Yongun Seong (성용운)
2017-19937
DB

Project 1-1: SQL Parser Report

For this part of the project, I implemented a basic parser for a subset of the SQL specification. My parser takes a well-formed SQL query and creates a syntax tree, reporting any syntax errors if found. I implemented this parser in two phases. First, I modified the provided Lark grammar specification to accept the entire subset of SQL that we are to implement. Second, I implemented a simple utility script to accept SQL queries as input to pass them into the parser generated by Lark, printing out an informational message on each query to verify the parser's correctness.

The Lark Python library performs most of the heavy lifting in the parsing, and I spent the majority of my time adjusting and adding onto the provided grammar specification to accept the entirety of the SQL specification as defined in the assignment. The provided grammar was already sufficient to parse the CREATE TABLE and SELECT queries, and it already contained most keywords and rules for syntax elements. In terms of keywords, I only defined some that had been omitted, such as UPDATE, SET, DESCRIBE, and EXPLAIN. I then implemented the rest of the queries, such as UPDATE, SHOW TABLES, and INSERT by defining the necessary parsing rules. As the grammar already contained most necessary syntactic elements, this was quite straightforward, only requiring mixing and matching existing terms to parse entire queries. Finally, I added each possible query type into the "query" rule, so that each newly added query type would be correctly parsed as a query.

Once I had a correctly constructed grammar file, I wrote a simple utility to take the grammar file, pass it into Lark to obtain a parser, and pass in the input to the generated parser to generate a syntax tree. During this phase, I chose to pass in single queries, delimited by semicolons (";"), separately into the parser. This is because I had to correctly handle syntax

errors in the middle of a query sequence, instead of rejecting the entire query sequence as erroneous. By passing in each query separately, I was able to handle every query up to, but excluding, the badly-formed query.

Once the Lark parser was done parsing the input, I passed the output tree into a custom Lark Transformer to report the query type. This was as simple as defining a method, named the same as the grammar term, that would print the correct query type.

As far as I am aware, my implementation satisfies the specification as laid out in the assignment, but I did take a few shortcuts that would normally be unacceptable in a fully-featured SQL parser. For example, I split the input on semicolons before any parsing, as our subset of SQL does not allow semicolons anywhere in the query except to end the query. Without this shortcut, the implementation would have had to become significantly more complex to handle badly-formed queries in the middle of the query sequence.

I had a few important learnings during the implementation of this parser. For instance, this was my first time working with a parser generator, and I found it very informative. I was able to understand the limitations of parser generators, how they worked, and that they are surprisingly easy to work with. Thanks to the experience I gained while working on this assignment, I gained yet another tool to apply when faced with a similar problem. I look forward to future project assignments, where I will build upon this basic parser to implement a fully working database application.