Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Классификация методов построения индексов в базах данных»

Студент: ИУ7-73Б _____ М. Д. Маслова
(группа) (подпись, дата) (И. О. Фамилия)

Преподаватель: _____ А. А. Оленев
(подпись, дата) (И. О. Фамилия)

2022 г.

РЕФЕРАТ

Расчетно-пояснительная записка 30 с., 9 рис., 0 табл., 16 источн., 0 прил. ИНДЕКСЫ, B-ДЕРЕВЬЯ, ХЕШ-ИНДЕКСЫ, БИТОВЫЕ ИНДЕКСЫ, ОБУЧЕННЫЕ ИНДЕКСЫ, БАЗЫ ДАННЫХ, СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Объектом исследования является построение индексов в базах данных.

Цель работы — классификация методов построения индексов в базах данных.

В разделе 1 рассмотрено понятие индекса в базах данных и его основные свойства, а также описаны типы индексов.

В разделе 2 проведен обзор методов построения индексов на основе B-деревьев, хеш-таблиц и битовых карт, а также соответсвующих обученных индексов.

В разделе 3 приведены критерии оценки качества описанных методов и проведено сравнение по этим критериям.

что-то про результат

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На протяжении последнего десятилетия происходит автоматизация все большего числа сфер человеческой деятельности [1]. Это приводит к тому, что с каждым годом производится все больше данных. Так, по исследованию компании IDC (International Data Corporation), занимающейся изучением мирового рынка информационных технологий и тенденций развития технологий, объем данных к 2025 году составит около 175 зеттабайт, в то время как на год исследования их объем составлял 33 зеттабайта [2].

Для хранения накопленных данных используются базы данных (БД), доступ к ним обеспечивается системами управления базами данных (СУБД), обрабатывающими запросы на поиск, вставку, удаление или обновление. При больших объемах информации необходимы методы для уменьшения времени обработки запросов, одним из которых является построение индексов [3].

Базовые методы построения индексов используют такие структуры, как деревья поиска, хеш-таблицы и битовые карты [4]. На основе данных методов проводятся исследования по разработке новых для уменьшения времени поиска и затрат на перестроение индекса при изменении данных, а также сокращения дополнительно используемой памяти. Одно из таких исследований [5] было проведено в 2018 году, авторы которого описраясь на идею, что обычные индексы не учитывают распределение данных, предложили новый вид индексов, основанный на машином обучении, и назвали их обученные индексы (learned indexes). За последние пять лет было проведено множество исследований [6–9] по совершенствованию обученных индексов в плане поддержки операций и улучшению производительности, поэтому в данной работе в сравнение к базовым приводятся методы построения обученных индексов.

Целью данной работы является ***классификация методов построения индексов в базах данных*** .

Для достижения поставленной цели требуется решить следующие задачи:

- провести анализ предметной области: дать основные определение, описать свойства индексов и их типы;
- описать методы построения индексов в базах данных;
- предложить и обосновать критерии оценки качества описанных методов и сравнить методы по предложенным критериям оценки.

1 Анализ предметной области

1.1 Основные определения

Индекс — это некоторая структура, обеспечивающая быстрый поиск записей в базе данных [10]. Индекс определяет соответствие значения аттрибута или набора аттрибутов — *ключа поиска* — конкретной записи с местоположением этой записи [11]. Это соответствие организуется с помощью индексных записей. Каждая из них соответствует записи в *индексируемой таблице* — таблице, по которой строится индекс — и содержит два поля: идентификатор записи или указатель на нее, а также значение индексированного поля в этой записи [12].

Индексы могут использоваться для поиска по конкретному значению или диапазону значений, а также для проверки существования элемента в таблице, однако обеспечение уменьшения времени доступа к записям в общем случае достигается за счет [11]:

- упорядочивания индексных записей по ключу поиска, что уменьшает количество записей, которые необходимо просмотреть;
- а также меньшего размера индекса по сравнению с индексируемой таблицей, сокращающего время чтения одного элемента.

В то же время индекс является структурой, которая строится в дополнение к существующим данным, то есть он занимает дополнительный объем памяти и должен соответствовать текущим данным. Последнее значит, что индекс необходимо изменять при вставке или удалении элементов, на что затрачивается время, поэтому индекс, ускоряя работу СУБД при доступе к данным, замедляет операции изменения таблицы, что необходимо учитывать[13].

Таким образом, индекс может описываться: [11]:

- *типом доступа* — поиск записей по аттрибуту с конкретным значением, или со значением из указанного диапазона;
- *временем доступа* — время поиска записи или записей;
- *временем вставки*, включающее время поиска правильного места вставки, а также время для обновления индекса;
- *временем удаления*, аналогично вставке, включающее время на поиск удаляемого элемента и время для обновления индекса;
- *дополнительной памятью*, занимаемая индексной стркутурой.

1.2 Типы индексов

Индексы могут быть:

- кластеризованные и некластеризованные;
- плотные и разреженные;
- одноуровневые и многоуровневые;
- а также иметь в своей основе различные структуры, что описывается в следующем разделе, так как исследуется в данной работе.

В *кластеризованных* индексах логический порядок ключей определяет физическое расположение записей, а так как строки в таблице могут быть упорядочены только в одном порядке, то кластеризованный индекс может быть только один на таблицу. Логический порядок *некластеризованных* индексов не влияет на физический, и индекс содержит указатели на записи таблицы [13].

Плотные индексы (рисунок 1.1) содержат ключ поиска и указатель на первую запись с заданным ключом поиска. При этом в кластеризованных индексах другие записи с заданным ключом будут лежать сразу после первой записи, так как записи в таких файлах отсортированны по тому же ключу. Плотные некластеризованные индексы должны содержать список указателей на каждую запись с заданным ключом поиска [11].

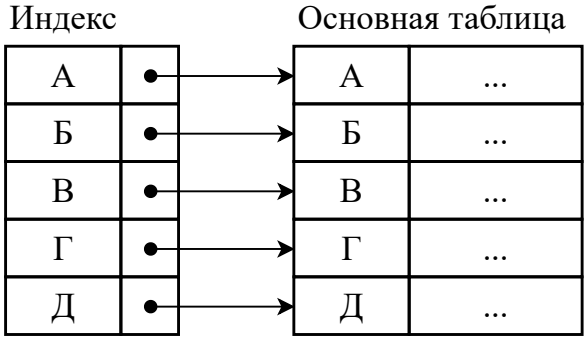

Рисунок 1.1 – Плотный индекс

В *разреженных* индексах (рисунок 1.2) записи содержат только некоторые значения ключа поиска, а для доступа к элементу отношения ищется запись индекса с наибольшим меньшим или равным значением ключа поиска, происходит переход по указателю на первую запись по найденному ключу и далее по указателям в файле происходит поиск заданной записи. Таким образом, разреженные индексы могут быть построены только на отсортированных последовательностях записей, иначе хранения только некоторых ключей поиска

будет недостаточно, так как будет неизвестно, после записи, с каким ключом будет лежать необходимый элемент отношения [11].

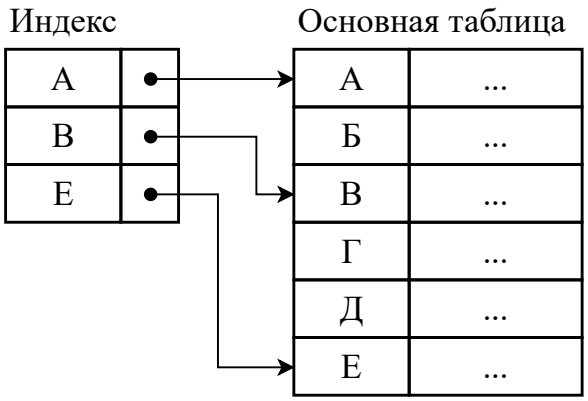

Рисунок 1.2 – Разреженный индекс

Поиск с помощью неразреженных индексов быстрее, так как указатель в записи индекса сразу приводит к необходимым записям. Однако разреженные индексы требуют меньше дополнительной памяти и сокращают время поддержания структуры индекса в актуальном состоянии при вставке или удалении [11].

Одноуровневые индексы ссылаются на данные таблице, индексы же верхнего уровня *многоуровневой* струкуры ссылают на индексы нижестоящего уровня [11] (рисунок 1.3).

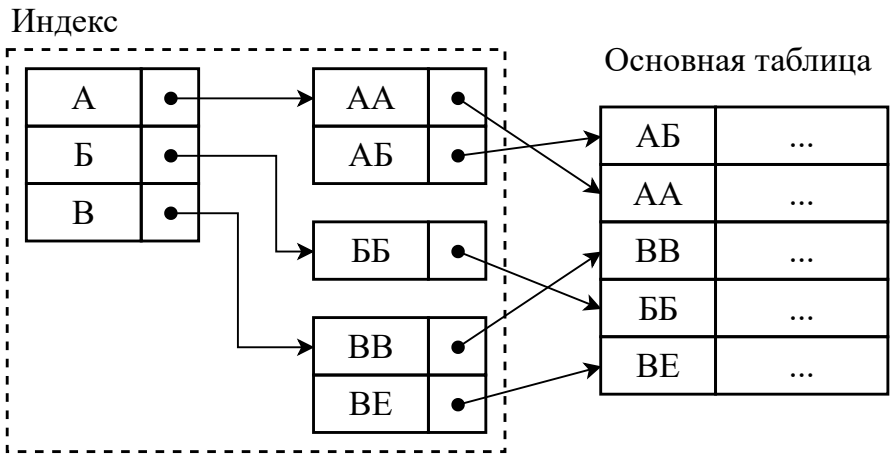

Рисунок 1.3 – Многоуровневый индекс

2 Описание существующих методов построения индексов

Как было сказано выше индексы обеспечивают быстрый поиск записей, поэтому в их основе лежат структуры, предназначенные для решения этой задачи. По данным структурам индексы подразделяются на

- индексы на основе деревьев поиска,
- индексы на основе хеш-таблиц,
- индексы на основе битовых карт.

2.1 Индексы на основе деревьев поиска

Дерево поиска — иерархическая структура, используемая для поска записей, в которой каждый переход на более низкий уровень иерархии уменьшает интервал поиска. При использовании деревьев поиска для построения индексов необходимо учитывать, что требуется обеспечить как ускорение поиска данных, так и уменьшение затрат на обновление индекса при вставках и удалениях. По этим причинам при решении задачи поиска в базах данных используют сбалансированные сильноветвящиеся деревья [14].

В данном случае сбалансированными деревьями назвают такие деревья, что длины любых двух путей от корня до листьев одинаковы [15]. Сильноветвящимися же являются деревья, каждый узел которых ссылается на большое число потомков [16]. Эти условия обеспечивают минимальную высоту дерева для быстрого поиска и свободное пространство в узла для внесения изменений в базу данных без необходимости изменения индекса при каждой операции.

Наиболее используемыми деревьями поиска, имеющими описанные свойства, являются B-деревья и их разновидность — B^+-деревья [14].

2.1.1 B-деревья

Что такое B-деревья?

Как они выглядят?

Как они строятся?

Как изменяются при вставке?

Как изменяются при удалении?

Для чего подходят?

Камбала B-дерево создает иерархию, которая помогает быстро переме-

щаться и находить искомые элементы.

B-деревья строятся на основе- использование сбалансированных деревьев поиска, поскольку они имеют более высокое разветвление (имеют больше дочерних узлов) и меньшую высоту.

B-деревья отсортированы: ключи внутри узлов B-дерева хранятся по порядку. Из-за этого, чтобы найти искомый ключ, мы можем использовать алгоритм, подобный бинарному поиску. Это также подразумевает , что поиск в B-деревьях имеет логарифмическую сложность. Например, для поиска искомого ключа среди 4 миллиардов элементов (4 x 10 9) требуется около 32 сравнений (см. "Поиск по B-дереву Сложность"на стр. 37 для получения дополнительной информации по этому вопросу). Если бы нам пришлось выполнять поиск по диску для каждого из этих сравнений, это значительно замедлило бы нас, но поскольку узлы B-дерева хранят десятки или даже сотни элементов, нам нужно выполнить поиск только по одному диску за прыжок на уровень.

Используя B-деревья, мы можем эффективно выполнять как точечные, так и диапазонные запросы. Точечные запросы, выражаемые предикатом равенства (=) в большинстве языков запросов, определяют местоположение одного элемента. С другой стороны, запросы диапазона, выраженные предикатами сравнения (<, >, s и 2), используются для запроса нескольких элементов данных по порядку.

B-деревья состоят из нескольких узлов. Каждый узел содержит до N ключей и N + 1 указателей на дочерние узлы. Эти узлы логически сгруппированы в три группы: Корневой узел У этого нет родителей, и это вершина дерева. Листовые узлы Это узлы нижнего уровня, у которых нет дочерних узлов. Внутренние узлы Это все остальные узлы, соединяющие корень с листьями. Обычно существует более одного уровня внутренних узлов.

B-деревья характеризуются их разветвлением: количеством ключей, хранящихся в каждом узле. Более высокая разветвленность помогает снизить затраты на структурные изменения, необходимые для поддержания сбалансированности дерева, и уменьшить количество запросов за счет хранения ключей и указателей на дочерние элементы узлы в одном блоке или нескольких последовательных блоках. Операции балансировки (а именно, разделения и слияния) запускаются, когда узлы заполнены или почти пусты.

Мы используем термин B-Tree в качестве общего обозначения семейства

структур данных, которые разделяют все или большинство упомянутых свойств. Более точное название для описанной структуры данных - B+-Tree. [KNUTH98] относится к деревьям с высоким разветвлением как к многополосным деревьям. B-деревья позволяют хранить значения на любом уровне: в корневых, внутренних и конечных узлах. B+-Деревья хранят значения только в конечных узлах. Внутренние узлы хранят только разделительные ключи, используемые для направления алгоритма поиска к соответствующему значению, хранящемуся на конечном уровне. Поскольку значения в B+-деревьях хранятся только на уровне листа, все операции (вставка, обновление, удаление и извлечение записей данных) влияют только на конечные узлы и распространяются на более высокие уровни только во время разделения и слияния. B+-деревья получили широкое распространение, и мы называем их B-деревьями, аналогично другой литературе по данной теме.

Разделительные ключи

Ключи , хранящиеся в узлах B-дерева, называются индексными записями, ключами-разделителями или ячейками-разделителями. Они разбивают дерево на поддеревья (также называемые ветвями или поддиапазонами), содержащие соответствующие диапазоны ключей. Ключи хранятся в отсортированном порядке, чтобы обеспечить двоичный поиск. Поддерево находится путем нахождения ключа и следования соответствующему указателю с более высокого уровня на более низкий. Первый указатель в узле указывает на поддерево, содержащее элементы, меньшие, чем первый ключ, а последний указатель в узле указывает на поддерево, содержащее элементы, превышающие или равен последнему ключу. Другие указатели являются ссылочными поддеревьями между двумя ключами: K1-1 < K, < K1, где K - набор ключей, а K - ключ, принадлежащий поддереву. На рисунке 2-10 показаны эти инварианты.

codeless

B-дерево - это тип дерева , который имеет функции для самобалансировки. Однако, в отличие от бинарного дерева поиска, B-дерево имеет родителей, которые могут иметь более двух дочерних узлов.

кабан

Тоже про B+

Наиболее широко используемая структура индексации совершенно иная: B-дерево.

Введенные в 1970 году [17] и названные "повсеместными"менее чем через 10 лет [18], B-деревья очень хорошо выдержали испытание временем. Они остаются стандартной реализацией индекса почти во всех реляционных базах данных, и многие нереляционные базы данных также используют их.

Как и SSTables, B-деревья сохраняют пары ключ-значение, отсортированные по ключу, что позволяет эффективно выполнять поиск по ключу- значению и запросы диапазона. Но на этом сходство заканчивается: у B-деревьев совсем иная философия дизайна.

Логически структурированные индексы, которые мы видели ранее, разбивают базу данных на переменные по размеру сегменты, обычно размером в несколько мегабайт или более, и всегда записывайте сегмент последовательно. В отличие от этого, B-деревья разбивают базу данных на блоки или страницы фиксированного размера, традиционно размером 4 КБ (иногда больше), и читают или записывают по одной странице за раз. Эта конструкция более точно соответствует базовому оборудованию, поскольку диски также расположены в блоках фиксированного размера.

Каждая страница может быть идентифицирована с помощью адреса или местоположения, что позволяет одной странице ссылаться на другую - аналогично указателю, но на диске, а не в памяти. Мы можем использовать эти ссылки на страницы позволяют построить дерево страниц, как показано на рисунке 3-6.

Одна страница обозначается как корень B-дерева; всякий раз, когда вы хотите найти ключ в индексе, вы начинаете здесь. Страница содержит несколько ключей и ссылок на дочерние страницы. Каждый дочерний элемент отвечает за непрерывный диапазон ключей, и ключи между ссылками указывают, где лежат границы между этими диапазонами. В примере на рис. 3-6 мы ищем ключ 251, поэтому мы знаем, что нам нужно следовать ссылке на страницу между границами 200 и 300. Это приводит нас к похожей странице, которая далее разбивает диапазон 200-300 на поддиапазоны.

В конце концов мы переходим к странице, содержащей отдельные ключи (листовая страница), которая либо содержит встроенное значение для каждого ключа, либо содержит ссылки на страницы, где можно найти значения.

Количество ссылок на дочерние страницы на одной странице B-дерева называется коэффициентом ветвления. Например, на рисунке 3-6 коэффициент

ветвления равен шести. На практике коэффициент ветвления зависит от объема пространства, необходимого для хранения ссылок на страницы , и границ диапазона, но обычно он составляет несколько сотен.

короче про поиск добавление и удаление

Если вы хотите обновить значение для существующего ключа в B-дереве, вы выполняете поиск конечной страницы, содержащей этот ключ, изменяете значение на этой странице и записываете страницу обратно на диск (все ссылки на эту страницу остаются действительными). Если вы хотите добавить новый ключ, вам нужно найти страницу, диапазон которой охватывает новый ключ, и добавить его на эту страницу. Если на странице недостаточно свободного места для размещения нового ключа, она разделяется на две страницы, заполненные наполовину, и родительская страница обновляется с учетом нового подраздела диапазонов ключей - см. рис. 3-7."

квадратики

Характеристики Btree: Что содержится в названии? Btree - это особый вид дерева, как показано на рисунке 8.12. Дерево - это структура, в которой каждый узел имеет не более одного родительского , за исключением корневого или верхнего узла. Структура Btree обладает рядом характеристик, обсуждаемых в следующем списке, которые делают ее полезной файловой структурой. Некоторые из характеристик являются возможными значениями буквы B’ в названии.

• Сбалансированный: все конечные узлы (узлы без дочерних элементов) находятся на одном уровне дерева. На рисунке 8.12 все конечные узлы расположены на два уровня ниже корневого. Сбалансированное дерево гарантирует, что все конечные узлы могут быть извлечены с одинаковой стоимостью доступа.

• Густой: количество ответвлений от узла велико, возможно, от 50 до 200 ветвей. Multivay, что означает более двух, является синонимом слова bushy. Ширина (количество стрелок от узла) и высота (количество узлов между корневым и конечным узлами) обратно пропорциональны: увеличьте ширину, уменьшите высоту. Идеальное дерево - широкое (кустистое), но короткое (несколько уровней).

• Блочно-ориентированный: каждый узел в Btree является блоком. Чтобы выполнить поиск в Btree, вы начинаете с корневого узла и следуете по пути

к конечному узлу, содержащему интересующие вас данные. Высота значение Btree важно, поскольку оно определяет количество обращений к физической записи для поиска.

• Динамический: форма Btree изменяется по мере вставки и удаления логических записей. Периодическая реорганизация никогда не является необходимой для Btree. Следующий подраздел описывает разделение и объединение узлов, изменения в Btree по мере вставки и удаления записей.

• Повсеместность: Btree - это широко реализованная и используемая файловая структура.

Прежде чем изучать динамическую природу, давайте более внимательно рассмотрим содержимое узла, как показано на рисунке 8.13. Каждый узел состоит из пар со значением ключа и указателем (адресом физической записи), отсортированных по значению ключа. Указатель идентифицирует физическую запись, содержащую логическую запись со значением ключа. Другие данные в логическая запись, помимо ключа, обычно не находится в узлах. Другие данные могут храниться в отдельных физических записях или в конечных узлах.

Важным свойством Btree является то, что каждый узел, кроме корневого, должен быть заполнен по крайней мере наполовину. Физический размер записи, размер ключа и размер указателя определяют пропускную способность узла. Например, если размер физической записи составляет 4096 байт, размер ключа - 8 байт, а размер указателя - 8 байт, максимальная емкость узла составляет 256 пар <ключ, указатель>. Таким образом, каждый узел должен содержать не менее 128 пар. Потому что разработчик обычно не имеет контроля над физическим размером записи и размером указателя, размер ключа определяет количество ветвей. Btrees обычно не подходят для больших размеров ключей из-за меньшего разветвления на узел и, следовательно, более высоких и менее эффективных Btrees.

Лошадь

Большинство баз данных структурируют свои данные в формате двоичного дерева, также известном как B- tree. B-дерево - это структура данных, которая самобалансируется при сохранении сортировки данных. B-дерево оптимизировано для чтения и записи блоков данных, именно поэтому B-деревья обычно встречаются в базах данных и файловых системах.

Вы можете представить таблицу или индекс B-дерева в виде перевернутого дерева. Существует корневая страница, которая является началом индекса, построенного на ключе. Ключ - это один или несколько столбцов. Большинство таблиц реляционной базы данных хранятся на первичном ключ, который может быть явно или неявно определен. Например, первичный ключ может быть целым числом. Если приложение ищет данные, которые соответствуют определенному идентификатору или диапазону идентификаторов, этот ключ будет использоваться для их поиска. В дополнение к B-дереву первичного ключа могут быть определены вторичные индексы для других столбцов или наборов столбцов. В отличие от исходного B-дерева, эти индексы хранят только те данные, которые проиндексированы, а не всю строку целиком. Это означает, что эти индексы намного меньше и могут гораздо легче помещаться в памяти.

B-дерево называется деревом, потому что, когда вы перемещаетесь по дереву, вы вы можете выбрать одну из двух или более дочерних страниц, чтобы получить доступ к нужным вам данным. Как только что обсуждалось, страница содержит строки данных и метаданных. Эти метаданные включают указатели на страницы под ними, также известные как дочерние страницы. Корневая страница имеет под собой две или более страниц, также известных как дочерние. Дочерняя страница или узел может быть внутренним узлом или конечным узлом. Внутренние узлы хранят сводные ключи и дочерние указатели и используются для направления операций чтения через индекс к тому или иному узлу. Конечные узлы содержат ключевые данные. Эта структура создает самость- балансирующее дерево, поиск по которому возможен только на нескольких уровнях, что позволяет выполнить всего несколько поисков по диску, чтобы найти указатели на нужные строки. Если необходимые данные находятся внутри самого ключа, вам даже не нужно следовать указателю на строку.

Двоичное дерево записывает

При вставке данных в B-дерево правильный конечный узел находится с помощью поиска. Узлы создаются с возможностью размещения дополнительных вставок, а не упаковываются в них . Если в узле есть место, данные вставляются в узел по порядку. Если узел заполнен, должно произойти разделение. При разделении определяется новая медиана и создается новый узел. Затем записи соответствующим образом перераспределяются. Данные об этой

медиане затем вставляются в родительский узел, что может вызвать дополнительные разбивается полностью до корневого узла. Обновления и удаления также начинаются с поиска правильного конечного узла с помощью поиска, за которым следует обновление или удаление. Обновления могут привести к разделению, если они увеличивают размер данных до такой степени, что они переполняют узел. Удаления также могут привести к перебалансировке.

Базы данных с нуля (новые) начинаются в основном с последовательной записи и чтения. Это проявляется в виде записи и чтения с низкой задержкой. По мере роста базы данных разделение приведет к тому, что ввод-вывод станет случайным. Это приводит к увеличению времени чтения и записи с задержкой. Вот почему мы должны настаивать на реалистичных наборах данных во время тестирования, чтобы убедитесь, что будут продемонстрированы долгосрочные эксплуатационные характеристики, а не эти наивные, ранние экспоненты.

Ниже приводится краткое описание атрибутов и преимуществ B-деревьев:
• Отличная производительность для запросов на основе диапазона.

•Не самая идеальная модель для поиска в одной строке.

• Ключи существуют в отсортированном порядке для эффективного поиска ключей и сканирования диапазона.

•Структура сводит к минимуму чтение страниц для больших наборов данных.

• Благодаря тому, что ключи не помещаются на каждую страницу, удаления и вставки выполняются эффективно, при этом требуется лишь случайное разделение и слияние.

• Производительность намного лучше, если вся структура может поместиться в памяти.

При индексации данных существуют и другие варианты. Наиболее распространенным из них является хэш-индекс.

Как мы упоминали ранее, B-дерево, как правило, довольно распространено в реляционных базах данных. Если вы работали в таких средах, то, вероятно, уже работали с ними. Однако существуют и другие варианты хранения данных, и они переходят от экспериментальных к зрелым. Давайте рассмотрим далее структуры журналов, доступные только для добавления.

2.1.2 B^+-деревья

Отличие от B-деревьев

Картинка

Камбала

Некоторые варианты B-дерева также имеют указатели родственных узлов, чаще всего на уровне листа, для упрощения сканирования диапазона. Эти указатели помогают избежать возвращения к родителю, чтобы найти следующего брата или сестру. Некоторые реализации имеют указатели в обоих направлениях, образуя двусвязный список на конечном уровне, что делает возможной обратную итерацию. Что отличает B-деревья от других, так это то, что вместо того, чтобы строиться сверху вниз (как бинарные деревья поиска), они строятся наоборот - снизу вверх. Количество конечных узлов растет, что увеличивает количество внутренних узлов и высоту дерева. Поскольку B-деревья резервируют дополнительное пространство внутри узлов для будущих вставок и обновлений, использование хранилища в дереве может достигать 50%, но обычно значительно выше. Более высокая заполняемость не влияет отрицательно на производительность B-Tree.

Поиск

Теперь, когда мы рассмотрели структуру и внутреннюю организацию B-деревьев, мы можем определить алгоритмы поиска, вставки и удаления. Чтобы найти элемент в B-дереве, мы должны выполнить один обход от корня к листу. Цель этого поиска - найти искомый ключ или его предшественника. Поиск точного совпадения используется для точечного запросы, обновления и удаления; поиск его предшественника полезен для сканирования диапазона и вставок.

Алгоритм начинается с корня и выполняет двоичный поиск, сравнивая искомый ключ с ключами, хранящимися в корневом узле, пока не будет найден первый разделительный ключ , который больше искомого значения. Это позволяет найти искомое поддерево. Как мы обсуждали ранее, индексные ключи разбивают дерево на поддеревья с границами между двумя соседними ключами. Как только мы находим поддерево, мы следуем указателю , который соответствует ему, и продолжаем тот же процесс поиска (находим разделительный ключ, следуем указателю), пока не достигнем целевого конечного узла, где мы либо находим искомый ключ , либо заключаем, что его нет, определяя

местоположение его предшественника.

На каждом уровне мы получаем более детальное представление о дереве: мы начинаем с самого грубозернистого уровня (корень дерева) и спускаемся на следующий уровень, где ключи представляют более точные, детализированные диапазоны, пока, наконец, не достигнем листьев, где расположены записи данных.

Во время точечного запроса поиск выполняется после нахождения или неудачи поиска искомого ключа. Во время сканирования диапазона итерация начинается с ближайшей найденной пары ключ-значение и продолжается, следуя указателям на родственные элементы, пока не будет достигнут конец диапазона или не будет исчерпан предикат диапазона.

Вставка

Чтобы вставить значение в B-дерево, мы сначала должны найти целевой лист и найти точку вставки. Для этого мы используем алгоритм, описанный в предыдущем разделе. После того, как лист найден, к нему добавляются ключ и значение. Обновления в B-деревьях работают путем определения местоположения целевого конечного узла с использованием алгоритма поиска и связывания нового значения с существующим ключом.

Если на целевом узле недостаточно свободного места, мы говорим, что узел переполнен [NICHOLS66] и должен быть разделен на две части, чтобы вместить новые данные. Более точно, узел разделяется, если выполняются следующие условия:

• Для конечных узлов: если узел может содержать до N пар ключ-значение, и вставка еще одной пары ключ-значение увеличивает его максимальную вместимость N.

• Для узлов без листа: если узел может содержать до N + 1 указателей, и вставка еще одного указателя увеличивает его максимальную вместимость N + 1.

Разделение выполняется путем выделения нового узла, переноса на него половины элементов из узла разделения и добавления его первого ключа и указателя на родительский узел. В этом случае мы говорим, что ключ продвигается. Индекс, по которому выполняется разделение, равен называется точкой разделения (также называемой средней точкой). Все элементы после точки разделения (включая точку разделения в случае разделения узла без

листа) переносятся во вновь созданный дочерний узел, а остальные элементы остаются в узле разделения.

Если родительский узел заполнен и в нем нет свободного места для продвинутого ключа и указателя на вновь созданный узел, его также необходимо разделить. Эта операция может распространяться рекурсивно вплоть до корня.

Как только дерево достигает своей пропускной способности (т.е. Разделение распространяется вплоть до корня), мы должны разделить корневой узел. Когда корневой узел разделен, выделяется новый корень, содержащий ключ точки разделения. Старый корень (теперь содержащий только половину записей) понижается на следующий уровень вместе со своим вновь созданным собратом, увеличивая дерево высота на единицу. Высота дерева изменяется, когда корневой узел разделяется и выделяется новый корень, или когда два узла объединяются, образуя новый корень.

На уровне листа и внутреннего узла дерево растет только горизонтально. На рисунке 2-11 показан полностью занятый конечный узел во время вставки нового элемента 11. Мы рисуем линию в середине полного узла, оставляем половину элементов в узле, а остальные элементы перемещаем в новый. Значение точки разделения помещается в родительский узел, чтобы служить разделительным ключом.

Поскольку разбиения на не листовые узлы всегда являются проявлением разбиений, распространяющихся с нижележащих уровней, у нас есть дополнительный указатель (на вновь созданный узел на следующем уровне). Если родительскому файлу не хватает места, его также необходимо разделить. Не имеет значения, разделен ли конечный или не-конечный узел (т.е. содержит ли узел ключи и значения или только ключи). В случае разделения листа ключи перемещаются вместе со связанными с ними значениями. Когда разделение завершено, у нас есть два узла, и мы должны выбрать правильный, чтобы закончить вставка. Для этого мы можем использовать инварианты ключа-разделителя. Если вставленный ключ меньше , чем продвигаемый, мы завершаем операцию вставкой в разделенный узел. В противном случае мы вставляем во вновь созданный файл.

Подводя итог, разделение узлов выполняется в четыре этапа:

1. Выделите новый узел.

2. Скопируйте половину элементов из узла разделения в новый.

3. Поместите новый элемент в соответствующий узел.

4. В родительском элементе разделяемого узла добавьте разделительный ключ и указатель на новый узел.

Удаление

Удаления выполняются путем предварительного определения местоположения целевого листа. Когда лист найден, ключ и значение, связанное с ним, удаляются.

Если соседние узлы имеют слишком мало значений (т.е. их заполняемость попадает под ограничение), родственные узлы объединяются. Такая ситуация называется недостаточным потоком. [БАЙЕР72] описывает два сценария недостаточного потока: если два соседних узла имеют общего родителя и их содержимое помещается в один узел, их содержимое должно быть объединено (конкатенировано).; если их содержимое не помещается в один узел, ключи перераспределяются между ними для восстановите баланс (см. раздел "Восстановление баланса"на стр. 70). Точнее, два узла объединяются , если выполняются следующие условия:

• Для конечных узлов: если узел может содержать до N пар ключ-значение, а общее количество пар ключ-значение в двух соседних узлах меньше или равно N.

• Для узлов, не являющихся листьями: если узел может содержать до N + 1 указателей, а общее количество указателей в двух соседних узлах меньше или равно N + 1.

На рисунке 2-13 показано слияние во время удаления элемента 16. Чтобы сделать это, мы перемещаем элементы от одного из братьев и сестер к другому. Как правило, элементы из правой родственные элементы перемещаются в левый, но это можно сделать и наоборот, если сохраняется порядок ключей.

квадратики

Последовательный поиск по диапазону B+tree может быть проблемой с Btrees. Чтобы выполнить поиск по диапазону, процедура поиска должна перемещаться вверх и вниз по дереву. Например, для извлечения ключей в диапазоне от 28 до 60 на рис. 8.15(a) процесс поиска начинается в корне, спускается к левому конечному узлу, возвращается к корню, а затем спускается к правому конечному узлу. Эта процедура имеет проблемы с сохранением физических записей в памяти. Операционные системы могут заменять физические записи,

если к ним не было недавних обращений. Потому что может пройти некоторое время, прежде чем к родительскому узлу снова будет получен доступ, операционная система может заменить его другой физической записью, если основная память заполнится. Таким образом, при повторном обращении к родительскому узлу может потребоваться другой доступ к физической записи .

Чтобы гарантировать, что физические записи не будут заменены, обычно реализуется вариант дерева B+ . На рисунке 8.16 показаны две части дерева B+. Треугольник (набор индексов) представляет собой обычный индекс Btree. Нижняя часть (набор последовательностей) содержит конечные узлы. Все ключи находятся в конечных узлах, даже если ключ присутствует в наборе индексов. Конечными узлами являются соединены вместе, так что последовательный поиск не требует перемещения вверх по дереву. После того, как исходный ключ найден, процесс поиска обращается только к узлам в наборе последовательностей.

only general

Дерево B + - это индексная структура, которая гарантирует, что все пути от корня до листа в данном дереве имеют одинаковую длину, то есть структура всегда сбалансирована по высоте.

2.1.3 Обученные индексы

B-tree индексы можно рассматривать как модель сопоставления ключа позиции искомой записи в отсортированном массиве (рисунок 2.1).

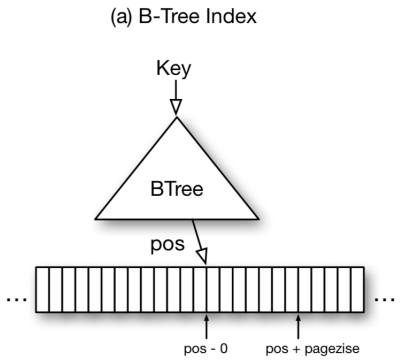

Рисунок 2.1 – B-деревья

Такие индексы как бы предсказывают положение записи с минимаксной ошибкой ($min\_err = 0$, $max\_err = page\_size$). Поэтому можем заминить

B-деревья на линейную модель также с минимаксной ошибкой (рисунок 2.2).

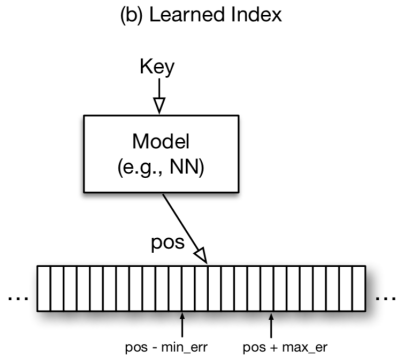

Рисунок 2.2 – Обученный индекс

Так для предсказания можно представлять Range Index Models как модели функции распределения (рисунок 2.3):

$$\text{position} = F(key) \cdot N, \tag{2.1}$$

где $F(key)$ — функция распределения, дающая оценку вероятности обнаружения ключа, меньшего или равного ключу поиска, то есть $P(X < key)$;

N — количество ключей.

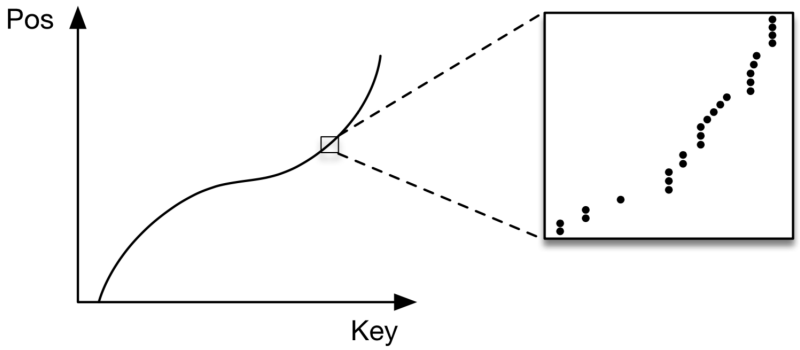

Рисунок 2.3 – Индекс как функция распределения

Можно построить индексы на основе рекурсивной модели (рисунок 2.4), в которой строится иерархия моделей из n уровней. Каждая модель на вход получает ключ, на основе которого выбирает модель на следующем уровне. Модели последнего этапа предсказывают положение записи.

Можно использовать различные модели: например, на верхнем использовать нейронные сети, а на нижних простые линейные регрессионные модели или даже простые B-деревья.

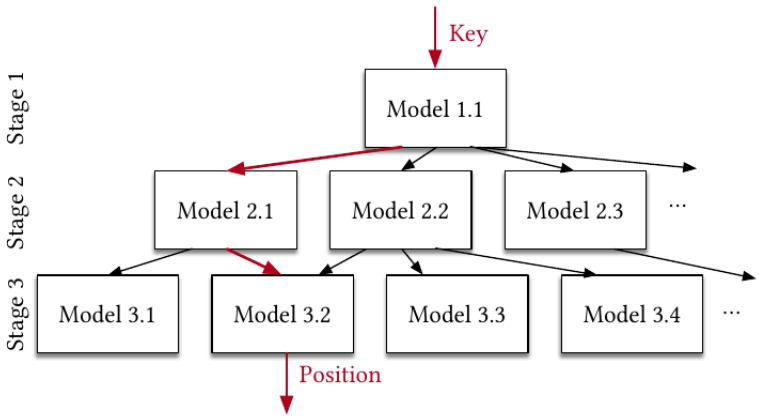

Рисунок 2.4 – Рекурсивная модель индекса

2.2 Индексы на основе хеш-таблиц

2.2.1 Хеш-индексы

Лошадь

Хэш-индексы Одной из простейших реализаций индекса является хэш-карта. Хэш -карта - это набор сегментов, содержащих результаты хэш-функции , примененной к ключу. Этот хэш указывает на местоположение, где можно найти записи. Хэш-карта пригодна только для поиска по одному ключу, поскольку сканирование диапазона было бы непомерно дорогим. Кроме того, хэш должен помещаться в память для обеспечения производительности. С учетом этих предостережений хэш-карты обеспечивают отличный доступ для конкретных случаев использования, для которых они работают. учше, когда индексируется небольшое количество значений.

only general

Мы можем упорядочивать записи, используя метод, называемый хэшированием, чтобы быстро находить записи , которые имеют заданное значение ключа поиска. Например, если файл записей сотрудников хэширован в поле имя, мы можем получить все записи о Джо. При таком подходе записи в файле группируются в сегменты, где сегмент состоит из основной страницы и, возможно, дополнительных страниц, связанных в цепочку. Корзина, к которой принадлежит запись, может быть определена путем применения специальной функции, называемой хэш-функцией, к ключу поиска. Задан номер корзины, структура индекса на основе хэша позволяет нам извлекать основную страницу для корзины в одном или двух дисковых вводах-выводах. При вставках

запись вставляется в соответствующую корзину, при этом страницы "переполнения"выделяются по мере необходимости. Чтобы выполнить поиск записи с заданным значением ключа поиска, мы применяем хэш-функцию для определения корзины, к которой принадлежат такие записи, и просматриваем все страницы в этой корзине. Если у нас нет значения ключа поиска для записи, например, индекс основан на sal, и нам нужны записи с заданным значением возраста, мы должны просканировать все страницы в файле.

2.2.2 Обученные хеш-индексы

Хеш-индексы можно рассматривать как модель сопоставления ключа позиции искомой записи в неупорядоченном массиве.

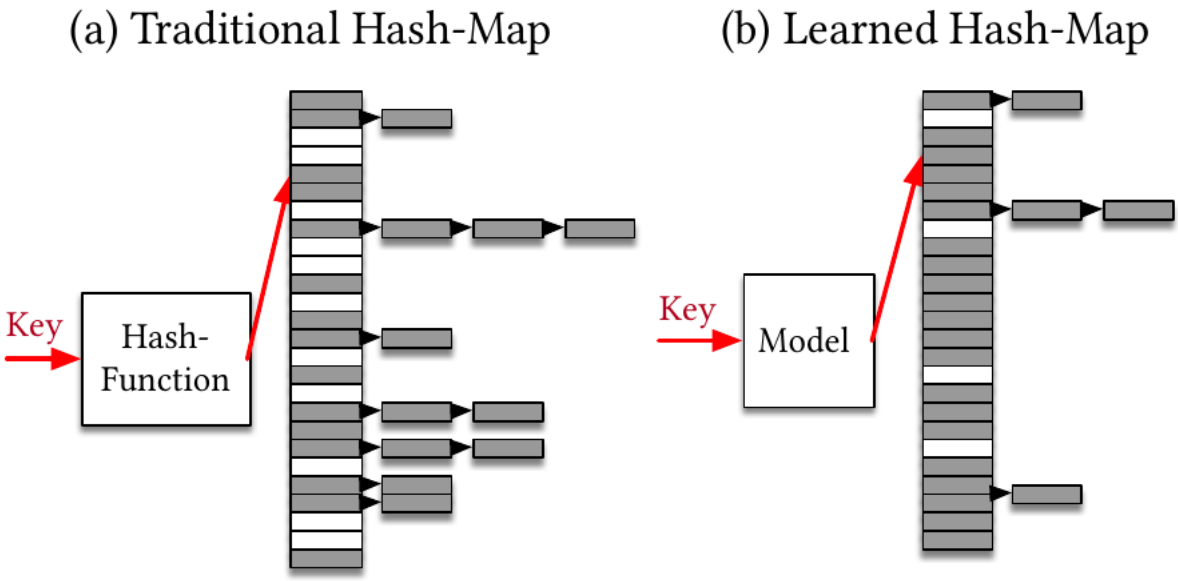

Рисунок 2.5 – Хеш-индексы

Функция распределения вероятностей распределения ключей один из возможных способов обучения хеш-индексов. Функция распределения масштабируется на размер хеш-таблицы M и для поиска положения записи аналогично случаю с B-деревьями используется формула:

$$h(K) = F(K) \cdot M, \tag{2.2}$$

где K — ключ.

2.3 Индексы на основе битовых карт

2.3.1 Фильтр Блума

Камбала 146

Лошадь

Индекс растрового изображения хранит свои данные в виде битовых массивов (растровых изображений). Когда вы проходите по индексу, это делается путем выполнения побитовых логических операций над растровыми изображениями. В B-деревьях индекс лучше всего работает со значениями, которые не повторяются часто. Это также известно как высокая мощность. Индекс растрового изображения работает намного л учше, когда индексируется небольшое количество значений.

2.3.2 Обученные индексы

Данные индексы можно рассматривать как модель проверки существования записи в массиве данных.

Фильтр Блума — алгоритм используемый для проверки существования записи.

Фильтр Блума использует массив бит размером m и k хеш-функций, каждая из которых сопоставляет ключ с одну из m позиций. Для добавления элемента в множество существующих значений ключ подается на вход каждой хеш-функции, возвращающих позицию бита, который должен быть установлен в единицу. Для проверки принадлежности ключа множеству, ключ также подается на вход k хеш-функций. Если какой-либо бит, соответствующий одной из возращенных позиций, равен нулю, то ключ не входит во множество. Из этого следует, что данный алгоритм гарантирует отсутсвие ложноотрицательных результатов.

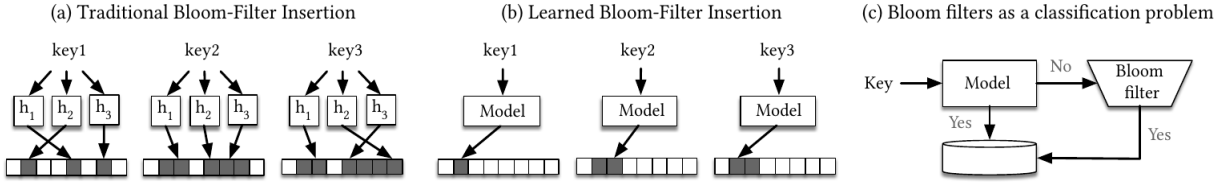

Рисунок 2.6 – Bitmap-индексы

Может со 100%-ной вероятностью сказать, что элемент отсутсвует в наборе, но то, что элемент присутсвует в наборе, со 100%-ной вероятностью он сказать не может (возможны ложноположительные результаты)

В случае индексов существования необходимо обучить функцию таким образом, чтобы среди возвращенных значений для множества ключей были коллизии, аналогично для множества неключей, но при этом не было коллизий возращенных значений для ключей и неключей.

В отличие от оригинального фильтра Блума, где $FNR = 0$, $FPR = const$, где const выбрано априори, при обучении достигается заданное значение FPR при $FNR = 0$ на реальных запросах.

3 Классификация существующих методов

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Исследование способов ускорения поисковых запросов в базах данных / Е. В. Коптенок [и др.] // Вестник образовательного консорциума среднерусский университет. Информационные технологии. — 2019. — N 1(13). — С. 24–27.
2. *Reinsel D.*, *Gantz J.*, *Rydning J.* The Digitization of the World From Edge to Core // IDC White Paper. — 2018.
3. *Носова Т. Н.*, *Калугина О. Б.* Использование алгоритма битовых шкал для увеличения эффективности поисковых запросов, обрабатывающих данные с низкой избирательностью // Электротехнические системы и комплексы. — 2018. — N 1(38). — С. 63–67.
4. DAMA-DMBOK : Свод знаний по управлению данными. — 2-е изд. — М. : Олимп-Бизнес, 2020. — 828 с.
5. The Case for Learned Index Structures / T. Kraska [et al.] // Proceedings of the 2018 International Conference on Management of Data. — SIGMOD'18, June 10–15, 2018, Houston, TX, USA, 2018. — P. 489–504.
6. ALEX: An Updatable Adaptive Learned Index / J. Ding [et al.] // Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. — 2020. — P. 969–984.
7. APEX: A High-Performance Learned Index on Persistent Memory (Extended Version) / B. Lu [et al.] // Proceedings of the VLDB Endowment. — 2022. — Vol. 15(3). — P. 597–610.
8. Updatable Learned Index with Precise Positions / J. Wu [et al.] // Proceedings of the VLDB Endowment. — 2021. — Vol. 14(8). — P. 1276–1288.
9. *Ferragina P.*, *Vinciguerra G.* The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds // PVLDB. — 2020. — Vol. 13(8). — P. 1162–1175.
10. *Григорьев Ю. А.*, *Плутенко А. Д.*, *Плужникова О. Ю.* Реляционные базы данных и системы NoSQL: учебное пособие. — Благовещенск : Амурский гос. ун-т, 2018. — 424 с.

11. *Silberschatz A.*, *Korth H. F.*, *Sudarshan S.* Database System Concepts. — New York : McGraw-Hill, 2020. — 1344 p.

12. *Эдвард Сьоре*. Проектирование и реализация систем управления базами данных. — М. : ДМК Пресс, 2021. — 466 с.

13. *Осипов Д. Л.* Технологии проектирования баз данных. — М. : ДМК Пресс, 2019. — 498 с.

14. *Lemahieu W.*, *Broucke S. vanden*, *Baesens B.* Principles of database management : the practical guide to storing, managing and analyzing big and small data. — Cambridge : Cambridge University Press, 2018. — 1843 p.

15. Encyclopedia of Database Systems / ed. by L. Liu, M. T. Özsu. — New York : Springer New York, 2018. — 4866 p.

16. *Mannino M. V.* Database Design, Application Development, and Administration. — Chicago : Chicago Business Press, 2019. — 873 p.