

Translator din PL-SQL în limbaj natural

Proiect la Limbaje Formale și Translatoare

Studenti: Alina Aurică, Horațiu Andrei Bologa, George-Emanuel Cucos-Artene

5/22/2023

Contents

Cerința proiectului:	2
Explicațiile proiectului:.....	3
Arborele de parsare:	5
Structura codului:.....	6
Bibliografie:	7

Cerința proiectului:

Crearea unui translator din PL-SQL în limbaj natural. La intrare se dă o interogare în PL-SQL, iar la ieșire se generează ”traducerea” în limbaj natural.

Explicațiile proiectului:

Limbajele pe care l-am ales pentru realizarea translatorului au fost: Lex și Yacc.

Partea de creare și identificare a token-ilor a fost realizată în Lex. Mai specific: Avem instrucțiunea:

```
SELECT table_name.  
  
CASE owner  
  
    WHEN 'SYS' THEN The_owner_is_SYS;  
  
    WHEN 'SYSy' THEN The_owner_is_tampit;  
  
    ELSE 'The_owner_is_another_value';  
  
END  
  
FROM all_tables;
```

Cuvintele SELECT, CASE, WHEN, THEN, ELSE, END și FROM, au fost identificate individual și trimise prin clauza return către parser, care este reprezentat de fișierul Yacc.

```
("CASE"|"case") {return CASE;}
```

Toate celelalte cuvinte care nu reprezintă KeyWord-uri specifice PL-SQL, au fost recunoscute conform următorului regex:

```
[a-zA-Z0-9\.\(\)\-\_\'\"%*+\/\=\:|>]+
```

Ca mai apoi să fie trimise către fișierul de tip Yacc prin yylval.word. yylval este o variabila de tip int, care trimite date de Lex la Yacc. În situația dată, noi doream să trimitem și să anexăm token-ului WORD, un șir de caractere, așa ca i-am atribuit lui yylval word, care este declarat într-o structură de date de tip union în fișierul Yacc:

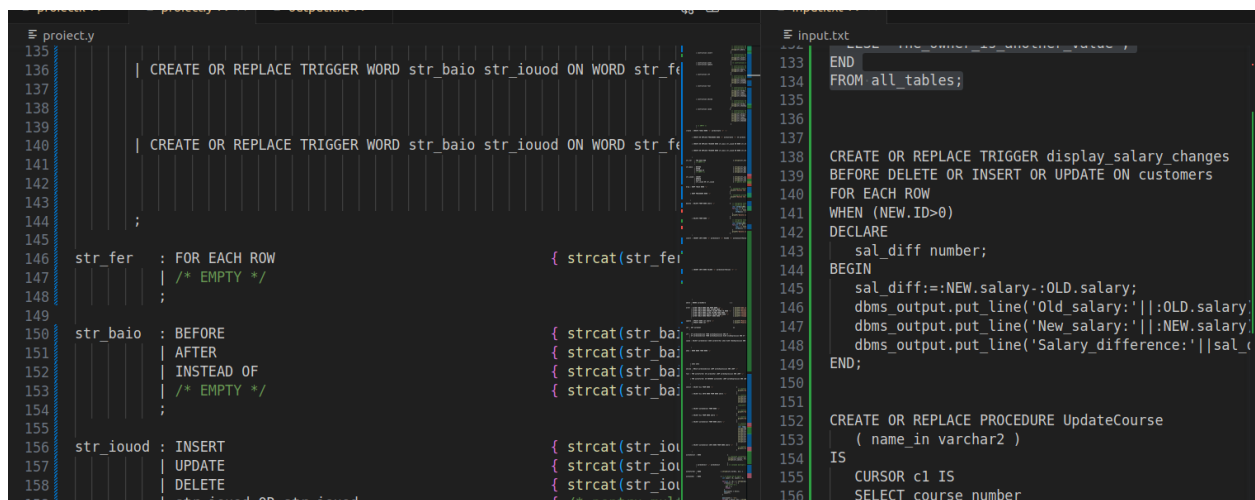
```
%union {char *word;}
```

Pentru a putea utiliza C și funcțiile aferente acestuia, am importat librăriile stdio.h, stdlib.h și string.h (pentru lucrul cu șiruri de caractere). De asemenea, am utilizat funcția

substr()¹ pentru a elimina ghilimelele('') sau apostroafele(') care încadrează string-uri specifice (cum ar fi numele unei persoane).

Pe lângă token-ii care conțin keyWord-uri, am trimis și token-i ca semne de punctuație, precum *, care în clauza SELECT reprezintă ideea de ALL (afișarea tuturor coloanelor din tabel).

Ca și parser s-a utilizat Yacc. Acesta preia token-ii din gramatică (care sunt simboluri terminale) și cu ajutorul lor creează producții (sau reguli) prin care recunoaște sintaxa limbajului de baze de date PL-SQL. Pentru crearea unor producții cât mai generale, acesta utilizează simboluri non-terminale care conțin ramificații pentru ceea ce reprezintă acesta. A se vedea formatul unui trigger, unde poate conține la început unul din cele 3 cuvinte: BEFORE, AFTER sau INSTEAD OF:



```
135 | CREATE OR REPLACE TRIGGER WORD_str_baio_str_iouod ON WORD_str_fe
136 |
137 |
138 |
139 |
140 | CREATE OR REPLACE TRIGGER WORD_str_baio_str_iouod ON WORD_str_fe
141 |
142 |
143 |
144 |
145 |
146 str_fer : FOR EACH ROW
147 | /* EMPTY */
148 |
149 |
150 str_baio : BEFORE
151 | AFTER
152 | INSTEAD OF
153 | /* EMPTY */
154 |
155 |
156 str_iouod : INSERT
157 | UPDATE
158 | DELETE
159 | str_iouod OR str_iouod

133 END
134 FROM all_tables;
135
136
137
138 CREATE OR REPLACE TRIGGER display_salary_changes
139 BEFORE DELETE OR INSERT OR UPDATE ON customers
140 FOR EACH ROW
141 WHEN (NEW.ID>0)
142 DECLARE
143 sal_diff number;
144 BEGIN
145 sal_diff:=NEW.salary-OLD.salary;
146 dbms_output.put_line('Old_salary'||:OLD.salary);
147 dbms_output.put_line('New_salary'||:NEW.salary);
148 dbms_output.put_line('Salary_difference'||sal_diff);
149 END;
150
151
152 CREATE OR REPLACE PROCEDURE UpdateCourse
153 ( name_in varchar2 )
154 IS
155 CURSOR c1 IS
156 SELECT course number
```

Simbolul de start (marcat cu %start) este instruction. Tokenii sunt marcați prin simbolul %token, declarați ca având același tip de date ca și word din union (adică char*), iar simbolurilor non-terminale le sunt asociate prin %type tipul de date al lui word.

```
%start instruction
%union {char *word;}
%token <word> CREATE DROP DELETE INSERT INTO VALUES WHERE TABLE WORD AND OR
ALTER COLUMN MODIFY RENAME ADD TO UPDATE IF THEN END SET EQ LTE GTE NQ ELSE
FOR LOOP REVERSE IN SELECT FROM ALL PROCEDURE IS EXCEPTION OTHERS BEGINN
REPLACE WHEN WHILE CASE TRIGGER OF AFTER INSTEAD EACH DECLARE ON ROW BEFORE
CURSOR COMMIT CLOSE OPEN FETCH
%type <word> wordsCreate
%type <word> wordsWhere
```

După identificarea structurii exemplului de PL-SQL, se generează traducerea utilizând ca limbaj de programare C, mai precis lucrul pe string-uri.

Mai multe detalii se află în comentariile din interiorul codului².

¹ Această funcție a fost preluată de pe internet, de pe site-ul: <https://www.techiedelight.com/implement-substr-function-c/>

² A se vedea fișierul proiect.y

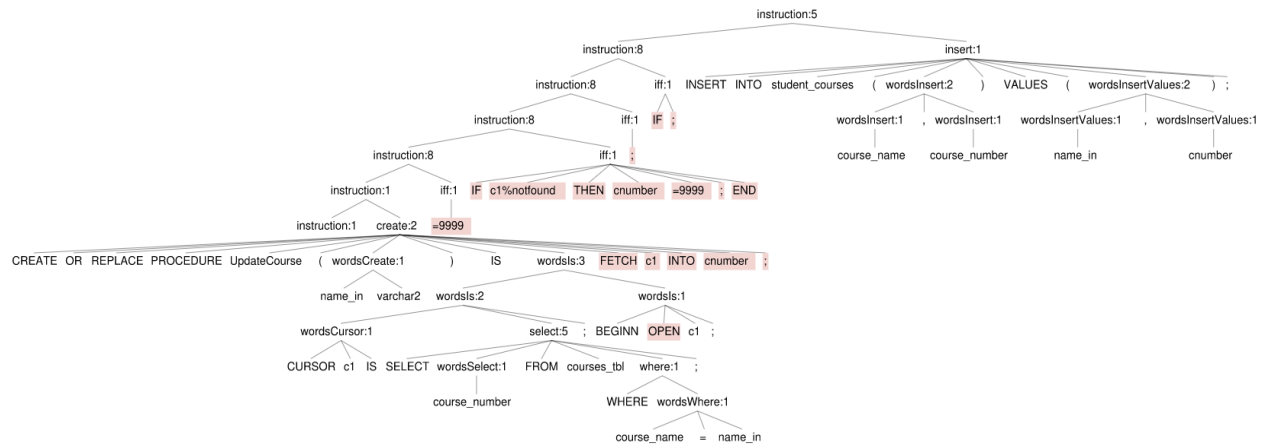
Se poate observa existența unor conflicte, care nu au fost reparate, pentru că Yacc-ul are un mecanism specific de tratare al acestora. Spre exemplu, dacă este vorba despre conflicte de tip shift/reduce, acesta face alege să facă shift.

Arborele de parsare:

Prin intermediul arborelui de parsare s-a analizat adâncimea lui și producțiile prin care a trecut pentru a fi tradus următorul exemplu:

```
CREATE OR REPLACE PROCEDURE UpdateCourse
( name_in varchar2 )
IS
CURSOR c1 IS
SELECT course_number
FROM courses_tbl
WHERE course_name = name_in;;
BEGIN
OPEN c1;
FETCH c1 INTO cnumber;
IF c1%notfound THEN
cnumber:=9999;
END IF;
INSERT INTO student_courses
( course_name, course_number )
VALUES
( name_in, cnumber );
COMMIT;
CLOSE c1;
EXCEPTION
WHEN OTHERS THEN
raise_application_error;
END;
```

Acesta este arborele de parsare generat:



Structura codului:

Codul a fost construit pe modelul oferit în cadrul lucrărilor de laborator³ și respectând regulile de bune practici sugerate de acestea.

³ Lucrările de laborator 4-8: <https://moodle.cs.utcluj.ro/course/view.php?id=569> – site-ul lor

Bibliografie:

1. <https://www.techiedelight.com/implement-substr-function-c/>
2. <https://moodle.cs.utcluj.ro/course/view.php?id=569>
3. <http://labantlr.org/> - pentru arborele de parsare