level of NLP support. However, general systems such as ELF are not yet good enough to achieve the required level of performance.

After pursuing the first goal, the MOVIES database NLP system is currently capable of handling simple queries, standard join conditions, aggregate functions and GROUP BY clauses. Because not all forms of SQL queries are supported, further development would be required before the system can be used within SQL-Tutor. Three levels of information were identified within the grammar for the MOVIES NLP system; database independent, database structure and database semantics. In some areas the MOVIES NLP system already performs better than the ELF demonstration system does, due to the additional database semantics that have been programmed into the MOVIES semantic grammar. This information is

virtually impossible for any general NLP system to extract from the database alone.

In pursuing the second goal, it has been argued that the method used by the ELF system to obtain third-level information is flawed because of the difficulty involved for the users, and the inadequate results achieved. An authoring tool was proposed to overcome this problem by targeting the user's knowledge of English and what the database represents rather than their programming abilities. If implemented, such an authoring tool would have the potential of creating a robust NLP system able to meet the requirements for use within SQL-Tutor.

Future work on the authoring tool would involve an evaluation to identify whether the database semantics could be obtained from casual users familiar with the database if they had a small amount of on-line help. Otherwise, a more experienced user, familiar with programming and/or linguistics would be required to supply the various phrases relating to the database.

Once the MOVIES NLP system can be used with SQL-Tutor, an evaluation of the benefits of tailor-made systems could be conducted. Such an evaluation could help to decide whether to create a tailor-made system for each of the databases within SQL-Tutor, or whether the additional benefits and convenience of a general NLP system would be required to justify making such additions to SQL-Tutor.

While there are still issues to be resolved for a complete integration of an NLP system with SQL-Tutor, these results show that it is a possibility. Further NLP enhancements could extend the proposed initial database NLP proposal. An SQL-to-English system could be designed to describe why a student's answer is incorrect in more detail than the existing hints. A dialogue system could be implemented to enable students to carry out a conversation with SQL-Tutor to discuss how to solve a problem or to ask for more information about SQL in general.

Acknowledgments

I would like to thank Tanja Mitrovic for her assistance throughout this project and Jane McKenzie for proof reading the report. Also thanks to Tania Shuker and the 1999 Computer Science Honours students for their moral support throughout the year.

6. Bibliography

Akama, S. (Ed.) *Logic, language and computation,* Kulwer Academic publishers, pp. 7-11, 1997.

ELF Software CO. *Natural-Language Database Interfaces from ELF Software Co,* cited November 1999, available from Internet: http://hometown.aol.com/elfsoft/

Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., Slocum, J. "Developing a natural language interface to complex data", in *ACM Transactions on database systems,* 3(2), pp. 105-147, 1978.

Joseph, S.W., Aleliunas, R. "A knowledge-based subsustem for a natural language interface to a database that predicts and explains query failures", in *IEEE CH,* pp. 80-87, 1991.

Mitrovic, A. *A knowledge-based teaching system for SQL,* University of Canterbury, 1998.

Moore, J.D. "Discourse generation for instructional applications: making computer tutors more like humans", in *Proceedings AI-ED,* pp.36-42, 1995.

Russell, S., Norvig, P. *Artificial intelligence a modern approach,* Prentice Hall, New Jersey, Chapter 23, pp. 691-719, 1995, *Language* sub-system source code available from Internet: http://www.cs.berkeley.edu/~russell/code/doc/overview-LANGUAGE.html

Suh, K.S., Perkins, W.C., "The effects of a system echo in a restricted natural language database interface for novice users", in *IEEE System sciences,* 4, pp. 594-599, 1994.

Weisenbaum, J., "ELIZA: A Computer program for the study of natural language communication between man and machine", *Communications of the ACM,* 9(1), 1966.

Whenhua, W., Dilts, D.M. "Integrating diverse CIM data bases: the role of natural language interface", in *IEEE Transactions on systems, man, and cybernetics,* 22(6), pp. 1331-1347, 1992.

Winnograd, T. *Understanding natural language.* Edinburgh University Press, 1972.

Woods, W. A. "Progress in Natural Language Understanding: An Application to Lunar Geology", in *AFIPS Conference Proceedings* 42, pp. 441-450, 1973.

Zickus, W.M., McCoy, K.F., Demasco, P.W., Pennington, C.A. *A lexical database for intelligent AAC systems.* RESNA Press, 1995, Available from Internet: http://www.cis.udel.edu/~mccoy/publications/1995/ZickMcCo95.ps