РОЗРОБКА ІНТЕРПРЕТАТОРА МОВИ РЕЛЯЦІЙНОЇ АЛГЕБРИ ЗАСОБАМИ ГЕНЕРАТОРА ПАРСЕРІВ ANTLR

УДК 004.62

ФІСУН Микола Тихонович
д.т.н., професор, завідувач кафедри інтелектуальних інформаційних систем, ЧДУ ім. Петра Могили.
Наукові інтереси: інтелектуальні інформаційні системи, реінжиніринг, бази даних, комп'ютерна лінгвістика.
e-mail: mykola.fisun@gmail.com

СТЕШОВ Іван Валерійович
інженер-програміст Глобал Лоджик Україна.
Наукові інтереси: компілятори, семантичне розпізнавання, технології програмування на Java.
e-mail: ivanlucky22@gmail.com

ВСТУП

Реляційна модель даних є основною в сучасних СКБД. Мови представлення структурних елементів моделі та маніпулювання ними є невід'ємною складовою будь-якої моделі даних. В реляційній моделі даних (РМД) мовами маніпулювання даними є, зокрема, мова реляційної алгебри (РА) і мова реляційного числення (РЧ). Вони, як і структура РМД, побудовані на формальному математичному базисі і послужили теоретичним підґрунтям для створення мов QBE і SQL. Мова реляційної алгебри (РА) [1] є важливою складовою частиною реляційної моделі даних. Опанування мови РА є запорукою глибинного розуміння найбільш розповсюдженої мови реляційних баз даних – мови SQL[2]. На даний час вивчення мови реляційної алгебри в навчальних закладах проводиться в пасивному режимі, оскільки відсутні компілятори цієї мови. Як правило, під час виконання лабораторних робіт студентам пропонується реалізація операцій РА засобами мов QBE та/або SQL [2].

В [3] було наведено приклад реалізації такого компілятора, однак на той час у ньому не були реалізовані додаткові оператори мови РА (semijoin, semiminus, summarize та інші [1]). Тому задача створення більш ефективних засобів засвоєння в навчальному процесі мови реляційної алгебри залишається актуальною.

Створення компілятора мови реляційної алгебри для навчального процесу можна здійснити як шляхом розробки «власного» програмного забезпечення, використовуючи відомі алгоритми, наприклад з [4, 5], так і за допомогою інструментальних засобів побудови компіляторів, таких як BIZON, YACC, ZUBR [6], ANTLR [7] т.і. В результаті проведеного аналізу було обрано генератор парсерів ANTLR, виходячи з того, що автори вже мали досвід розробки невеликого компілятора на цій платформі [8].

ПОСТАНОВКА ЗАДАЧІ

Основна ідея проекту полягає в доповненні реляційної СКБД такими засобами маніпулювання даними, як мови реляційної алгебри із можливістю її трансляції у скрипти мови SQL. Основні *проблеми*, що потребують вирішення в процесі роботи, пов'язані із забезпеченням взаємодії між різними мовами маніпулювання даними, тому що інтерпретатор мови РА реалізується програмними модулями, а SQL є „внутрішньою" мовою СКБД [1]. Оскільки у більшості сучасних СКБД діагностика помилок в SQL-програмах не розвинута, то потрібно буде розробити розвинуті засоби виявлення помилок граматичного розбору як для мови РА, так і для SQL-коду.

Структура інтерпретатора мови реляційної алгебри представлена на Рис.1, з якого видно, що для автоматизації розробки лексичного аналізатору було використано одну з найбільш популярних систем - ANTLR, вхідною

мовою якого є регулярні вирази [7]. ANTLR, як і багато інших подібних засобів, складається з бібліотеки класів, що полегшують основні операції при розборі (буферизація, пошук і т. д.), і утиліти, які генерують код парсера на основі файлу, що описує граматику мови, що розбирається. Сам ANTLR написаний на Java [7], але дозволяє генерувати код на Java, C++, C#, JavaScrip та ще деяких популярних мовах.

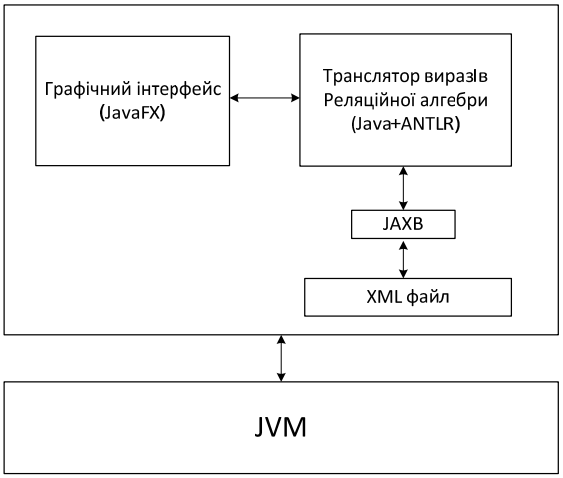

Рисунок 1 – Загальна структура інтерпретатора РА

Реляційна алгебра Кодда [1], для вивчення якої було розроблено програмне забезпечення у якості інтерпретатора РА, складається з набору операторів, що використовують відношення як операнди і повертають відношення як результат. Кодд визначив так звану "початкову" алгебру – набір з восьми операторів, що становлять дві групи, по чотири оператори в кожній:

– Перша група – "Операції над множинами": *об'єднання, переріз, віднімання, декартовий добуток.*

– Друга група – "Спеціальні реляційні операції": *вибірка, проекція, сполучення, ділення.*

В основі операцій реляційної алгебри лежить класична теорія множин, але відповідні операції реляційної алгебри мають деякі особливості.

Почнемо з операції об'єднання (те ж саме стосується перерізу й різниці). Зміст операції об'єднання в реляційній алгебрі в цілому залишається теоретико-множинним, але відношення мають бути сумісними для об'єднання, тобто мати однакові заголовки.

Для взяття добутку відношень, вони повинні бути *сумісні по взяттю розширеного прямого добутку*. Два відношення сумісні по взяттю прямого добутку в тому і тільки в тому випадку, якщо множини імен атрибутів цих відносин не перетинаються. Ці особливості були враховані при розробці програмного продукту, оскільки їх врахування є необхідним елементом правильності дій, що виконуються над відношеннями [1].

Лексичний та синтаксичний аналіз, які є обов'язковими при роботі з природними мовами і мовами програмування, вимагають формалізації мови. Засобом представлення реляційної алгебри у формальній формі було обрано регулярні вирази [4], деякі з них наведено нижче:

*attr_name_list ⟶ attr_name (',' attr_name)**

attr_name ⟶ (table_name '.')? IDENTIFIER;

table_name ⟶ IDENTIFIER;

sign ⟶ |'>'|'<'|'='|'<='|'>=';

Як відомо, процес інтерпретації та компіляції складається з трьох фаз: лексичного, синтаксичного та семантичного аналізу. Отже, наступним етапом розробки програмного комплексу було створення засобів лексичного і синтаксичного аналізу.

ЛЕКСИЧНИЙ АНАЛІЗ

Основна задача лексичного аналізу - розбити вхідний текст, що складається з послідовності одиночних символів, на послідовність слів, чи лексем, тобто виділити ці слова з безперервної послідовності символів [4]. Усі символи вхідної послідовності з цього погляду розділяються на символи, що належать яким-небудь лексемам, і символи, що поділяють лексеми (роздільники).

Для автоматизації розробки лексичного аналізатору було використано одну з найбільш популярних систем - ANTLR, вхідною мовою якого є регулярні вирази. ANTLR, як і багато інших подібних засобів, складається з бібліотеки класів, що полегшують основні операції при розборі (буферизація, пошук і т. д.), і утиліти, які генерують код парсера на основі файлу, що описує граматику розбираємої мови. Сам ANTLR написаний на Java, але дозволяє генерувати код на Java, C++, C#, JavaScrip та ще деяких популярних мовах. Як джерело даних, використовується клас потоку, що дозволяє розбирати дані, що знаходяться у файлі, в пам'яті процесу або приходять через мережу [7]. Результатом роботи ANTLR є два класи – Лексер і парсер. Лексер розбиває потік символів на потік токенів відповідно до правил, а парсер обробляє потік токенів у відповідності з іншими правилами.

ANTLR відноситься до так званих LL(*)-аналізаторів – не обмежених скінченним числом лексем для попереднього перегляду, а здатних приймати рішення визначаючи, чи належать вхідні лексеми регулярній мові.

Коротка довідка елементів мови граматики наведена нижче в табл. 1.

Таблиця 1 –

Довідка елементів мови граматики

Символ	Описання
(...)	підправило
(...)*	повтор підправила 0 чи більше разів
(...)+	повтор підправила 1 чи більше разів
(...)?	підправило може бути відсутнім
{...}	семантичні дії (на мові, яка використовується в якості вихідної, напр. Java)
[...]	параметри правила
|	оператор альтернативи
..	оператор діапазона
~	заперечення
.	будь-який символ
=	присвоювання
:	мітка, початок правила
;	кінець правила
class	клас граматики
extends	визначає базовий клас для граматики
returns	опис повертаємого значення для правила
options	секція опцій
tokens	секція токенів
header	заголовок

Лексичний аналізатор, генерований ANTLR, взаємодіє з синтаксичним аналізатором таким чином: при виклику його синтаксичним аналізатором лексичний аналізатор посимвольно читає залишок входу [4], поки не знаходить найдовший префікс, що може бути співставлений одному з регулярних виразів. Потім він виконує відповідну дію. Як правило, дія повертає керування синтаксичному аналізатору. Якщо це не так, тобто у відповідній дії немає повернення, то лексичний аналізатор продовжує пошук лексем до тих пір, поки дія не поверне керування синтаксичному аналізатору. Повторний пошук лексем аж до явної передачі керування дозволяє лексичному аналізатору правильно обробляти проміжки та коментарі. Синтаксичному аналізатору лексичний аналізатор повертає єдине значення - тип лексеми [8].

У файлі правил ANTLR усі лексичні правила повинні починатись з великої літери. Ключовим словом «*fragment*» позначаються найменші неподільні лексичні правила або токени. Наведемо частину ANTLR-файлу, що реалізує функцію лексичного аналізу реляційної алгебри:

```
WHERE : W H E R E;
JOIN: J O I N;
UNION: U N I O N;
RENAME: R E N A M E;
INTERSECT: I N T E R S E C T;
TIMES: T I M E S;
MINUS: M I N U S;
AS: A S;

LITERAL: STRING_LITERAL | NUMERIC_LITERAL ;

STRING_LITERAL
 : '\'' ( ~'\'' | '\'\'' )* '\''
 | '\"' ( ~'\"' | '\"\"' )* '\"'
 ;

 NUMERIC_LITERAL
 : DIGIT+ ( '.' DIGIT* )? ( E [-+]? DIGIT+ )?
 | '.' DIGIT+ ( E [-+]? DIGIT+ )?
 ;

IDENTIFIER
 : [a-zA-Z_] [a-zA-Z_0-9]*
 ;

SPACES
 : [ \u000B\t\r\n] -> channel(HIDDEN)
 ;

fragment DIGIT : [0-9];
fragment A : [aA];
fragment B : [bB];
fragment C : [cC];
fragment D : [dD];
fragment E : [eE];
fragment F : [fF];
fragment G : [gG];
fragment H : [hH];
fragment I : [iI];
fragment J : [jJ];
fragment K : [kK];
```

fragment *L* : [lL];
fragment *M* : [mM];
fragment *N* : [nN];
fragment *O* : [oO];
fragment *P* : [pP];
fragment *Q* : [qQ];
fragment *R* : [rR];
fragment *S* : [sS];
fragment *T* : [tT];
fragment *U* : [uU];
fragment *V* : [vV];
fragment *W* : [wW];
fragment *X* : [xX];
fragment *Y* : [yY];
fragment *Z* : [zZ];

СИНТАКСИЧНИЙ АНАЛІЗ

Ієрархічний аналіз називається розбором (parsing) або синтаксичним аналізом, котрий включає групування токенів початкової програми в граматичні фрази, які використовуються компілятором (чи інтерпретатором) для синтезу виводу [4].

Ієрархічна структура програми зазвичай виражається рекурсивними правилами. Наприклад, при обумовлені виразів можна дотримуватись таких правил.

Будь-який ідентифікатор (*identifier*) є виразом (*expression*).

Будь-яке число (*number*) є виразом (*expression*).

Якщо *expression1* та *expression2* є виразами, то виразами також є:

expression1 - expression2
expression1 expression2*
(expression1).

Правила (1) і (2) є базовими (нерекурсивними), в той час як (3) обумовлює вираз за допомогою операторів , які застосовуються до інших виразів.

У файлі правил ANTLR всі синтаксичні правила починаються з маленької букви. Аналізуєма мова (в нашому випадку - Реляційна алгебра) описується за допомогою граматики в вигляді форми Бекуса-Наура (БНФ). Результат роботи ANTLR є 2 Java класи – *Lexer* та *Parser* .

Кожен з виразів реляційної алгебри має свої еквіваленти на мові SQL [1]. При трансляції ці еквіваленти будуть слугувати запитами до бази даних.

Вираз Реляційної Алгебри:

A RENAME Attr_old1, Attr_old2 AS Attr_new1, Attr_new2;

Еквівалент на SQL:

SELECT Attr_old1 AS Attr_new1, Attr_old2 AS Attr_new2 FROM A;

Вираз Реляційної Алгебри:

A WHERE Attr_A > 123;

Еквівалент на SQL:

SELECT * FROM A WHERE Attr_A > 123;

Вираз Реляційної Алгебри:

A[Attr1_A, Attr2_A, Attr3_A];

Еквівалент на SQL:

SELECT DISTINCT Attr1_A,Attr2_A,Attr3_A FROM A;

Вираз Реляційної Алгебри:

A UNION B;

Еквівалент на SQL:

(SELECT * FROM A) UNION (SELECT * FROM B);

Вираз Реляційної Алгебри:

A INTERSECT {Attr_A, Attr_B} B;

Еквівалент на SQL:

SELECT * FROM A WHERE Attr_A IN (SELECT Attr_B FROM B);

Вираз Реляційної Алгебри:

A MINUS {Attr_A, Attr_B} B;

Еквівалент на SQL:

SELECT * FROM A WHERE Attr_A NOT IN (SELECT Attr_B FROM B);

Вираз Реляційної Алгебри:

A TIMES B;

Еквівалент на SQL:

SELECT * FROM A, B;

Вираз Реляційної Алгебри:

Money1 JOIN {Money1, Money2} Price;

Еквівалент на SQL:

SELECT * FROM Money1 LEFT JOIN Price ON Money1.Money1 = Price.Money2;

Вираз Реляційної Алгебри:

(ORDERS[pnum, dnum]) DIVIDEBY{pnum, dnum} (Details[dnum]);

Еквівалент на SQL:

SELECT DISTINCT SYSTEM_TABLE_1.pnum FROM (SELECT DISTINCT pnum,dnum FROM ORDERS) SYSTEM_TABLE_1 WHERE NOT EXIST (SELECT * FROM (SELECT

DISTINCT dnum FROM Details) SYSTEM_TABLE_2 WHERE NOT EXIST (SELECT * FROM (SELECT DISTINCT pnum,dnum FROM ORDERS) SYSTEM_TABLE_3 WHERE SYSTEM_TABLE_1.pnum = SYSTEM_TABLE_3.pnum AND SYSTEM_TABLE_3.dnum = SYSTEM_TABLE_2.dnum));[5]

Ядром ANTLR-специфікації є набір граматичних правил. Кожне правило описує синтаксичну конструкцію і дає їй ім'я:

```
parse returns [List<Statement> statements]
@init{$statements = new LinkedList<Statement>();}
:    ( stmt ';' {$statements.add($stmt.statement);} )+
EOF
;
stmt returns [Statement statement]
:
 | union_stmt {$statement =
$union_stmt.statement;}
 | where_stmt {$statement =
$where_stmt.statement;}
 | rename_stmt {$statement =
$rename_stmt.statement;}
 | select_stmt {$statement = $select_stmt.statement;}
 | intersect_stmt {$statement =
$intersect_stmt.statement;}
 | minus_stmt {$statement =
$minus_stmt.statement;}
 | times_stmt {$statement = $times_stmt.statement;}
 | join_stmt {$statement = $join_stmt.statement;}
;
union_stmt returns [Statement statement]
:    t1 = table_name UNION t2 = table_name
     {$statement = new UnionStatement($t1.text,
$t2.text); }
;
where_stmt returns [Statement statement]
:    table_name WHERE attr_name sign LITERAL
     {$statement = new
WhereStatement($table_name.text, $attr_name.text,
$sign.text, $LITERAL.text); }
;
select_stmt returns [Statement statement]
:    table_name '[' attr_name_list ']'
     {$statement = new
SelectStatement($table_name.text,
$attr_name_list.text); }
;
intersect_stmt returns [Statement statement]
:    t1 = table_name INTERSECT '{' attr_name_list '}'
t2 = table_name
     { $statement = new
IntersectStatement($t1.text,$t1.text,$attr_name_list.text)
; }
;
minus_stmt returns [Statement statement]
:    t1 = table_name MINUS '{' attr_name_list '}' t2
= table_name
     { $statement = new
MinusStatement($t1.text,$t1.text,$attr_name_list.text); }
;
join_stmt returns [Statement statement]
:    a1 = attr_name JOIN '{' t1 = table_name ',' t2 =
table_name '}' a2 =attr_name
     { $statement = new
JoinStatement($t1.text,$t2.text,$a1.text,$a2.text); }
;
```

У кожному правилі можна використовувати звичайний Java-код, що можливо писати тільки у фігурних дужках. Також, ANTLR надає можливість оперувати його ж змінними, ставлячи знак «$» на початку змінної [7]. Завдяки цьому коду можна зрозуміти процес трансляції виразів. Головним правилом, з якого починається процес трансляції є правило *parse*, що повертає список виразів (*statement*) які вдалося розпізнати [10]. У правилі «*stmt*» (від *statement*) перелічені види виразів що можуть бути розпізнані. Кожен вираз – нове синтаксичне правило, що повертає Java об'єкт типу *Statement (інтерфейс)*. У кожному правилі змінна *statement* ініціалізується окремим конструктором, в який передаються параметри середи ANTLR, та стає доступною у «батьківському правилі», як атрибут правила, що викликалось.

Після передачі списку об'єктів *statement* у основну Java програму, виконується логіка згідно з кожним типом виразу що були отримані.

Замість бази даних використовується звичайний XML файл. Для того, щоб налагодити роботу Java з XML використовується технологія JAXB, що дозволяє працювати з Java об'єктами як з тегами XML файлу. Приклад такого файлу представлено нижче:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<database>
    <table name="Students">
        <column type="char">Second_Name</column>
        <column type="int">Age</column>
        <row>
            <value>Shevchenko</value>
            <value>20</value>
        </row>
    </table>
</database>
```

Для проекції тегів у об'єкти були створенні такі класи: *Database*, *Table*, *Column*, *Row*.

Інтерфейс програми створено на основі JavaFX бібліотеки, що вбудована у JDK, починаючи з 8-ої версії. В цієї бібліотеки є всі необхідні примітиви інтерфейсу сучасного графічного програмного продукту. Більш того, графічний інтерфейс можливо створювати у спеціальному *fxml* файлі, що дуже нагадує створення вебсторінки засобами HTML. Також компоненти JavaFX підтримують CSS для стилізації. Для роботи з інтерпретатором інтерфейсом, зокрема, передбачені вікна для визначення бази даних (DB name) і оператора реляційної алгебри (RA operator), а також кнопки Execute і Exit.

ВИСНОВКИ

Засобами компілятора компіляторів ANTLR вдалося розроблено синтаксичні правила мови реляційної алгебри та інтерпретатор цієї мови, що розпізнає запити мови РА та, згідно з розпізнаними запитами, імплементує логіку мовою програмування Java та здійснює відповідне маніпулювання реальними даними, представленими у вигляді XML-структури. В подальшому передбачається розробка граматики мови реляційного числення (РЧ) та розширення розробленого інтерпретатора за рахунок мови РЧ. Планується впровадження цих інтерпретаторів у навчальний процес.

ЛІТЕРАТУРА:

1. Christopher J. Date, Vvedeniye v systemy baz dannyh. – M.: Izdatelsky dom «Williams», 2001. – 1071 p.
2. Pasichnik V.V. , Reznichenko V.A., Organizatcia baz danyh ta znan: pidruchnyk dlia VUZiv. – K.: BHV, 2006. – 384 s.
3. Zasoby pidtrymky protsesiv nadbannia znan' i umin' pry opanuvanni reliatsiynoii algebry ta reliatsiynogo //M.T. Fisun, O.V. Gnezdionova, I.V. Suprun, D.O. Kozachenko //Proceeding of the First International Conference «New Information Technologies in Education for All. ITEA-2006». – K.: Vydannia dim «Fkademaperiodyka», 2006. – 403 c.
4. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman Aho. Kompiliatory: printsypy, technlogii, instrumenty: Pereklad z angliiskoii. – M.: Izdatelsky dom «Williams», 2001. – 768 s.
5. J.E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. Vvedenie v teoriyu avtomatov, yazykov i vichesliniy, 2-e izdaniye: Pereklad z angliiskoii. – M.: Izdatelsky dom «Williams», 2008. – 528 s.
6. Kosteltsev A.V. Postroeniye interpretatorov i kompiliatorov. Ispolzovaniye program BIZON, BYACC, ZUBR. – SPB: «Nauka I tehnika», 2001. – 219 s.
7. Terence Parr: Definitive ANTLR Reference. «ANTLR dlia razrabotki kompiliatora na jadzyke Java»[Elektronnyi resurs]. – Regym dostypy : URL: https://vimeo.com/25753384. – Zagol. z ekranu.
8. Steshov I.V., Ukrainomovnyi kompiliator z naborom funktsiy dlia vyrishennia matematychnyh zadach //Vseukr. nauk.-pract. konf . molod. uchenyh, aspirantiv i studentiv «Intelectualni informatciyni systemy».– Mykolaiv: ChDU im. P. Mogyly, 2015. – S.43-44.
9. Serebriakov V.A. Lektsii po konstruyuvanniyu kompiliatoriv. – M., 1993. – 175 s.
10. V.L. Bogdanov, V.S. Gordeev. Practychniy dustup napisannia syntaksichnogo analizatora movy programuvannia Cobol, 2000. – 7 s.

Рецензент: *д.т.н., доц. Шерстюк В.Г., Херсонський національний технічний університет.*