РАЗРАБОТКА И ВНЕДРЕНИЕ ОБЩЕГО АНАЛИЗАТОРА SQL

М.В. Болябкин[1], *магистрант*
А.И. Панов[2], *студент*
Н.А. Башмуров[2], *студент*
[1]Московский технический университет связи и информатики
[2]Нижегородский государственный университет им. Н.И. Лобачевского
[1](Россия, г. Москва)
[2](Россия, г. Нижний Новгород)

DOI:10.24412/2500-1000-2022-1-1-55-61

Аннотация. *С быстрым развитием компьютерных сетей информационная безопасность баз данных становится все более и более важной. В целях защиты баз данных необходимо проанализировать и восстановить запросы SQL. Однако в продуктах аудита баз данных, представленных на рынке, отсутствует точный анализ SQL-запросов. Поэтому в данной статье анализ SQL строится с учетом четырех аспектов: лексического анализа, синтаксического анализа, оптимизации связующего дерева SQL и обнаружения ошибок.*

Ключевые слова: *SQL, базы данных, анализатор, синтаксический анализ, безопасность.*

В настоящее время широко используются реляционные базы данных, и язык SQL является главным приоритетом реляционной базы данных. Большинство продуктов для аудита баз данных на рынке пропускают аудит SQL-запросов длиной более 1,5 кб, то есть они напрямую теряют пакеты данных без анализа. Некоторые хакеры используют эти уязвимости для запуска атак. Точная технология синтаксического анализа длинных предложений определяет эффект защиты от таких атак. Поэтому разработка общего анализатора SQL для поддержки восстановления операторов SQL и обработки ошибок стала общим требованием многих приложений.

В этой статье строится общий синтаксический анализатор SQL с учетом аспектов лексического анализа, синтаксического анализа, оптимизации связующего дерева SQL и обнаружения ошибок синтаксического анализатора SQL, чтобы избежать пропусков анализа операторов SQL и улучшить способность синтаксического анализатора обрабатывать ошибки.

Создание синтаксического анализатора

1. Лексический анализ

Лексический анализ является первым этапом процесса компиляции и основой компиляции. Существует два лексических правила: EBNF и BNF. В этой статье используется парадигма EBNF для описания стандартного синтаксиса SQL.

Задача лексического анализа на этом этапе состоит в том, чтобы прочитать исходную программу символ за символом слева направо, идентифицировать слова в соответствии с правилами словообразования, а выходным результатом является последовательность слов (токен), которая обеспечивает основу для синтаксического анализа [1]. Кроме того, также необходимо выполнить предварительную обработку исходной программы, такую как фильтрация бесполезных пробелов, пропуск, возврат каретки, перевод строки и комментарии в SQL. Кроме того, ANTLR позволяет встроить язык хоста в лексический файл. Код C, встроенный в лексический анализатор в этой статье, помогает анализатору сопоставлять информацию об ошибках, найденную с помощью лексического анализа, с местоположением ошибки в исходном коде SQL. Лексический анализатор анализирует пять видов токенов: (1) зарезервированные слова, такие как «создать» и «таблица» в SQL; (2) Идентификатор, такой как имя константы и имя таблицы; (3) Различные типы констант, такие как

2,72, 3,14 и т. д. (4) Операторы, такие как "+","<", "и"и т. д.; (5) Разделители, такие как круглые скобки, скобки и т. д.

После определения этих слов нам также необходимо определить морфологию специального употребления, потому что в процессе синтаксического анализа все символы должны быть обработаны. Когда пользователь вводит другие символы, анализатор грамматики выдает ошибку неопределенной сегментации слов. Чтобы избежать таких ошибок, нам необходимо дополнительно определить морфологию использования, "~ (...)"означает слова, отличные от любых ранее определенных слов.

2. Синтаксический анализ

После лексического анализа переходим к грамматическому анализу. Любой неразрешенный конфликт может привести к тому, что анализатор не сможет точно распознать правила, и программа не сможет работать корректно. Только разработав четкие и бесконфликтные грамматические правила, мы сможем создать грамматическое дерево. Однако большинство грамматик неоднозначны. Во-вторых, в большинстве языковых спецификаций используется специальный рекурсивный метод, называемый левой рекурсией. В настоящее время классическая форма синтаксиса и синтаксического анализа сверху вниз не может обрабатывать левую рекурсию.

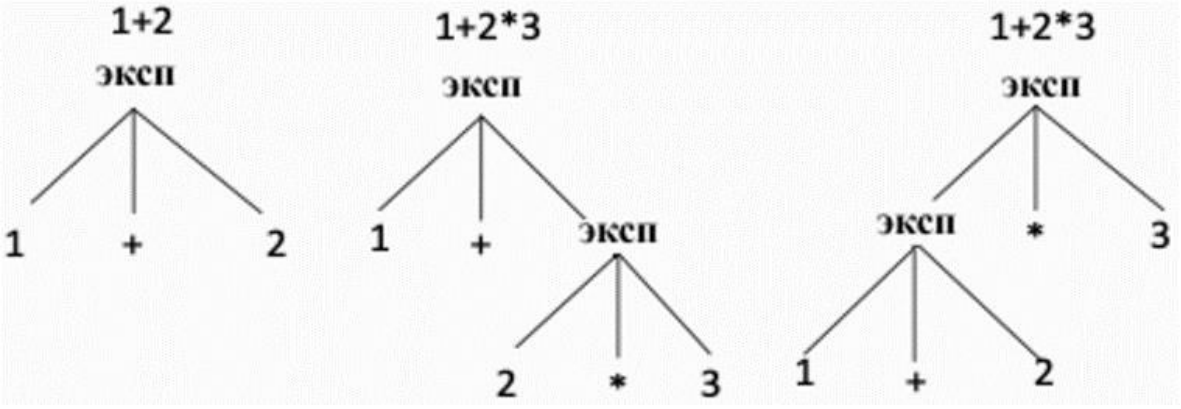

Рис. 1. Дерево синтаксического анализа, представленное различными способами

Однако для ввода таких данных, как 1 + 2 * 3, входные данные могут быть интерпретированы двумя способами. Разница между деревом синтаксического анализа в середине и справа на рисунке 1 заключается в том, что дерево синтаксического анализа, добавленное в середину, указывает, что 1 добавляется к результату умножения 2 и 3, в то время как дерево синтаксического анализа справа указывает, что результат сложения 1 и 2 умножается на 3. Это проблема приоритета оператора. Традиционный синтаксис не может указать приоритет. ANTLR решает проблему неоднозначности, отдавая предпочтение передним альтернативным ветвям, что неявно позволяет указывать приоритет оператора. Например, в правилах ЭКСП правила умножения предшествуют правилам сложения, поэтому ANTLR будет отдавать приоритет умножению при решении проблемы неоднозначности 1 + 2 * 3. В сочетании с логической структурой и прикладными требованиями синтаксиса SQL, на основе работы с присущими синтаксическим правилам SQL, пользовательские правила добавляются в анализатор SQL в соответствии с функциональными требованиями. Синтаксический анализатор автоматически генерируется в соответствии с синтаксической парадигмой. Синтаксический анализатор вызывает лексический анализатор, чтобы получить маркер следующего слова. В процессе синтаксического анализа определяемые пользователем правила и действия встраиваются в планирование анализа.

Оптимизация синтаксического дерева

1. Оптимизация реляционной алгебры

Применение некоторых полезных алгебраических законов реляционной алгеб-

ры к базе данных может эффективно повысить скорость запросов[2]. В соответствии с теорией закона реляционной алгебры получается следующее: 1) при преобразовании запросов мы должны учитывать другие бинарные операции, такие как сначала выбор проекции, а затем подключение; 2) При подключении вы должны сначала подключить небольшие отношения, а затем подключить большие отношения. Если есть связь: Workerinfor (идентификатор работника, имя, телефон, адрес, пол); Positioninfor (должность, год, идентификатор работника); Запрос должности и места рождения сотрудников мужского пола, принятых на работу в 2021 году в этих двух отношениях, выбор должности, addr из workerinfor, positioninfor, где год = 2021 и пол = m, а идентификатор работника = ID;

Слева находится синтаксическое дерево после завершения этапа компиляции запроса, а справа - оптимизированное синтаксическое дерево, как показано на рисунке 2.

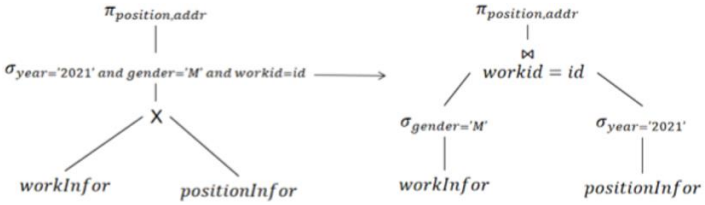

Рис. 2. Синтаксическое дерево и оптимизированное синтаксическое дерево

В процессе оптимизации, сначала, происходит построение связей между возможными вариантами; Далее, два других условия разделены на два варианта σ. Операции перемещаются вниз к соответствующим отношениям их соответствующих деревьев. Установлено, что оптимизированное синтаксическое дерево может сэкономить место для хранения.

2. Устранение бесполезных условий

Удаление бесполезных условий может быть выполнено только в соответствии с самим SQL и структурой таблицы, и существует множество случаев оптимизации. Чтобы избежать слишком громоздкого описания, следующие два случая анализируются в соответствии с рисунком 3.

а) 1 = 1 и (m > 3 и n > 4)

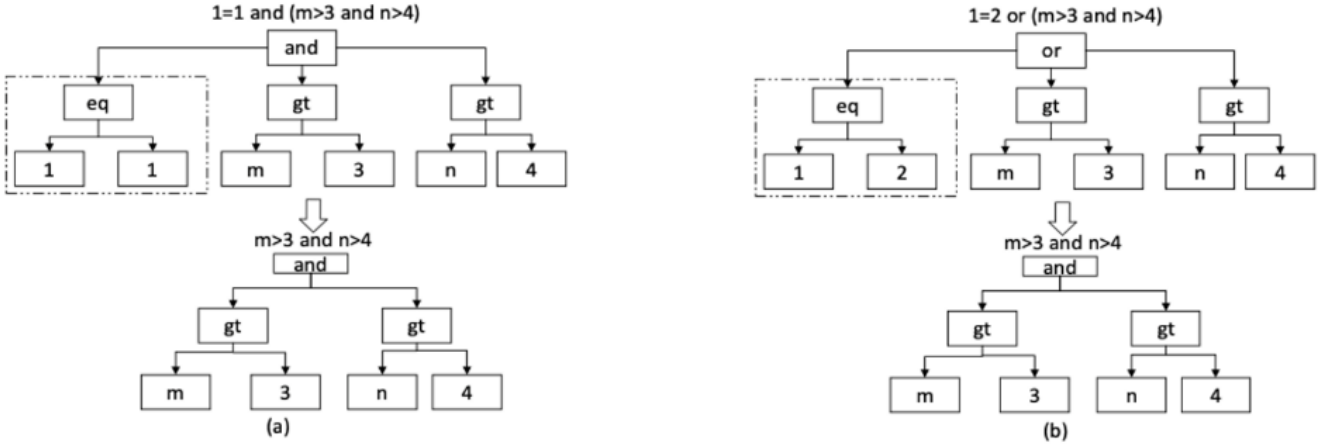

Рис. 3. Синтаксическое дерево (а) и (б)

Из синтаксического дерева рисунка 3 (а) видно, что значение в пунктирной рамке всегда равно true, а верхний слой равен "и", поэтому пунктирная рамка является бесполезным условием. Непосредственное удаление этого условия из синтаксического дерева позволяет его оптимизировать.

б) 1 = 2 или (m > 3 и n > 4)

Из синтаксического дерева рисунка 3 (b) видно, что значение в пунктирном поле всегда равно false, а верхний слой равен

"или", поэтому пунктирное поле является бесполезным условием, которое непосредственно удаляется из синтаксического дерева, чтобы избежать SQL-инъекции и защитить информацию базы данных.

Обнаружение ошибок в различных кодах

Код SQL анализируется в дереве идиом, а затем обнаружение семантических ошибок кода SQL преобразуется в обнаружение ошибок синтаксического дерева SQL. Обнаружение ошибок синтаксического дерева может быть решено путем сопоставления древовидной структуры. Алгоритм обнаружения использует расстояние редактирования дерева [3]. Расстояние редактирования дерева также называется расстоянием Левенштейна. В качестве примера возьмем строку, расстояние редактирования между строкой a и строкой b – это минимальное количество операций для преобразования a в b. Существует три операции: вставка символа, удаление символа и замена символа.

Формула используется для представления расстояния редактирования строк a и b, где I представляет длину a, а j представляет длину b.

$$ed_{a,b}(i,j) = \begin{cases} \min(i,j) = 0 & \max(i,j) \\ a_i = b_j & ed_{a,b}(i-1,j-1) \\ a_i < b_j & \min(ed_{a,b}(i-1,j-1)+1, ed_{a,b}(i-1,j\_+1, ed_{a,b}(i,j-1)+1) \end{cases}$$

min (i, j) = 0, что означает, что строка пуста, а расстояние редактирования равно длине другой непустой строки;

Когда значение строки ai равно bj, расстояние редактирования равно расстоянию редактирования одного символа в строках a и b;

Если значение ai не равно bj, отредактировать минимальное значение в следующих трех случаях:

1) eda,b (i − 1, j)+1 удалить ai;
2) eda,b (i, j − 1)+1 вставить bj;
3) eda,b (i − 1, j − 1) заменить bj.

	0	k	i	t	t	e	n
0	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

	0	k	i	t	t	e	n
0	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

Рис. 4. Матрица редактирования дерева

Смотря на рисунок 4 слева, можно обнаружить, что значения элементов в правом верхнем углу и левом нижнем углу таблицы большие, потому что элементы в правом верхнем углу производят больше операций удаления, в то время как элементы в левом нижнем углу производят больше операций вставки [4].

Анализ показывает, что максимальное значение минимального расстояния редактирования между двумя строками равно длине более длинной строки в двух строках. Независимо от того, как изменяется содержимое строки, минимальное расстояние редактирования не будет превышать длину более длинной строки. Поэтому пропуск редактирования состояний редак-

тирования в таблице расстояний, что приведет к тому, что конечный шаг редактирования превысит максимальную длину.

Минимальное расстояние = |2 ∗ (j − i) + len(цель) − len(источник)| (2)

Значение этой формулы - минимальное расстояние редактирования шага редактирования, содержащего операцию позиционирования. Где I и j - индексы строк и столбцов таблицы соответственно. Способ определения того, следует ли пропускать, таков: когда список результатов вычисления больше длинной строки, нет необходимости вычислять расстояние редактирования в этой позиции.

Все ячейки слева на рисунке 4, для которых не требуется вычислять расстояние, отмечены областью наклонного подчеркивания: справа на рисунке 4 видно, что ячейки таблицы, для которых не требуется вычислять расстояние, распределены в небольшой верхней и нижней треугольной матрице. Когда разница в длине между двумя последовательностями очень мала, две полуматрицы также будут увеличиваться. Когда две последовательности равны по длине, длина стороны под прямым углом полуматрицы достигнет половины последовательности, а элементы, которые не нужно вычислять, будут составлять 1/4 всех элементов.

Заключение

В анализаторе SQL частота ложных срабатываний и частота ложных срабатываний являются двумя важными показателями для оценки качества анализатора [5]. Частота ложных срабатываний относится к вероятности того, что система выдаст правильный запрос SQL за неправильный. Частота ложных срабатываний относится к операторам SQL, которые не обнаруживаются, но оцениваются как обычный ввод. Чем меньше частота ложноположительных и ложноотрицательных срабатываний, тем лучше способность синтаксического анализа SQL-анализатора. Расчет частоты ложных срабатываний и частоты ложных срабатываний анализатора SQL предназначен для сбора данных, сгенерированных при работе анализатора SQL в режиме реального времени, поэтому результат расчета является более точным.

Таблица 1. Таблица функциональных испытаний

Метод фильтрующего анализа	Частота ложноположительных результатов	Показатель занижения отчетности
Фильтрация на основе ключевых слов	26 %	19 %
Фильтрация на основе регулярных выражений	15 %	11 %
Фильтрация на основе синтаксического дерева	6 %	7 %

Коэффициент ложного отрицания вычисляет коэффициент ложного срабатывания анализатора SQL в соответствии со статистикой результатов теста. Результаты в приведенной выше таблице показывают, что метод синтаксического анализа SQL, предложенный в этой статье, имеет более низкую частоту ложных отрицательных и ложных положительных результатов,

чем другие методы фильтрации.

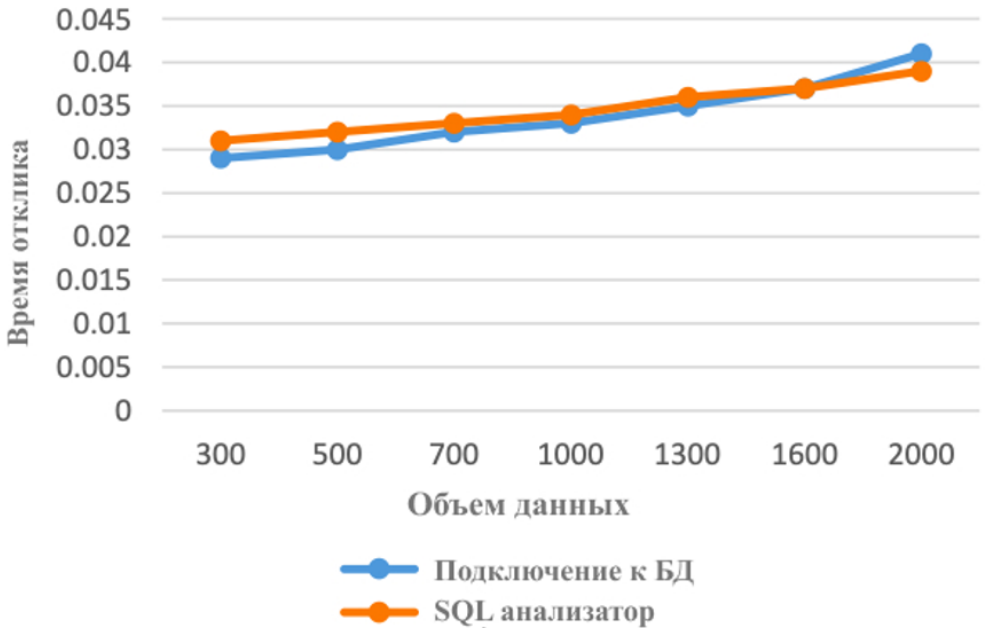

Рис. 5. Схема тестирования производительности

В базе данных SQL время отклика оператора SQL является еще одним важным показателем для оценки качества анализатора SQL. Как видно из рисунка 5, когда объем данных в запросе SQL невелик, время отклика при прямом подключении к базе данных быстрее, чем при использовании анализатора SQL. Причина в том, что в процессе синтаксического анализа SQL требуется синтаксический анализ протокола, преобразование кода, синтаксический анализ и построение синтаксического дерева. Несмотря на некоторую задержку в ответе системы, с увеличением объема данных время отклика анализатора SQL быстрее, чем у прямой базы данных. Это связано с тем, что оптимизация инструкций SQL используется для ускорения времени отклика и повышения эффективности выполнения запросов SQL.

Проанализированы словарный запас и синтаксические правила стандартного языка SQL, разработан и реализован анализатор SQL на основе ANTLR; Анализатор SQL может отслеживать активность базы данных и предотвращать выполнение опасных операторов SQL, чтобы защитить базу данных.

Библиографический список

1. Мюллер, Р.Д. Проектирование баз данных и UML. – М.: Лори, 2013. – 420 с.

2. Мартишин С.А. Проектирование и реализация баз данных в СУБД MySQL с использованием MySQL Workbench: Методы и средства проектирования информационных систем и технологий / С.А. Мартишин, В.Л. Симонов, М.В. Храпченко. – М.: Форум, 2018. – 61 с.

3. Лукин, В.Н. Введение в проектирование баз данных. – М.: Вузовская книга, 2015. – 144 с.

4. Эмблер, С. Рефакторинг баз данных: эволюционное проектирование / С. Эмблер, П. Садаладж. – М.: Вильямс, 2007. – 672 с.

5. Коннолли Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. – М.: Вильямс И.Д., 2017. – 1440 с.

DEVELOPMENT AND IMPLEMENTATION OF A COMMON SQL ANALYZER

M.V. Bolyabkin[1], *Graduate Student*
A.I. Panov[2], *Student*
N.A. Bakhmurov[2], *Student*
[1]Moscow Technical University of Communications and Informatics
[2]Nizhny Novgorod State University named after N.I. Lobachevsky
[1](Russia, Moscow)
[2](Russia, Nizhny Novgorod)

Abstract. *With the rapid development of computer networks, the information security of databases is becoming more and more important. In order to protect databases, it is necessary to analyze and restore SQL queries. However, in the database audit products on the market, there is no accurate analysis of SQL queries. Therefore, in this article, SQL analysis is based on four aspects: lexical analysis, syntactic analysis, optimization of the SQL spanning tree and error detection.*

Keywords: *SQL, databases, analyzer, parsing, security.*