International Journal of Research In Science & Engineering e-ISSN: 2394-8299
Volume: 3 Issue: 2 March-April 2017 p-ISSN: 2394-8280

SQL Query Generator For Natural Language

Amit Kumar Jaiswal[1], Vivek Yadav[2], Vidya Sagar Singh[3]

[1]*Student, Computer Science & Engineering, UIET CSJM University Kanpur,*
amitkumarj441@gmail.com
[2]*Student, Computer Science & Engineering, UIET CSJM University Kanpur,*
vivekyadavofficial@gmail.com
[3]*Student, Computer Science & Engineering, UIET CSJM University Kanpur,*
vidyasagarsingh000@gmail.com

ABSTRACT

This research concerns with translating natural language into SQL queries by exploiting the Perl DBI library for both database construction and thesis verification in the task of question answering. We built SQGNL which uses linguistic dependencies and metadata to build sets of possible SELECT and WHERE clauses and is designed to be database and platform independent with multi-user support. It can be used by users with no knowledge of SQL to translate natural language to SQL queries. The program has the ability to learn new grammar.

SQGNL application is written in Perl language with a simple user interface implemented using Tk module. It uses Perl recursive descent module to build the underlying parser. Our algorithm can be recursively applied to deal with complex questions, requiring nested SELECT instructions.

Our preliminary results are encouraging as they show that our system generates the right SQL query among the first five in the 80% of the cases. This result can be greatly improved by re-ranking the queries with a machine learning algorithms.

Keywords*: Natural language translation, Perl module, database functionalities*

---

1. INTRODUCTION

SQGNL propose a large body of work based manual work for grammar specification and dataset annotation. However the task of question answering, translating natural language (NL) into something understandable by a machine, in an automatic way is rather challenging as it is not possible to hand-crafting all the needed rules. Database Management Systems are very powerful means of cumulating and retrieving large amounts of data quickly and efficiently. There are many different commercially available database management systems used around the world. However retrieving data out of the database is not an easy task. A special database interaction language called SQL (Structured Query Language) is used to communicate with these databases. Even though there is an ANSI standard for SQL, there are still minor differences between various database management systems, making it more difficult for even an experienced user to access data. SQGNL is aimed at shortening this complexity of database querying.First it is necessary to use a language that is understood by anybody, whether an expert database programmer or person with non-technical background. The best-pertinent language for this purpose is the English language. This means SQGNL has to translate English language queries into SQL statements before retrieving data from database.

In order to translate natural language, SQGNL needs to know the database architecture. This process is automated as much as possible to minimize the complexity to the user. Our new idea assort well with on how and where this matching can be found, i.e., availing existing knowledge that comes along with each database (metadata). The resulting matching between metadata and words destine for building sets of query components (clauses). These are combined together using a smart algorithm to generate a set of SQL queries, taking into account also the structure of the starting NL.

1.1 MOTIVATIONS AND PROBLEM DEFINITION

A database is not just a collection of data: at design time, domain experts organize entities and relationships giving proper names to tables and columns, defining constraints and specifying the type of the stored data. This additional information is known as metadata and is stored in an underlying database called Information Schema (IS, for brevity) that contains, for each database, tables containing columns referring to table names and column names. This self-reference allows for querying metadata with the same technique and technology used to query the database embedding data that answer a given question. In other words, we can execute several SQL queries over IS and a target database and combine their result sets to generate the final SQL query in a straightforward and very intuitive way.

1.2 DATABASES QUERYING USING SQL

SQL commands (or commonly referred to as SQL statements) are issued to DBMS, which then execute these commands and return the results (if any). In order to query or retrieve some specific data from the database, SELECT statement has to be issued. The syntax for SELECT statement is,

SELECT column_list
FROM table_list
[WHERE conditional_expression]
[GROUP BY group_by_column_lis]
[HAVING conditional_expression]
[ORDER BY order_by_column_list]

The expressions in square brackets are optional and the words in capital letters are the SQL keywords. All the lists in the statement are comma separated. Column names and table names are usually case sensitive. The most common SQL statements involve only the SELECT, FROM and WHERE clauses.

Columns_list in SELECT clause

This list contains what columns to be displayed in the results. The asterisk (*) represents all the columns. If the tables list contain more than one table, then each column needs to be referred to as <table_name>.<column_name> format.

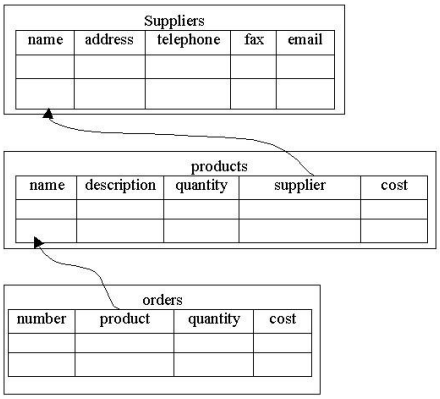

Fig. 1 : Sample Database Structure

Examples -

show all columns in suppliers table

SELECT * FROM suppliers

show the 'name' and 'address' columns from the suppliers table

SELECT name, address FROM suppliers

show the 'name' column from suppliers table and 'description' column from products table.

SELECT suppliers.name, products.description FROM suppliers join products ON suppliers.primarykey = products.foreignkey

Columns list can also contain expressions or functions. Expressions have the format of:

<column_name> <operator> <column_name> AS <name_for_the_returned_column>

2. SQGNL Design

Inteneded design of SQGNL can be broken down into three sections: database functionalities, natural language translation and user interface and the details of which are explained below.

2.1 **Required Database functionalities**

Numerous database functions need to be implemented for the operation of SQGNL. First it is required to establish a connection to the chosen data source. Since SQGNL is intended to be database independent, the implementation must support establishing connections to different types of databases. The data source may require user login. Therefore the program needs to know the current SQGNL user and his/her password to access the database. Except for password, all the other information (database type, data source and user

name) needs to be saved for future use. Password however is required to be entered by the user every time user runs the program.

Before translating any English statements, it is essential to know the database structure. This structure can then be used to match tables and columns to the natural language queries. In order to know the database structure, SQGNL needs to retrieve table names, column names and column data types. Column data type is required to format the values in SQL statements as string values need to be quoted and date values need hashes (#) around the value. Since each user have different access privileges to the same database, the database structure visible to the user may differ from one another. SQGNL only needs to consider what is visible to the user, not the complete database structure.

The third database functionality required is the execution of the translated SQL statements. Only SELECT queries are executed and other types of SQL statements are not implemented in SQGNL. One of the main reasons for this is that the SELECT queries do not alter the database and therefore it is more secure against accidental changes. Once the SQL statement is executed, the results have to be displayed to the user. The results could either be an SQL error or a number of database records. The maximum number of database records to be displayed must be limited, as it is possible that the resulting record set may contain thousands of records. Displaying such huge information is not required and it will consume lots of computer resources.

2.2 NATURAL LANGUAGE TRANSLATION

We used Parse::RecDescent module to implement the natural language parser. Parse::RecDescent is a top-down parser, which gives all the advantages of top-down parsing. There are other bottom-up parsers such as Parse::Yapp and perl-byacc , which I could have used. But the Parse::RecDescent has some special features that are more suited for this application. Most importantly, the RecDescent module can,

1. Generate run-time parsers - so we can embed the database structure during the run-time. This is important as we do not know the database structure until we connect to the database during run-time.
2. Modify or extend the parser during run-time - useful for SQGNLs learning functionality as we need to extend the parser to learn new grammar
3. Save the parser object to a file and reload it in future - this is another important feature as this enables the parser object to be created only once and new extensions to the grammar can be saved in the same file as the parser rather than a separate file for the new grammar learned. Also loading the parser from a file is significantly faster than creating the parser.

The use of Parse::RecDescent module in SQGNL can be explained using grammar given below.

Grammar : [show|list|display} (me) (all) (our) <table_name>
SQL : SELECT * FROM <table_name>

The above grammar can be translated to perl code that the parser can understand.

```
my $grammar = q{
    translate : select
    select : ask /(me)?/ /(all)?/ /(our)?/ table_name { "SELECT * FROM $item[5]" }
    ask : /show|list|display/
    table_name: TABLES };
```

2.3 USER INTERFACE DESIGN

The basic user interface consists of at least four windows. First we need a user login window to get the user password to access the database. This window should contain two areas for the user to enter the user name and password.

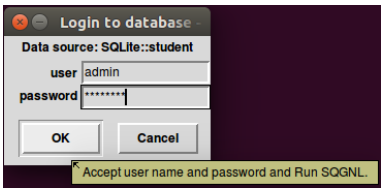

Fig.2 : User Login Interface

Secondly a configuration window is required to get the information about the database from the user. This information consists of database type, data source, user name and the password. Additional information such as maximum number of records to display and choice of enabling the learning process can also be entered in the same window.

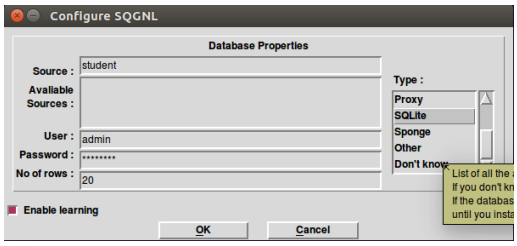

Fig.2 : Database Configuration

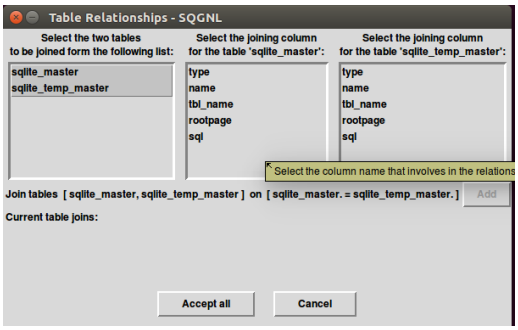

Fig.3 : Database Structure

Another window is required to get extra information about the database structure. This information includes related words for tables and columns and relationships between tables.Finally the SQGNL main window, which contains areas for English statement entry, SQL output of the translation and area to display play the resulting records from the database.

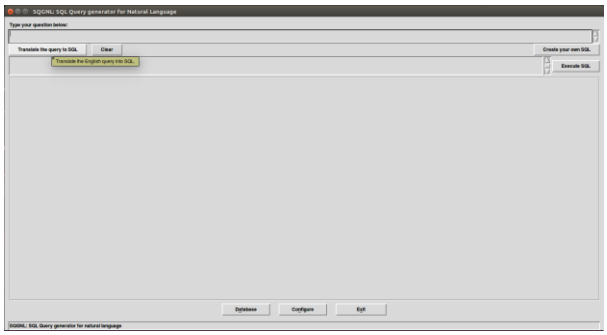

Fig.4 : SQGNL Application Interface

3. SQGNL Implementation

SQGNL application is written in Perl. Perl is chosen as the preferred programming language because of its simple and powerful string manipulation capabilities. Since SQGNL relies much on string manipulation tasks, this is the major factor of considering the implementing language. Furthermore Perl already has many useful modules written by other people, which can be used in SQGNL. In particular the use of a parser module will simplify the process of coding, as I do not have to spend time on writing my own code. Also there are many different modules for database communications and implementing user interfaces in Perl. Perl can also write platform independent program and this is suited for the purpose of SQGNL. Considering all the above factors, Perl is the best candidate for writing SQGNL.
DBI module is used to implement the required database functions. DBI is chosen as it can be used to write database independent code without repeating similar code to different types of databases. In order for DBI module to work, the corresponding DBD module is required to be installed in the system. These DBD modules have the database specific code, which is encapsulated by the DBI module. If the appropriate database driver (or DBD module) is not installed in the user system, the program cannot progress any further. If the user does not know the type of database, SQGNL can then try all the available database types until a valid connection is established.Table names can be accessed using the DBI:table_info() method. However for DBD::ADO module, table_info() method is not yet implemented. SQGNL currently does not support for ADO type databases and hence cannot access table names from ADO type databases.

4. CONCLUSION

We implemented most of the aims of SQGNL successfully. The program can translate simple natural language in to SQL queries. It can translate to different types of SELECT queries, which include retrieving data from single or two tables with or without a single condition. Learning capability of SQGNL is also been implemented with some success. It is not as efficient as expected because it can only detect table names, field names and conditions in the queries but cannot generalise other words such as determining which words can be optional and may omitted in queries. SQGNL is almost database independent with the exception of ADO databases. The program successfully runs on both Windows 32/64 platforms and Linux environments. Multiple user access is also supported as a new parser is created for each user and saves in different files.

5. ACKNOWLEDGEMENT

The research described in this paper has been reviewed by the Assistant Professor; Alok Kumar and Deepak Kumar Verma, of Department of Computer Science & Engineering. We also thank our mentor; Deepak Kumar Verma for consistent feedback on our research.

6. REFERENCES

[1] Marie-Catherine de Marneffe, B.M., Manning, C.D.: Generating typed dependency parses from phrase structure parses. In: Proceedings LREC 2006. (2006).

[2] Akeel I Din, "Structured query language (SQL) A practical Introduction", NCC Blackwell Ltd, 1994.

[3] Fred R. McFadden, Jeffery A. Hoffer, Mary B. Prescoh, "Modern Database Management", 5th Edition, Wesley Educational Publishers Inc, 1999

[4] Alligator Descartes and Tim Bunce, "Programming the Perl DBI" Cambridge, MA : O'Reilly, 2000

[5] Nancy Walsh, "Learning Perl/TK", Beijing ; Cambridge : O'Reilly, c1999

[6] Damian Conway, "The man(1) of descent", The Perl Journal, Issue 12, Vol. 3, No. 4, Winter 1998, pg 46-58

[7] Damian Conway, "Practical Parsing with Parse::RecDescent" Perl Conference 3.0, 1999