CSB 353: Compiler Design

Project Report (Parser for P3SQL)

Submitted By:

Prashant Borkar (191210036)

Prem Kumar (191210037)

Prince Kumar (191210038)

Submitted To: Dr. Shelly Sachdeva

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY DELHI

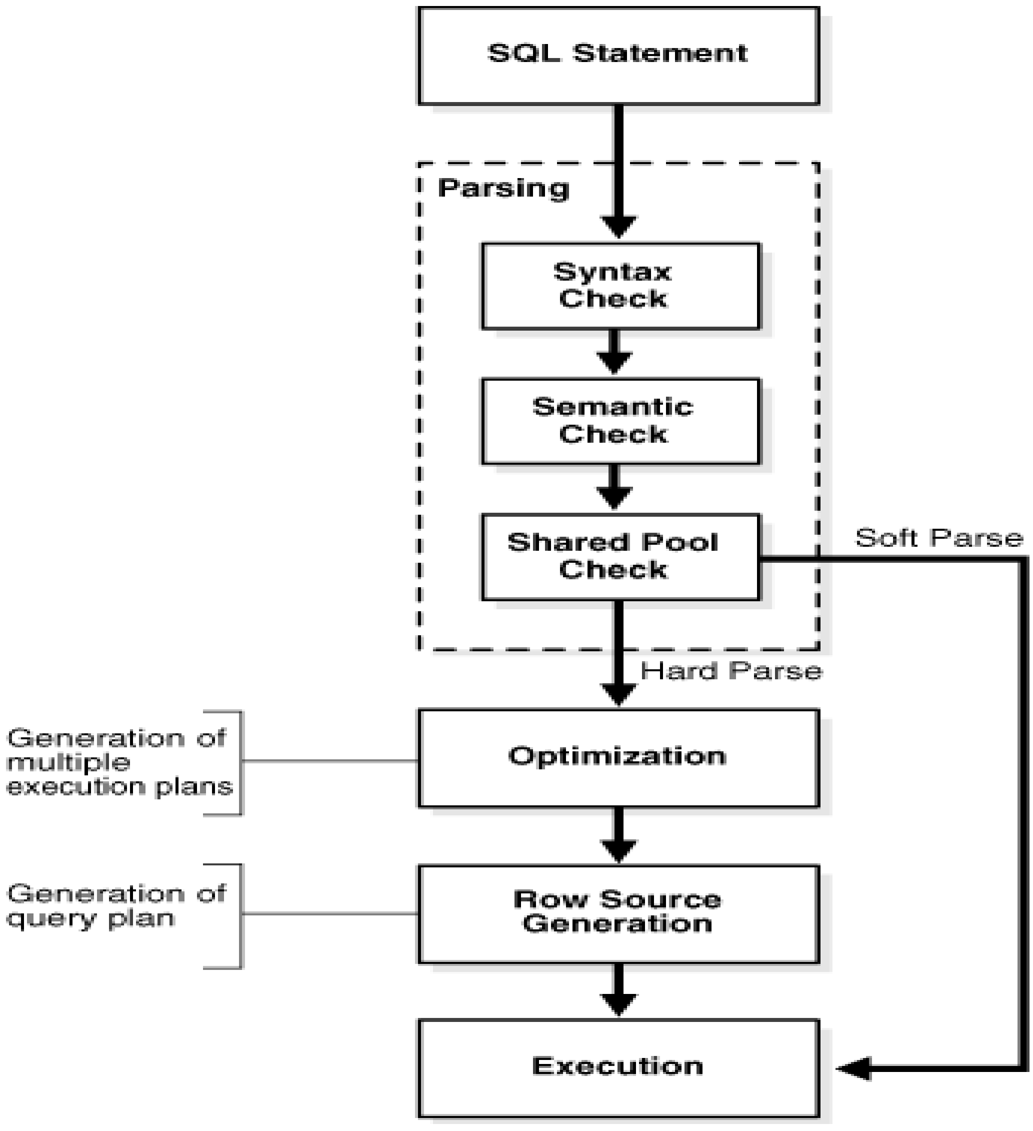
2019-2023

Table of Content

Introduction

Our project aims to undertake a sequence of experiments to design and implement various phases of a Parser for a Structured Query language which we have named as P3SQL, it is a subset of SQL.

We have developed this in python making the use of lex and yacc and we use pymysql to connect with MySQL Database.



Covered Scope:

- **Type**
    - STRING
    - INTEGER
- **Statement**
    - CREATE DATABASE
    - DROP DATABASE
    - SHOW DATABASES
    - USE DATABASE
    - CREATE TABLE
    - DROP TABLE
    - SHOW TABLES
    - INSERT
    - DELETE
    - UPDATE
    - SELECT

Inputs can be given through the input file or through terminal.

Requirements:

- Python 3
- PLY

  PLY is yet another implementation of lex and yacc for Python. Some notable features include the fact that it is implemented entirely in Python and it uses LALR(1) parsing which is efficient and well suited for larger grammars.
- PyMySQL

  This package contains a pure-Python MySQL client library.
- MySQL
- VS Code(ide)

Phases:

- Lexical Analyzer:

Identification of Keywords, Identifiers, Operators (Relational, Logical and Arithmetic), Punctuators, Constants (Integer, Character) and String Literals with invalid string error handling. Parenthesis matching with error reporting.

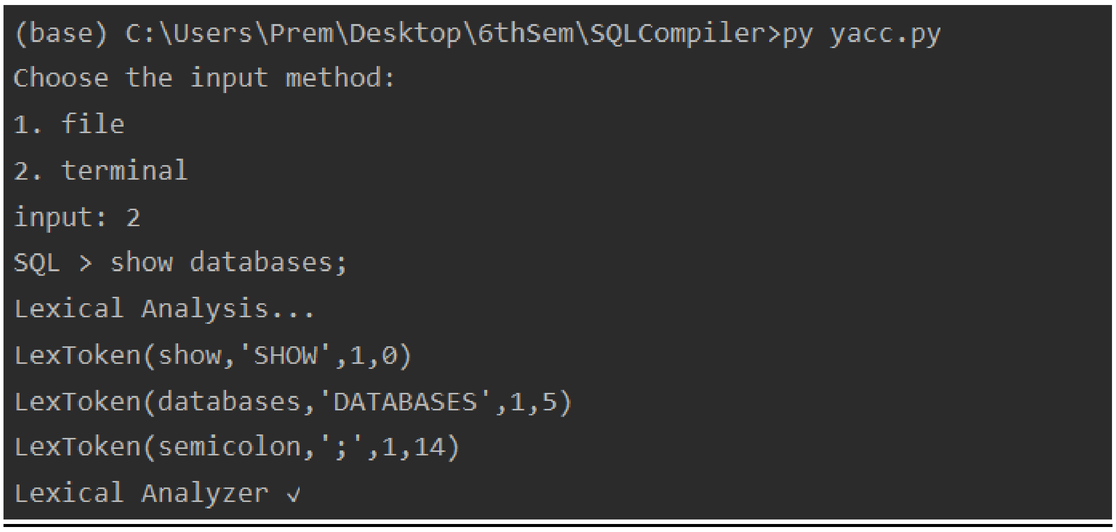
Following are the valid tokens:

```
tokens = [
    'create',
    'use',
    'show',
    'insert',
    'select',
    'update',
    'delete',
    'drop',
    'exit',
    'databases',
    'database',
    'tables',
    'table',
    'into',
    'values',
    'from',
    'all',
    'where',
    'set',
    'compare',
    'logic',
    'char',
    'int',
    'id',
    'number',
    'string',
    'comma',
    'semicolon',
    'left_paren',
    'right_paren'
]
```

And tokens are defined as follows using regular expression -

```
def t_create(t):
    r"""(C|c)(R|r)(E|e)(A|a)(T|t)(E|e)"""
    t.value = "CREATE"
    return t
```

Output for Lexical Analyzer



```
(base) C:\Users\Prem\Desktop\6thSem\SQLCompiler>py yacc.py
Choose the input method:
1. file
2. terminal
input: 2
SQL > show databases;
Lexical Analysis...
LexToken(show,'SHOW',1,0)
LexToken(databases,'DATABASES',1,5)
LexToken(semicolon,';',1,14)
Lexical Analyzer √
```

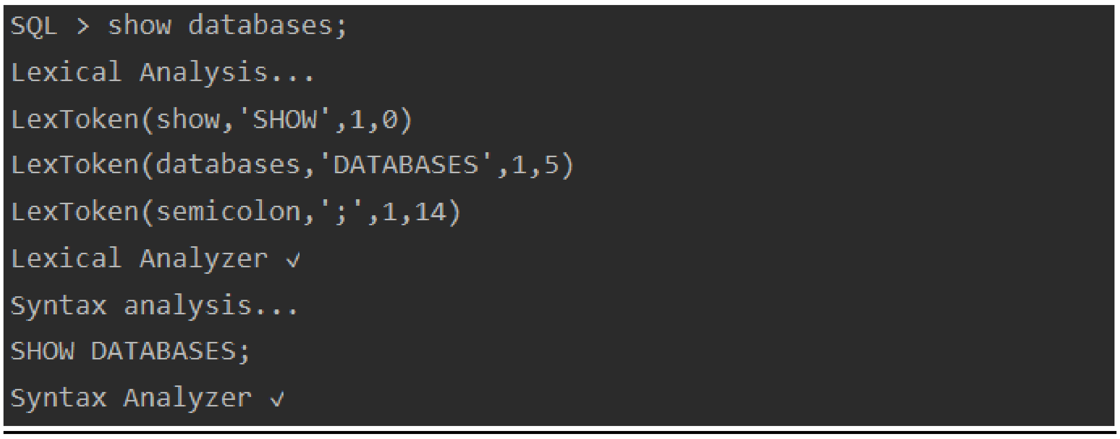
- Syntax Analyzer:

In this phase it receives the inputs, in the form of tokens, from lexical analyzers. Checks if the expressions from these tokens are syntactically correct or not.

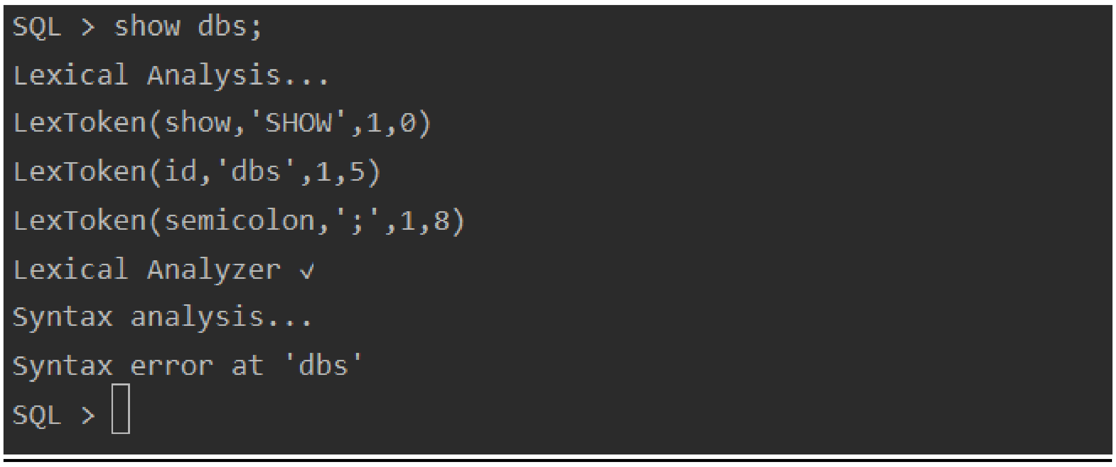
Production rules are as follows -

```
Rule 0     S' -> start
Rule 1     start -> statement
Rule 2     statement -> <empty>
Rule 3     statement -> create_db statement
Rule 4     statement -> show_db statement
Rule 5     statement -> drop_db statement
Rule 6     statement -> use_db statement
Rule 7     statement -> create_tb statement
Rule 8     statement -> show_tb statement
Rule 9     statement -> drop_tb statement
Rule 10    statement -> insert_tb statement
Rule 11    statement -> delete_tb statement
Rule 12    statement -> update_tb statement
Rule 13    statement -> select_tb statement
Rule 14    statement -> exit_db statement
Rule 15    create_db -> create database id semicolon
Rule 16    show_db -> show databases semicolon
Rule 17    drop_db -> drop database id semicolon
Rule 18    use_db -> use id semicolon
Rule 19    create_tb -> create table id left_paren cols right_paren
semicolon
Rule 20    cols -> id type col
Rule 21    type -> int
Rule 22    type -> char left_paren number right_paren
Rule 23    col -> <empty>
Rule 24    col -> comma id type col
Rule 25    show_tb -> show tables semicolon
Rule 26    drop_tb -> drop table id semicolon
Rule 27    insert_tb -> insert into tb_name values left_paren value_cols
right_paren semicolon
Rule 28    tb_name -> id
Rule 29    tb_name -> id left_paren id_cols right_paren
Rule 30    id_cols -> id id_col
Rule 31    id_col -> <empty>
Rule 32    id_col -> comma id id_col
Rule 33    value_cols -> string value_col
Rule 34    value_cols -> number value_col
Rule 35    value_col -> <empty>
Rule 36    value_col -> comma string value_col
Rule 37    value_col -> comma number value_col
Rule 38    delete_tb -> delete from id where conditions semicolon
Rule 39    conditions -> condition_col
Rule 40    conditions -> left_paren conditions right_paren condition
Rule 41    condition_col -> id compare id
Rule 42    condition_col -> id compare number
Rule 43    condition_col -> id compare string
Rule 44    condition_col -> number compare id
Rule 45    condition_col -> number compare string
Rule 46    condition_col -> string compare id
Rule 47    condition_col -> string compare string
Rule 48    condition -> <empty>
Rule 49    condition -> logic left_paren condition_col right_paren
condition
Rule 50    update_tb -> update id set update_cols semicolon
Rule 51    update_tb -> update id set update_cols where conditions
semicolon
Rule 52    update_cols -> id compare number update_col
Rule 53    update_cols -> id compare string update_col
Rule 54    update_col -> <empty>
Rule 55    update_col -> comma id compare number update_col
Rule 56    update_col -> comma id compare string update_col
Rule 57    select_tb -> select all from id_cols semicolon
Rule 58    select_tb -> select id_cols from id_cols semicolon
Rule 59    select_tb -> select all from id_cols where conditions semicolon
Rule 60    select_tb -> select id_cols from id_cols where conditions
semicolon
Rule 61    exit_db -> exit semicolon
```

Output for Syntax Analyzer



```
SQL > show databases;
Lexical Analysis...
LexToken(show,'SHOW',1,0)
LexToken(databases,'DATABASES',1,5)
LexToken(semicolon,';',1,14)
Lexical Analyzer √
Syntax analysis...
SHOW DATABASES;
Syntax Analyzer √
```



```
SQL > show dbs;
Lexical Analysis...
LexToken(show,'SHOW',1,0)
LexToken(id,'dbs',1,5)
LexToken(semicolon,';',1,8)
Lexical Analyzer √
Syntax analysis...
Syntax error at 'dbs'
SQL >
```

- Semantic Analyzer:

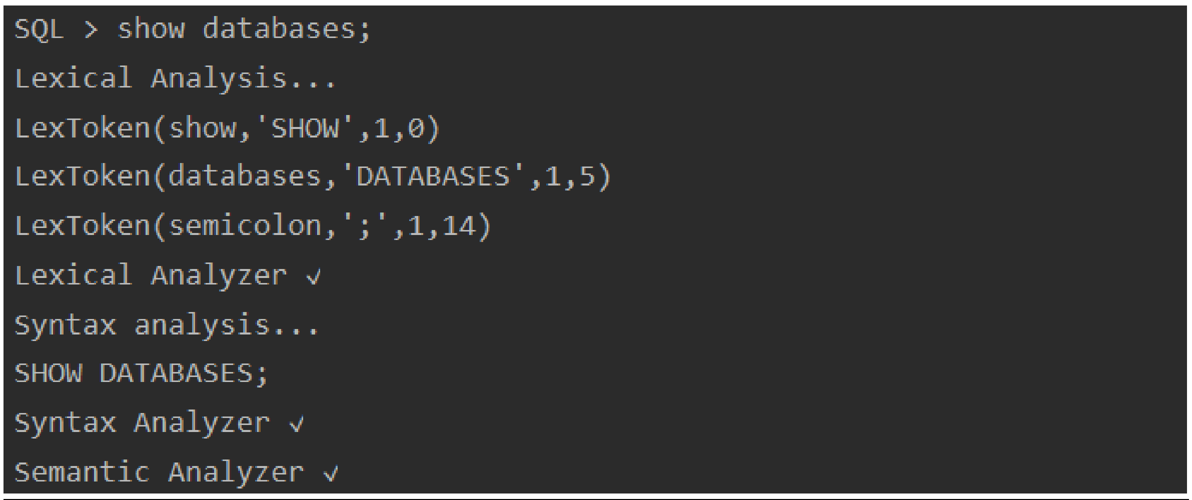
In this phase, we extract necessary semantic information from the P3SQL queries.

We are validating the following things –

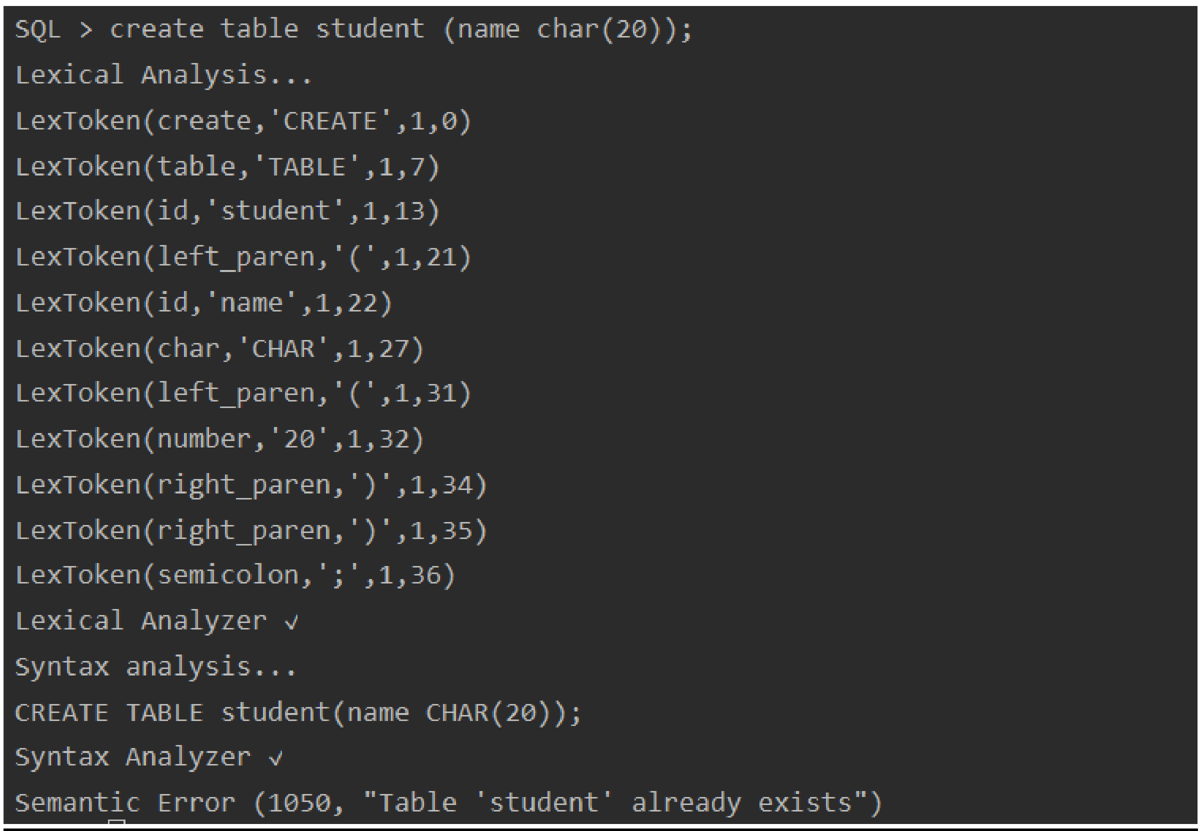
Validation of Databases and Tables if already exists or not.

Validating values for insert and update queries.

Output for Semantic Analyzer



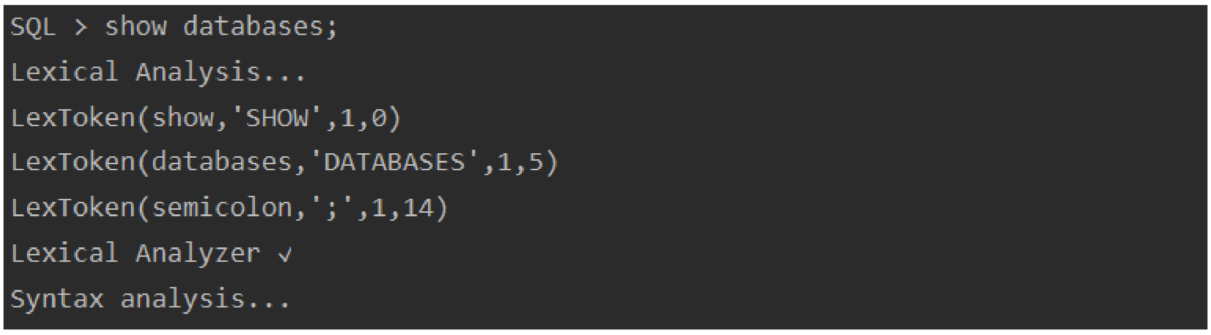
```
SQL > show databases;
Lexical Analysis...
LexToken(show,'SHOW',1,0)
LexToken(databases,'DATABASES',1,5)
LexToken(semicolon,';',1,14)
Lexical Analyzer √
Syntax analysis...
SHOW DATABASES;
Syntax Analyzer √
Semantic Analyzer √
```



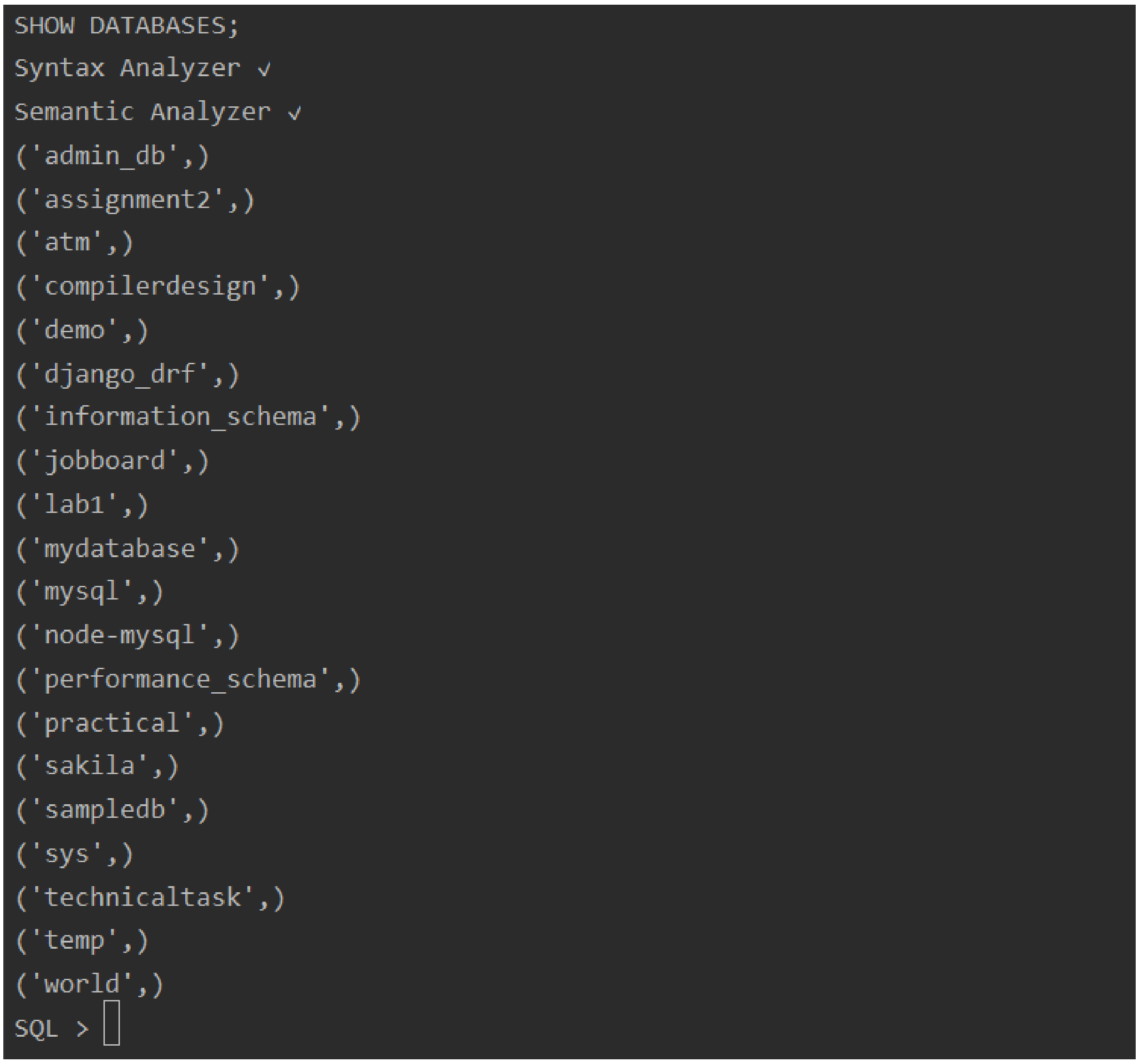
```
SQL > create table student (name char(20));
Lexical Analysis...
LexToken(create,'CREATE',1,0)
LexToken(table,'TABLE',1,7)
LexToken(id,'student',1,13)
LexToken(left_paren,'(',1,21)
LexToken(id,'name',1,22)
LexToken(char,'CHAR',1,27)
LexToken(left_paren,'(',1,31)
LexToken(number,'20',1,32)
LexToken(right_paren,')',1,34)
LexToken(right_paren,')',1,35)
LexToken(semicolon,';',1,36)
Lexical Analyzer √
Syntax analysis...
CREATE TABLE student(name CHAR(20));
Syntax Analyzer √
Semantic Error (1050, "Table 'student' already exists")
```

Execution:

Execution of P3SQL Query:



```
SQL > show databases;
Lexical Analysis...
LexToken(show,'SHOW',1,0)
LexToken(databases,'DATABASES',1,5)
LexToken(semicolon,';',1,14)
Lexical Analyzer ✓
Syntax analysis...
```



```
SHOW DATABASES;
Syntax Analyzer ✓
Semantic Analyzer ✓
('admin_db',)
('assignment2',)
('atm',)
('compilerdesign',)
('demo',)
('django_drf',)
('information_schema',)
('jobboard',)
('lab1',)
('mydatabase',)
('mysql',)
('node-mysql',)
('performance_schema',)
('practical',)
('sakila',)
('sampledb',)
('sys',)
('technicaltask',)
('temp',)
('world',)
SQL > 
```

References

https://www.dabeaz.com/ply/ply.html

https://ply.readthedocs.io/en/latest/