Lexer.lex

```
%option noyywrap
%{
    #include <stdio.h>
    #include "parser.tab.h"
%}

%s HASH

%%
"SELECT"                {yylval.stringVal = strdup(yytext);return SELECT;}
"<"                    {yylval.stringVal = strdup(yytext);return LT;}
">"                    {yylval.stringVal = strdup(yytext);return GT;}
"("                 {yylval.stringVal = strdup(yytext);return OB;}
")"                 {yylval.stringVal = strdup(yytext);return CB;}
"PROJECT"               {yylval.stringVal = strdup(yytext);return
PROJECT;}
"CARTESIAN_PRODUCT"             {yylval.stringVal = strdup(yytext);return
CARTESIAN;}
"EQUI_JOIN"                {yylval.stringVal = strdup(yytext);return
EQUI;}
"AND"                   {yylval.stringVal = strdup(yytext);return AND;}
"OR"                    {yylval.stringVal = strdup(yytext);return OR;}
[a-zA-Z_][a-zA-Z0-9_]*  {yylval.stringVal = strdup(yytext);return NAME;}
[0-9]+                      {yylval.stringVal = strdup(yytext);return
DIGIT;}
["]+[a-zA-Z0-9_ ]*["]+          {yylval.stringVal = strdup(yytext);return
STRING;}
"="                    {yylval.stringVal = strdup(yytext);return EQ;}
"<="                    {yylval.stringVal = strdup(yytext);return LE;}
">="                    {yylval.stringVal = strdup(yytext);return GE;}
","                 {yylval.stringVal = strdup(yytext);return COMMA;}
"."                   {yylval.stringVal = strdup(yytext);return DOT;}
[ \n\t]+                {}
";"                 {yylval.stringVal = strdup(yytext);return SEMI;}
.                       {printf("error\n");}
```

```
"!="                    {yylval.stringVal = strdup(yytext);return NEQ;}

%%
```

Parser.y

```
%{
#include <stdio.h>
void yyerror(char *s){
  printf ("Invalid Syntax\n");
}
char type[1000];
char line[1000];
char tempName1[1000];
char tempCond[1000];
char line1[1000];
char tableName[100];
char tableName1[100];
char col_list[1000];
char tempCol[100];
char sp_table1[100];
char sp_table2[100];
char sp_col1[100];
char sp_col2[100];
int label;
FILE * f;
%}
%union{ char * stringVal; }
%token <stringVal> SELECT LT GT LE GE EQ OB CB PROJECT CARTESIAN NAME EQUI
AND OR COMMA DOT SEMI NEQ STRING DIGIT
%start STATEMENTS
%%
STATEMENTS :            STATEMENT STATEMENTS   {}
                   |   STATEMENT              {}

STATEMENT :             SELECT LT CONDITIONS GT OB TABLENAME CB SEMI
{printf("Valid Syntax\n");fprintf(f, "cout<<\"Query %d:\"<<endl;f1 =
```

```
fopen(\"%s.csv\", \"r\");\nif(!f1){\ncout<<\"Table not
found!\"<<endl;\ngoto label%d;\n}\nfgets(line1, 1000,
f1);\ncout<<line1;\ncol1 = getTokens(line1);\nfgets(line1, 1000,
f1);\ntype = getTokens(line1);\nfor(int
i=0;i<col1.size();i++){\nmap1.insert({col1[i], i});\nmapt.insert({col1[i],
type[i]});\n}\ncond1 = strdup(\"%s\");\ncon = getConditions(cond1);\ntp =
strdup(\"%s\");\ntp1 = getConditions(tp);for(int i=0;i<tp1.size();i+=2){\n
int flag = 0;\n  if(tp1[i+1]==\"string\"){\n
if(mapt[tp1[i]]!=\"string\"){\n      flag = 1;\n    }\n  }else
if(tp1[i+1]==\"int\"){\n    if(mapt[tp1[i]]!=\"int\"){\n      flag = 1;\n
}\n  }else{\n    if(mapt[tp1[i]]!=mapt[tp1[i+1]]){\n      flag = 1;\n
}\n  }\n  if(flag){\n    cout<<\"(Semantic Error)\"<<endl;\n    goto
label%d;\n  }\n}\nwhile(fgets(line1, 1000,
f1)){\nline1[strlen(line1)-1]='\\0';\nstring l = line1;\nint i = 0;\nrow1
= getTokens(line1);\nwhile(i<con.size()){\nstring var1 =
con[i];\nif(map1.find(var1)==map1.end()){\ncout<<\"Coloumn not
found!\"<<endl;\goto
label%d;\n}\ni+=1;\n}\nif(%s){\ncout<<l<<endl;\n}\n}\n", label+1,
tableName,label, line,type, label, label,
line1);type[0]='\0';line[0]='\0';tempName1[0]='\0';tempCond[0]='\0';line1[
0]='\0';tableName[0]='\0';tableName1[0]='\0';col_list[0]='\0';tempCol[0]='
\0';sp_table1[0]='\0';sp_table2[0]='\0';sp_col1[0]='\0';sp_col2[0]='\0';fp
rintf(f, "fclose(f1);\nlabel%d :\ncout<<endl;\n", label); label++;}
                  |   PROJECT LT ATTR_LIST GT OB TABLENAME CB SEMI
{printf("Valid Syntax\n");fprintf(f, "cout<<\"Query %d:\"<<endl;f1 =
fopen(\"%s.csv\", \"r\");\n    if(!f1){\n        cout<<\"Table name not
found!\"<<endl;\n        goto label%d;\n    }\n    fgets(line1, 1000,
f1);\n    col1 = getTokens(line1);\n    for(int i = 0; i < col1.size();
i++){\n        map1.insert({col1[i],i});\n    }\n    cond1 =
strdup(\"%s\");\n    col2 = getConditions(cond1);\n    for(int i = 0; i <
col2.size(); i++){\n        if(map1.find(col2[i])==map1.end()){\n
cout << \"Coloumn not Found!\" << endl;\n             goto label%d;\n
}\n        indexs.push_back( map1[col2[i]] );\n    }\n    for(int i = 0; i
< col2.size(); i++){\n        cout << col2[i];\n
if(i!=col2.size()-1)\n            cout << \",\";\n        else\n
cout << endl;\n    }\n    fgets(line1, 1000, f1);\n    while(fgets(line1,
1000, f1)){\n        row1 = getTokens(line1);\n        for(int i = 0; i <
```

```
indexs.size(); i++){\n            cout << row1[indexs[i]];\n
if(i!=indexs.size()-1)\n                cout << \",\";\n            else\n
cout << endl;\n        }\n    }\n    fclose(f1);\n    label%d :\n
cout<<endl;\n",label+1, tableName,label, col_list,label,
label);label++;line[0]='\0';tempName1[0]='\0';tempCond[0]='\0';line1[0]='\
0';tableName[0]='\0';tableName1[0]='\0';col_list[0]='\0';tempCol[0]='\0';s
p_table1[0]='\0';sp_table2[0]='\0';sp_col1[0]='\0';sp_col2[0]='\0';}
                    |   CARTESIAN_QUERY1 CARTESIAN_QUERY CB SEMI
{printf("Valid Syntax\n");fprintf(f, "cout<<\"Query %d:\"<<endl;f1 =
fopen(\"%s.csv\", \"r\");\n    f2 = fopen(\"%s.csv\", \"r\");\n    if(!f1
|| !f2){\n        cout<<\"Table name not found!\"<<endl;\n        goto
label%d;\n    }\n    fgets(line1, 1000, f1);\n    fgets(line2, 1000,
f2);\n    line1[strlen(line1)-1] = '\\0';\n
cout<<line1<<\",\"<<line2;\n    fgets(line1, 1000, f1);\n   fgets(line2,
1000, f2);\n     while(fgets(line1, 1000, f1)){\n
line1[strlen(line1)-1] = '\\0';\n        while(fgets(line2, 1000, f2)){\n
cout<<line1<<\",\"<<line2;\n        }\n        fseek(f2, 0, SEEK_SET);\n
fgets(line2, 1000, f2);\n        fgets(line2, 1000, f2);\n    }",label+1,
tableName, tableName1,
label);line[0]='\0';tempName1[0]='\0';tempCond[0]='\0';line1[0]='\0';table
Name[0]='\0';tableName1[0]='\0';col_list[0]='\0';tempCol[0]='\0';sp_table1
[0]='\0';sp_table2[0]='\0';sp_col1[0]='\0';sp_col2[0]='\0';fprintf(f,
"\nfclose(f1);\nfclose(f2);\nlabel%d :\ncout<<endl;\n", label);label++;}
                    |   CARTESIAN_QUERY1 EQUI_QUERY CB SEMI
{printf("Valid Syntax\n");fprintf(f, "cout<<\"Query %d:\"<<endl;f1 =
fopen(\"%s.csv\", \"r\");\n    f2 = fopen(\"%s.csv\", \"r\");\n    if(!f1
|| !f2){\n        cout<<\"Table name not found!\"<<endl;\n        goto
label%d;\n    }\n    fgets(line1, 1000, f1);\n    fgets(line2, 1000,
f2);\n    line1[strlen(line1)-1] = '\\0';\n
cout<<line1<<\",\"<<line2;\n    col1 = getTokens(line1);\n    col2 =
getTokens(line2);\n        fgets(line1, 1000, f1);\n    fgets(line2, 1000,
f2);\n    line1[strlen(line1)-1] = '\\0';\n    type = getTokens(line1);\n
type1 = getTokens(line2);\nfor(int i=0;i<col1.size();i++){\n
map1.insert({col1[i], i});\n    }\n    for(int i=0;i<col2.size();i++){\n
map2.insert({col2[i], i});\n    }\n    cond1 = strdup(\"%s\");\n    cond2
= strdup(\"%s\");\n    if(map1.find(cond1)==map1.end() ||
map2.find(cond2)==map2.end()){\n        cout<<\"Coloumn not
```

```
found!\"<<endl;\n    goto label%d;\n}\n    index1 =
map1.find(cond1)->second;\n    index2 = map2.find(cond2)->second;\n
if(type[index1]!=type1[index2]){\n        cout<<\"(Semantic
Error)\"<<endl;\n        goto label%d;\n}\nwhile(fgets(line1, 1000,
f1)){\n        line1[strlen(line1)-1] = '\\0';\n        row1 =
getTokens(line1);\n        while(fgets(line2, 1000, f2)){\n
row2 = getTokens(line2);\n            if(row1[index1] == row2[index2]){\n
for(int i=0;i<row1.size();i++){\n
cout<<row1[i]<<\",\";\n                }\n                for(int
i=0;i<row2.size();i++){\n                    cout<<row2[i];\n
if(i!=row2.size()-1)cout<<\",\";\n                }\n
cout<<endl;\n            }\n        }\n        fseek(f2, 0, SEEK_SET);\n
fgets(line2, 1000, f2);\n        fgets(line2, 1000, f2);\n    }",label+1,
tableName, tableName1,label, sp_col1, sp_col2, label,
label);line[0]='\0';tempName1[0]='\0';tempCond[0]='\0';line1[0]='\0';table
Name[0]='\0';tableName1[0]='\0';col_list[0]='\0';tempCol[0]='\0';sp_table1
[0]='\0';sp_table2[0]='\0';sp_col1[0]='\0';sp_col2[0]='\0';fprintf(f,
"\nfclose(f1);\nfclose(f2);\nlabel%d :\ncout<<endl;\n", label);label++;}
                  |   error SEMI { label++;yyerrok; }

CARTESIAN_QUERY1    :   OB NAME {tableName[0] = '\0';strcpy(tableName,
yylval.stringVal);}
CARTESIAN_QUERY :       CB CARTESIAN OB NAME {tableName1[0] =
'\0';strcpy(tableName1, yylval.stringVal);}
EQUI_QUERY      :       CB EQUI LT EQUI_CONDITION GT OB NAME
{tableName1[0] = '\0';strcpy(tableName1,
yylval.stringVal);if(!((strcmp(sp_table1,tableName)==0 &&
strcmp(sp_table2,tableName1)==0) || (strcmp(sp_table2,tableName)==0 &&
strcmp(sp_table1,tableName1)==0))){printf("(Semantic
Error)");yyerror("Semantic Error!\n");YYERROR;}}

TABLENAME  :            NAME {tableName[0] = '\0';strcpy(tableName,
yylval.stringVal);}


CONDITIONS :            CONDITION AND1 CONDITIONS        {}
                  |   CONDITION OR1 CONDITIONS         {}
```

```
                    |   CONDITION                          {}

AND1        :           AND                      {strcat(line, "
");strcat(line1, " && ");}
OR1         :           OR                       {strcat(line, "
");strcat(line1, " || ");}

CONDITION  :            NAME1 EQ STRING           {tempCond[0]='\0';
strcat(type, "string "); strcpy(tempCond,tempName1); strcat(line1,
"row1[map1.find(\""); strcat(line, tempCond); strcat(line1, tempName1);
strcat(line1, "\")->second"); strcat(line1, "] == ");strcat(line1,
yylval.stringVal);}
                     |   NAME1 NEQ STRING          {tempCond[0]='\0';
strcat(type, "string "); strcpy(tempCond,tempName1); strcat(line1,
"row1[map1.find(\""); strcat(line, tempCond); strcat(line1, tempName1);
strcat(line1, "\")->second"); strcat(line1, "] != ");strcat(line1,
yylval.stringVal);}
                    |   NAME1 EQ NAME             {tempCond[0]='\0';
strcat(type, yylval.stringVal); strcat(type, " ");
strcpy(tempCond,tempName1); strcat(tempCond, " "); strcat(tempCond,
yylval.stringVal); strcat(line1, "row1[map1.find(\""); strcat(line,
tempCond); strcat(line1, tempName1); strcat(line1, "\")->second");
strcat(line1, "] == "); strcat(line1, "row1[map1.find(\""); strcat(line1,
yylval.stringVal);strcat(line1, "\")->second]");}
                    |   NAME1 NEQ NAME             {tempCond[0]='\0';
strcat(type, yylval.stringVal); strcat(type, " ");
strcpy(tempCond,tempName1); strcat(tempCond, " "); strcat(tempCond,
yylval.stringVal); strcat(line1, "row1[map1.find(\""); strcat(line,
tempCond); strcat(line1, tempName1); strcat(line1, "\")->second");
strcat(line1, "] != "); strcat(line1, "row1[map1.find(\""); strcat(line1,
yylval.stringVal);strcat(line1, "\")->second]");}

                    |   NAME1 LT NAME             {tempCond[0]='\0';
strcat(type, yylval.stringVal); strcat(type, " ");
strcpy(tempCond,tempName1); strcat(tempCond, " "); strcat(tempCond,
yylval.stringVal);strcat(line1,"sttoi("); strcat(line1,
"row1[map1.find(\""); strcat(line, tempCond); strcat(line1, tempName1);
```

```
strcat(line1, "\")->second");  strcat(line1, "]) < sttoi("); strcat(line1,
"row1[map1.find(\""); strcat(line1, yylval.stringVal);strcat(line1,
"\")->second])");}
                    |   NAME1 GT NAME              {tempCond[0]='\0';
strcat(type, yylval.stringVal); strcat(type, " ");
strcpy(tempCond,tempName1); strcat(tempCond, " "); strcat(tempCond,
yylval.stringVal); strcat(line1,"sttoi("); strcat(line1,
"row1[map1.find(\""); strcat(line, tempCond); strcat(line1, tempName1);
strcat(line1, "\")->second");  strcat(line1, "]) > sttoi("); strcat(line1,
"row1[map1.find(\""); strcat(line1, yylval.stringVal);strcat(line1,
"\")->second])");}
                    |   NAME1 LE NAME              {tempCond[0]='\0';
strcat(type, yylval.stringVal); strcat(type, " ");
strcpy(tempCond,tempName1); strcat(tempCond, " "); strcat(tempCond,
yylval.stringVal); strcat(line1,"sttoi("); strcat(line1,
"row1[map1.find(\""); strcat(line, tempCond); strcat(line1, tempName1);
strcat(line1, "\")->second"); strcat(line1, "]) <= sttoi("); strcat(line1,
"row1[map1.find(\""); strcat(line1, yylval.stringVal);strcat(line1,
"\")->second])");}
                    |   NAME1 GE NAME              {tempCond[0]='\0';
strcat(type, yylval.stringVal); strcat(type, " ");
strcpy(tempCond,tempName1); strcat(tempCond, " "); strcat(tempCond,
yylval.stringVal); strcat(line1,"sttoi("); strcat(line1,
"row1[map1.find(\""); strcat(line, tempCond); strcat(line1, tempName1);
strcat(line1, "\")->second"); strcat(line1, "]) >= sttoi("); strcat(line1,
"row1[map1.find(\""); strcat(line1, yylval.stringVal);strcat(line1,
"\")->second])");}
                    |   NAME1 LT DIGIT             {tempCond[0]='\0';
strcat(type, "int "); strcpy(tempCond,tempName1); strcat(line1,"sttoi(");
strcat(line1, "row1[map1.find(\""); strcat(line, tempCond); strcat(line1,
tempName1); strcat(line1, "\")->second");  strcat(line1, "]) < ");
strcat(line1, yylval.stringVal);}
                    |   NAME1 GT DIGIT             {tempCond[0]='\0';
strcat(type, "int "); strcpy(tempCond,tempName1); strcat(line1,"sttoi(");
strcat(line1, "row1[map1.find(\""); strcat(line, tempCond); strcat(line1,
tempName1); strcat(line1, "\")->second");  strcat(line1, "]) > ");
strcat(line1, yylval.stringVal);}
```

```
                  |   NAME1 LE DIGIT            {tempCond[0]='\0';
strcat(type, "int "); strcpy(tempCond,tempName1);
strcat(line1,"sttoi(");strcat(line1, "row1[map1.find(\""); strcat(line,
tempCond); strcat(line1, tempName1); strcat(line1, "\")->second");
strcat(line1, "]) <= "); strcat(line1, yylval.stringVal); }
                  |   NAME1 GE DIGIT            {tempCond[0]='\0';
strcat(type, "int "); strcpy(tempCond,tempName1);
strcat(line1,"sttoi(");strcat(line1, "row1[map1.find(\""); strcat(line,
tempCond); strcat(line1, tempName1); strcat(line1, "\")->second");
strcat(line1, "]) >= "); strcat(line1, yylval.stringVal); }
                  |   NAME1 EQ DIGIT              {tempCond[0]='\0';
strcat(type, "int "); strcpy(tempCond,tempName1);
strcat(line1,"sttoi(");strcat(line1, "row1[map1.find(\""); strcat(line,
tempCond); strcat(line1, tempName1); strcat(line1, "\")->second");
strcat(line1, "]) == "); strcat(line1, yylval.stringVal);}
                  |   NAME1 NEQ DIGIT             {tempCond[0]='\0';
strcat(type, "int "); strcpy(tempCond,tempName1);
strcat(line1,"sttoi(");strcat(line1, "row1[map1.find(\""); strcat(line,
tempCond); strcat(line1, tempName1); strcat(line1, "\")->second");
strcat(line1, "]) != "); strcat(line1, yylval.stringVal);}

NAME1      :            NAME
{tempName1[0]='\0';strcpy(tempName1, yylval.stringVal);strcat(type,
yylval.stringVal);strcat(type, " ");}

ATTR_LIST  :            NAMEZ COMMAZ ATTR_LIST          {}
                  |   NAMEZ                          {}

COMMAZ     :            COMMA                       {strcat(col_list,"
");}
NAMEZ      :            NAME
{strcat(col_list,yylval.stringVal);}
```

```
EQUI_CONDITION :        EQUI_CONDITION1 EQUI_CONDITION2 EQUI_CONDITION3
EQUI_CONDITION4  {if(strcmp(sp_table1, tableName)!=0){ char temp[100];
strcpy(temp, sp_col1); strcpy(sp_col1, sp_col2); strcpy(sp_col2, temp);}}
EQUI_CONDITION1 :       NAME    {sp_table1[0] = '\0';strcpy(sp_table1,
yylval.stringVal);}
EQUI_CONDITION2 :       DOT NAME    {sp_col1[0] = '\0';strcpy(sp_col1,
yylval.stringVal);}
EQUI_CONDITION3 :       EQ NAME     {sp_table2[0] = '\0';strcpy(sp_table2,
yylval.stringVal);}
EQUI_CONDITION4 :       DOT NAME    {sp_col2[0] = '\0';strcpy(sp_col2,
yylval.stringVal);}

%%

int main(){
    f = fopen("output.cpp", "w");
    label = 0;
   fprintf(f, "#include<bits/stdc++.h>\nusing namespace
std;\nvector<string> getTokens(char line[]){\nchar * token;\ntoken =
strtok (line,\",\\n\");\nvector<string> col;\nwhile (token !=
NULL)\n{\ncol.push_back(token);\ntoken = strtok (NULL,
\",\\n\");\n}\nreturn col;\n}\nvector<string> getConditions(char
line[]){\nchar * token;\ntoken = strtok (line,\" \");\nvector<string>
col;\nwhile (token != NULL)\n{\ncol.push_back(token);\ntoken = strtok
(NULL, \" \");\n}\nreturn col;\n}\nint sttoi(string s){\n int flag = 0;\n
int t = 0;\n      for(int i=0;i<s.length();i++){\n    t += s[i];\n
if(!isdigit(s[i])){\n       flag = 1;\n    }\n      }\n   if(flag==0){\n
return stoi(s);\n }else{\n     return t;\n    }\n   }\nint main(){\nFILE
* f1, *f2;\nvector<string> row1, row2;\nvector<int> indexs;\nchar * line1
= new char[1000];\nchar * line2 = new char[1000];\nvector<string> col1,
col2;\nmap<string, int> map1, map2;\nmap<string, string> mapt;\nchar *
cond1;\nchar * cond2;\nchar * tp;\nvector<string> con, type, type1,
tp1;\nint index1, index2;\n");
   yyparse();
   fprintf(f, "}\n");
    fclose(f);
   return 0;
```

```
}
```

Run1.sh

```
bison -d -v parser.y
flex lexer.lex
gcc -w parser.tab.c lex.yy.c -o main
./main < input.in
echo
g++ output.cpp
./a.out
```