BCSE III-Compiler Design Lab Report

Assignment V, Project -6

Team Members:-

Roll:	Name:
1.	Siddhartha Pal
6.	Anindya Chakraborty
14.	Praneet Sikdar
27.	Abhishek De

Problem statement:

An SQL parser can check syntax errors in SQL statement. Consider a simple set of SQL statement consisting of only CREATE TABLE ,INSERT and SELECT. The SELECT statement supports “where” clause with the following operator=,>,<, BETWEEN,LIKE,IN. No nesting of SELECT statement are supported.

For CREATE TABLE, only the following format is supported:

CREATE TABLE table_name(

Column1 datatype,

Column2 datatype,

Column3 datatype,

.....

PRIMARY KEY(column)

);

For INSERT statement ,the following formats are supported:

INSERT INTO table_name(column1,column2,column3,.....)

VALUES(value1,value2,value3,.....);

Or

INSERT INTO table_name

VALUES(value1,value2,value3,.....);

Part I- Construct a CFG for this language.Part II- Write a lexical analysis to scan the stream of characters from a SQL.

Query(as above) and generate stream of tokens.

Part III-Write a top-down parser to detect syntax errors in the SQL queries(modules include FIRST, FOLLOW, parsing table construction and parsing).

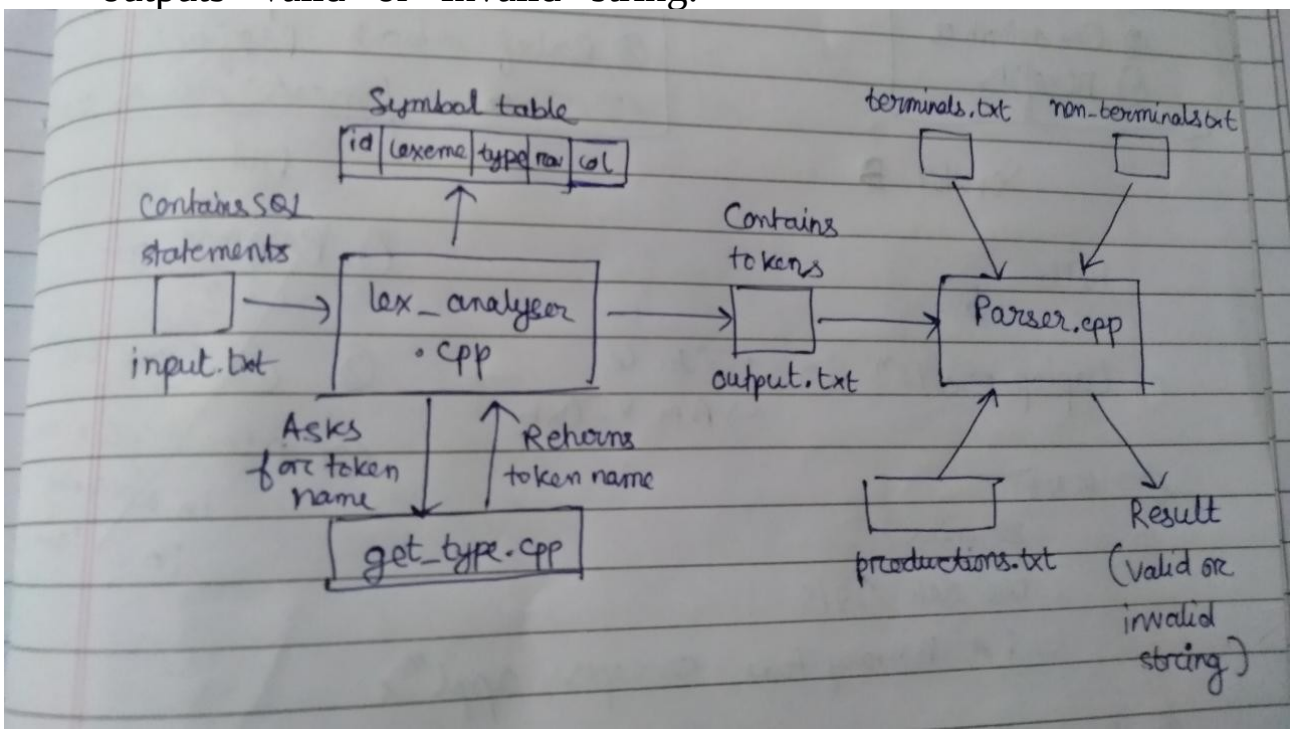
DESIGN DETAILS

The program consists of the following cpp files :

- **lex_analyser.cpp**
  This program reads SQL statements from input.txt, and creates the symbol table, and stores the tokens in output.txt. To determine the token type from the lexeme it calles the function getType() from get_type.cpp.
- **get_type.cpp**
  It contains the function getType() which takes the lexeme as input and determines the token type. The token types are as follows:
  - For keywords such as CREATE, INSERT, etc, token name is same as lexeme.
  - For punctuations, token names are as follows:
    - * STAR
    - , COMMA
    - ; EOS
    - (BOPEN
    -) BCLOSE
    - >,<,= RELOP
  - For CHAR, VARCHAR, NUMBER, token name is datatype
  - For integer, float, quoted string, identifier(table or attribute name), token names are INTEGER, FLOAT, STRING, ID respectively. These types are determined by the final state of a DFA.
- **parser.cpp**
  Performs the function of an LL(1) parser. Contains the class Parser, with the following methods:
- **get_terminals(string);**
  Reads the set of terminal symbols from a file(defaulted to teminals.txt)
- **get_non_terminals(string);**
  Reads the set of non-terminal symbols from a file(defaulted to non_teminals.txt)
- **get_productions(string);**
  Reads the set of production rules from a file(defaulted to productions.txt) and stores in a map
- **calculate_first();**
  Calculates FIRST of terminal and non-terminal symbols.
- **calculate_follow();**

Calculates FOLLOW of non-terminal symbols.

- **create_parsing_table();**
  Creates the parsing table
- **show_details();**
  Displays the FIRST, FOLLOW and parsing table
- **parse_input(string);**
  Reads string from a file, uses the parsing table and a stack and outputs “Valid” or “Invalid” string.



GRAMMAR USED

Start symbol: stmt
Terminal symbols

```
terminals.txt
CREATE
TABLE
INSERT
INTO
SELECT
FROM
PRIMARY
KEY
VALUES
BETWEEN
LIKE
IN
WHERE
AND
RELOP
STAR
COMMA
BOPEN
BCLOSE
EOS
TYPE
INTEGER
FLOAT
ID
STRING
```

Non-terminal symbols

```
non_terminals.txt
stmt
S
S1
DATATYPE
S2
S6
VALS
DATA
S3
IDS
S4
S5
S7
S8
S9
S10
S11
S12
```

Production rules

productions.txt

```
stmt : S EOS ;

S : CREATE TABLE ID BOPEN S1 S10 BCLOSE | INSERT INTO ID S2 | SELECT S3 FROM S4 S5 ;

S1 : ID DATATYPE COMMA S11 ;

S10 : PRIMARY KEY BOPEN ID BCLOSE ;

S11 : ID DATATYPE COMMA S11 | eps ;

DATATYPE : TYPE BOPEN INTEGER BCLOSE ;

S2 : VALUES BOPEN VALS BCLOSE | BOPEN S6 BCLOSE ;

S6 : ID S12 ;

S12 : BCLOSE VALUES BOPEN DATA | COMMA S6 COMMA DATA ;

VALS : DATA S8 ;

DATA : INTEGER | FLOAT | STRING ;

S3 : STAR | IDS ;

IDS : ID S9 ;

S4 : IDS ;

S5 : WHERE ID S7 | eps ;

S7 : RELOP DATA | BETWEEN DATA AND DATA | IN BOPEN VALS BCLOSE | LIKE STRING ;

S8 : COMMA DATA S8 | eps ;

S9 : COMMA ID S9 | eps ;
```

CONVENTIONS USED

- For simplicity, we have used 3 datatypes for attributes, NUMBER, CHAR and VARCHAR.
- In production rules, “eps” denotes epsilon.
- Multiple SQL statemens can be written in the input file(input.txt), anything written upto a semicolon is taken as a statement.
- First symbol in the non-terminals file is taken as start symbol.
- A valid identifier is one which contains atleast an alphabet or underscore.

COMPILATION AND EXECUTION DETAILS

- To compile and run the lexical analyser, from the project folder:
  make run_lex
- To compile and run the parser, from the project folder:
  make run_parser
- To compile and run both, sequentially:
  make run
- To clean the .out files:
  make clean
- Edit the input.txt file for new input.

RESULTS

input.txt

```
CREATE TABLE STUDENT
(
    NAME VARCHAR(30),
    DEPT VARCHAR(25),
    ROLL NUMBER(20),
    PRIMARY KEY(ROLL)
);

INSERT INTO STUDENT VALUES("ABHIK", "CSE", 224);

INSERT INTO STUDENT(NAME, ROLL) VALUES ("SUPRIYA", 24);

SELECT * FROM STUDENT WHERE ROLL > 30;

SELECT NAME, ROLL FROM STUDENT WHERE ROLL BETWEEN 50;

INSERT INTO STUDENT(NAME, ROLL) VALUES ("ROHAN", "CIVIL", 24);

CREATE TABLE 453
(
    NAME VARCHAR(30),
    DEPT VARCHAR(25),
    ROLL NUMBER(20),
    PRIMARY KEY(ROLL)
);
```

Symbol table(truncated due to large size)

Token ID	Token type	Lexeme	Row	Column
1	CREATE	CREATE	0	0
2	TABLE	TABLE	0	7
3	ID	STUDENT	0	13
4	BOPEN	(	1	0
5	ID	NAME	2	4
6	TYPE	VARCHAR	2	9
7	BOPEN	(	2	16
8	INTEGER	30	2	17
9	BCLOSE	)	2	19
10	COMMA	,	2	20
11	ID	DEPT	3	4
12	TYPE	VARCHAR	3	9
13	BOPEN	(	3	16
14	INTEGER	25	3	17
15	BCLOSE	)	3	19

First and Follow

```
FIRST of AND: AND
FIRST of BCLOSE: BCLOSE
FIRST of BETWEEN: BETWEEN
FIRST of BOPEN: BOPEN
FIRST of COMMA: COMMA
FIRST of CREATE: CREATE
FIRST of DATA: FLOAT INTEGER STRING
FIRST of DATATYPE: TYPE
FIRST of EOS: EOS
FIRST of FLOAT: FLOAT
FIRST of FROM: FROM
FIRST of ID: ID
FIRST of IDS: ID
FIRST of IN: IN
FIRST of INSERT: INSERT
FIRST of INTEGER: INTEGER
FIRST of INTO: INTO
FIRST of KEY: KEY
FIRST of LIKE: LIKE
FIRST of PRIMARY: PRIMARY
FIRST of RELOP: RELOP
FIRST of S: CREATE INSERT SELECT
FIRST of S1: ID
FIRST of S10: PRIMARY
FIRST of S11: ID eps
FIRST of S12: BCLOSE COMMA
FIRST of S2: BOPEN VALUES
FIRST of S3: ID STAR
FIRST of S4: ID
FIRST of S5: WHERE eps
FIRST of S6: ID
FIRST of S7: BETWEEN IN LIKE RELOP
FIRST of S8: COMMA eps
```

```
FIRST of S9: COMMA eps
FIRST of SELECT: SELECT
FIRST of STAR: STAR
FIRST of STRING: STRING
FIRST of TABLE: TABLE
FIRST of TYPE: TYPE
FIRST of VALS: FLOAT INTEGER STRING
FIRST of VALUES: VALUES
FIRST of WHERE: WHERE
FIRST of eps: eps
FIRST of stmt: CREATE INSERT SELECT

FOLLOW of DATA: AND BCLOSE COMMA EOS
FOLLOW of DATATYPE: COMMA
FOLLOW of IDS: EOS FROM WHERE
FOLLOW of S: EOS
FOLLOW of S1: PRIMARY
FOLLOW of S10: BCLOSE
FOLLOW of S11: PRIMARY
FOLLOW of S12: BCLOSE COMMA
FOLLOW of S2: EOS
FOLLOW of S3: FROM
FOLLOW of S4: EOS WHERE
FOLLOW of S5: EOS
FOLLOW of S6: BCLOSE COMMA
FOLLOW of S7: EOS
FOLLOW of S8: BCLOSE
FOLLOW of S9: EOS FROM WHERE
FOLLOW of VALS: BCLOSE
FOLLOW of stmt: $
```

Parsing table

(Table contains the index number of the corresponding rule, -1 for no rule)

_		$	AND	BCLOSE	BETWEEN	BOPEN	COMMA	CREATE	EOS	FLOAT	FROM	ID	IN	INSERT	INTEGER	INTO	KEY	L
IKE	PRIMARY	RELOP	SELECT	STAR	STRING	TABLE	TYPE	VALUES	WHERE									
DATA		-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	0	-1	-1	-
1	-1	-1	-1	-1	2	-1	-1	-1	-1									
DATATYPE			-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-
1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1								
IDS		-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-
1	-1	-1	-1	-1	-1	-1	-1	-1	-1									
S		-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	1	-1	-1	-1	-
1	-1	-1	2	-1	-1	-1	-1	-1	-1									
S1		-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-
1	-1	-1	-1	-1	-1	-1	-1	-1	-1									
S10		-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-
1	0	-1	-1	-1	-1	-1	-1	-1	-1									
S11		-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-
1	1	-1	-1	-1	-1	-1	-1	-1	-1									
S12		-1	-1	0	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-
1	-1	-1	-1	-1	-1	-1	-1	-1	-1									
S2		-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-
1	-1	-1	-1	-1	-1	-1	-1	0	-1									
S3		-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-
1	-1	-1	-1	0	-1	-1	-1	-1	-1									
S4		-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-
1	-1	-1	-1	-1	-1	-1	-1	-1	-1									
S5		-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-
1	-1	-1	-1	-1	-1	-1	-1	-1	0									
S6		-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-
1	-1	-1	-1	-1	-1	-1	-1	-1	-1									
S7		-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	2	-1	-1	-1	-1	3
-1	0	-1	-1	-1	-1	-1	-1	-1										
S8		-1	-1	1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-
1	-1	-1	-1	-1	-1	-1	-1	-1	-1									
S9		-1	-1	-1	-1	-1	0	-1	1	-1	1	-1	-1	-1	-1	-1	-1	-
1	-1	-1	-1	-1	-1	-1	-1	-1	1									
VALS		-1	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	0	-1	-1	-
1	-1	-1	-1	-1	0	-1	-1	-1	-1									
stmt		-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	0	-1	-1	-1	-
1	-1	-1	0	-1	-1	-1	-1	-1	-1									

Output

```
Valid string
Valid string
Valid string
Valid string
Invalid string
Invalid string
Invalid string
```