МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ПРАВИЛ ГРАММАТИКИ ПОСТРОЕНИЯ АБСТРАКТНОГО СИНТАКСИЧЕСКОГО ДЕРЕВА ДЛЯ ЯЗЫКА PL/PGSQL И ГЕНЕРАЦИЯ ЕДИНОЙ ХРАНИМОЙ ПРОЦЕДУРЫ ДЛЯ ОПТИМИЗАЦИИ РАБОТЫ БАЗЫ ДАННЫХ

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

Студентки 2 курса 273 группы
направления 02.04.03 — Математическое обеспечение и администрирование
информационных систем
факультета КНиИТ
Шаймардановой Анар Галиевны

Научный руководитель
зав. каф., к. ф.-м. н., доцент _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одной из самых популярных бесплатных реляционных СУБД на текущий момент является PostgreSQL, которая для обработки данных предоставляет целый ряд процедурных языков, в том числе plpgsql — Postgres procedural language. При помощи plpgsql разработчики создают наборы функций или хранимых процедур (ХП), которые существенно облегчают решение производственных задач, выполняя обработку данных непосредственно на самом сервере СУБД, что значительно увеличивает время отклика в современной многослойной архитектуре приложения.

Разработчики, создавая такие функции обычно стремятся к тому, чтобы каждая функция выполняла строго свои задачи и была обозримой, т. е. содержала как можно меньше строк кода. Это позволяет быстрее отладить код и уменьшить количество ошибок типа «copy-past». Но так как одна маленькая функция не имеет бизнес смысла, то создаются управляющие функции, которые вмещают в себя конкретную бизнес-логику и соответственно выполняют вызовы других функций, затем выполняют финальную обработку данных и возвращают результат в точку вызова.

Таким образом формируется иерархический стек вызовов хранимых процедур, который как правило усугубляется работой в цикле. В результате на СУБД обрушивается шквал вызовов хранимых процедур, заставляя ее без конца переключать контекст вызовов. Это неизбежно приводит к деградации производительности.

Решение данной проблемы может состоять в том, чтобы, проанализировав код входной управляющей функции, создать ее клон, в котором будут помещены коды вызываемых ею подфункций.

В этом случае приложение, решающее данную проблему, должно получать на вход список имен хранимых процедур, включающий основную ХП и вызываемые подпроцедуры, генерировать объединенное тело ХП, в котором вместо вызовов подпроцедур будут вставлены тела соответствующих процедур. Такое переопределение тела хранимой процедуры значительно ускорит ее выполнение, особенно если подпроцедуры хранимой процедуры вызываются в цикле.

Целью настоящей работы является разработка правил грамматики построения абстрактного синтаксического дерева для языка PL/pgSQL и генера-

ция единой хранимой процедуры из набора вызываемых хранимых процедур для оптимизации работы базы данных. Отметим, что в результате работы библиотеки ANTLRv4 для языка Java по описанным правилам грамматики генерируется абстрактное синтаксическое дерево, информацию из которого мы в дальнейшем используем для склейки хранимых процедур.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- определиться с выбором инструмента лексического анализа;
- создать для него словарь ключевых слов (для языка PL/pgSQL);
- описать правила построения абстрактного синтаксического дерева;
- создать набор тестов ХП для проверки абстрактного синтаксического дерева.

Далее необходимо реализовать приложение, которое будет:

- фиксировать нужные данные, полученные в результате парсинга ХП;
- выделять тела подпроцедур;
- правильно переименовывать передаваемые параметры в подпроцедурах;
- инициализировать локальные переменные подпроцедур;
- скомпоновывать тело ХП, параметры и локальные переменные в код и заменять вызов ХП на этот новый код.

Таким образом на выходе мы получим удлиненный код хранимой процедуры, который будет работать быстрее, т. к. будет исключены множественные обращения к подфункциям.

Магистерская работа состоит двух глав. Работа включается в себя 96 страниц, 29 библиографических источников. Первая глава называется «Обзор, теоретические сведения и используемые технологии» и включает в себя обзор существующих аналогов решения, а также описание технологий, использованных при разработке решения. Вторая глава называется «Реализация функционала и примеры работы приложения» и включает описание кода, реализованного в работе, примеры работы приложения, а также сравнение времени работы хранимых процедур с вызовами подпроцедур и объединенной хранимой процедуры.

1 Обзор, теоретические сведения и используемые технологии

1.1 Обзор существующих решений

На сегодняшний день не существует решений по ускорению вызовов хранимых процедур в цикле. Существует несколько способов по ускорению выполнения запросов в PostgreSQL, по ускорению работы самой СУБД. Однако данные решения не помогут значительно ускорить выполнение запросов в нашем случае, при вызове хранимых процедур в цикле. Тем не менее, их стоит рассмотреть и, возможно, учитывать в дальнейшем при проектировании программы. В работе представлено 14 похожих решений.

1.1.1 Теоретические сведения

1.1.2 Библиотека ANTLRv4

Одним из удобных инструментов для парсинга программы с использованием словаря ключевых слов и формирования AST дерева является ANTLR 4. Его мы и собираемся использовать в при реализации парсера языка plpgsql.

1.1.3 Абстрактное синтаксическое дерево

Для того, чтобы реализовать парсер plpgsql, необходимо описать, каким образом выполняется лексический и синтаксический анализ программы, и как формируется абстрактное синтаксическое дерево. Также абстрактное синтаксическое дереве генерируется с помощью библиотеки ANTLRv4, основываясь на описанных правилах грамматики, которое мы в дальнейшем используем для получение нужной нам информации, полученной в результате парсинга хранимых процедур.

1.1.4 Структура данных связанный список в Java

Отметим, что структура реализованного в практической части класс PgTreeNode была выбрана, основываясь на логике построения абстрактного синтаксического дерева, генерируемого средствами библиотеки ANTLRv4, а также основываясь на структуре данных связанный список.

1.2 Технологии языка Java

Были описаны следующие технологии языка Java: интерфейс Comparable, класс ArrayList, контракт equals, контракт hashcode, классы StringBuilder и StringBuffer, некоторые классы библиотеки ANTLRv4 для Java.

2 Реализация функционала и примеры работы приложения

Опишем план, который необходимо выполнить для достижения поставленной цели магистерской работы:

1. реализовать правила грамматики построения абстрактного синтаксического дерева для языка PL/pgSQL;
2. реализовать функционал, с помощью которого мы сможем генерировать тело объединенной хранимой процедуры;
3. написать тесты к полученному функционалу;
4. сравнить время работы процедур с подвызовами и объединенной хранимой процедуры.

2.1 Реализованные классы

Реализованы:

— перечисление PgPlSqlElEnum. Данное перечисление используется для описания типов элементов;
— класс PgFuncReplacementPart. В данном классе мы храним результаты прохождения по AST дереву хранимой процедуры;
— класс PgVarDefinition. Данный класс используется для хранения информации о переменных, определяемых в хранимых процедурах;
— класс PgFuncBodyPartsBag. Представляет собой класс-контейнер, который фиксирует результаты прохода по AST дереву хранимой процедуры в списке типа PgFuncReplacementPart;
— класс PgFuncDefined. Используется для хранения информации по хранимой процедуре;
— класс PgFuncInvoked. Используется для хранения информации по вызываемой хранимой подпроцедуре;
— класс PgParsingResult. Используется для хранения информации по: самой ХП (с телом ХП) в экземпляре PgFuncDefined; по вызываемым ХП: хранится список вызываемых подпроцедур типа PgFuncInvoked; хранится флаг parsingErrorHappened, определяющий, произошла ли ошибка в результате парсинга;
— класс PgParsingResutBag. Используется для складывания результатов успешного разбора ХП в списке типа PgParsingResult:;
— функциональный интерфейс ITreeHandler, содержащий единственный ме-

тод apply(T t). Данный интерфейс используется при построении дерева хранимой процедуры;

— собой функциональный интерфейс ITreeState, содержащий единственный метод doState. Данный интерфейс используется далее при построении дерева хранимой процедуры;
— класс PgTreeNodeWalker позволяет переходить от объекта PgTreeNode к другому PgTreeNode при визуальном построении дерева вызовов;
— класс PgTreeNode. Данный класс представляет собой дерево, формируемое в результате построения AST дерева хранимой процедуры;
— класс ParserHelper. Он используется для описания некоторых методов для работы с парсером;
— класс PgSqlIncludeListenerResult расширяет абстрактный класс PgSqlIncludeListener. Данный класс фиксирует результаты прохождения AST дерева хранимой процедуры;
— класс PgGenGlueFunctions. Данный класс используется для склейки общего тела ХП и подпроцедур, основываясь на полученных результатах парсинга.

2.2 Пример работы приложения

Рассмотрим на тестах примеры работы приложения.

В тестовом классе TestPlPgSql определим метод testGeneratingOutThreeLevel. В данном методе на трех процедурах мы проверим, будет ли сгенерирована общая ХП из трех процедур. Будем проверять работу приложения на следующих ХП:

— основная процедура p02_void_call_perf:

```
CREATE OR REPLACE FUNCTION p02_void_call_perf(p1 int4)
    RETURNS void AS
$$
DECLARE
   m_par1 int2 := 30;
   m_par2 int2 := 30;
   m_res int4 := null;
BEGIN

   m_res := p02_int4_v4(m_par1, m_par2);
```

```
    RAISE NOTICE 'Result = %', m_res;

  EXCEPTION
    WHEN others THEN RAISE NOTICE 'Catch exeption for
    ↪ p02_void_call_perf';
  END;
  $$
  LANGUAGE 'plpgsql';
```

— вызываемая подпроцедура второго уровня p02_int4_v4:

```
  CREATE OR REPLACE FUNCTION p02_int4_v4(int4, int2)
  RETURNS int4
  LANGUAGE plpgsql AS $$
  BEGIN
    RAISE NOTICE 'Called p02_int4_v4(int4, int2)';

    PERFORM p01_void_v2($2);

    RETURN $1 + $2;
  END; $$;
```

— вызываемая подпроцедура третьего уровня p01_void_v2:

```
  CREATE OR REPLACE FUNCTION p01_void_v2(int2)
      RETURNS void AS
  $BODY$
  DECLARE
    m_res int4;
  BEGIN
    m_res := $1;
    RAISE NOTICE 'Result p01_void_v2 = %', m_res;
  END;
  $BODY$
  LANGUAGE plpgsql;
```

Внутри теста мы создаем stream из вызываемых подпроцедур, вызываем парсинг ХП с помощью вызова метода parseFromEnum:

```
@Test
public void testGeneratingOutThreeLevel() {
    Stream<PgPlSQLEnums> stream =
    ↪ PgPlSQLEnums.getPlPgSQLResources("p01_void_v2.sql",
          "p02_int4_v4.sql", "p02_void_call_perf.sql");
```

```
PgParsingResultBag pgParsingResultBag = new
↪ PgParseFunctions().parseFromEnum(stream);
```

далее вызываем метод makeTree построения дерева ХП, выводим полученное дерево основной ХП и подпроцедур на экран, после чего вызываем склейку ХП вызовом метода glue:

```
PgTreeNode root = ParserHelper.makeTree(pgParsingResultBag);
root.drawTree();

new PgGenGlueFunctions().glue(root);
```

Видим, что в результате работы теста в консоль выводится три уровня вызова хранимых процедур (см. рисунок 1):

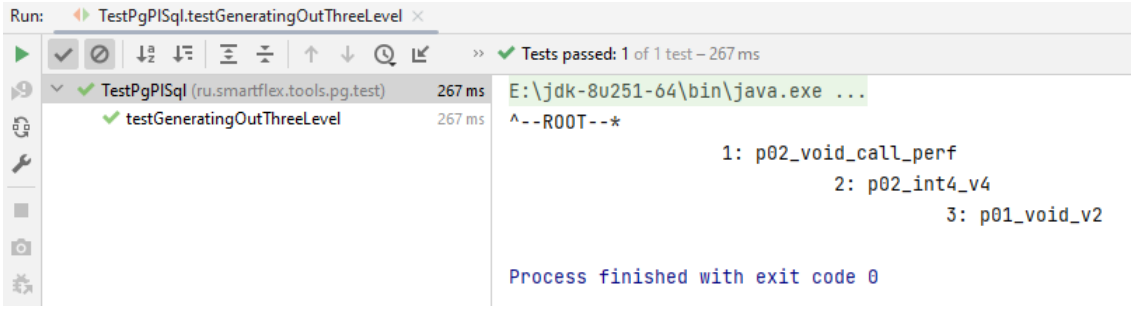

Рисунок 1 – Результат работы теста testGeneratingOutThreeLevel()

2.3 Сравнение времени работы хранимой процедуры с вызовами подпроцедур и объединенной хранимой процедуры

Рассмотрим следующий пример, на котором мы сравним время работы процедуры, в которой вызывается подпроцедура в цикле, и процедуры, которая была получена в результате работы приложения и представляющая собой объединенную с подпроцедурой процедуру.

Расчеты производились на компьютере с оперативной памятью 32 Гб, с процессором AMD Ryzen 5 3400G, используя графический клиент pgAdmin 4 версии 6.10.

Пусть у нас дана следующая задача: мы передаем в качестве параметра p1 в процедуру число, и хотим посчитать сумму всех чисел, кратных двум, от 1 до p1. У нас есть основная ХП p10_void_call_loop, в которую мы передаем параметр p1 и считаем сумму:

```
CREATE OR REPLACE FUNCTION p10_void_call_loop(p1 int8)
    RETURNS void AS
$$
DECLARE
   ind int8 := 1;
   sum_res int8 := 0;
BEGIN

   LOOP
       IF ind % 2 = 0 THEN
          sum_res := p11_int8_sum(ind, sum_res);
       END IF;
       ind := ind + 1;
   EXIT WHEN ind > p1;
   END LOOP;

   RAISE NOTICE 'Sum = %', sum_res;

END;
$$
LANGUAGE 'plpgsql';
```

и дочерняя ХП p11_int8_sum, в которой мы передаем в качестве параметров два числа и возвращаем их сумму:

```
CREATE OR REPLACE FUNCTION p11_int8_sum(int8, int8)
RETURNS int8
LANGUAGE plpgsql AS $$
BEGIN
   RETURN $1 + $2;
END; $$;
```

В результате запуска приложения мы получим следующую объединенную ХП:

```
CREATE OR REPLACE FUNCTION p10_void_call_loop_un(p1 int8)
    RETURNS void AS
$$
DECLARE
   ind int8 := 1;
   sum_res int8 := 0;
BEGIN
```

```
   LOOP
        IF ind % 2 = 0 THEN

--
-- ***** insert sub-body BEGIN *****: "p11_int8_sum"
--
      BEGIN
         sum_res := ind + sum_res;
      END;
--
-- ***** insert sub-body END *****: "p11_int8_sum"
--

       END IF;
       ind := ind + 1;
   EXIT WHEN ind > p1;
   END LOOP;

   RAISE NOTICE 'Sum = %', sum_res;

END;
$$
LANGUAGE 'plpgsql';
```

Создадим данные три ХП в базе данных и сравним их время работы на разных входных данных.

В таблице 1 можем увидеть полное сравнение времени работы:

Таблица 1 – Сравнение времени работы хранимых процедур

Значение p1	t ХП без объед., мс	t объед. ХП, мс
4	82	60
100	51	46
1000	55	46
5000	58	47
10000	61	48
50000	120	65
100000	120	77
300000	210	125
500000	329	190
750000	500	262
1000000	5744	2836

Как можем заметить, при небольших входных данных время работы объединенной ХП не сильно отличается от ХП с вызовом подпроцедуры. При больших значениях разница во времени выполнения более заметна. Начиная с p1 = 300000 можем заметить значительный прирост по времени работы объединенной ХП.

ЗАКЛЮЧЕНИЕ

Все задачи, поставленные в магистерской работе работе, а именно:

- определиться с выбором инструмента лексического анализа;
- создать для него словарь ключевых слов (для языка PL/pgSQL);
- описать правила построения абстрактного синтаксического дерева;
- создать набор тестов ХП для проверки абстрактного синтаксического дерева.
- реализовать приложение, которое выполняет следующие функции:
  - фиксирует нужные данные, полученные в результате парсинга ХП;
  - выделяет тела подпроцедур;
  - правильно переименовывает передаваемые параметры в подпроцедурах;
  - инициализирует локальные переменные подпроцедур;
  - скомпоновывает тело ХП, параметры и локальные переменные в код и заменяет вызов ХП на этот новый код.

были выполнены. Цель была достигнута.

Результаты работы были представлены в студенческой научно-практической конференции «Научное сообщество студентов XXI столетия. ТЕХНИЧЕСКИЕ НАУКИ» (https://sibac.info/studconf/tech/cxxiii/282040).

Данное приложение позволит значительно ускорить время выполнение хранимых процедур в базах данных для СУБД PostgreSQL.