ISSN: 2229-6956 (ONLINE)
DOI: 10.21917/ijsc.2019.0282

SQL_NL - A PARSER THAT CONVERTS SQL QUERY TO NATURAL LANGUAGE

Smita Paira and Sourabh Chandra

Department of Computer Science and Technology, Indian Institute of Engineering Science and Technology, Shibpur, India

Abstract

With the rapid development of technology, Natural Language processing acts as a technique where machine and human can interact simultaneously. Besides learning a new programming language, human being can rather talk with the machine. However, the non-expert users neither understand any computer command nor have any idea about the underlying technology. For this reason, a new parser, SQL_NL has been developed to convert any Structured Query Language (SQL) query to its corresponding natural language version. The parser is much efficient to produce the results within milliseconds with error detection and debugging capabilities. The output produced by the parser is easily understandable and unambiguous with fast conversion rate.

Keywords:

LEX, YACC, SQL, Parser, NL, Lexical Analyzer, Grammar, Database

1. INTRODUCTION

Generally, programmers manage databases very carefully and perform certain operations on the databases using languages like SQL. It takes structured data as input and produces structured data as output. The fundamental concept behind this language is to define or write certain queries. Based on those queries, the databases are being created, updated or deleted. There are various platforms for this language like MySQL, Oracle 11g, Oracle 9i, etc.

The programmers can easily write and execute SQL queries as and when required. But the problem is that, it becomes difficult for common public to understand those queries. Therefore, a parser that takes natural language as input and produces SQL query as output or a parser that takes SQL query as input and produces natural language as output can make them better understand the queries and operations. Efficient compilation process plays a vital role for the execution of different programming languages. Lexical Analyzer (LEX) and Yet Another Compiler Compiler (YACC) are the two fundamental tools those are used for designing Compilers [16].

This paper deals with the design of a parser called SQL_NL that translates an SQL query into Natural Language. The Compiler is designed to compile and SQL query and reproduce the same in Natural Language, which any human being can easily understand. LEX and YACC have been used here for designing the parser. LEX takes a Regular Grammar (RG) as input [14] and converts it into a recognizer yylex(). On the other hand, YACC takes a Context Free Grammar (CFG) and converts it into yyparse().

Given a set of regular expressions (RE), the task is to generate a Lexical Analyzer. It takes an input stream, scans it, identifies a part of it, matches it with the set of RE and returns the associated integer as token to the yyparse(). The Fig.1 shows how a Lexical Analyzer works.

The Fig.2 presents the block diagram of yylex() and yyparse() and how they work hand in hand. LEX on receiving the regular expression generates a number. The number and inputted SQL query pass through yylex() which further generates a stream of tokens. The stream of token goes through yyparse() which creates the required output after compilation through YACC.

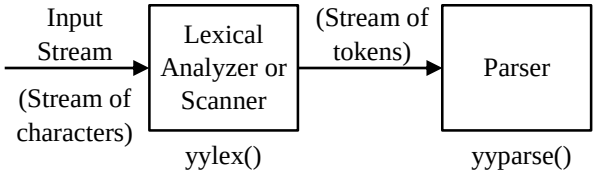

Fig.1. Principle of yylex()

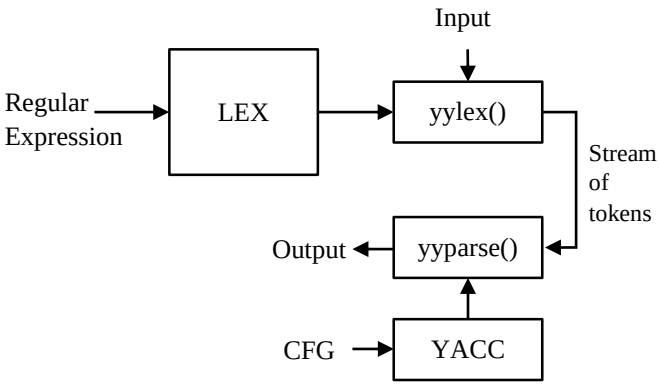

Fig.2. Block diagram of LEX and YACC

The structure of LEX and YACC are shown in Fig.3 and Fig.4 respectively. In case of LEX, the definitions of all variables are kept first, followed by the rules which are further followed by user subroutines. Even though variable description and subroutines are optional, rules are mandatory to be mentioned.

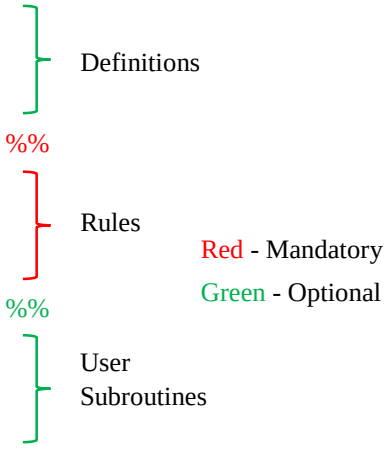

Fig.3. Structure of LEX [14]

Similarly, structure of YACC consists of two optional parts namely variable declaration and programs and one mandatory section consisting of Context Free Grammar (CFG) rules.

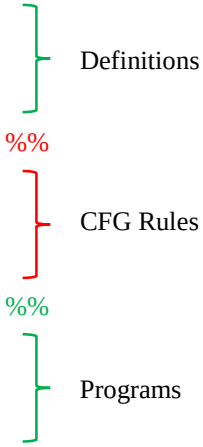

Fig.4. Structure of YACC [14]

Consider a grammar with production rules as follows:

$$E \rightarrow (E)$$
$$E \rightarrow E+E$$
$$E \rightarrow id$$
$$E \rightarrow int\_const$$

Corresponding to each production rule, there is an associated action routine both in the lexical analyzer (.l) and parser (.y) files. The parse tree for the above grammar is shown in Fig.5.

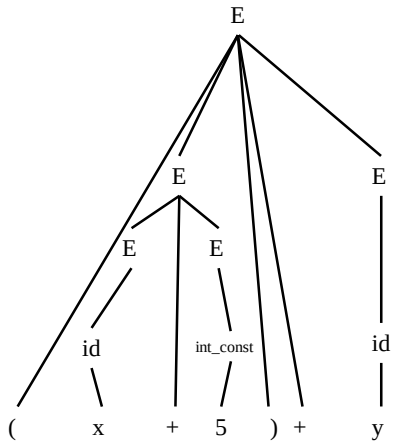

Fig.5. Parse tree of the given grammar

The reduction of the leaf nodes like id $E \rightarrow x$, $int\_const \rightarrow 5$, $id \rightarrow y$, etc. are taken care of by yylex(). The reductions like $E \rightarrow$ id, $E \rightarrow int\_const$, $E \rightarrow (E)$ and $E \rightarrow E+E$ are taken care of by yyparse().

When main() calls yyparse(), the latter calls yylex(). Here, yylex() scans the input stream from left to right and identifies tokens [15] [16]. If yylex () provides some value to the identifier through yylval() in the action routine. This value becomes available in action routine of YACC as $1, $2, etc. which in turn will be available in $$ of the left hand side non terminal of the grammar rules. This value uniquely identifies each identifier.

In this paper, SQL_NL takes SQL query as input and produces its corresponding natural language expression as the output. The parser is also flexible enough to check the syntax or semantic errors if present in the input SQL query. The underlying process has been elaborated in the remaining sections.

2. RELATED WORK

Alok Parlikar et al. in [1] proposed a Natural Query Markup Language called NQML. It is an extension of XML [2] that converts questions in English into SQL queries. The algorithm works on SELECT queries with multiple WHERE clauses. NQML returns an exception in case of unknown cases but it is quite nascent.

Yellin and Mueckstein [3] proposed an automatic inversion process of attribute grammars which generate compatible two-way translators from single description. They have implemented such algorithm on an SQL database to paraphrase the queries into English. They want to specify that if an attribute grammar can translate a language L1 into L2, then its inversion can do the opposite.

Another Natural Language Query (NQL) procession technique has been proposed by Xuan et al. [4]. NQL works on Remote sensing query text by extracting the keywords, extending them and then generating the SQL query.

Axita Shah et al. [5] has proposed a natural language and keyword based interface called NLKBIDB. The keyword based interface helps to provide solutions to incorrect natural language input queries. They have implemented this algorithm on an agriculture database in which out of 75 syntactically incorrect queries, 40 are solved.

Reinaldha and Widagdo [6] worked on the extension of the NLIDB System. The method can process question type query as well as it can handle unit conversion. It applies different modifying rules and directive sentences to process different type of question queries and recognize and translate data into units as asked by the user.

Mohite and Bhojane [7] developed another method of converting query in Natural Language to SQL query by forming a co-occurrence matrix. The steps that they used are syntactic parsing, extraction of the keywords, matrix generation and conversion into SQL query. They have done their work on a Java platform but unfortunately the algorithm does not work for all form of queries.

Joseph et al. [8] provided a natural language interface to XML Database. In this method, query in natural language is converted in XQuery by a mapping technique from the dependency parse tree. Such an XQuery statement is used to retrieve data from XML database.

C-Phrase is a Natural Language interface to database that is proposed in [9]. It uses semantic grammars inspired by X-bar theory [10], encode those in λ-SCFG. Such encoded grammars map user inputs to an extended version of Codd's tuple calculus which automatically maps to SQL. Sometimes it happens that the queries entered by the user are contradictory that return empty sets which is not desirable. A method was proposed by Brass and

Goldberg in [11] that checks the semantic errors present in a query. This helps the DBMS to check the contradictory queries before execution.

According to Llopis and Ferrández [12], Natural language interfaces to database are not properly adopted. They designed a system called AskMe* that provides the adoption and some extra query authoring services. It lowers the entry barrier of end users. Frank S.C. Tseng and Chun-Ling Chen, in their work [13] have extended the representations in UML class diagram to capture queries in Natural language with fuzzy semantics. The proposed method converts natural language constructs into class diagrams and employ SOM methodology to transform into SQL statements.

It is felt that accessing data from a database using Natural Language is quite convenient and easy compared to some formal languages likes SQL [8]. A lot of works has been done to convert a query given in Natural Language into SQL. In this paper, we have proposed a novel approach of applying the reverse technique i.e. we have developed an algorithm that can translate a given SQL query into Natural Language. So that common user can not only feel the essence of accessing data from the database but also can understand what operations are being performed by different SQL queries.

3. UNDERLYING ALGORITHM

SQL_NL creates three source files: lexfile.l, yaccfile.y and abc.h. The algorithm for SQL_NL is described below:-

Step 1: Create the lex file and save it with .l extension. Here, we have named as lexfile.l. The lexfile.l contains the action routines of each lexeme. The file should return values to yyparse() using yylval().

Step 2: Create the yacc file and save it with .y extension. Here, we have named as yaccfile.y

Step 3: Create a union of terminals and non-terminals in yaccfile.y file. Design the CFG with associated action routines.

Step 4: Create themain() function takes the input and output files names as command line arguments. The main() function calls yyparse().

Step 5: Create another file abc.h. This file contains the actual syntax of how to detect the type of query and how to handle it. The file is included in the yaccfile.y along with other header files.

Step 6: Each time a query in inputted, the lexfile.l tokenizes it. The tokens with associated values are sent to the yaccfile.y file.

Step 7: In yaccfile.y the grammar for each type of query is written. Simple queries do not have many complexities.

Step 8: For large queries, the parser calls abc.h which contains code for handling them. The abc.h file also contains code for handling the semantic errors.

Step 9: The yyparse() itself identifies the syntax errors if any but the type of error is coded elaborately in abc.h file. The parser prints the errors efficiently with type and line numbers.

Step 10: End.

4. EXECUTION RESULTS

The SQL_NL parser has been tested and executed on a Linux Terminal. Different test cases with outputs are shown in Fig.6-Fig.8.

Case 1: Input file: in

Output file: opt

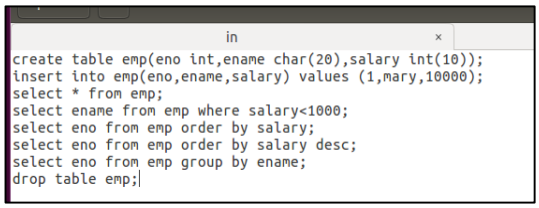

Fig.6(a). Input file for case 1

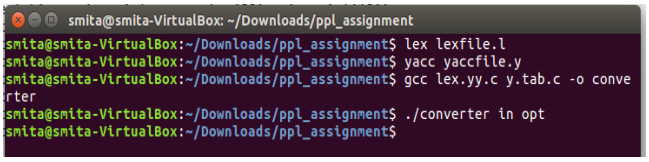

Fig.6(b). Terminal for case 1

The compiler compiles the input file (shown in Fig.6(a)) and cannot find any error. Upon execution it generates the output file (shown in Fig.6(b)) which converts the queries into normal sentences. Hence, the output file (shown in Fig.6(c)) is empty.

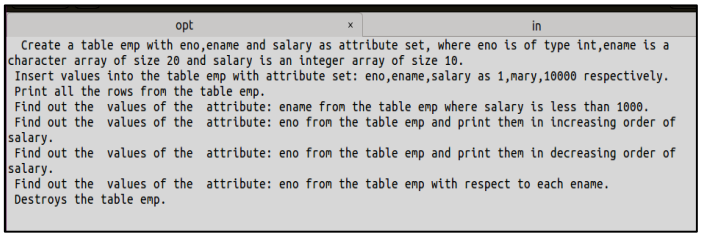

Fig.6(c). Output file for case 1

Case 2: Input file: input

Output file: opt

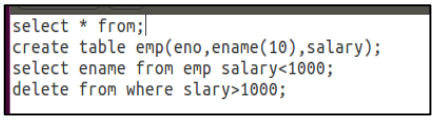

Fig.7(a). Input file for case 2

```
smita@smita-VirtualBox:~/Downloads/ppl_assignment$ ./converter inpt opt
 Line no 1 : syntax error : Expected table name after 'from'

select * from;



 Line no 2 : syntax error : Required 'char' or 'int' after eno
               Required 'char' or 'int' after ename
               Required 'char' or 'int' after salary

create table emp(eno,ename(10),salary);



 Line no 3 : syntax error : Expected 'where' after table name

select ename from emp salary<1000;



 Line no 4 : syntax error : Expected table name after 'from'

delete from where slary>1000;
```

Fig.7(b). Terminal for case 2

The input file shown in Fig.7(a) contains some syntax errors. The SQL queries upon compilation produces syntax errors which are shown on the terminal (in Fig.7(b)) indicating line numbers and error positions. Hence, the output file (shown in Fig.7(c)) is empty.

Fig.7(c). Output File for case 2

Case 3: Input File: input1

Output File: opt

The third case scenario is to show how SQL_NL works in case there are semantic errors in the input file and how it handles them. The input file illustrated in Fig.8(a) contains some queries those are semantically incorrect.

input1

```
create table emp(eno int);
create table emp(eno int,ename char(10));
create table dept(dno int(3), dname char(20));
drop table dept;
drop table employee;
drop table emp;
drop table emp;
```

Fig.8(a). Input File for case 3

SQL_NL compiles and points out all semantic errors with line numbers (shown in Fig.8(b)). It neither allows duplicate table/relation to be created nor does try to delete or drop a table that does not exist.

```
smita@smita-VirtualBox:~/Downloads/ppl_assignment$ lex lexfile.l
smita@smita-VirtualBox:~/Downloads/ppl_assignment$ yacc yaccfile.y
smita@smita-VirtualBox:~/Downloads/ppl_assignment$ gcc lex.yy.c y.tab.c -o con
r
smita@smita-VirtualBox:~/Downloads/ppl_assignment$ ./converter input1 opt

 Line no : 2 : Error : Table 'emp' has already been created
       create table emp(eno int,ename char(10));


 Line no : 5 : Error : Table 'employee' does not exists
       drop table employee;


 Line no : 7 : Error : Table 'emp' does not exists
       drop table emp;

smita@smita-VirtualBox:~/Downloads/ppl_assignment$
```

Fig.8(b). Terminal for case 3

```
 Create a table emp with eno as attribute, where eno is of type int.
 Create a table dept with dno and dname as attribute set, where dno is an integer array of size 3 and
dname is a character array of size 20.
 Destroys the table dept.
 Destroys the table emp.
```

Fig.8(c). Output File for case 3

The output file for the third case is shown in Fig.8(c). Queries those are syntactically as well as semantically correct, are converted into simple English sentences for the better understanding of common people.

The parser has been tested several times taking different data sets. Each data is compiled and executed around 20 times with an average conversion of 0.348 milliseconds without any ambiguity.

5. CONCLUSION AND FUTURE WORK

In this paper, a new parser called SQL_NL has been proposed with several benefits associated with it. The parser can compile an SQL query and generate its corresponding natural language within fraction of seconds. It also has the capability to detect and debug any syntax or semantic errors present in the query code.

It acts not only as a language converter but also solves numerous problems for different segments of society. It increases interactions among business and social community. It strengthens client-salesman relationship as well as improves various management skills like leadership, communication and motivation

In future, we shall extend our efforts to design efficient compilers that could process any programming language.

REFERENCES

[1] Alok Parlikar, Dr. Sudip Sanyal, Nishant Shrivastava and Varun Khullar, "NQML: Natural Query Markup Language", *Proceedings of International Conference on Natural Language Processing and Knowledge Engineering*, pp. 184-188, 2005.

[2] Tim Bray, Jean Paoli, C. Michael Sperberg McQueen, Eve Maler and Francois Yergeau, "*Extensible Markup Language (XML) 1.0*", 1st Edition, World Wide Web Consortium, 2000.

[3] Daniel M. Yellin and Eva-Maria M. Mueckstein, "The Automatic Inversion of Attribute Grammars", *IEEE*

Transactions on Software Engineering, Vol. 12, No. 5, pp. 590-599, 1986.

[4] Liu Jianbo, Xuan Xuan and Yang Jin, "Research on the Natural Language Querying for Remote Sensing Databases", *Proceedings of IEEE International Conference on Computer Science and Service System*, pp. 228-231, 2012.

[5] Axita Shah, Hemal Patel, Jyoti Pareek and Namrata Panchal, "NLKBIDB - Natural Language and Keyword Based Interface to Database", *Proceedings of IEEE International Conference on Advances in Computing, Communications and Informatics*, pp. 1569-1576, 2013.

[6] Filbert Reinaldha and Tricya E. Widagdo, "Natural Language Interfaces to Database (NLIDB): Question Handling and Unit Conversion", *Proceedings of IEEE International Conference on Data and Software Engineering*, pp. 1-7, 2014.

[7] Anuradha Mohite and Varunakshi Bhojane, "Natural Language Interface to Database Using Modified Co-Occurrence Matrix Technique", *Proceedings of IEEE International Conference on Pervasive Computing*, pp. 1-6, 2015.

[8] Janu R Panicker, Jiffy Joseph and M. Meera, "An Efficient Natural Language Interface to XML Database", *Proceedings of International Conference on Information Science*, pp. 207-212, 2016.

[9] Michael Minock, "C-PHRASE: A System for Building Robust Natural Language Interfaces to Databases", *Data and Knowledge Engineering*, Vol. 69, No. 3, pp. 290-302, 2010.

[10] R. Jackendoff, "*X-Bar-Syntax: A Study of Phrase Structure, Linguistic Inquiry Monograph*", MIT Press, 1977.

[11] Christian Goldberg and Stefan Brass, "Semantic Errors in SQL Queries: A Quite Complete List", *Journal of Systems and Software*, Vol. 79, No. 2, pp. 630-644, 2006.

[12] Antonio Ferrandez and Miguel Llopis, "How to Make A Natural Language Interface to Query Databases Accessible to Everyone: An Example", *Computer Standards and Interfaces*, Vol. 35, No. 5, pp. 470-481, 2013.

[13] Chun Ling Chen and Frank S.C. Tseng, "Extending the UML Concepts to Transform Natural Language Queries with Fuzzy Semantics into SQL", *Information and Software Technology*, Vol. 48, No. 3, pp. 901-914, 2006.

[14] Saumya K. Debray, "Lex and Yacc: A Brisk Tutorial", Technical Report, Department of Computer Science, University of Arizona, 2006.

[15] H. Altay Guvenir, "Lex and Yacc", Available at: www.cs.bilkent.edu.tr/~guvenir/courses/CS315/lex-yacc/lex-yacc.pdf