Национальный исследовательский ядерный университет «МИФИ» (Московский Инженерно–Физический Институт)
Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №2-4 : «Индексы»

Цель работы

Изучить возможности PostgreSQL по ускорению доступа к данным за счёт использования различных типов индексов.

Подготовка к работе

Индексы — дополнительные структуры данных, предназначенные для ускорения доступа к данным. В отсутствие индексов СУБД вынуждена искать требуемые значения полным перебором. Это, фактически, требует прочитать всю таблицу и проверить каждое значение. При этом, если данные из таблицы не находятся полностью в кэше, это потребует чтения с диска. Чтение с диска не только гораздо медленнее, но и связано с бóльшими задержками. Ситуация многократно усугубляется, если такие столбцы участвуют в операции соединения (JOIN).

Поэтому, по мере роста БД, индексы превращаются из «нечта желательного» в «нечто совершенно необходимое». Поэтому, у архитектора БД есть возможность создавать (условно) неограниченное число индексов для любой таблицы. Также, индекс автоматически создаётся для полей, которые объявлены как основной ключ (PRIMARY KEY) при создании сущности: это единственное существенное отличие PRIMARY KEY от обычного ограничения целостности.

Разумеется, у мендали есть и обратная сторона. Главный недостаток индексов состоит в том, что их содержимое должно регулярно обновляться по мере того, как меняются данные в самой таблице. Если в таблицу без индексов достаточно просто «дописать один ряд в конце», то при добавлении данных в таблицу с индексами требуется ещё и перестроить все индексы, которые связаны с изменившимися данными (читать: все индексы). Причём, изменения в индексах могут быть гораздо более существенными, чем в самой таблице.

Ситуация усугубляется по мере того, как к таблице добавляются дополнительные индексы: чем их больше, тем большую работу нужно проводить СУБД, чтобы поддерживать их в актуальном состоянии. Иногда, при обновлении БД через атомарные инструкции INSERT приходится даже временно «отключать» индексы и затем включать обратно.

Ещё одно — меньшее — зло при использовании индексов: создать индекс, который не будет в действительности применяться при выполнении запросов. Понятно, что таблицу из 4 столбцов и 2 строк быстрее прочитать целиком, чем обработать с использованием индекса. Аналогично, если выбран неверный тип индекса, либо неверное сочетание столбцов, либо запрос, для которого был создан индекс, просто больше не актуален, — индекс превращается в пустую трату вычислительного времени. При разработке системы индексирования для конкрентной БД, архитектору следует контролировать реальное использование созданных индексов при помощи EXPLAIN PLAN.

Разумеется, индексы также занимают и часть памяти; это касается и оперативной памяти, и постоянных запоминающих устройств.

В зависимости от конкретной модели СУБД, архитектору может быть доступен один или несколько различных типов индексов.

В качестве «индекса по умолчанию» в СУБД (и во многих других областях информационных технологий) лежат индексы на основе B-деревьев (обычно, B+деревьев). Эти структуры данных славятся простотой, универсальностью и отсутствием специфических эксплуатационных требований. Это объясняет их широкое применение в различных видах баз данных (включая файловые системы). Индекс на основе B+дерева представляет собой дерево из нескольких ярусов. Двигаясь от корня, СУБД в каждом узле дерева отвечает на вопрос: больше или меньше? В зависимости от отвера, выбирает один или

второй путь вниз по дереву. В узлах последнего яруса хранятся указатели на место хранения исходных данных.

Чем хорош этот тип индексов? Во-первых, сложность поиска очевидно сокращается с линейной до логарифмической. Во-вторых, они позволяют искать не только отдельные значения, но и диапазоны значений: логика «больше-меньше» тут работает так же эффективно. Недостаток: нахождение нужных данных всё ещё требует нескольких операций чтения (одного чтения индекса может не хватить). Также, операция обновления такого индекса может быть трудоёмкой: важно, чтобы на каждом узле ветви по обе стороны были примерно одинакового размера. Другими словами, половина значений должна быть больше, чем указано в узле, половина меньше. Если, например, добавлять в таблицу возрастающую последовательность чисел, все они будут попадать на одну сторону дерева. Чем больше дисбаланс, тем больше времени занимает поиск нужного узла. Устранение дисбаланса же требует частичной перестройки дерева: сложно и долго.

Также распространены индексы на основе hash-таблиц. Такие индексы работают иначе. Каждый раз, когда в таблицу добавляются новые данные, вычисляется их (в случае с postgreSQL, 32-битная hash-сумма). Эта сумма определяет, где будет храниться указатель на данные (а в некоторых реализациях и сами данные). Если в дальнейшем необходимо найти какое-либо значение, достаточно только посчитать его hash-сумму: её значение укажет на место, где хранится указатель на эти данные (и другие данные, с которыми может быть hash-коллизия; это придётся проверить непосредственно).

Основным достоинством hash-индекса является очень быстрый поиск значения. Вычислив hash-сумму искомого значения, можно сразу узнать, где его искать. Поэтому, в отличие от B+деревьев, hash-таблицы не требуют нескольких операций чтения. Операция обновления такого индекса также очень простая: при появлении новых данных нужно просто добавить значение на нужную позицию, при удалении — убрать. Ещё один небольшой плюс: размер hash-индекса не страдает от индексирования даже очень больших величин.

На фоне ярко выраженных достоинств hash-индексы имеют так же ярко выраженные недостатки. Самое главное: такие индексы занимают много места. При добавлении любого значения в таблице, запись в индексе должна быть размещена в строго определённом месте. Как следствие, приходится резервировать под такой индекс всю необходимую ему память, вне зависимости от степени заполнения таблицы. При этом, использование hash-функций с маленьким числом образов повышает число коллизий, а коллизии необходимо разрешать линейным поиском.

Второй недостаток: hash-индексы поддерживают только поиск по строгому равенству. Если необходимо искать диапазон значений, hash-индексы не помогут, так как непрерывному диапазону значений, конечно же, не будет соответствовать непрерывный диапазон hash-значений. Поэтому, в запросах, не использующих прямое сравнение, hash-индексы бесполезны.

Индексы на основе битовых карт (BITMAP INDEX) представляют собой таблицы, где по строкам располагаются возможные значения, а по столбцам номера строк индексируемой таблицы. На пересечении строки и столбца ставится 1 тогда и только тогда, когда значение в этом столбце соответствует указанному значению в строке.

Такой индекс может ускорить линейный поиск, если число возможных значений столбца невелико. Но, в силу того, что такой индекс занимает много места, а случаи, когда пространство значений столбца ограничено, не так уж и много, такие индексы используются редко. Обновление индекса обычно происходит довольно быстро (достаточно изменить один или два бита), но если в столбец добавилось новое, не встречавшееся ранее значение, индекс на основе битовых карт существенно вырастет. postgreSQL не поддерживает создание индексов на основе битовых карт, но может строить их сама для себя на основе других индексов, если сочтёт это эффективным для выполнения конкретного запроса.

postgreSQL дополнительно предоставляет ещё несколько типов индексов. Индексы BRIN (Block Range Index) работают схожим с B+деревьями образом, но хранят только максимальное и минимальное значение индексируемого диапазона. Такой подход может быть существенно более эффективным, чем B+-деревья, в операциях сравнения с диапазоном, но при условии, что данные в таблице изме-

няются монотонно. Напротив, если значения в столбце случайны и равномерно распределены, такой тип индекса почти бесполезен: диапазон значений в каждом блоке индекса оказывается примерно одинаковым.

Индексы GiST, SP-GiST и GIN, — усложнённые реализации индексов на основе B+деревьев, предоставляемые postgreSQL. Они предназначены для работы с составными типами данных, в частности, массивами и данными геолокации.

Помимо понятия «индекс» существует понятие «таблица, организованная по индексу». Индекс — независимый, самостоятельный объект, который может храниться, удаляться, обновляться и удаляться независимо от данных самой таблицы. Если речь идёт о таблице, организованной по индексу, индекс как отдельный объект не существует, но сами ряды в таблице переставляются таким образом, чтобы оптимизировать поиск. Это может катастрофически замедлить операцию добавления или изменения данных, потому что для того, чтобы поместить ряды на положенное им место, придётся переместить большой объём данных. Положительная сторона: индекс не занимает дополнительного места на диске, а вместо указателя на нужные данные — сами данные.

postgreSQL, по всей видимости, не поддерижвает таблицы, организованные по индексу.

hash-кластер — объект, получаемый путём объединения нескольких таблиц в одну сущность, ряды которой хранятся в hash-таблице. При этом hash-значение выполняется по нескольким полям, принадлежащим нескольким таблицам. Такой тип индекса очень упрощает соединение таблиц в кластере по индексируемым значениям. Но последовательное сканирование всех этих таблиц замедляется: так как они хранятся вместе, сканировать их приходится тоже вместе.

postgreSQL не поддерживает hash-кластеры. Более того, слово «кластер» имеет другое значение в postgreSQL и относится к группе (часто) независимых пользовательских баз данных, работающих под управлением одного экземпляра СУБД.

Самым важным инструментом архитектора при проработке стратегии индексирования базы данных явялется инструкция (в postgreSQL) EXPLAIN. С её помощью можно узнать план выполнения запроса, который СУБД подготовила на основе написанного SQL-запроса. По плану видно, собирается ли СУБД использовать индекс, и какую выгоду намерена за счёт этого получить. Если СУБД решила, что использование индекса неэффективно, упоминаний этого индекса не будет в плане.

Важно понимать, что для принятия решения о применении той или иной стратегии выполнения запроса важно обладать знаниями о данных, хранящихся в таблице. Если поле в 10.000 записях может принимать только 2 различных значения, может быть эффективно использование битовой карты: потребуется проверить всего 2.5 килобайта информации, чтобы найти все вхождения искомой величины. При том, что данные на диске обычно хранятся в блоках по 4КБ, это всего одна операция чтения. Если же 10.000 записей соответствуют 10.000 различных значений поля, об индексах на основе битовых карт — ради своего же блага — лучше забыть.

Вывод: для выбора оптимальной стратегии необходимо знать не только структуру таблицы, но и свойства хранящихся в ней данных. СУБД (в частности, postgreSQL) хранят такие данные и следят за их обновлением. Но это происходит не мгновенно и не всегда (статистический анализ данных вслепую не относится к основным функциям СУБД и не должен потреблять существенных ресурсов вычислительной системы). Если в таблицу недавно был добавлен большой объём данных, статистические данные могут устареть и потерять актуальность. В этом случае можно обновить их явно инструкцией ANALYZE.

Вопросы для самоконтроля и самостоятельного изучения

1. Какой цели служат индексы в системах управления базами данных?
2. Являются ли индексы отдельными объектами базы данных?
3. Сколько индексов можно построить по одной таблице?
4. Сколько индексов можно построить по одному столбцу таблицы?
5. Какие индексы создаются автоматически?
6. Можно ли построить индекс по нескольким столбцам? Что в этом случае будет представлять собой индекс?

7. Верно ли, что индексы можно строить только по основным ключам?
8. Верно ли, что индексы можно строить только по основным и внешним ключам?
9. Можно ли построить индекс по уже существующей таблице?
10. Можно ли построить индекс по пустой таблице?
11. Стандартизированы ли индексы стандартом ISO/IEC 9075:2016 (SQL)?
12. Что такое индекс на основе B+дерева?
13. Что такое индекс на основе hash-таблиц?
14. Что такое индекс на основе битовых карт?
15. Что такое block range index (BRIN)?
16. Насколько сложной является процедура обновления индекса на основе B+дерева? В чём состоит основная сложность?
17. Насколько сложной является процедура обновления индекса на основе hash-таблиц? В чём состоит основная сложность?
18. Насколько сложной является процедура обновления индекса на основе битовых карт? В чём состоит основная сложность?
19. Верно ли, что состояние индекса может не отражать текущее состояние таблицы?
20. Верно ли, что индекс занимает столько же места, сколько и сама таблица?
21. Верно ли, что индекс позволяет извлечь из БД нужные данные, не выполняя операций чтения с диска?
22. Каковы причины и последствия разбалансировки B+дерева?
23. К каким последствиям может привести использование повреждённого индекса?
24. Что такое hash-кластер?
25. Что такое кластер в терминологии postgreSQL?
26. Может ли поиск по неиндексированной таблице быть быстрее, чем по индексированной?
27. Может ли СУБД игнорировать индекс несмотря на то, что он создан и подходит для выполнения текущего запроса?
28. Почему индекс на основе hash-таблиц не подходит для поиска диапазонов значений?
29. В каком случае неэффективен индекс на основе B+деревьев?
30. В каком случае неэффективно использование любых индексов?
31. Какова функция директивы ANALYZE?
32. При каких обстоятельствах postgreSQL вызывает ANALYZE автоматически?
33. Верно ли, что ANALYZE проверяет все значения в таблице и определяет статистические показатели?
34. Верно ли, что postgreSQL корректирует статистические показатели таблицы при добавлении в неё новых данных?
35. Как функция ANALYZE связана с индексами?
36. Приведите пример последствий пренебрежения актуальностью статистических данных.

Ход работы

1. Разработайте систему индексов для существующей базы данных. Важно, чтобы существовала полноценная схема данных и были известны типичные запросы к ней, поэтому рекомендуется использовать схему данных из лабораторных работ 1-1 .. 1-3;
2. (дополнительное кармическое задание) Найдите в каталоге postgreSQL статистическую информацию о таблицах схемы данных. Попробуйте актуализировать её инструкцией ANALYZE. Изменится ли она?
3. Добавьте существенный объём данных к таблицам и выполните ANALYZE, чтобы убедиться в актуальности статистических данных. Можно использовать ключевое слово VERBOSE;
4. Используя инструкцию EXPLAIN для построения планов выполнения запросов, убедитесь, что предложенные Вами индексы действительно используются;

5. (дополнительное кармическое задание) Меняя статистические свойства данных в одной из таблиц, заставьте postgreSQL изменить сценарий выполнения одного из запросов;
6. (дополнительное кармическое задание) Определите объём пространства, занимаемого созданными индексами. Соотнесите его с размером самих таблиц.

Оформление отчёта

1. Титульный лист: название института, название лабораторной работы, имя, фамилия, номер группы, год,...
2. Схема данных;
3. Список типичных запросов к БД с краткими пояснениями;
4. Список предложенных индексов с обоснованием их полезности/эффективности;
5. Листинги планов выполнения запросов с использованием индексов;
6. (дополнительно) Листинги планов выполнения тех же запросов, но без индексов;
7. (дополнительно) Результаты выполнения кармических заданий;
8. Заключение: краткое описание проделанной работы.

Справочные материалы

1. Документация postgreSQL, раздел 11. Индексы;
2. Документация postgreSQL, раздел 64. Особенности реализации индексов;
3. Документация postgreSQL, команда ANALYZE;
4. Документация postgreSQL, раздел 14. Использование директивы EXPLAIN.