Design and implementation of a general SQL parser

Kehe Wu\textsuperscript{1,a}, Dan Di\textsuperscript{2,b*}, Wei Li\textsuperscript{3,c}, Wenchao Cui\textsuperscript{4,d}

\textsuperscript{1} School of control and Computer Engineering, North China Electric Power University, City, Beijing, 102206, China
\textsuperscript{2} School of control and Computer Engineering, North China Electric Power University, City, Beijing, 102206, China
\textsuperscript{3} School of control and Computer Engineering, North China Electric Power University, City, Beijing, 102206, China
\textsuperscript{4} School of control and Computer Engineering, North China Electric Power University, City, Beijing, 102206, China
\textsuperscript{a} wkh@ncepu.edu.cn
\textsuperscript{b}Corresponding author’s e-mail: 120192227048@ncepu.edu.cn
\textsuperscript{c} epulw@126.com, \textsuperscript{d} czzz@qq.com

Abstract: With the rapid development of computer network, the information security of database becomes more and more important. In database security protection, SQL statements need to be parsed and reconstructed. However, the database audit products on the market lack accurate analysis of SQL statements. Therefore, this paper constructs an SQL parser from four aspects: lexical analysis, syntax analysis, SQL spanning tree optimization and error detection.

1. Introduction
At present, relational database is widely used, and SQL language is the top priority of relational database. Most database audit products on the market miss the audit of SQL statements with a length of more than 1.5k, that is, they directly lose data packets without analysis. Some hackers will use these vulnerabilities to launch attacks. The precise parsing technology of long sentences determines the protection effect against such attacks. Therefore, developing a general SQL parser to support SQL statement reconstruction and error handling has become a common requirement of many applications. This paper constructs a general SQL parser from the aspects of lexical analysis, syntax analysis, SQL spanning tree optimization and error detection of SQL parser, so as to avoid the omission of SQL statement analysis and improve the error processing ability of the parser.

2. Parser generation
2.1 Lexical analysis
Lexical analysis is the first stage of the compilation process and the basis of compilation. There are two lexical rules: EBNF and BNF. This paper uses EBNF paradigm to describe the standard syntax of SQL.

The task of lexical analysis in this stage is to read the source program character by character from left to right, identify words according to word formation rules, and the output result is a word
sequence (token), which provides the basis for syntax analysis. In addition, it is also necessary to preprocess the source program, such as filtering out useless spaces, skip, carriage return, line feed and comments in SQL. Moreover, ANTLR allows the host language to be embedded in the lexical file. The C code embedded in the lexical analyzer in this paper helps the parser to correspond the error information found by lexical analysis with the error location of SQL source code. There are five kinds of tokens analyzed by lexical analyzer: (1) reserved words, such as create and table in SQL;(2)Identifier, such as constant name and table name;(3)Various types of constants, such as 25, 3.14, etc;(4)Operators, such as +, <, and, etc;(5)Delimiters, such as parentheses, brackets, etc.

After defining these words, we also need to define a special use morphology. Because in the process of syntax parsing, all characters should be able to be processed. When the user enters other characters, the grammar parser will produce an error of undefined word segmentation. In order to avoid such errors, we need to define the use morphology additionally, "~ (...)" means words other than any previously defined words.

2.2 Syntax analysis

After lexical analysis, enter grammar analysis. Any unresolved conflict may cause the parser to not accurately recognize the rules and the program cannot execute correctly. Only by designing clear and conflict free grammar rules can we establish a grammar tree. However, most grammars are ambiguous. Secondly, most language specifications use a special recursive method called left recursion. At present, the classic form of top-down syntax and parser cannot handle left recursion.

However, for input such as $1 + 2 \times 3$, the input can be interpreted in two ways. The difference between the syntax analysis tree in the middle and on the right of Figure 1 is that the syntax analysis tree added to the middle indicates that 1 is added to the result of multiplying 2 and 3, while the syntax analysis tree on the right indicates that the result of adding 1 and 2 is multiplied by 3. This is the problem of operator priority. The traditional syntax cannot specify the priority. ANTLR solves the ambiguity problem by preferring the front alternative branches, which implicitly allows the operator priority to be specified. For example, in expr rules, multiplication rules precede addition rules, so ANTLR will give priority to multiplication when solving the ambiguity problem of $1 + 2 \times 3$.

Combined with the logical structure and application requirements of SQL syntax, on the basis of dealing with the inherent syntax rules of SQL, custom rules are added to the SQL parser according to the functional requirements. The parser is automatically generated according to the syntax paradigm. The parser calls the lexical analyzer to obtain the next word token. In the process of syntax analysis, the user-defined rules and actions are embedded into the analysis scheduling. The analysis process is shown in Figure 2.

Figure 1 syntax analysis tree explained in different ways

Figure 2 SQL parser process diagram
3. Syntax tree optimization

3.1 relational algebra optimization

Applying some useful algebraic laws in relational algebra to database can effectively improve the query speed. The following is obtained according to the theory of the law of Relational Algebra: 1) when transforming queries, we should consider other binary operations such as selecting projection first and then connecting; 2) when connecting, you should first connect small relationships, and then connect large relationships. If there is a relationship: workerinfo R (workerid, name, phone, addr, gender); Positioninfo (position, year, workerid); Queries the position and place of birth of male employees recruited in 2021 in these two relationships, select position, addr from workerinfo, positioninfo where year = 2021 and gender = 'M' and workerid = ID;

On the left is the syntax tree after the query compilation phase is completed, and on the right is the optimized syntax tree, as shown in Figure 3.

![Figure 3 syntax tree and optimized syntax tree](image)

In the optimization process, first when you can make a selection, Tend to connect rather than product; Secondly, the other two conditions of the where clause are split into two choices, Operations are pushed down to the corresponding relationship of their respective trees. It is found that the optimized syntax tree can save the storage space.

3.2 removal of useless conditions

The removal of useless conditions can be completed only according to the SQL itself and table structure, and there are many optimization cases. In order to avoid too cumbersome description, the following two cases are analyzed through the figure [2].

a) 1=1 and (m > 3 and n > 4)

![Figure 4 Syntax Tree (a) and (b)](image)

From Figure 4 Syntax Tree (a), the value in the dashed box is always true, and the upper layer is and, so the dashed box is a useless condition. Directly remove this condition from the syntax tree and optimize the syntax tree.

b) 1=2 or (m > 3 and n > 4)
From Figure 4 Syntax Tree (b), the value in the dashed box is always false, and the upper layer is or, so the dashed box is a useless condition, which is directly removed from the syntax tree to avoid SQL syntax injection and protect the security of database information.

4. Error detection of different codes

The SQL code is parsed into the idiom tree, and then the semantic error detection of SQL code is transformed into the error detection of SQL syntax tree. The error detection of syntax tree can be solved by tree structure matching. The detection algorithm adopts tree editing distance [3]. Tree editing distance is also called Levenshtein distance. Taking string as an example, the editing distance between string a and string b is the minimum number of operations to convert a into b. There are three operations: insert a character, delete a character, and replace a character.

The formula is used to represent the editing distance of a and b strings, where I represents the length of a and j represents the length of b.

\[
ed_{ab}(i, j) = \begin{cases} 
\min(i, j) = 0 & \text{if } a_i = b_j \\
\max(i, j) & \text{if } a_i < b_j \\
\min(e_{a,b}(i−1, j−1) + 1, e_{a,b}(i−1, j) + 1, e_{a,b}(i, j−1) + 1) & \text{otherwise}
\end{cases}
\]

(1)

deab (i, j) is described as follows:

- \( \min (i, j) = 0 \), which means that a string is empty, and the editing distance is the length of another non empty string;
- When the value of string \( a_i \) is equal to \( b_j \), the editing distance is the editing distance of one character on string a and b;
- When the \( a_i \) value is not equal to \( b_j \), edit the minimum value of the following three cases

  1) \( e_{a,b}(i−1, j−1) + 1 \) delete \( a_i \);
  2) \( e_{a,b}(i−1, j) + 1 \) insert \( b_j \);
  3) \( e_{a,b}(i, j−1) + 1 \) replace \( b_j \);

By observing Figure 5 on the left, it can be found that the element values near the upper right corner and the lower left corner of the table are large, because the elements near the upper right corner produce more deletion operations, while the elements in the lower left corner produce more insertion operations [4].

The analysis shows that the maximum value of the minimum editing distance between two strings is the length of the longer string in the two strings. No matter how the content of the string changes, the minimum editing distance will not be greater than the length of the longer string. Therefore, skip editing the editing states in the distance table, which will make the final editing step greater than the maximum length.

\[
\text{mindist} = \left| 2 \times (j - i) + \text{len(target)} - \text{len(source)} \right|
\]

(2)

The meaning of this formula is the minimum editing distance of the editing step containing the position operation. Where I and j are the row and column subscripts of the table, respectively. The method to judge whether to skip is: when the calculation result mindist is greater than a long string, it is not necessary to calculate the editing distance at this position.

All cells on the left of Figure 5 that do not need to calculate the distance are marked with an oblique underline area, as shown in the following figure: it can be seen from the right of Figure 5 that the cells in the table that do not need to calculate the distance are distributed in a small upper and
lower triangular matrix. When the length difference between two sequences is very small, the two semi matrices will also increase. When the two sequences are equal in length, the right angle side length of the semi matrix will reach half of the sequence, and the elements that do not need to be calculated will account for 1/4 of all elements.

5. Conclusions

In SQL parser, false positive rate and false negative rate are two important indicators to evaluate the quality of parser [5]. False positive rate refers to the probability that the system reports the correct SQL statement as the wrong SQL statement. False positive rate refers to SQL statements that are not detected but judged as normal input. The smaller the false positive rate and false negative rate, the better the parsing ability of SQL parser. The calculation of false positive rate and false negative rate of SQL parser is to collect the data generated by real-time operation of SQL parser, so the calculation result is more accurate.

| Filter analysis method       | False positive rate | Underreporting rate |
|------------------------------|---------------------|---------------------|
| Keyword based filtering      | 26%                 | 19%                 |
| Regular expression based filtering | 15%             | 11%                 |
| Syntax tree based filtering  | 6%                  | 7%                  |

The false negative rate calculates the false negative rate and false positive rate of the SQL parser according to the statistics of the test results. The results in the above table show that the parser SQL method proposed in this paper has lower false negative rate and false positive rate than other filtering methods.

Figure 6 performance test chart

In SQL database, the response time of SQL statement is another important index to evaluate the quality of SQL parser. As can be seen from Figure 6, when the amount of data in the SQL statement is small, the response time of directly connecting to the database is faster than that of using the SQL parser. The reason is that protocol parsing, code conversion, syntax parsing and syntax tree construction are required in the process of SQL parser. Although there is a certain delay in the response to the system, with the increase of the amount of data, the response time of SQL parser is faster than that of direct database. This is because the optimization of SQL statements is used to speed up the response time and improve the execution efficiency of SQL statements.

The vocabulary and syntax rules of standard SQL language are analyzed, and an SQL parser based on ANTLR is designed and implemented; The SQL parser can monitor the activity of the database and prevent the execution of dangerous SQL statements, so as to protect the database.
References
[1] Xia Ming, Lu Yang, Sheng Yexing, Li Dayong. Design of SQL compiler using lex and yacc in embedded database [J]. Computer technology and development, 2007, 4 (11): 121-124
[2] Sun Zhaoyu, Zhu Hongyu, Huang Yuguang. A strategy and implementation of SQL syntax analysis [J]. Computer applications, 2007 (S1): 18-20 + 23
[3] Wang Jie. Research on SQL injection defense based on abstract syntax tree [D]. Wuhan Institute of Posts and telecommunications, 2018
[4] Cui Na. SQL statement parsing and translation for database performance [J]. Modern electronic technology, 2016,39 (11): 99-102 + 107
[5] Song Jianlong. SQL statement parsing and translation for database security [D]. Beijing University of technology, 2015