COMPILER DESIGN

PROJECT REPORT

Parser for CREATE and DROP sql commands

Submitted by:

Hardik Chawda

Shubham Paliwal

Sakshi Jain

Submitted to:

Dr. Ankit Rajpal

INDEX

PROBLEM STATEMENT:

The aim of this project is to implement a parser using Lex and Yacc to validate the syntax of Create and Drop commands in SQL.

SQL is a widely used programming language used to manage and manipulate databases. In order to ensure that the SQL commands entered are syntactically correct, a parser can be used to validate the input.

The parser will be able to recognize the following types of SQL commands:

- **CREATE**
- **DROP**

Overall, this project will help to ensure that SQL commands entered are valid and will prevent errors when executing SQL queries

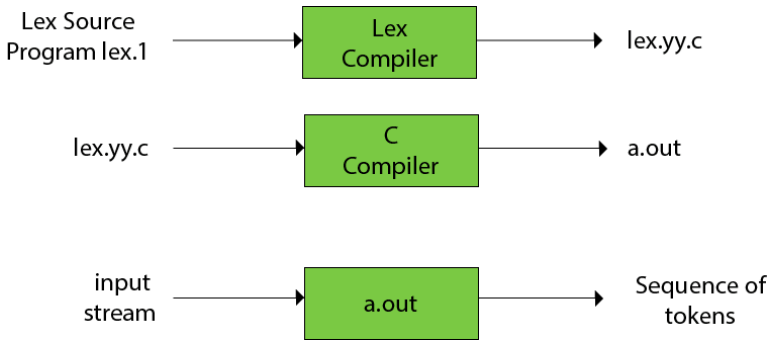
INTRO TO LEX AND YACC

Lex and Yacc are tools used for writing compilers and interpreters for programming languages.

Lex (short for lexical analyzer) is a program that generates a lexical analyzer, which is responsible for breaking input text into tokens. Tokens are the smallest units of meaning in a language, such as keywords, identifiers,

operators, and literals. Lex uses regular expressions to recognize patterns in the input text and converts them into tokens.

- Firstly lexical analyzer creates a program filename.l in the Lex language. Then Lex compiler runs the lfilename.l program and produces a C program lex.yy.c.
- Finally C compiler runs the lex.yy.c program and produces an object program a.out.
- a.out is lexical analyzer that transforms an input stream into a sequence of tokens.



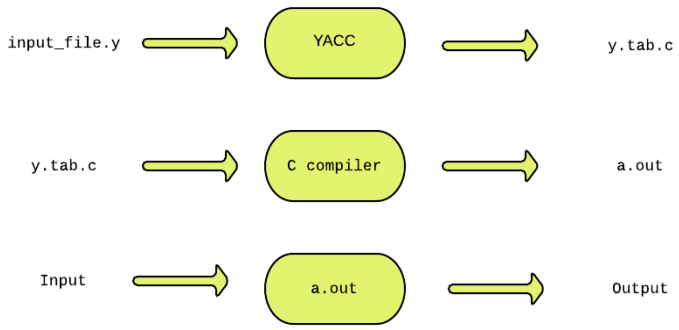
Yacc (short for yet another compiler compiler) is a tool that generates a parser, which is responsible for analyzing the structure of the input text according to the rules of the language's grammar. Yacc uses a formal grammar specification to generate a parser that can recognize and parse the input text according to the language's syntax rules.

These are some points about YACC:

Input: A CFG- file.y

Output: A parser y.tab.c (yacc)

- The output file "file.output" contains the parsing tables.
- The file "file.tab.h" contains declarations.
- The parser called the yyparse ().
- Parser expects to use a function called yylex () to get tokens.



Together, Lex and Yacc provide a powerful toolset for building compilers and interpreters for programming languages, making it easier to detect complex language constructs and generate efficient and accurate parsers.

SYNTAX & ASSUMPTIONS FOR GIVEN COMMANDS

ASSUMED SQL DATATYPES

We have divided attributes into 3 categories for our help , category -A which does not require any additional parameter , category -B which requires 1 additional parameter , category -C which requires 2 additional parameters.

DATATYPE ‘A’

DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
TIME	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
TEXT	Holds a string with a maximum length of 65,535 bytes
BIT	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.

INT	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER	Equal to INT(size)

DATATYPE ‘B’

FLOAT(*size*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum string length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1

DATATYPE ‘C’

DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
NUMERIC(precision, scale)	Here, numeric is a data type that takes 5-17 bytes storage. Precision or p is an integer representing the total number of allowed digits in the specified column.

CREATE DATABASE

The **CREATE DATABASE** command is used to create a new SQL database.

The following SQL query creates a database called "DUCSDATA":

```
CREATE DATABASE DATABASE_NAME;
```

```
CREATE DATABASE SCHEMA.DATABASENAME;
```

CREATE TABLE

The **CREATE TABLE** command creates a new table in the database.

The following SQL creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

```
CREATE TABLE Persons (
```

```
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

```
CREATE TABLE SCHEMA.DATABASENAME.TABLENAME(
    ...
);
```

PRIMARY KEY

A Primary key is a unique column we set in a table to easily identify and locate data in queries. A table can have only one primary key.

The primary key column has a unique value and doesn't store repeating values. A Primary key can never take NULL values.

For example, in the case of a student when identification needs to be done in the class, the roll number of the student plays the role of Primary key.

```
CREATE TABLE tableName (
    col1 int NOT NULL,
     col2 varchar(50) NOT NULL,
    col3 int,
    CONSTRAINT PK_Person PRIMARY KEY (ID)
);
```

ANOTHER WAY:

```
CREATE TABLE Persons (
     ID int NOT NULL PRIMARY KEY,
     LastName varchar(255) NOT NULL,
     FirstName varchar(255),
     Age int
);
```

FOREIGN KEY

A Foreign key is beneficial when we connect two or more tables so that data from both can be put to use parallelly.

A foreign key is a field or collection of fields in a table that refers to the Primary key of the other table. It is responsible for managing the relationship between the tables.

The table which contains the foreign key is often called the child table, and the table whose primary key is being referred by the foreign key is called the Parent Table.

For example: When we talk about students and the courses they have enrolled in, now if we try to store all the data in a single table, the problem of redundancy arises.

```
CREATE TABLE childTable (

    col1 int NOT NULL,

    col2 int NOT NULL,

    col3 int,

    CONSTRAINT PK PRIMARY KEY (col1),

    CONSTRAINT FK FOREIGN KEY (col3) REFERENCES

    parentTable(parent_Primarykey)

);
```

ANOTHER WAY:

```
CREATE TABLE Orders (
     OrderID int NOT NULL PRIMARY KEY,
     OrderNumber int NOT NULL,
     PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

ASSUMPTIONS

There are few assumptions that we have made while creating tables and databases.these are as follows:

- Table name,Column name,database name can not be “ ”(empty).
- Table & Database and attribute name should not start with digit(0-9) , and can only contain alphabets, underscore and digits.
- Few constraints are provided such as foreign key, primary key ,not null , unique and default.
- Rules like “NOT USING KEYWORDS TO NAME TABLES,COLUMNS AND DATABASES” will be considered as misleading syntax.
- More than one constraints are allowed on a column in a table.
- Order of constraints are as follows: Not NULL, Unique, Primary Key, Foreign Key and in last Default.
- If the user gives the constraint in different order, our parser will generate the syntax error.
- Both the syntaxes of primary key and foreign key are assumed to be correct.
- Data types provided should be the ones which we’ve mentioned above.
- Semicolon is Compulsory.
- Check for the range of data types like varchar and char is not assured.
- Special characters other than { , ; ( ) ” } are not allowed

DROP DATABASE

The **DROP DATABASE** command is used to delete an existing SQL database.

The following SQL drops a database named "testDB":

```
DROP DATABASE testDB;
```

DROP TABLE

The DROP TABLE command deletes a table in the database.

The following SQL deletes the table "Shippers":

```
DROP TABLE Shippers;
```

ASSUMPTIONS

There are few assumptions that we have made while using DROP for tables and databases.these are as follows:

- Table name, database name can not be “ ”(empty).
- Table & Database names should not start with digit(0-9) and can only contain alphabets underscore and digit.
- Rules like “NOT USING KEYWORDS TO NAME TABLES,COLUMNS AND DATABASES” will be considered as misleading syntax.
- Semicolon is Compulsory.

LOGIC

To implement a parser for Create and Drop commands in SQL using Lex and Yacc, we can follow these steps:

1. **Define a grammar:** We need to define a grammar that describes the syntax of Create and Drop commands in SQL. The grammar should include rules for all the valid syntax constructs and should also detect any possible error scenarios. For example, the grammar could include rules for table and database creation, handling of invalid syntax & errors etc.
2. **Write a Lex file:** The next step is to write a Lex file that can tokenize input based on the defined grammar. The Lex file should read input from the user, and tokenize it based on the defined rules in the grammar. The tokens generated by Lex can then be passed to Yacc for further analysis.
3. **Write a Yacc file:** The Yacc file should take the tokens generated by Lex and use them to build a parse tree based on the defined grammar. The Yacc file should detect any possible error scenarios and generate error messages accordingly.
4. **Build the parser:** Once the Lex and Yacc files have been written, we need to build the parser by running them through Lex and Yacc. This will generate a parser executable that can be used to validate the syntax of Create and Drop commands in SQL.

5. **Test the parser:** We can do this by providing input that is valid according to the grammar and input that contains syntax errors. The parser should correctly identify errors and generate error messages accordingly.0

GRAMMAR:

```
/*start of grammar rules */
S : {command_start_line = yylloc.first_line; command_start_col =
yylloc.first_column;} CREATE C Semicolon  S
  | {command_start_line = yylloc.first_line; command_start_col =
yylloc.first_column;} DROP  D Semicolon  S
  |/*empty*/
  ;
// Error recovery i user misses to write semicolon
Semicolon : ';' { if( valid == 1) printf("\nGiven command from line
%d:%d-%d:%d is
valid!!\n",command_start_line,command_start_col,yylloc.last_line,yylloc.la
st_column); valid = 1;  /*for case when user enters multiple command*/}
          | { if( yychar == 0) {yyerror("missing semicolon(;).");} else
{yyerror("you might be missing semicolon(;) or there is a syntax
error.\nMisleading syntax, parsing stopped!!!"); }  exit(0);}
          ;
// rules for drop database | table
D : DATABASE SingleDBName      { if( name_provided == 0)  yyerror("you
missed Database name."); }
  | TABLE SingleTableName    { if( name_provided == 0)  yyerror("you
missed table name."); }
  | DATABASE NUM             { yyerror("invalid database name,expecting
Identifier number provided");} //rule when number is passed as database
name
  | TABLE NUM                { yyerror("invalid table name,expecting
Identifier number provided");}  //rule when number is passed as table name
  | DATABASE mulDBName         { yyerror("multiple DATABASE name provided,
command requires only 1."); }  /*to detect multiple name errors*/
```

```
  | TABLE mulTableName       { yyerror("multiple TABLE name provided,
command requires only 1.");  }    /*to detect multiple name errors*/
  ;
/*here we have duplication in rules because we want to be sepcific what
error to show
database name missing or table name? , multiple name for database or
table..
*/
C : DATABASE SingleDBName  { if( name_provided == 0)  yyerror("you missed
database name."); }
  | DATABASE mulDBName     { yyerror("multiple Database name provided,
command requires only 1.");  }       {/*to detect multiple name errors*/}
  | TABLE  SingleTableName { if( name_provided == 0){ checkTableName();
yyerror("you missed table name.");} } '('  A {checkCommaError(); } ')'
  | TABLE  mulTableName  { yyerror("multiple TABLE name provided, command
requires only 1.");  } '(' A {checkCommaError(); } ')'
  ;
//grammar rules when multiple table name are provided in place of 1
mulTableName :  ID DbName
  | DbName DbName
  | ID ID
  | SchDBName DbName
  | DbName SchDBName
  | SchDBName ID
  | ID SchDBName
  | SchDBName SchDBName
  | DbName ID
  |  ID mulTableName
  | DbName mulTableName
  | SchDBName mulTableName
  ;
//grammar rules for table name
SingleTableName : ID   { name_provided = 1;  }
  | DbName   { name_provided = 1;  }
  | SchDBName  { name_provided = 1;  }
  | /*empty*/     { name_provided = 0;  }
  ;
```

```
//grammar rules for database name
SingleDBName : ID    { name_provided = 1;  }
  | DbName    { name_provided = 1;  }
  | /*empty*/      { name_provided = 0;  }
  ;
//grammar rules for multiple database name
mulDBName : DbName DbName
  | ID ID
  | DbName ID
  | ID DbName
  | ID mulDBName
  | DbName mulDBName
  ;
//grammar rule for attribute and database name and all other that reuired
ID
SingleName : ID    { name_provided = 1;  }
  | /*empty*/      { name_provided = 0;  }
  ;
mulName : ID mulName //grammar rule when multiple attribute and database
name and all other that reuired ID
  | ID ID
  ;
//rule for attributes declaration of tables
A : Attr ',' A
  | Attr /*for the last attribute which don't require ',' */
  | error //to detect errors raised by invalid tokens during attribute
declaration
  ;
Attr : SingleName    { if( name_provided == 0) { checkAttributeName();
yyerror("you missed attribute name");}}  DAT {  if(name_provided==0 &&
datatype_provided==0){yyerror("Invalid attribute declaration\nMisleading
syntax, parsing stopped!!!");exit(0);}} NN Unq PK FK  DF
     | mulName       { yyerror("multiple attribute name provided, command
requires only 1.");  } DAT NN Unq PK FK DF
     | NUM           { { yyerror("Invalid attribute name,expecting
Identifier number provided");}}  DAT {  if(name_provided==0 &&
```

```
datatype_provided==0){yyerror("Invalid attribute declaration\nMisleading
syntax, parsing stopped!!!");exit(0);}} NN Unq PK FK  DF
     | CPK    //non terminal for primary key constraint
     | CFK    //non terminal for foreign key constraint
     ;
// rule for datatypes
DAT : DATATYPE_A                      {datatype_provided = 1; }
    | DATATYPE_B '(' NUM ')'            {datatype_provided = 1; }
    | DATATYPE_C '(' NUM COMMA NUM ')'  {datatype_provided = 1; }
    | /*error*/                       {datatype_provided =
0;yyerror("missing datatype.");}
    ;
NN : NOT NULLL {}
   | /*empty*/ {}
   ;
//rule for primnAry key
PK : PRIMARY KEY {}
   | /*empty*/ {}
  ;
//rule for unique key
Unq : UNIQUE
    |/*empty*/
    ;
//rule for foreign key
FK : FOREIGN KEY REFERENCES SingleTableName {if( name_provided == 0){
checkTableName(); yyerror(" missed table name for Reference");}} '('
SingleName {if( name_provided == 0)  yyerror("missed column name for
references.");} ')'
   | /*empty*/
   ;
//rule for default
DF : DEFAULT Val
   | /*empty*/
   ;
//rule for default values it can be a string value(ID) , function() or an
integer(NUM)
Val : NUM
```

```
    | ID '(' ')'
    | '"' ID '"'
    | '"' NUM '"'
    ;
//grammar rules for primary constraint
CPK : CONSTRAINT SingleName  {if( name_provided ==
0){checkPKconstraintName(); yyerror("missing constraint name ");}} PRIMARY
KEY '(' COL ')'
    | CONSTRAINT mulName  {yyerror("multiple constraint name provided,
command requires only 1.");} PRIMARY KEY '(' COL ')'
    ;
//grammar rules for foreign key constraint
CFK : CONSTRAINT SingleName  {if( name_provided ==
0){checkFKconstraintName(); yyerror("missing constraint name ");}} FOREIGN
KEY '(' COL ')' REFERENCES SingleTableName {if( name_provided == 0){
checkTableName(); yyerror("you missed table name.");}} '(' COL ')'
    | CONSTRAINT mulName  {yyerror("multiple constraint name provided,
command requires only 1.");} FOREIGN KEY '(' COL ')' REFERENCES
SingleTableName {if( name_provided == 0){ checkTableName(); yyerror("you
missed table name.");}} '(' COL ')'
    ;
//grammar rule for defining multiple columns seprated by comma for
constraint
COL : ID COMMA COL { /*COL.count = COL1.count + 1;*/}
    | ID         { /*COL.count = 1;*/}
    ;
//rule for error recovery is user misses comma(,)
COMMA : ','
    | /* error */ {yyerror("missing comma.");}
    ;
```

GRAPH:

`graphviz.svg`

Click on the above link to see the go-to graph for this grammar.

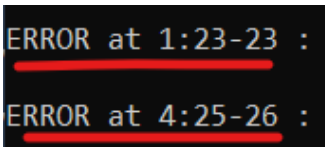
ABOUT OUR PARSER:

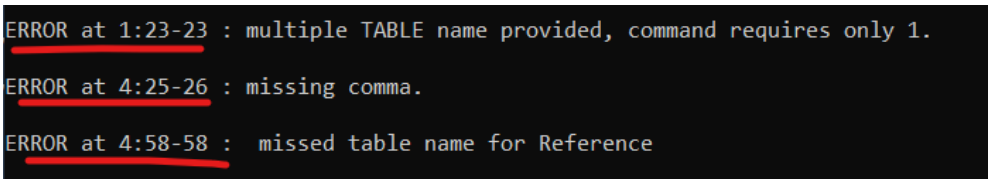
- IT has ***0 conflicts.***

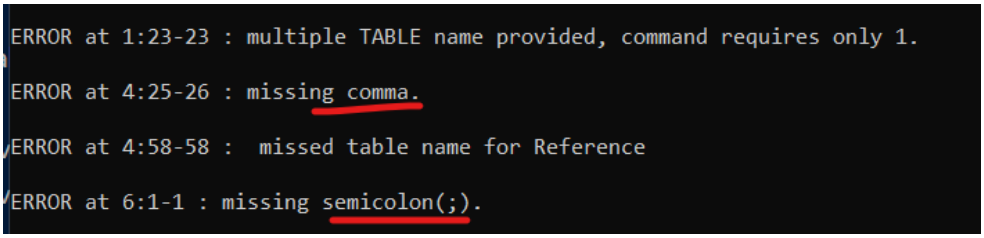
- This parser is able to detect various syntax errors and provide meaningful error messages to the user.

- This parser is designed to be extensible, making it easy to add new commands in the future.

- This Parser Precisely tells the user where the exact error occurred by pointing out the line number and column number.
    - for eg



- It Tries to find and list multiple errors to a user in single parsing , which helps users such that they can fix all errors, before compiling next time.
    - For eg
    - 

- Sometimes while writing complex and larger commands, users may forget little things like semicolon, open and closing parenthesis, and comma between attributes, our parser guides users by taking care of these syntax mistakes.
    - For eg
    - 

- Users can parse multiple commands in single parsing, only need to end all commands by semicolon.
    - For eg

```
C:\Users\shubham\OneDrive\Documents\4th sem\compiler design\cd_project>a

Given String :
drop database d ;
drop database Sch1.d ;
drop database ;
create table d();
create table shc2.db1.d (
    A int not null,
    ab int not null Primary Key,
    ab int Primary Key,
    ab int ,
);



Given command from line 1:1-1:20 is valid!!

Given command from line 2:1-2:25 is valid!!

ERROR at 3:17-17 : you missed Database name.

ERROR at 4:18-18 : syntax error, unexpected ), expecting Identifier
```

TEST CASES:

CREATE DATABASE

```
CREATE DATABASE S.DB;
```

```
Given String :
CREATE DATABASE S.DB;


Given command from line 1:1-1:23 is valid!!

C:\Users\Lenovo\Desktop\compiler design\project>
```

```
CREATE DATABASE 123;
```

```
Given String :
CREATE DATABASE 123;

ERROR at 1:17-19 : you missed database name.

ERROR at 1:17-19 : you might be missing semicolon(;) or there is a syntax error.
Misleading syntax, parsing stopped!!!

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```
CREATE DATABASE D1();
```

```
Given String :
CREATE DATABASE D1 ();


ERROR at 1:23-23 : you might be missing semicolon(;) or there is a syntax er
ror.
Misleading syntax, parsing stopped!!!

C:\Users\Lenovo\Desktop\compiler design\project>
```

CREATE TABLE

```
CREATE TABLE T(
    AGE INT,
    NAME VARCHAR,
    ADDR VARCHAR(255),
    ID INT PRIMARY KEY;
    CONSTRAINT P PRIMARY KEY(ID,ID),
    CONSTRAINT F FOREIGN KEY (ADDR) REFERENCES ()
);
```

```
Given String :
CREATE TABLE T(
    AGE INT,
    NAME VARCHAR,
    ADDR VARCHAR(255),
    ID INT PRIMARY KEY;
    CONSTRAINT P PRIMARY KEY(ID,ID),
    CONSTRAINT F FOREIGN KEY (ADDR) REFERENCES ()
);


ERROR at 3:22-22 : syntax error, unexpected ',', expecting '('

Misleading syntax, parsing stopped!!!

C:\Users\Lenovo\Desktop\compiler design\project>
```

```
CREATE TABLE T(
    AGE INT,
    ADDR VARCHAR(255),
    ID INT PRIMARY KEY,
    CONSTRAINT FK FOREIGN KEY (NAME) REFERENCES (NAME2,)
);
```

```
C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>a

Given String :
CREATE TABLE T(
    AGE INT,
    ADDR VARCHAR(255),
    ID INT PRIMARY KEY,
    CONSTRAINT FK FOREIGN KEY (NAME) REFERENCES (NAME2,)
);

ERROR at 5:49-49 : you missed table name.

ERROR at 5:56-56 : syntax error, unexpected ')', expecting ID

Misleading syntax, parsing stopped!!!

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```sql
CREATE TABLE T(
    A INT,
    B FLOAT(3) NOT NULL,
    C DECIMAL(4,5) PRIMARY KEY,
    D INT FOREIGN KEY REFERENCES T(C1),
    E CHAR(12) NOT NULL PRIMARY KEY FOREIGN KEY REFERENCES T(C2),
    F INT NOT NULL UNIQUE,
    CONSTRAINT F PRIMARY KEY (A,B),
    CONSTRAINT G FOREIGN KEY (A,B) REFERENCES (C1,C2)
);
```

```
Given String :
CREATE TABLE T(
    A INT,
    B FLOAT(3) NOT NULL,
    C DECIMAL(4,5) PRIMARY KEY,
    D INT FOREIGN KEY REFERENCES T(C1),
    E CHAR(12) NOT NULL PRIMARY KEY FOREIGN KEY REFERENCES T(C2),
    F INT NOT NULL UNIQUE,
    CONSTRAINT F PRIMARY KEY (A,B),
    CONSTRAINT G FOREIGN KEY (A,B) REFERENCES (C1,C2)
);

ERROR at 9:47-47 : you missed table name.

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```
CREATE TABLE T(
    A INT PRIMARY
)
```

```
Given String :
CREATE TABLE T(
    A INT PRIMARY
)

ERROR at 3:1-1 : syntax error, unexpected ')', expecting KEY

Misleading syntax, parsing stopped!!!

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```
CREATE TABLE T(
    A INT FOREIGN KEY
```

```
);
```

```
Given String :
CREATE TABLE T(
    A INT FOREIGN KEY
);

ERROR at 3:1-1 : syntax error, unexpected ')', expecting REFERENCES

Misleading syntax, parsing stopped!!!

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```
CREATE TABLE(CREATE DATABASE);
```

```
Given String :
CREATE TABLE(CREATE DATABASE);
ERROR at 1:13-13 : you missed table name.

ERROR at 1:14-19 : syntax error, unexpected CREATE, expecting Identifier

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```
Given Commands:
CREATE TABLE T(
    ID INT,
    AGE INT,
    NAME VARCHAR(255),
    MARKS FLOAT(255),
    POINT DECIMAL(2,3),
    CONSTRAINT PK PRIMARY KEY (ID),
    CONSTRAINT FK FOREIGN KEY (POINT) REFERENCES sc1.db.t2 (POINT)
);
Given command from line 1:1-9:2 is valid!!
```

```
CREATE TABLE sc2.db3.T(
    ID INT,
    AGE INT,
    NAME VARCHAR(255),
    MARKS FLOAT(255),
    POINT DECIMAL(2,3),
    CONSTRAINT PK PRIMARY KEY (ID),
    CONSTRAINT FK FOREIGN KEY (POINT) REFERENCES sc1.db.t2 (POINT)
);
Given command from line 1:1-9:2 is valid!!
```

```
C:\Users\hp\Downloads\shubham>a

Given String :
create table t(
    a int default
);


ERROR at 3:1-1 : syntax error, unexpected ')', expecting ID or NUM or '"'

Misleading syntax, parsing stopped!!!
```

```
C:\Users\hp\Downloads\shubham>a

Given String :
create table t(
    a int default 8
);


Given command from line 1:1-3:2 is valid!!
```

DROP DATABASE

```
Drop DATABASE db;
```

```
C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>a

Given String :
Drop DATABASE db;
Given command from line 1:1-1:17 is valid!!

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```
DROP DATA@BASE D;
```

```
Given String :
DROP DATA@BASE D;
ERROR at 1:6-9 : syntax error, unexpected ID, expecting DATABASE or TABLE

Misleading syntax, parsing stopped!!!
```

```
DROP DATABASE DATABASE;
```

```
Given String :
DROP DATABASE DATABASE;
ERROR at 1:15-22 : you missed Database name.

ERROR at 1:15-22 : you might be missing semicolon(;) or there is a syntax error.
Misleading syntax, parsing stopped!!!

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```
DROP DATABASE 1 D2 D3;
```

```
Given String :
DROP DATABASE 1 D2 D3;
ERROR at 1:15-15 : invalid database name,expecting Identifier number provided

ERROR at 1:17-18 : you might be missing semicolon(;) or there is a syntax error.
Misleading syntax, parsing stopped!!!

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

DROP TABLE

```
DROP ;
```

```
Given String :
DROP ;
ERROR at 1:6-6 : syntax error, unexpected ';', expecting DATABASE or TABLE

Misleading syntax, parsing stopped!!!

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```
drop TABLE  a23;
```

```
C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>a

Given String :
drop TABLE  a23;
Given command from line 1:1-1:16 is valid!!

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```
Drop TABLE a34 b123;
```

```
C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>a

Given String :
Drop TABLE a34 b123;
ERROR at 1:20-20 : multiple TABLE name provided, command requires only 1.

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

```
Drop TABLE 123;
```

```
C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>a

Given String :
Drop TABLE 123;
ERROR at 1:12-14 : invalid table name,expecting Identifier number provided

C:\Users\Hardik\OneDrive\Desktop\COMPILER PROJECT>
```

CONCLUSION:

In conclusion, the parser project developed using Lex and Yacc has been successful in parsing the “Create” and “Drop” SQL commands. Lex was used to perform lexical analysis and generate tokens, while Yacc was used to perform syntax analysis and generate a parse tree. The parser was able to detect various syntax errors and provide meaningful error messages to the user. Additionally, the parser was designed to be extensible, making it easy to add new commands in the future. Overall, the parser project has demonstrated the power and flexibility of Lex and Yacc in building efficient and effective parsers for complex languages such as SQL.