Informatyka, 3 rok

Wydział EAIiIB

Teoria Kompilacji i Kompilatory

Projekt zaliczeniowy

[**Translator skryptów SQL na reprezentację graficzną**](http://home.agh.edu.pl/%7Ejpi/dokuwiki/doku.php?id=dydaktyka:kompilatory:2015:sql_entity)

Marek Ryznar

Michał Dyrkacz

1. **Opis działania translatora**

Celem projektu jest przedstawienie działania prostego translatora skryptów tworzących bazę danych w języku SQL na jego reprezentację graficzną w postaci encji. Skaner skanuje plik wejściowy w poszukiwaniu tokenów, następnie parser za pomocą gramatyki zapisuje wykryte tabele w Liście tabel. Na końcu rysowane są wszystkie encje.

Dokładniejsze działanie programu jest opisane w dalszych częściach dokumentacji.

1. **Skaner**

Skaner został skonstruowany tak, aby na podstawie gramatyki opisanej w następnym punkcie wyodrębnić z otrzymanego łańcucha znaków tokeny charakterystyczne dla języka SQL. Tokeny są reprezentowane przez instancję klasy Yytoken, która posiada pola ‘type’ i ‘attribute’ oznaczające odpowiednio rodzaj tokena opcjonalnie jego atrybut, np.:

{INTEGER} { return symbol(sym.INTEGER,new Integer(yytext())); }

Do stworzenia skanera zostało wykorzystane narzędzie JFlex, które pozwala na generowanie analizatorów leksykalnych w języku Java w oparciu o plik z rozszerzeniem ‘.flex’, ‘.lex’ itd. Plik ten dzieli się na trzy części oddzielone znakiem ‘%%’.

Pierwsza część zawiera kod w języku Java, który zostanie wygenerowany bez zmian do wynikowych klas javy. Dzięki temu można dodać dodatkowe elementy potrzebne w programie (np. klasy,bufory) lub dodatkowe biblioteki.

Druga sekcja to miejsce na definiowanie makr dla skanera. Podaje się nazwe makra i odpowiadające mu wyrażenie regularne.

W ostatniej części pliku określa się konkretne zachowanie w zależności od dopasowanego wzorca. Dla poszczególnych wzorców podane są instrukcje, które są podejmowane w momencie wykrycia (zwracany jest dany token).

1. **Parser**

Parser został utworzony przy pomocy narzędzia CUP. Na podstawie pliku sql.cup wygenerowana została klasa javy, w której zawarte są wygenerowane metody oraz metody odziedziczone po interfejsie java\_cup.runtime.Parser. Parser korzysta ze stworzonej wcześniej klasy SqlLexer (Skanera). W konstruktorze przyjmowany jest obiekt skanera.

Plik parsera składa się z kilku części:

- parser code – część kodu dołączone do klasy wynikowej.

parser code {:

Tabela tab = new Tabela();

List<Tabela> tabele = new LinkedList<Tabela>();

String nazwa;

String typ;

public Tabela getTabela()

{

return this.tab;

}

public List<Tabela> getTabele()

{

return this.tabele;

}

:}

- init/scan with – części odpowiadające za inicjalizację i wczytywanie kolejnych tokenów.

- gramatyka – definicja symboli terminalnych, nieterminalnych, priorytetów, reguł gramatyki.

input ::= input:a new\_table {: RESULT=a; :}

| new\_table:a {: RESULT=a; :}

;

new\_table ::= def:a APOSTROPHE {: RESULT=a; :}

;

def ::= CREATE TABLE IF NOT EXISTS ID:var dec

{: RESULT = "test";

parser.tab.setNazwa(var.toString());

parser.tabele.add(parser.tab);

parser.tab=new Tabela();

:}

|

CREATE TABLE ID:var dec

{: parser.tab.setNazwa(var.toString());

parser.tabele.add(parser.tab);

parser.tab=new Tabela();

:}

;

dec ::= LBRACKET mid RBRACKET;

mid ::= mid variable

| variable {: :}

| key {: :}

;

variable ::= ID:var var\_type COMMA

{: parser.nazwa=var.toString();

parser.tab.getAtrybuty().put(parser.nazwa,parser.typ);

:}

| ID:var var\_type

{: parser.nazwa=var.toString();

parser.tab.getAtrybuty().put(parser.nazwa,parser.typ);

:}

;

var\_type ::= INTEGER\_TYPE key {: parser.typ = "integer "+parser.typ; :}

| INTEGER\_TYPE {: parser.typ = "integer"; :}

| INTEGER\_TYPE NOT NULL {: parser.typ = "integer"; :}

| REAL\_TYPE key {: parser.typ = "real "+parser.typ; :}

| REAL\_TYPE {: parser.typ = "real"; :}

| REAL\_TYPE NOT NULL {: parser.typ = "real"; :}

| SMALLINT key {: parser.typ = "smallint "+parser.typ; :}

| SMALLINT {: parser.typ = "smallint"; :}

| SMALLINT NOT NULL {: parser.typ = "smallint"; :}

| CHAR LBRACKET INTEGER:var RBRACKET key

{: parser.typ = "char("+var+") "+parser.typ; :}

| CHAR LBRACKET INTEGER:var RBRACKET

{: parser.typ = "char("+var+")"; :}

| CHAR LBRACKET INTEGER:var RBRACKET NOT NULL

{: parser.typ = "char("+var+")"; :}

| VARCHAR LBRACKET INTEGER:var RBRACKET key

{: parser.typ = "varchar("+var+") "+parser.typ; :}

| VARCHAR LBRACKET INTEGER:var RBRACKET

{: parser.typ = "varchar("+var+")"; :}

| VARCHAR LBRACKET INTEGER:var RBRACKET NOT NULL

{: parser.typ = "varchar("+var+")"; :}

| NUMERIC LBRACKET INTEGER:var COMMA INTEGER:var2 RBRACKET

{: parser.typ = "numeric("+var+","+var2+")"; :}

|NUMERIC LBRACKET INTEGER:var COMMA INTEGER:var2 RBRACKET NOT NULL

{: parser.typ = "numeric("+var+","+var2+")"; :}

| NUMERIC LBRACKET INTEGER:var COMMA INTEGER:var2 RBRACKET key

{: parser.typ = "numeric("+var+","+var2+")"+parser.typ; :

| TEXT key {: parser.typ = "text "+parser.typ; :}

| TEXT {: parser.typ = "text"; :}

| TEXT NOT NULL {: parser.typ = "text"; :}

| BOOLEAN key {: parser.typ = "boolean "+parser.typ; :}

| BOOLEAN {: parser.typ = "boolean"; :}

| BOOLEAN NOT NULL {: parser.typ = "boolean"; :}

| TIMESTAMP key {: parser.typ = "timestamp "+parser.typ; :} | TIMESTAMP {: parser.typ = "timestamp"; :}

| TIMESTAMP NOT NULL {: parser.typ = "timestamp"; :}

| DATE key {: parser.typ = "date "+parser.typ; :}

| DATE {: parser.typ = "date"; :}

| DATE NOT NULL {: parser.typ = "date"; :}

| SERIAL key {: parser.typ = "serial "+parser.typ; :}

| SERIAL {: parser.typ = "serial"; :}

| SERIAL NOT NULL {: parser.typ = "serial"; :}

;

key ::= PRIMARY KEY {: parser.typ = "primary key"; :}

| REFERENCES ID:var LBRACKET ID:var2 RBRACKET

{:parser.typ ="references "+var.toString()+"("+var2.toString()+")"; :}

| PRIMARY KEY REFERENCES ID:var LBRACKET ID:var2 RBRACKET

{:parser.typ ="primary key references "+var.toString()+"("+var2.toString()+")"; :} | PRIMARY KEY LBRACKET ID:var COMMA ID:var2 RBRACKET {: :} ;

1. **Interfejs graficzny**

Miejsce dla Misiaczka :\*