COMPILER DESIGN PROJECT

SQL PARSER

MOHD. SADIQ HUSSAIN	13103406
MANJITA SHARMA	13103411
EKLAVYA HINGORANI	13103435
SHUBHAM AGGARWAL	13103437

SQL Parser

Abstract

The aim of our project is to design an SQL parser that parses given SQL grammar, which solves the close relationship between the data query and interface design of management information system.

Introduction

Here we have designed a parser that parses whole SQL grammer and prepares an in depth analysis of the same. The implementation of this parser has been done in yacc which is an standard parser generator for the Unix operating system. Sometimes it is necessary to apply a custom filter to an existing SQL query to perform search by a custom criteria or to order the query results depending on the user action. In this control the user should be able to specify a filter and order the search results as he needs. A SQL query is specified in the control's data source and it can contain any parts including sub-queries.

YACC

Yacc (for "yet another compiler compiler.") is the standard parser generator for the Unix operating system. It is an open source program, which generates code for the parser in the C programming language. It is a LALR parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to BNF. Yacc itself used to be available as the default parser generator on most Unix systems, though it has since been supplanted as the default by more recent, largely compatible, programs.

Lexical analyser

In computer science, lexical analysis is the process of converting a sequence of characters (such as in a computer program or web page) into a sequence of tokens (strings with an identified "meaning"). A program that performs lexical analysis may be called a lexer, tokenizer, or scanner (though "scanner" is also used to refer to the first stage of a lexer). Such a lexer is generally combined with a parser, which together analyze the syntax of programming languages, web pages, and so forth.

- **Lexical grammar**

  The specification of a programming language often includes a set of rules, the lexical grammar, which defines the lexical syntax. The lexical syntax is usually a regular language, with the grammar rules consisting of regular expressions; they define the set of possible character sequences that are used to form individual tokens or lexemes. A lexer recognizes strings, and for each kind of string found the lexical program takes an action, most simply producing a token.

- **Tokenization**

  Tokenization is the process of demarcating and possibly classifying sections of a string of input characters. The resulting tokens are then passed on to some other form of processing. The process can be considered a sub-task of parsing input.

Objectives

- The parser parses the SQL grammar.
- The output is provided with a SQL query parse tree.
- Returns the table row, column and object name.

Code for lex file

```
%{
#include"y.tab.h"
%}
%%
"SELECT" {printf("TABLES:\n");return SELECT;}
"FROM" {printf("\nCOLUMNS:\n");return FROM;}
"ORDERBY" {printf("\nORDERED:\n");return ORDER;}
"GROUPBY" {printf("\nGROUPED:\n");return GROUPBY;}
"WHERE" {printf("\nCONDITIONS:\n");return WHERE;}
"HAVING" {printf("\nCONDITIONS:\n");return HAVING;}
"ASC" {return ASC;}
"DSC" {return DSC;}
"MIN"|"MAX"|"AVG" {printf(" ->CONDITIONS:\n");return FUN;}
"OR"|"AND"|"NOT" {printf("\n");return FUNN;}
"UNION" {printf("\n\n OR \n\n");return OR;}
"INTERSECT" {printf("\n\n AND \n\n");return OR;}
"(" {return OB;}
")" {return CB;}
"," {printf("\n");return MORE;}
([a-z])* {printf("%s",yytext); return ID;}
"+"|"<"|"="|">"|"-"|"%"|"!"|"*"|"=="|"!=" {printf("%s",yytext);return OP;}
[\t]
[\n] {return 0;}
%%
```

Yacc file

```
%{
#include<stdio.h>
%}
%token SELECT ID FROM ID
%token SELECT FUN OB ID CB FROM ID GROUPBY ID
%token SELECT OP
%token MORE ID
%token ID ASC
%token ID DSC
%token ORDER
%token WHERE ID OP ID
%token ID OP ID FUNN
%token HAVING ID
%token OR
%%
variable: SELECT ID b
            | SELECT ID MORE a
            | SELECT OP b
            | SELECT FUN OB ID CB FROM ID GROUPBY ID
            | SELECT FUN OB ID CB FROM ID GROUPBY ID OR variable
;
a: ID b
      | ID MORE a
;
b: FROM ID d|e|f
      | FROM ID MORE c
;
c: ID d|e|f
      | ID MORE c
;
d: WHERE ID OP ID
      | WHERE OB variable CB
      | WHERE ID OP ID e|f
```

```
        | WHERE ID OP ID FUNN x
        | WHERE ID OP ID OR variable
        | WHERE OB variable CB OR variable
;
e: ORDER w
;
f: HAVING g
;
g: MORE ID g
        |ID
        |ID e
        |ID OR variable
;
w: ID ASC|DSC
;
x: ID OP ID FUNN x
    | ID OP ID
    | ID OP ID OR variable
;
%%

main()
{
yyparse();
printf("\nQuery ACCEPTED \n");
}
int yyerror(char *s)
{
printf("\nQuery REJECTED\n");
exit(0);
}
```