РАЗРАБОТКА НОВЫХ ВИДОВ ИНДЕКСОВ ДЛЯ СУБД POSTGRESQL С ИСПОЛЬЗОВАНИЕМ ПРОГРАММНОГО ИНТЕРФЕЙСА СЕРВЕРА *

***Андрианов Игорь Александрович**, к.т.н., доцент*
***Корякин Дмитрий Сергеевич**, аспирант*
Вологодский государственный университет

В статье описывается способ разработки новых видов индексов для СУБД PostgreSQL с использованием программного интерфейса сервера для управления данными в индексе. Выполнено сравнение предложенного подхода с альтернативными. В качестве примера описана реализации нового метода доступа для повышения скорости исполнения запросов, содержащих оператор LIKE.

Ключевые слова: СУБД, индексирование, методы доступа, PostgreSQL.

Индексирование данных заключается в выполнении их предварительной обработки с целью более быстрого выполнения в дальнейшем многократных поисковых запросов. Структура данных, сгенерированная в процессе такой предобработки, называется индексом. Алгоритмические и программные средства СУБД для поддержки индексов определённого вида образуют метод доступа к данным. Реализация метода доступа, как правило, содержит процедуры создания индекса, добавления и удаления из него данных, выполнения определённых видов поиска и др.

Во всех СУБД поддерживается некоторый базовый набор методов доступа к данным. Так, широко используются индексы на основе B-деревьев и хеш-таблиц [1]. СУБД, работающие с геометрическими данными, часто поддерживают индексы на базе R-деревьев или похожих структур [2]. СУБД, поддерживающие полнотекстовый словарный поиск, часто используют индексы на основе инвертированных файлов. Несмотря на такое разнообразие, нередко возникают ситуации, когда перечисленных способов не хватает. Приведём несколько примеров.

1). Мы создали новый тип данных, поддерживающий определённый набор операций. Существующие методы доступа при этом могут быть не применимы – частично или полностью. Для примера, пусть мы создаём новый тип данных для представления множеств. Нам хочется быстро выполнять запросы наподобие: “получить все множества, пересечения которых с данным содержит не менее k элементов”.

2). Похожий пример: вводится новая операция, но для уже имеющегося типа данных. Пример – вариант задачи об “интересных” подстроках: требуется найти, содержатся ли в данном тексте подстроки длиной не менее k, число которых намного больше, чем могло получиться случайно.

3). Также нас может не устраивать время выполнения некоторых операций и над стандартными типами данных. Пример – оператор LIKE из стандарта языка SQL. При использовании данного оператора в условии

*Работа частично поддержана грантом РФФИ 18-47-350001 р_а

WHERE оператора SELECT СУБД в большинстве случаев выполняет полное сканирование данных, что может занимать значительное время.

В данной статье мы предлагаем подход, позволяющий разработчику, использующему СУБД PostgreSQL, но не являющемуся специалистом по особенностям её внутренней работы, создавать новые методы доступа с минимальными усилиями. Выбор именно PostgreSQL объясняется тем, что она изначально спроектирована как система с расширяемой архитектурой, в которой предусмотрены возможности по созданию новых методов доступа к данным.

Основными способами создания новых методов доступа к данным в PostgreSQL являются следующие.

1). Использование обобщённого дерева поиска информации (Generalized Search Tree – GiST) [3]. В GiST для хранения элементов индекса используется разновидность сбалансированного дерева. В его листьях лежат ссылки на строки индексируемой таблицы, во внутренних вершинах – некоторые данные (причём мы сами решаем, какие именно). Эти данные позволяют эффективно определять при выполнении поиска, могут ли в поддереве текущего узла найтись листы, соответствующие запросу, либо это поддерево можно пропустить. Чем большая часть дерева будет пропущена, тем меньше будет время выполнения поиска.

У данного индекса можно выделать два недостатка. Во-первых, это ”жёсткость” его внутренней структуры – ведь далеко не всегда представление данных в виде сбалансированного дерева будет самым эффективным вариантом. Во-вторых, в GiST предполагается, что индексируемые записи являются неделимыми, один листовой узел индекса ссылается только на одну запись в исходной таблице. Поэтому, например, если мы хотим построить индекс из всех слов, содержащихся в индексируемых текстовых данных, то с помощью GiST реализовать это будет сложно.

2). Использование обобщённого инвертированного индекса (Generalized Inverted Index – GIN) [4]. Данный индекс позволяет осуществлять индексирование сложных объектов с произвольным разбиением на ключи, то есть второй недостаток GiST в данном методе отсутствует.

Однако, на структуру хранения данных мы, опять-таки, повлиять не можем. Например, элементы, однажды попавшие в индекс, никогда из него не удаляются. Это сделано для упрощения алгоритмов, обеспечивающих параллельную работу с индексом нескольких процессов [4]. Предполагается, что набор элементов, из которых состоят значения, довольно статичен, и для большинства задач (в том числе для полнотекстового поиска) это вполне оправданно. Однако, вполне могут встретиться практические случаи, в которых это свойство не выполняется – тогда индекс, по видимости, придётся время от времени удалять и создавать заново.

3). Ещё один способ – это разработка метода доступа “с нуля” путём реализации заданного набора функций. При этом подходе нет таких ограничений, как в предыдущих вариантах: мы можем реализовать совершенно

произвольные структуры данных и алгоритмы их обработки. Тем не менее, данный способ обладает серьёзным недостатком: разработчик должен хорошо представлять себе особенности внутреннего устройства и работы СУБД. В частности, потребуется вручную управлять блокировками, чтобы гарантировать правильную работу в условиях параллельной обработки транзакций (не допускать нарушения целостности, появления тупиков и др.). Для сравнения, при использовании GiST или GIN-индексов вникать в такие детали работы СУБД не требуется.

В качестве модификации третьего варианта мы предлагаем следующий достаточно простой подход. Его суть состоит в том, чтобы воспользоваться уже имеющимся набором возможностей, которые предоставляет СУБД. Индекс создаётся согласно третьему способу – то есть мы реализуем необходимый набор функций и регистрируем новый тип индекса индекс в системных таблицах PostgreSQL. Однако, при разработке функций мы не будем работать с операциями низкого уровня (для управления блокировками, дисковыми страницами и др.). Вместо этого мы воспользуемся стандартными средствами СУБД (таблицы и стандартные индексы), к которым можно получить доступ с помощью языка SQL через программный интерфейс сервера (Server Programming Interface – SPI). Важно отметить, что корректность параллельной работы при этом будет обеспечивать сам сервер.

Заметим, что с точки зрения пользователя полученный индекс ничем не будет отличаться от стандартных индексов СУБД. В частности, для увеличения скорости выполнения запросов нет необходимости перекомпилировать существующие пользовательские программы: если в программе встречается запрос на выборку, то при его исполнении СУБД будет автоматически подключать наш индекс.

Для примера рассмотрим создание нового индексного метода доступа для ускорения поиска в текстах по LIKE-шаблонам. LIKE-шаблон – это некоторая строка, в которой, кроме обычных символов, могут содержаться два со специальным значением: ‘%’ – любая подстрока, ‘_’ – любой один символ. Например, под шаблон ‘%@.mail.ru%’ подходят все тексты, в которых встречается подстрока ‘@.mail.ru’. В PostgreSQL (и большинстве других СУБД) поиск по LIKE выполняется простой проверкой соответствия шаблону всех записей (исключением является случай, когда шаблон начинается с префикса, не содержащего подстановочных символов, а над полем построен индекс на базе B-дерева).

Для ускорения поиска воспользуемся методом триграмм. Если взять стоку и перемещать по нему окно длины 3, то все полученные подстроки будут являться триграммами для этого текста. Например, для строки “текст” получатся следующие триграммы: “тек”, “екс”, “кст” (часто также в начало и конец строки дополнительно добавляют по два пробела). Выделим из всех индексируемых записей триграммы, после чего с каждой триграммой свяжем множество записей, в которых она встречалась. Как мож-

но заметить, в результате получится структура, очень напоминающая инвертированный файл – то есть для рассмотренной задачи GIN также вполне бы подошёл.

Для уменьшения размера индекса и ускорения поиска множество записей для каждой триграммы будем хранить в виде битового масива (*i*-й бит равен единице, если в *i*-м документе встречалась данная триграмма) – такой подход известен как метод битовых срезок. Саму триграмму для удобства будем представлять как целое число, в котором первые 3 байта соответствуют её трём символам. Данную индексную информацию будем хранить в таблице вида:

create table ptrgms_srctable_srcfld (trgm integer, documents integer[]);

Для ускорения выборки из этой таблицы создадим над ней индекс (в результате получается использование одного индекса внутри другого):

create index trgms_srctable_srcfld_idx on trgms_srctable_srcfld (trgm);

Поисковый запрос будет выполняться следующим образом. Сначала выделим из LIKE-запроса допустимые триграммы (без подстановочных символов). Далее выполним выборку из таблицы trgms множеств записей, где они встречались, и найдём пересечение этих множеств. После этого осталось проверить записи, которые остались в этом пересечении.

Существенно снижает трудоёмкость написания нашего метода доступа тот факт, что имеется возможность проинформировать СУБД специальной опцией о том, что мы не гарантируем, что наш индекс будет возвращать только подходящие записи (но, разумеется, гарантируем, что отброшенные записи точно не подходят). В этом случае СУБД выполнит дополнительную проверку – применит оператор LIKE к каждой записи, которую возвратит наш индекс.

Интересным моментом при реализации стал вопрос о том, как выполнить первоначальное построение индекса над большим набором записей за приемлемое время. В настоящий момент он решился следующим образом. Входные данные обрабатываются за несколько проходов, при этом на каждом проходе обрабатывается не более чем *TMax* триграмм (где параметр *TMax* определяется количеством доступной оперативной памяти). На каждом проходе мы открываем курсор, проходим по всем исходным записям, выделяем из них триграммы и помещаем их в словарь (используя шаблонный класс std::map библиотеки STL). По завершении прохода готовая порция триграмм сбрасывается во временный файл, а оперативная память, занятая под битовые массивы, освобождается для следующего прохода. По окончании мы используем специальные средства СУБД PostgreSQL (SQL-оператор COPY) для перемещения большого объёма данных из внешнего файла в таблицу базы данных.

Оценим число операций, выполняемых в нашем методе доступа. Функции вставки записей в индекс и удаления можно без труда реализовать за время $O(n \cdot \log k)$, где n – длина вставляемой записи, k – текущее число триграмм в БД. Впрочем, поскольку максимально возможное число триграмм

в худшем случае составляет около 17 миллионов (а для реальных текстов обычно не превышает 200 тысяч), то $\log_2 k$ можно считать константой. Процедура поиска потребует $O(p \cdot \log k)$ операций (где p – длина поискового шаблона), чтобы выдать набор записей, возможно, соответствующих запросу. После этого каждую из этих записей СУБД должна проверить на соответствие шаблону. Конечно, несложно привести пример, когда данный индекс не даст никакого ускорения поиска. Однако, на практике такие случаи встречаются редко.

Тестирование показало следующее. Эксперимент проводился над достаточно небольшим набором из примерно 6500 HTML-документов общим объёмом около 180 мегабайт. Размер индекса составил 120 мегабайт, т.е. 67% от размера данных. Скорость выполнения поискового запроса при этом в среднем возросла более чем в 10 раз.

Таким образом, можно сделать вывод, что предложенный подход к разработке методов доступа к данным может быть достаточно полезным в зависимости от решаемых задач. При его использовании мы получаем несколько большую гибкость по сравнению с GIST и GIN индексами, при этом также отсутствует необходимости изучать особенности внутренней работы СУБД и заниматься программированием на низком уровне.

Список литературы

1. Андрианов, И.А. Базы данных. Программирование и администрирование / И. А. Андрианов, С. Ю. Ржеуцкая. – Вологда: ВоГУ, 2018. – 71 с.

2. Андрианов, И.А. Индексирование и поиск в последовательностях для больших баз данных: монография / И. А. Андрианов, А. Ф. Чернов. – Вологда: ВоГТУ, 2013. – 167 с.

3. Индексы в PostgreSQL – 5. GiST. [Электронный ресурс]. URL: https://habr.com/ru/company/postgrespro/blog/333878 (дата обращения: 29.03.2020)

4. Индексы в PostgreSQL – 7. GIN. [Электронный ресурс]. URL: https://habr.com/ru/company/postgrespro/blog/340978 (дата обращения: 29.03.2020)