Text to SQL with Natural Language Processing

Christopher Gauffin, cgau@kth.se
Erik Rehn, erikreh@kth.se

January 10, 2021

Contents

1 Introduction

Grammar is a set of rule which describes the structure of a language. The purpose of the project was get insight into how to design a context free grammar and provide an more intuitive way to access information in a relational database. To do this we created program which translates ”everyday” English into SQL.

2 Background

Creating an translator between two languages have many perks. A translator between two programming languages can help with the migration from one code base to another. A translator between a natural language and programming language can provide insight and reduce labor on a developer. With the growth programming as a whole the need to make it more accessible also grew. The problem of transforming English to a programming language is not at all unique. Converting text into discrete API calls is used all over the place in both virtual assistants and in chat bots. However it is not an easy task at all. Even though programming languages are derived from speaking languages there are many differences between them making it difficult to transform one to another. Many efforts have been made but still the transformation is far from providing a complete working solution. Our project tries to use artificial intelligence to give a more intuitive way to access data inside a database. In this project we convert a subsection of English to SQL(MySQL syntax).

However like previously mentioned the problem we are giving a shot at is not at all unique. Like many before us the first step to making a project like to limit the scope. Since SQL is a vast and complex language trying to create a complete language interpreter would be extremely complex(if not impossible) and the language would not be anything close to how a human would speak making the entire project redundant(since you would have to learn our quasi language). Our restrictions where that we choose to not allow for any mutation of the data, meaning that inserts and updates where not allowed. This limits the possible queries allowing us to focus only on the ”question part” of the query. Even thought this restriction removes many of the operations associated with SQL it still leaves us with the selecting part which arguably are the most complex and useful part. The other restriction was only allowing a single table. In this project we only used a single table since joining in SQL is not how we tend to think about objects.The very structured and table based view on data in a relational compared to the real world would creates a fundamental issue. In this project we choose to completely opt out of that question since that would lead to many more difficult questions about understanding which affected tables are used based on the requested entity. For example using a one to many, one to one or many to many needs different implementations in most databases but we humans tend to not give any thought about which relationship type it is. A question about ”get all pets of the personnel with id 5” would need to be interpreted different based on the structure of the tables in SQL, since it might require several joins.

2.1 Context free grammar

Context free grammar is a set of finite rule which describes all possible strings in a formal language. The set of rules in a language must be finite be can allow for recursion. In this project we use context-free grammar along with NLTK to easily update and add new rules. [1]

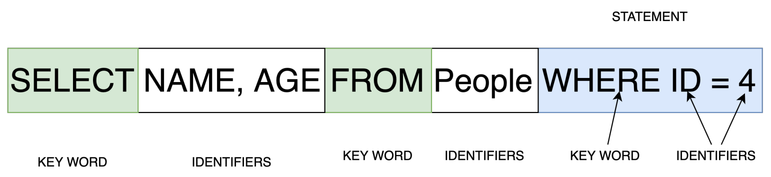
2.2 SQL

SQL, stands for structured query language. It was developed at IBM in the early 1970[2]. The purpose for the language was to provide a way to access to an internal project but as other companies saw the potential it overtime became the standard in many of the relational databases we use today. There are many variants of the language today which provide specific tools for the database. The variants of the language is called dialects, the project is tested on the MySQL 5.7.32 dialect but since the scope of the project is relatively small we are confident that it should work on most modern relational databases.

SQL grammar consists of elements including

- Keywords, are words that are specifically use in the database which have a specific function associated with them.
- Identifiers, which are the names of the database or of which value a certain query is searching for.
- Predicates are statements which are evaluated to be either true or false and decide the scope of the query.[3]

More elements exist in the language, many associated with updating and creating data which is out the scope for this project. Therefore it is enough that we represent a query as the following.



3 Problem

The problem of converting English to SQL have some complications. Natural languages have ambiguities. Word we use in everyday context can mean different things when used in different contexts, have hidden meaning which are not apparent until later on and so forth. The order in which we say them can sometimes be crucial and sometimes not. Generally speaking our language tends to have a lot of things which are not explicitly said but are generally accepted anyways. For example if we are at a restaurant and someone asks ”could you pass me the salt?”, the question is if I could, but the underlying meaning is an action. This is very different compared to programming languages. When it comes to programming languages nothing should be able to be misunderstood or left up to the computer. Every word should have a purpose and an explicit meaning.

4 Requirements

The requirements for the project is the following:

- Implement a parsing grammar that can at least support SQL queries with keywords MAX, MIN, $=$, $<$, $>$, $*$, WHERE, AND, SELECT and FROM.
- Populate the grammar with tags corresponding to the selected database.
- Program a builder function which can construct an SQL query with a parsed sentence tree as input.
- Specify 10 sentences of varying intention and structure and generate a query for each sentence that match the intent of the user.
- Build a test function which takes two input files, the sentences and their respective queries.

5 Implementation

5.1 Method

5.1.1 Libraries

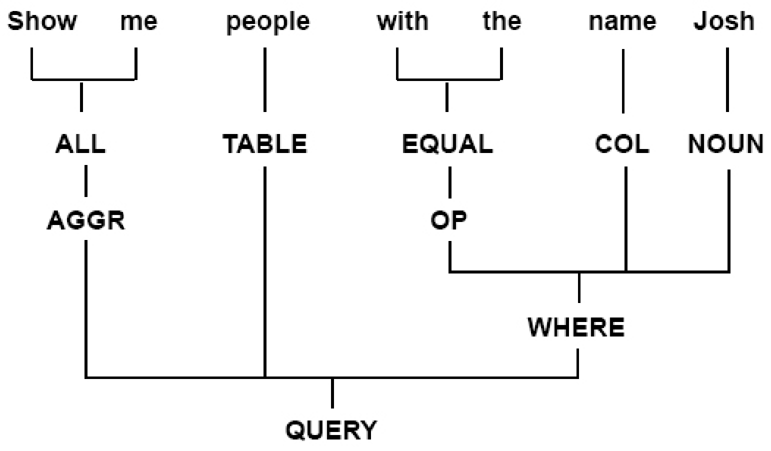
We used NLTK (Natural Language Toolkit) to aid in parsing and tagging sentences [4]. Python and SQL are the only programming languages used in the project. MySQL is the type of database used for storing and querying dummy data.

5.1.2 Testing

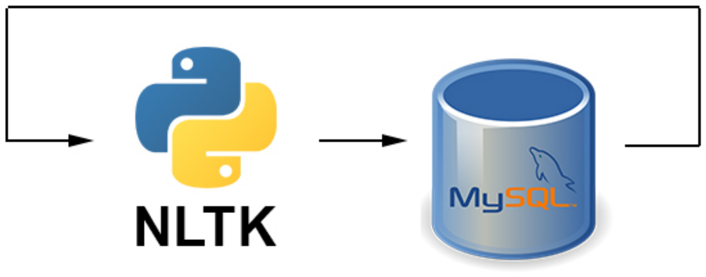
To develop the algorithm it was important to test previous queries continuously because if the grammar is changed it could also change the result of other inputs, not just the one being implemented. Hence we used test-driven development with a test function that served as a validator where input was a file with sentences that we wanted to parse. In order to check whether they were correct we compared the output from the algorithm with another file of SQL queries as what should be expected after building the query from the sentence.

5.2 Design

Parsing a sentence in NLTK requires that a grammar is defined. This is done by a string defining which values are mapped to which key. We can see from here that a query can consist of an AGGR, aggregator, TABLE and WHERE query. We also see that on the next level the WHERE key consists of and OP, operator, COL, column name and a NOUN, this would for example be NUM if "Josh" would be a number instead of a name. Defining this grammar lets the parser know what is an accepted sentence or not and parse it accordingly.



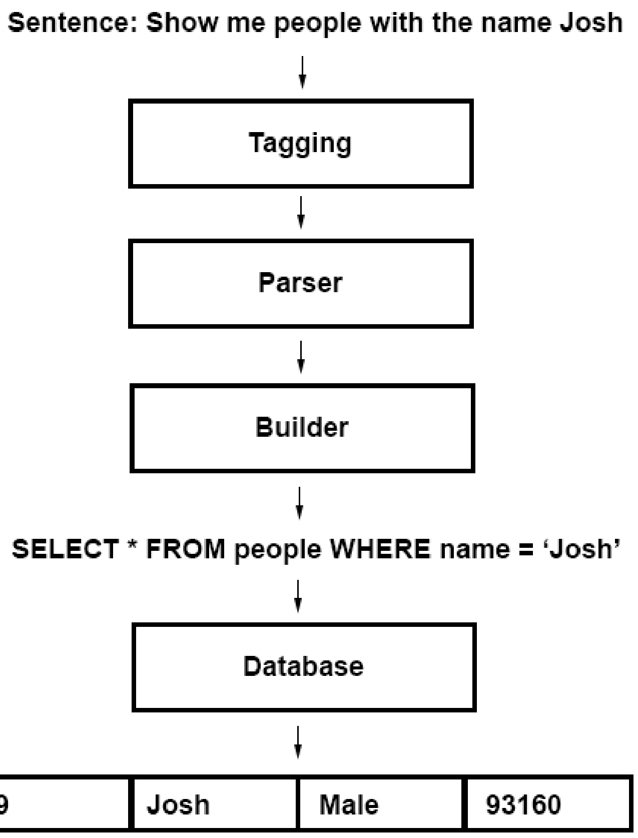
In order for a sentence to make pragmatical sense we need to know the database context. We do this by specifying the database we want to connect to and the table we want to use when parsing sentences. By doing this we can map values to keys before we parse a sentence such as $TABLE \rightarrow$ "*people*" or $COL \rightarrow$ "*name*". This way the algorithm is more aware of the context and knows what the user is referring to when parsing the sentence. We also add the words from the input sentence to the grammar with the help of PoS (part-of-speech) tagging [5] to be able to identify keys like NOUN, NUM or ADJ.



5.3 Architecture

The flow of the program is the following:

- **Tagging** We generate tags from the sentence with NLTK and push them on to our grammar to formalize our semantics.
- **Parser** The NLTK parser uses this grammar to generate a parsed tree data structure.
- **Builder** Knowing the structure of this tree we can build a sentence with logical if-statements and loops to construct an SQL query.
- **Database** The SQL query is run against the database and we print out the response from the query to the user.
- **Test** We might run tests for each query to see if we receive the expected response.


Sentence: Show me people with the name Josh
Tagging
Parser
Builder
SELECT * FROM people WHERE name = 'Josh'
Database
89
Josh
Male
93160

6 Results

6.1 Support

The project meets the requirements of simple translator from plain English to a syntactically correct SQL query that could be run against a MySQL database. Questions or requests including column values, numbers, conjunctions and different variations of them can be parsed by the algorithm. If asked for a highest, lowest or value equal to something the algorithms build a WHERE query correctly. If a conjunction such as ”and” or ”or” is multiple WHERE statements will be build with AND or OR respectively.

For the general case the algorithm could be used to respond to the most common queries that are usually asked because we know that most questions will start with, for example ”who is”, ”show me” and contain operators such as ”over”, ”under”, ”is” and column/table names that is known to the parser.

6.2 Limitations

The algorithm does not support more advanced queries such as combining tables and columns with INNER JOIN statements. We intentionally left out more uncommon operators such as LIKE, HAVING, BETWEEN, GROUP BY etc.

7 Conclusion

The problem that we have attempted to solve is of course much more complex than what it may seem from the results of this project. The truth is that there can be a lot of variations of a word depending on the context as well as questions where the intention may be unclear. This could for example be ”How old is Josh?”, the part we have not implemented is the word ”old”, in order to parse this sentence we would need to map words relating to a column such as ”old”, ”older”, ”younger” to column ”age”, or ”male” and ”female” to column ”gender”. One way to achieve this would be to add values from fields in the database to the parsing grammar requiring a bit more prepossessing. Some words can also have different forms, for example ”name” is ”names” in plural, which could be parsed with the help of NLTKs stemming functions.

In conclusion the task of translating any type of sentence to a SQL query is a very difficult and is already widely studied by major companies such as Google and is not done in an afternoon. It is however possible to create such an algorithm for the general case in a much more feasible time.

To further improve the algorithm, one could implement machine learning in order to further populate the grammar and also give a better probabilistic support for the parser. Given more time this would be the next step in the development curve of this project.

References

[1] Context free grammars.

[2] Raymond F (1974) Chamberlin, Donald D; Boyce. *SEQUEL: A Structured English Query Language.*

[3] ANSI/ISO/IEC International Standard (IS). *Database Language SQL—Part 2: Foundation (SQL/Foundation).*

[4] Natural language toolkit.

[5] Part of speech (pos) tagging.