

PARSER PROJECT

(SQL INSERTION QUERY)

SAKSHI MAAN

SYNTAX OF SQL INSERTION QUERY

It is possible to write the **INSERT INTO** statement in two ways:

1. Specify both the column names and the values to be inserted where total no. of columns and total no. of values should be equal.

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

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

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table.

```
INSERT INTO table_name
```

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

ASSUMPTIONS

1. Query can be in lower case or in upper case or can be the combination of both.
2. Table names could be in the combination of alphabets, digits and symbols. Symbols include ('\/-_@#.?), Alphabets include (A-Z or a-z) and digits include (0-9).
3. Data values can be of any type, either string type, integer type, float type or can be symbols.
4. Data values can not be null. Also data values do not include comma as comma is specified for ending of one data value.
5. Column name should also be as data value. It can also be in combination of alphabets, digits and symbols. But it does not include float type.
6. Number of Data values to be inserted is equal to the number of columns
7. Table name, column name and data values could also be in single quotation or double quotation .

TEST CASES

VALID STATEMENTS

Test case 1: All Lowercase

```
insert into table(col1,col2)values(val1,val2);
```

Test case 2: All Uppercase

```
INSERT INTO table(col1,col2)VALUES(val1,val2);
```

Test case 3: Mixed Lowercase and Uppercase

```
INSERt Into table(col)VALUES(val);
```

Test case 4: without column names

```
insert into table values(123,"abc",val3,val4);
```

INVALID STATEMENTS

Test case 5: No. of columns is not equal to values

```
insert into table(col1,col2)values(val1);
```

Test case 6: Table is missing

```
insert into (col1,col2)values(val1,val2);
```

LEXICAL CODE (sql.l)

```
%[ #include <stdio.h>

#include "y.tab.h"

%}

alpha [A-Za-z]

digit [0-9]+([0-9]+)?

symbols ['\-_@#\.\? ]

%%

[ \t\n]

[iI][nN][sS][eE][rR][tT]      {return INSERT;}

[iI][nN][tT][oO]              {return INTO;}

[vV][aA][lL][uU][eE][sS]      {return VALUES;}

{digit}+                      {return NUMBER;}

'({alpha}|{digit}|{symbols}|,)+ ' {return ID;}

\"({alpha}|{digit}|{symbols}|,)+\" {return ID;}

({alpha}|{digit}|{symbols})+    {return ID;}

.                              {return yytext[0];}

%%

int yywrap() {

    return 1;

}
```

YACC CODE (sql.y)

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    void yyerror();
    int yylex();
    int colCount = 0, valCount = 0;
}%

%token ID
%token NUMBER
%token INSERT
%token INTO
%token VALUES

%%

S : ST1'{
    printf("CORRECT STATEMENT");
    exit(0);}
    ;

ST1 :INSERT INTO table '(' attributeList ')' VALUES '(' valuesList ')'
    {
        if(valCount != colCount) {
            printf("Wrong Statement \n No. of Values should be equal to
total no. of Columns \n");
            return 0;
        }
    }
```

```

    }
    | INSERT INTO table VALUES '(' valuesList ')'
    ;

```

```

attributeList: ID ',' attributeList {colCount++;}
              IID                  {colCount++;}
              ;

```

```

valuesList: ID ',' valuesList {valCount++;}
           | NUMBER ',' valuesList {valCount++;}
           | IID              {valCount++;}
           | NUMBER           {valCount++;}
           ;

```

```

table : ID;
%%

```

```

int main()
{
    printf("Enter the Insert Query:\n");
    yyparse();
}

```

```

void yyerror () {
    printf("INCORRECT STATEMENT");
}

```

REPORT

Lexical Analysis(sql.l)

Lexical file is used to generate rules for the tokens we have used. So, here we have five rules for the tokens INSERT, INTO, VALUES, NUMBER and ID.

their regular expressions are as follows,


1. INSERT [iI][nN][sS][eE][rR][tT]
2. INTO [iI][nN][tT][oO]
3. VALUES [vV][aA][lL][uU][eE][sS]
4. NUMBER {digit}+
5. ID ({alpha}|{digit}|{symbols}|,)+

where, alpha is [A-Za-z] , digit is [0-9]+([0-9])? and symbol is [\'\"_@#\\.\\?]

Syntax Analysis(Sql.y)

The main objective of this code is to validate the SQL insertion query, the yacc code uses the tokens returned by the lexical parser.

First of all the code prints the statement to indicate the user to input a query. Then the lex code returns tokens to the yacc code.



In the rules section, first we define what the code should do if the query is accepted. Then we move on to define what tokens should a statement have to get accepted as a valid query.

```
S : ST1';{  
    printf("CORRECT STATEMENT");  
    exit(0);}  
;
```

If the statement has the right sequence of the tokens, then the program goes on to check whether the number of values provided are equal to the number of columns.


```
ST1 :INSERT INTO table '(' attributeList ')' VALUES '(' valuesList '  
    {if(valCount != colCount) {  
        printf("Wrong Statement \n No. of Values should be equal to total  
no. of Columns \n");  
        return 0; } }
```

The assignment to the variables is done in further rules.

If the number of attributes is not the same as the number of values provided then the statement is declared as a wrong statement.

Also, we define another statement which can be accepted as a query where the user does not provide the name of columns, just the name of table and the values to be entered.

```
| INSERT INTO table VALUES '(' valuesList '  
;
```



In the next rules we determine the number of attributes(count saved in colCount) in the attribute list and the number of values(count saved in valCount) provided in the value list. Also we are separating different column values and different data values with comma (,).

```
attributeList:ID ',' attributeList {colCount++;}
            IID                    {colCount++;}
            ;
valuesList:ID ',' valuesList      {valCount++;}
            INUMBER ',' valuesList {valCount++;}
            IID                    {valCount++;}
            INUMBER                {valCount++;}
            ;
table : ID;
```

Further on we determine the output and in case of error , yyerror will execute in which it will print “incorrect statement”.

```
void yyerror () {
    printf("INCORRECT STATEMENT");
}
```

COMPILATION PROCESS

The first job is done by the lexical scanner, `lex`. It recognizes certain patterns of symbols using regular expressions, and assigns them to tokens.

So, for this we choose the alphanumeric RE `([a-z][AZ0-9]*)` for identifying the token `ID`, to correctly get table names with form `(tab1,table2 ...)` or columns like `(col1, col2 ...)` or values like `(val1,val2,...)` for the other tokens we impose the same symbol pattern used by the token.

The grammar task is done by `yacc`, in which we mainly specify the order. The lexical tokens we decide to use are: `INSERT`, `INTO`, `VALUES`, `NUMBERS`, `ID`.

The lexical analysis is done by giving **`flex sql.l`** to the command line, in doing so the `lex` program automatically generates a C file (**`lex.yy.c`**) that follows the rules we provide in `sql.l`.

The syntax analysis is done by `yacc` file by giving **`yacc -vyd sql.y`** to the command line, in doing so the `yacc` program automatically generates a C file (**`y.tab.c`**) that follows the rules we provide in `sql.y`.

We compile both the C files using command **`gcc lex.yy.c y.tab.c`** In doing this we get an executable **`a.exe`** file.

OUTPUT

Test case 1:

All Lowercase

```
insert into table(col1,col2)values(val1,val2);
```

```
D:\MSC_CS\sem2\CD\Programs\14>yacc -vyd sql.y
D:\MSC_CS\sem2\CD\Programs\14>gcc lex.yy.c y.tab.c
D:\MSC_CS\sem2\CD\Programs\14>a.exe
Enter the Insert Query:
insert into University(stu_name,rollno,course)values("abc",123,xyz);
CORRECT STATEMENT
D:\MSC_CS\sem2\CD\Programs\14>_
```

Test case 2:

All Uppercase

```
INSERT INTO table(col1,col2)VALUES(val1,val2);
```

```
CONNECT STATEMENT
D:\MSC_CS\sem2\CD\Programs\14>flex sql.l
D:\MSC_CS\sem2\CD\Programs\14>yacc -vyd sql.y
D:\MSC_CS\sem2\CD\Programs\14>gcc lex.yy.c y.tab.c
D:\MSC_CS\sem2\CD\Programs\14>a.exe
Enter the Insert Query:
INSERT INTO Uiversity(stu_name,marks)VALUES("ABC",23.5);
CORRECT STATEMENT
D:\MSC_CS\sem2\CD\Programs\14>
```

Test case 3:

Mixed Lowercase and Uppercase

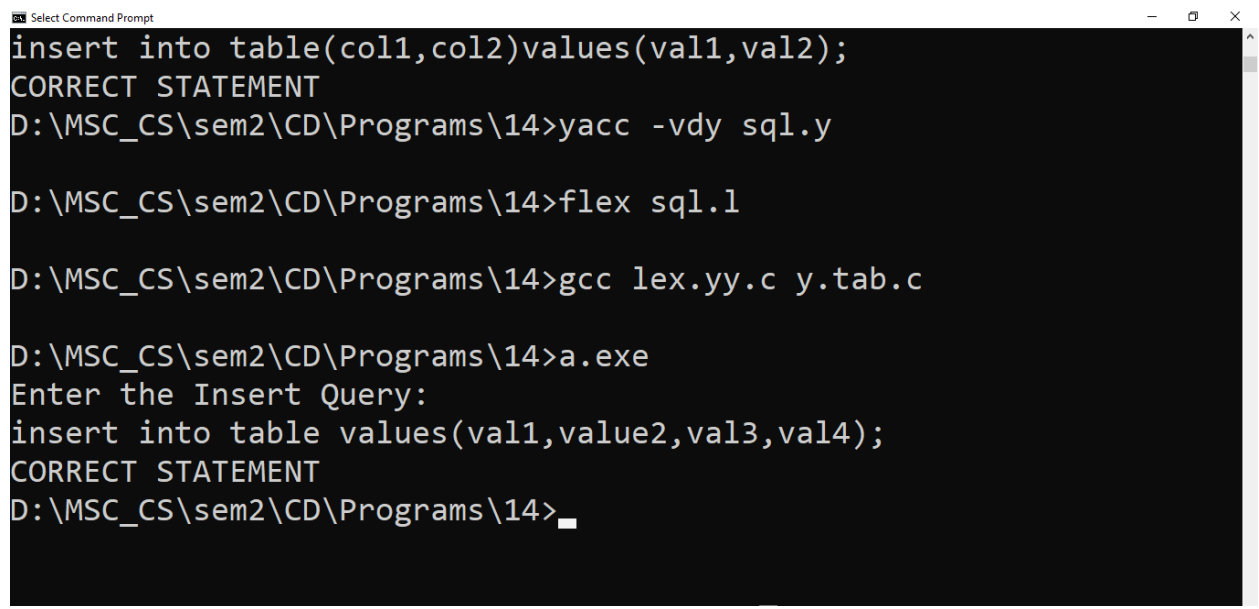
INSERt Into table(col)VALUES(val);

```
D:\MSC_CS\sem2\CD\Programs\14>flex sql.l
D:\MSC_CS\sem2\CD\Programs\14>yacc -vyd sql.y
D:\MSC_CS\sem2\CD\Programs\14>gcc lex.yy.c y.tab.c
D:\MSC_CS\sem2\CD\Programs\14>a.exe
Enter the Insert Query:
INsert INTO University(stu_name,DOB)valueS("AbC",17-09-1776);
CORRECT STATEMENT
D:\MSC_CS\sem2\CD\Programs\14>_
```

Test case 4:

Without column names

```
insert into table values(123,"abc",val3,val4);
```



```
Select Command Prompt
insert into table(col1,col2)values(val1,val2);
CORRECT STATEMENT
D:\MSC_CS\sem2\CD\Programs\14>yacc -vdy sql.y

D:\MSC_CS\sem2\CD\Programs\14>flex sql.l

D:\MSC_CS\sem2\CD\Programs\14>gcc lex.yy.c y.tab.c

D:\MSC_CS\sem2\CD\Programs\14>a.exe
Enter the Insert Query:
insert into table values(val1,value2,val3,val4);
CORRECT STATEMENT
D:\MSC_CS\sem2\CD\Programs\14>_
```

INVALID STATEMENT TEST CASES

Test case 5:

No. of columns is not equal to values

```
insert into table(col1,col2)values(val1);
```

```
D:\MSC_CS\sem2\CD\Programs\14>flex sql.l
D:\MSC_CS\sem2\CD\Programs\14>yacc -vyd sql.y
D:\MSC_CS\sem2\CD\Programs\14>gcc lex.yy.c y.tab.c
D:\MSC_CS\sem2\CD\Programs\14>a.exe
Enter the Insert Query:
INSERT INTO University(Stu_name,age,marks)VALUES("abc",21);
Wrong Statement
No. of Values should be equal to total no. of Columns
D:\MSC_CS\sem2\CD\Programs\14>_
```


Test case 6:

Table is missing

```
insert into (col1,col2) values (val1,val2);
```

```
D:\MSC_CS\sem2\CD\Programs\14>flex sql.1
D:\MSC_CS\sem2\CD\Programs\14>yacc -vyd sql.y
D:\MSC_CS\sem2\CD\Programs\14>gcc lex.yy.c y.tab.c
D:\MSC_CS\sem2\CD\Programs\14>a.exe
Enter the Insert Query:
INSERT INTO (stu_name,age,marks)VALUES("abc",21,29.5);
INCORRECT STATEMENT
D:\MSC_CS\sem2\CD\Programs\14>
```