

IMPLEMENTING A NATURAL LANGUAGE TO  
STRUCTURED QUERY LANGUAGE TRANSLATOR

A Thesis Submitted  
in Partial Fulfillment  
of the Requirements for the Designation  
University Honors

Kevin J. Shannon  
University of Northern Iowa  
May 2011

This Study by: Kevin J. Shannon

Entitled: Implementing a Natural Language to Structured Query Language Translator

has been approved as meeting the thesis or project requirement for the Designation

University Honors

\_\_\_\_\_  
Date

\_\_\_\_\_  
Eugene Wallingford, Honors Thesis/Project Advisor

\_\_\_\_\_  
Date

\_\_\_\_\_  
Jessica Moon, Director, University Honors Program

I would first like to thank my Honors Thesis Project Advisor, Dr. Eugene Wallingford, for his guidance, motivation, and creativity while embarking on this journey.

Gratitude should also go to my roommates, especially fellow Honors Student Katelyn Haw and Michelle Cross for their patience as they deal with my stress and poor punctuation.

Last, but not least, I would like to show my appreciation to my parents, Kerry and Kathy Shannon, and my brother, Adam UNI '14, for raising me to be the person I am today!

## **Introduction**

Since the beginning of computing, humans have been trying to make computers understand each other more completely. Humans have always had difficulties encoding their thoughts and computational problems and in turn, computers have had difficulty understanding the subtle nuances of the human mind. The first computers were nothing more than large calculators, but as time went on, they became become more sophisticated. Various technologies and techniques have been created that add features and capabilities to modern computer systems, and many of these will be discussed in this paper.

This paper describes my Honors Thesis project to design and implement a natural language to a structured query language translator. This system is being implemented in an academic environment, so its domain will match the types of data and queries it will be able to understand. After the source review, this paper will describe the system in more detail and reflect on the development process. This paper concludes with appendices including screenshots, source code, and a language definition.

## **Source Review**

The first attempts at creating systems with natural language access to a database were mostly unsuccessful and very constrained. A computer scientist at the University of Illinois created a system to query a military maintenance database for information about particular maintenance requests (Waltz, 1975). Questions had to follow a very specific format, like “Were/Was <maintenance action> performed/done on/for <specific plane>?” About twenty questions could be asked of this system, but each question format had to be hand coded before it was going to be used and the system was unable to recover with grace if it encountered a question it did not understand. Although the system used a constrained form of English which is

more user-friendly than a structured programming language, the system was not very flexible. The author noted that this approach to “language parsing” is not really parsing because “[many] words are defined implicitly within each pattern in which they occur in the pre-stored network. For example, the system does not require subject-verb number agreement” (Waltz, 1975, p. 871). However, the author did build artificial intelligence (AI) into other parts of the system. The system can infer the meaning of certain phrases and ambiguous terms like dates and ranges. This system did not perform well with new types of queries because each query was hard coded in the computer as was the data format.

A second attempt at creating “an intelligent interface that provides natural language access to a large body of data” was investigated by four new researchers at the Stanford Research Institute, later known as SRI International (Hendrix, Sacerdoti, Sagalowicz, & Slocum, 1978, p. 105). Building on prior research, these researchers clearly knew that the focus of their investigation was to build a system that could extract specific data from databases without employing the services of a technician. Avoiding a technician is important because most of his or her time is spent comprehending the request and creating the necessary extracts and linkages from knowledge of the system and data store. The researchers cited the needs of business executives and other decision makers for current, accurate information as the reason for creating a new system (Hendrix et al., 1978, p. 106). This particular business need is common and will be mentioned in the context of other future systems.

Although the researchers wanted to build a cross-discipline system, again their system had a Department of Defense focus. This time, the researcher’s system, called LADDER, extracted data for Navy command and control users. Hendrix et al. (1978) described the architecture chosen for this problem. They linked multiple computers and computer files to be

able to extract data from certain locations within those files. Then, the system of record responded with the answer to the query, returning only the data requested by the query. Fortunately, LADDER worked well and its pieces were compartmentalized enough so the designers could replace one piece as better technology developed.

In addition, the researchers from SRI International (Hendrix et al., 1978) touted LADDER because if an incomplete input was given, the system would infer that the user simply wanted an answer to the previous question with a changed data set or changed data piece. Hendrix et al. described a conversation between a human and computer as follows: “First the user would enter ‘What is the length of the Constellation?’ LADDER would respond with the length of the ship. Then the user would input ‘Of the Nautilus?’ The system would infer that the complete second question was ‘What is the length of the Nautilus?’” (1978, p. 109). The system could also correct minor spelling errors. Unfortunately, the language used for data querying was “Datalanguage,” which is no longer the popular method of input for querying databases. The natural language component of this system also could only recognize a limited number of words because of its small vocabulary.

A large part of the ability to recall data from a database is based in part on E.F. Codd’s research into databases. Codd wrote a series of papers in the 1970s about data storage and normalized data. One paper in particular included a discussion about relations (how complex three-or-more dimensional data should be stored in a two-dimensional table) and relational calculus (how the data can be used and aggregated to answer specific questions) (Codd, 1970). This paper also specified which operations all companies’ database implementations should include. All of this is important to the field of natural language processing and database querying because it specified a generalized form for querying that many database systems have

chosen to implement. This allows new application developers to create a generalized program that will most likely work with all systems one will encounter. Many of the major database systems including IBM's DB2 (Chamberlin, 1998) and Microsoft's SQL Server cite Codd's paper when discussing their implementations (IBM Archives: Edgar F. Codd, 2003).

A number of advancements in understanding natural language have made developing the solution to this problem easier and made the solution itself more similar to the linguistic complexity of English. A system named "User Specialty Languages" was created in which queries were split by word and identified by parts of speech (Lehmann, 1978, p. 560). USL assumed that its users would be experts in their field, but not experts in the underlying database or data constructs. This system sent a user query to separate parsers and interpreters. The parser was taken from another project, but the interpreter was custom designed and developed to cover popular questions in three languages. Some concepts were added for the ease of translation and a few more concepts were added because of the specific nature of the system to be developed. This USL system was able to handle time inputs and use them for data comparison. It also caught plural nouns and used both a formal language interpreter and a world view of the data, as defined as a bit of knowledge about what was around the system, such as what data was available, to create database queries (Lehmann, 1978, p. 566). When USL was developed, it was the most complex and the least restrictive, but other systems would soon come to overtake it in terms of complexity.

A second great advancement in understanding natural language came when researchers worked toward the goal of understanding language in its current state rather than translating it into a set of rules. Again, the researchers built a natural language front end called a "Discourse Module" and connected it to a logic and data back end (de A. Barros & DeRoeck, 1994). The

natural language front end is built on top of another front end called SQUIRREL. In this instance, the front end of the system “takes the input sentence, producing syntactic and semantic representations, which it maps into First Order Logic” (de A. Barros & DeRoeck, 1994, p. 120). First order logic is a type of predicate logic where sentences can be represented as facts, objects, and relations. These sentences can either be true, false, or unknown (Schafer, 2010). In this case, the natural language front end was responsible for transforming queries into models understandable by the database. The back end uses the logic-formatted query and domain specific calculus along with Codd’s tuple-relational calculus to create a SQL query which is sent to the database for a database answer. The front end addition to SQUIRREL was developed in 1994, and it seems to work well because the creators allow for removal of ambiguity in at least three stages of the data transformation process.

It can also be noted that significant research and feature compilation was done by a research team in the paper “Natural Language Interfaces to Databases – An Introduction” (Androutsopoulos, Ritchie, & Thanisch, 1995). This detailed paper gave a great general overview of natural language interfaces to databases. The researchers continued to discuss advantages and disadvantages to building such a system and linguistic problems encountered by individuals building similar systems. One important distinction made by the authors is that the users often expect the systems to be intelligent or to possess some form of artificial intelligence. Unfortunately, this type of database lookup can often be performed through straight English for Structured Query Language, or SQL, translation. Many artificial intelligence functions can be performed on a query. Functions are unnecessary and the problem can be solved by simply writing an algorithm that requests additional information from the user if the query is unclear. Similarly to the SQUIRREL system, this system parses a query using English semantic grammar.

Grammar categories such as noun, pronoun, verb, and adjective are pulled from the query.

Parsing the query in this way will solve some of the primary problems in interpreting English speech.

A recent paper by researchers in Sweden built from Codd's calculus and current language parsing techniques; from there, they implemented a design for a natural language interface for geographical data (Minock, Olofsson, & Naaslund, 2008). Minock et al. (2008) described how relations between data elements are translated from words like "with" or "in" into union statements between tables in the database. When queries have union, intersection, and other set operations, they can easily be placed over top of a database to get a response with certain limiting criteria. These researchers also focused on solving two common and complex problems: noun phrases with complex clauses and choosing the right data. Again, by implementing parsing trees, the developers were able to parse a complex clause into simpler yet still difficult parts of speech. The complex problem with choosing the right data was solved by creating a user interface that allows users the ability to choose which data they wish to recover. A click and drag interface was created to drill down and allow the users to select particular data values from tables. Users were also allowed to choose from a number of sample queries like "How many \_\_\_ are \_\_\_" where users of the system filled in the blanks. This made the system much easier to use, but the programming time to convert databases into webpages was complex. Another interesting note is that by this time, other approaches to parsing the geographical database in question had been discovered, developed, and documented in scholarly journals. Minock, Olofson, & Naaslund found that their method of user interface string selection performed better in tests when compared to methods including string entry where the system filled in blanks based on information it had previously learned (machine learning).

A small number of inference engines are currently being developed to answer questions that would have been sent to natural language interfaces. One such inference engine is the Wolfram|Alpha computation engine. Wolfram is the company best known for its Mathematica software program for computing, visualizing, and allowing use of special math algorithms. Wolfram took the computing power of Mathematica and the knowledge of the Internet and put it behind a familiar search engine interface. The engine can calculate the solution to a number of different types of equations including most that can be drawn on a graphing calculator. Because of the type of display used (computer monitor), the website also draws 2-D and 3-D graphs of any equation entered. However, the developers cited the computational power when applied to data as being the feature that sets this engine apart (Gray, 2009). Gray said that computation matters “Because computation is what turns generic information into specific answers (2009, para. #5).” A sample query of “undergraduate university of northern iowa iowa state university university of iowa” creates a table with the number of undergraduate students at each state university. Computations can be done for chemical, biological, geographical, socioeconomic and web page data sets as well as for dates, times, money, and finance data. In short, this website will become every math and science student’s best friend.

A second computational engine developed earlier this year battled contestants on Jeopardy this year. A system named “Watson” was developed at IBM (Thompson, 2010); this question-answering machine is the smartest of its type in the world. Watson will do more than Google’s simple return of a list of websites where the answer is likely to be found. Watson has actually read and stored over 10 million sources and will return a Jeopardy-style question to a Jeopardy answer. IBM was also the company that built the machine that took on Chess champion Gary Kasparov. IBM is building this machine to run on one of its one million dollar

super computers. To build the machine, fifteen scientific team members were given three to five years. In just three years, they have developed a system that can signal the buzzer and give an answer in the game of Jeopardy in just six to eight seconds. However, Watson only buzzes in when it feels confident in its answer (Thompson, 2010).

Watson builds confidence by solving the problem multiple ways, literally using multiple algorithms. When two or more responses come back with the same answer, the system builds confidence in that answer. IBM's scientists wanted to build a computer that was not limited by the type of data it was to be an expert on. Instead of building a custom natural language interface to a database, IBM built a system that can understand knowledge from all disciplines. Watson has stored information from 10 million articles and is not connected to the internet when it plays Jeopardy, much like the human contestants (Thompson, 2010). Surprisingly, the computer has read enough cultural sources to do well at Jeopardy. IBM hopes to eventually market this system to law firms that need to sift through piles of legal cases or medical teams that need more experienced and knowledgeable sources.

### **Summary of Work Completed**

I am using the knowledge gained through the Source Review to design and implement a natural language to structured query language translator. This translator embodies many of the features seen in other previous systems. It includes a parser that looks for record types and attributes or adjectives about those records. It also includes a method for the user of the system to indicate they would like a mathematical calculation performed or results grouped by an adjective. These are all features which were given as examples in other systems previously built by researchers and scientists.

The purpose of my system is to supplement the work of an office like Institutional Research on the typical college campus. The University of Northern Iowa (UNI) can be used as an example institution; the Office of Institutional Research (IR) receives requests daily for information that is available in the University Fact Book or on the IR website. My system is intended to supplement the services provided by offering educational users a search box to input their requests. A system like this is intended to only answer the most basic of questions proposed to an office like this. More complex questions may work and this type of interaction is recommended in future projects.

The translator takes a string of words as input and outputs a data structure that can be understood by another module. The input string can consist of letters and the space character. Other characters, like the ampersand, question mark, and parentheses might be useful and helpful at a later date, but they are not helpful at this time. Incorporating these special symbols would require extra programming and the meanings of these symbols can be covered by another phrasing of the input string.

Because computers are logical systems which only take facts into account, they cannot answer subjective questions very well. For this reason, this system is not designed to handle subjective questions. Databases are data storage buckets stored with facts, so it would be difficult for a computer to guess or weigh two options unless it was somehow trained to do so. I can envision a computer asking what the next value would be in a series of 5 values, but other than that, questions about attitude and opinions should be left to the humans prepared to answer those questions.

Most human users will input a string in the form of an interrogative clause which the system will interpret and respond to. An interrogative clause is a category for all types of

questions. Specifically, this system is designed to answer non-polar questions, such as who or what or how many. The system uses the selection criteria at the end of the string to limit, sort, and group responses rather than using wh-clauses + adjectives as in how big or how long (Trotta, 2000). In addition, the system does the best with queries that can be easily answered by using the data located in the databases it is connected to.

In order to limit the quantity of variations of the English language that would be accepted by this system, a very usable subset of English was chosen for use in interacting with the system. A full description of this language can be found in Appendix A. For the most part, this custom subset of language matches common syntax for asking a wh-clause question in English. This type of clause has a wh-word, like whom, what, or where, followed by an assertion or fact about a person or group of people. This assertion can come in the form of a statement about a particular group of people or a comparison of a group of people to an arbitrary base. For example, an assertion can be “has a declared major in the College of Education” or “has a grade point average greater than 3.5.” In either scenario, the language subset should be able to handle both.

One inspiration for this project is the way that Wolfram|Alpha, the computational knowledge engine, parses an input string to look for comparisons or mathematical equations. As far as can be seen without decomposition or reverse engineering, the system looks for proper nouns and special words that signify relationships between these words. This style of decomposition caused the development to start with a simple mix and match grammar and expand to include relationship words. At this point, words implemented include “less than”, “greater than”, “and”, “or”, & “not”. These powerful words are translated using the same phrasing and in the same order as they are written.

A second part of the translator takes the output data structure of the first module and converts it into a computer query that can be run against a database. The intermediate data structure resembles SQL, but not in its entirety. SQL has a number of different manufacturer variations depending on the features built into the particular system. Most companies have implemented a core set of SQL keywords and these are the ones that are implemented in this translator. Select, count, sum, average, where, sort by, group by, and a few others have all been implemented in this translator. Uncoincidentally, these are some of the most common questions asked by users of academic or demographic data.

Manipulation of the data is expected to be done by the database management system, or DBMS. Because the goal of this project is to create a translator and not a data manipulation or Extract-Transform-Load tool, also called an ETL tool, the data manipulation is not handled inside the translator. An ETL tool is used to copy or move data from one database to another. An ETL tool usually must change the data formatting because the receiving data layout is usually different. ETL tools are very popular in data warehousing. Any data manipulation that is needed is handled inside the database query. Using keywords like group by, sort by, average, and sum, some basic calculations can be initiated from the SQL statement alone. The requirements for the project did not dictate the creation of any additional keywords beyond those implemented by the DBMS.

My system is built so that it can be later expanded into a website or other input/output system. The initial idea for this system is that it would be part of a larger database retrieval system. An input for the translator would come from a search box similar in look and feel to Google's home page. The string would be processed and results downloaded from the database. The results could be sent back to the browser including a table if needed to show a set of results

or simply a number if such a number will answer the query. It would also be useful to show the user how his or her string was translated. If a word or phrase isn't being recognized, perhaps the user will want to rephrase or reword his or her request so that it is properly understood.

The system that has been developed is being called a translator, but it could just as easily be called a compiler. The definitions of the two terms are unclear, especially with regard to the differences between them. A course on Compilers and Translators at Virginia Tech (Virginia Polytechnic Institute and State University) uses each of the terms in the definition of the other (Lee, 2001). A compiler is defined in this instance as a “program that translates between programming languages” and a translator is a “[thing] that changes a sentence from one language to another without change of meaning (Lee, 2001, para. callout box).” In my point of view, a translator is a linguistic term and a compiler is a computer science term. Both fit this project, but a translator is a more recognizable term for non-scientists, so that is what I have chosen to use to describe the project.

I decided early on that this system would not try to use machine learning or artificial intelligence to understand queries. This may be appropriate for some projects, but the goal of this project is to build a translator. This includes the ability to interpret the data structure and schema of a target database in order to import its settings without manual intervention. Each database only has to be set up once and it can stay that way until the underlying table structure changes or the data mappings change.

### **Reflection**

Overall, I am happy with the outcome of this project. I think that it was a valuable exercise in both computer science and linguistics to learn how people ask questions for this subset of data and understand what they are looking for. I am a numbers person and I like to

remember certain random facts about UNI. Not everyone has this ability or interest, so I wanted to make it easier for people to be able to access this data. In addition, during two internships I completed with State Farm and Principal Financial Group, I saw how hard it was for everyday people to access data. Those who did use data on a regular basis were very technical business people or they had to have other more technical people prepare the reports on their behalf. I wanted to solve this problem by making an easier interface for these one-off data requests or curious people to use.

An analytical and data query system is important, because businesses across the country are struggling to find ways to get meaning and use out of their data. A recent Burton Group report showed that 60% of responders state they do not have the ability to get good analytical data from their data stores and data warehouses (Santos, 2009). In addition, that same report showed that business intelligence is critical to virtually all Fortune 500 organizations. A number of offerings from companies such as Business Objects and Microsoft exist to help solve the problem of insufficient data, but only the largest companies have the budgets and manpower needed to implement these solutions.

Even if a company has the ability to purchase these products, they are often difficult to use. During my two internships, I heard stories from IT customers about the software products they use and the problems they had in making them work. It would be nice to use a program that answered a simple question like “How many contracts will renew in the next 7 days?” No knowledge of database structure or computer programming would be necessary to make this application work.

I was able to create a translator that recognizes many of the common nouns and adjectives in the higher education domain and translates them into a pseudo-SQL statement.

Eventually, after receiving schematics from Information Technology Services, or “ITS”, I hope to fully translate the pseudo-SQL into usable SQL that will return data if connected to a database. After meeting with an individual in Institutional Research, I learned that many of the common questions that are asked are simple enough that they could possibly be answered by a program like this one. I also learned how questions are asked and refined. I used this information to develop the system further by encoding the ability to include ANDs and ORs in the selection criteria section of the system. The system I created looks specifically for these types of questions so it can answer them based on the information about the data encoded into the system.

Although other systems of similar type and function have been created in the past, I believe this system is unique because of the domain in which it was implemented and the speed at which it was created. First, I believe this system was unique because it was built for an academic environment. Most systems would be built for military or commercial customers. A similar system to mine could be developed for a class assignment, but I don not think that it is likely due to the large size of the project compared to the typical coursework given in an undergraduate computer science course. Second, this system is unique because it was developed by one person in a matter of months, not years. The first systems used complex algorithms and were not flexible; this system is more flexible because it will work with a varied sentence structure.

My system could later be modified in order to handle business environments and information in addition to higher education environments. I could easily see an expansion covering business customers or the products offered. A few simple modifications would need to be made to the proper nouns and a few other terminal words, or words that are directly

translated, like record types. In addition, each database implementation is different, so further modifications would have to be made in alignment with those table layouts.

From this point, I feel there are a number of avenues where this project can lead me. I would eventually like to connect my system to a web form and a SQL database with scrubbed student information; this would prove that the system works. Furthermore, I think that the system can be built by one person in a matter of weeks, because the computing technologies have advanced significantly in the past number of years. I would also like to implement a number of the features discussed in this reflection. First, I could implement the parenthesis and bracketing for understanding the “not phrases.” Second, I would like to expand this to include as many terminal phrases as possible. This may mean additional database work to extend the data available to the system. However, this may also be completed by understanding the data more completely. It is hard to build a system without a framework or requirements document to start with. Finally, I would want to test this more comprehensively by bringing in test users from all over the university to try out the system. This would give me an opportunity to explain computer science and the ways in which computers can help us in everyday life. This would also give me a test set of questions to run against the system.

I believe this project, if implemented, will help the university because it will provide a faster turnaround for secretaries completing surveys for outside entities and reports for other UNI offices and the Board of Regents, State of Iowa. The current process requires staff to fill out a form online or call the research office and request information. Then the office has to verify that the recipient is authorized to receive the information and ask clarifying questions about the request. After this is done, the office sends a programmer to find the information. This whole process can take a number of days. By automatically allowing certain specific queries to be

automatically answered, this process can be sped up. For the same reasons, this project also has the potential to free up some staff in the research offices around campus from answering simple questions that a computer could answer.

This project advances the field of computer science by showing how a system that used to take years to implement can now be implemented in a matter of months using updated programming techniques. Included in this project are my class diagrams and source code, in order that others can start where I left off. They may choose to extend this project to match their environments or add additional features that I did not think of. I also think my language definition can be used by others so they do not have to create their own for their projects involving SQL.

Throughout this project, I enjoyed learning about the process to make a compiler, including choosing a language to compile. I had taken a class in developing a programming language, but not one in making a full compiler. I learned the best way to organize the methods inside this program. There are a few major ways to run a compiler, and I learned about and implemented the one that made the most sense for my project. I also had the opportunity to spend some time working in the field of linguistics to understand what a question was to linguists, what was included in a question, and how to process the individual parts of a question. Overall, I have really enjoyed experiencing the process of development on my own and I look forward to designing new systems after the completion of this project.

## Appendices

### Appendix A

#### Natural Language Query – Language Definition

<database selection> ::= <selector interrogative> <selection> <record type> (“not”) <selection criteria> { <conjunction> <selection criteria> };

<selector interrogative> ::= “how many” | “how much” | “average” | “sum” | “who” | “which one”;

<record type> ::= “student(s)” | “major(s)” | “course(s)” ;

<selection criteria> ::= [<selection> “in” <value> ] | <selection> <selection comparison> <value> | <grouping criteria>;

<selection comparison> ::= “greater than” | “more than” | “less than” | “smaller than”;

<value> ::= a numerical value which fits the range of the comparison domain. For example, 0.0 to 4.0 is in the range of acceptable values for comparing GPAs to. This can also be a proper noun or code number that matches the specific data stored

<selection> ::= <grade> | <college> | <department number> | <course number> | <section number> | <credit hours> | <cumulative GPA> | <cumulative UNI GPA> | <session number (20081, 20113, etc.)> | <residency> | <degree> | <major sought> | <minor sought> | <academic characteristic> | <ethnicity> | <generic record>;

<grouping criteria> := “group by” [“major” | “course number” | “ethnicity” | “college”] | “”;

<conjunction> ::= “and” | “or” | “and not” | “or not”;

## Appendix B – Sample Interactions

User input is color coded in gray between “Enter the query:” and “QueryRepresentation”

### Scenario 1:

Enter the query:

How many student records are in the College of Social and Behavioral Sciences?

```
QueryRepresentation [originalString=How many student records are in the
College of Social and Behavioral Sciences?, cleanString=how many student
records are in the college of social and behavioral sciences, question=COUNT,
record=STUDENT, firstConjunctionNot=false, selection=[WhereSelection
[attribute=RECORD, value=, numWords=1], WhereSelection [attribute=COLLEGE,
value=S, numWords=6], null, null, null, null, null, null, null, null],
selectionComparison=[null, EQUALS, null, null, null, null, null, null, null,
null], conjunctions=[null, null, null, null, null, null, null, null, null]]
```

```
SELECT COUNT(SIMACTRL) FROM sima WHERE COLLEGE = S;
```

### Scenario 2:

Enter the query:

How many major records are there in the graduate college

```
QueryRepresentation [originalString=How many major records are there in the
graduate college, cleanString=how many major records are there in the
graduate college, question=COUNT, record=MAJOR, firstConjunctionNot=false,
selection=[WhereSelection [attribute=RECORD, value=, numWords=1],
WhereSelection [attribute=COLLEGE, value=G, numWords=2], null, null, null,
null, null, null, null, null], selectionComparison=[null, EQUALS, null, null,
null, null, null, null, null], conjunctions=[null, null, null, null,
null, null, null, null, null]]
```

```
SELECT COUNT(SIMACTRL) FROM sivb JOIN sima WHERE COLLEGE = G;
```

### Scenario 3:

Enter the query:

how many student records are not in the college of business administration and not in college of humanities and fine arts group by major?

```
QueryRepresentation [originalString=how many student records are not in the
college of business administration and not in college of humanities and fine
arts group by major?, cleanString=how many student records are not
in the college of business administration and not in college of humanities
and fine arts group by major, question=COUNT, record=STUDENT,
firstConjunctionNot=true, selection=[WhereSelection [attribute=RECORD,
value=, numWords=1], WhereSelection [attribute=COLLEGE, value=B, numWords=4],
WhereSelection [attribute=COLLEGE, value=H, numWords=6], GroupBySelection
[attribute=MAJOR_SEEK], null, null, null, null, null, null],
selectionComparison=[null, EQUALS, EQUALS, null, null, null, null, null,
null, null], conjunctions=[null, AND_NOT, null, null, null, null, null,
null]]
```

```
SELECT COUNT(SIMACTRL) FROM sima WHERE NOT COLLEGE = B AND NOT NOT COLLEGE =
H;
```

#### Scenario 4:

Enter the query:

How many student records have a gpa greater than 3.50

```
QueryRepresentation [originalString=How many student records have a gpa
greater than 3.50, cleanString=how many student records have a gpa greater
than 3.50, question=COUNT, record=STUDENT, firstConjunctionNot=false,
selection=[WhereSelection [attribute=RECORD, value=, numWords=1],
WhereSelection [attribute=CUM_GPA, value=3.50, numWords=1], null, null, null,
null, null, null, null, null], selectionComparison=[null, GREATER_THAN, null,
null, null, null, null, null, null, null], conjunctions=[null, null, null,
null, null, null, null, null]]
```

```
SELECT COUNT(SIMACTRL) FROM sima WHERE CUM_GPA > 3.50;
```

#### Scenario 5:

Enter the query:

How many student records are in the Honors program and have a gpa greater than 3.50?

```
QueryRepresentation [originalString=How many student records are in the
Honors program and have a gpa greater than 3.50?, cleanString=how many
student records are in the honors program and have a gpa greater than 3.50,
question=COUNT, record=STUDENT, firstConjunctionNot=false,
selection=[WhereSelection [attribute=RECORD, value=, numWords=1],
WhereSelection [attribute=ACADEMIC_CHARACTERISTIC, value=A OR B OR H OR M,
numWords=2], WhereSelection [attribute=CUM_GPA, value=3.50, numWords=1],
null, null, null, null, null, null, null, null], selectionComparison=[null, EQUALS,
GREATER_THAN, null, null, null, null, null, null, null], conjunctions=[null,
AND, null, null, null, null, null, null, null]]
```

```
SELECT COUNT(SIMACTRL) FROM sima WHERE ACADEMIC_CHARACTERISTIC = A OR B OR H
OR M AND CUM_GPA > 3.50;
```

**Scenario 6:**

Enter the query:

what is the average uni gpa for students in the graduate college and uni gpa is greater than 3.5

```
QueryRepresentation [originalString=what is the average uni gpa for students
in the graduate college and uni gpa is greater than 3.5, cleanString=what is
the average uni gpa for students in the graduate college and uni gpa is
greater than 3.5, question=AVERAGE, record=STUDENT,
firstConjunctionNot=false, selection=[WhereSelection [attribute=UNI_CUM_GPA,
value=, numWords=2], WhereSelection [attribute=COLLEGE, value=G, numWords=2],
WhereSelection [attribute=UNI_CUM_GPA, value=3.5, numWords=2], null, null,
null, null, null, null, null], selectionComparison=[EQUALS, EQUALS,
GREATER_THAN, EQUALS, null, null, null, null, null, null],
conjunctions=[null, AND, null, null, null, null, null, null, null]]
```

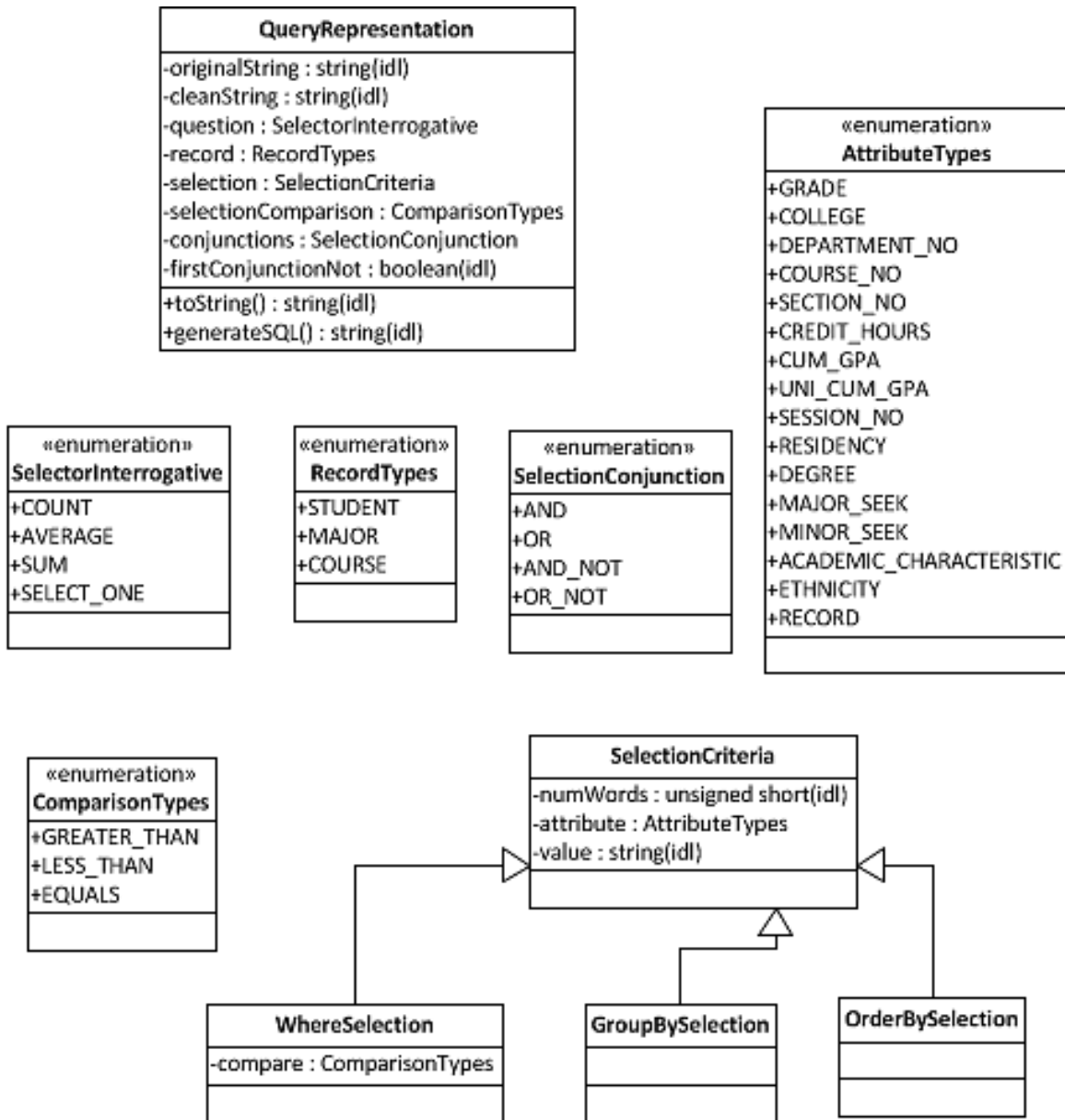
```
SELECT AVG(UNI_CUM_GPA) FROM sima WHERE COLLEGE = G AND UNI_CUM_GPA > 3.5;
```

**Appendix C – Source Code**

Source code for this project is available online at:

<http://www.kevinjshannon.com/thesis/index.html>

### Appendix D – Class Diagram



## References

- Androutsopoulos, I., Ritchie, G. D., & Thanisch, P. (1995). Natural language interfaces to databases – an introduction. *Journal of Natural Language Engineering* , 1(01), 29-81.
- Chamberlin, D. (1998). *A complete guide to DB2 universal database*. San Francisco: Morgan Kauffman Publishers.
- Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM* , 13(6), 377-387.
- de A. Barros, F., & DeRoeck, A. (1994). Resolving Anaphora in a Portable Natural Language Front End to Databases. *Proceedings of the 4th Conference on Applied Natural Language Processing (pp. 119-124)*. Stuttgart, Germany: Applied Natural Language Conferences.
- Gray, T. (2009, May 1). The Secret behind the Computational Engine in Wolfram|Alpha [Web log post]. Retrieved from <http://blog.wolframalpha.com/2009/05/01/the-secret-behind-the-computational-engine-in-wolframalpha/>
- Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D., & Slocum, J. (1978). Developing a natural language interface to complex data. *ACM Transactions on Database Systems* , 3(2), 105-147.
- IBM Archives: Edgar F. Codd*. (2003, April 23). Retrieved October 23, 2010, from International Business Machines: [http://www-03.ibm.com/ibm/history/exhibits/builders/builders\\_codd.html](http://www-03.ibm.com/ibm/history/exhibits/builders/builders_codd.html)
- Lee, J. (2001, April 9). The Basic Structure of a Compiler. Retrieved April 11, 2011, from CS 1104 Introduction to Computer Science: <http://courses.cs.vt.edu/~cs1104/Compilers/TOC.html>

Lehmann, H. (1978, September). *Interpretation of Natural Language in an Information System*.

IBM Journal of Research and Development , 560-572.

Minock, M., Olofsson, P., & Naaslund, A. (2008). *Towards Building Robust Natural Language*

*Interfaces to Databases*. Natural Language and Information Systems (pp. 187-198).

London: Springer.

Santos, J. (2009). *Is Business Intelligence Relevant?* Midvale, Utah: Burton Group.

Schafer, B. (2010). *Session 22: The Powerpoint Slides* [PowerPoint slides] . Retrieved from

<http://www.cs.uni.edu/~schafer/courses/161/sessions/s22/>.

Thompson, C. (2010, June 20). What is I.B.M's Watson? *The New York Times* , p. MM30.

Trotta, J. (2000). *Wh-clauses in English*. Amsterdam: Rodopi.

Waltz, D. (1975). *Natural language access to a large data base: an engineering approach*.

Proceedings of the 4th international joint conference on Artificial intelligence. I, pp. 868-872. Tblisi, USSR: Morgan Kaufmann Publishers Inc.