Базы данных

Лекция 03.1 – Основные концепции. Индексирование

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2022

2022.09.24

Оглавление

Индексирование

При выполнении запроса к таблице пользователя часто интересуют только некоторые записи в ней, например записи, имеющие определенное значение в некотором поле.

Индекс – это файл, помогающий движку базы данных быстро найти такие записи, не просматривая всю таблицу.

Наиболее распространенными способами реализации индексов являются:

- статическое хеширование;
- расширяемое хеширование;
- B-деревья.

Эффективность выполнения некоторых запросов может значительно повысить подходящая организация таблиц. Представим телефонный справочник – по сути, большая таблица, записи которой содержат имена, адреса и номера телефонов абонентов.

Фамилия Имя Отчество	Адрес	Тел. номер

Эта таблица отсортирована сначала по фамилиям, а затем по именам.

Предположим, что нам нужно узнать номер телефона конкретного человека.

Существенно ускорить поиск помогает тот факт, что записи отсортированы по имени.

Например, можно выполнить бинарный поиск, который в худшем случае потребует просмотреть log2N записей, где N – общее число записей в справочнике. Это очень быстро. Например, если N = 1 000 000, то log2N < 20, то есть чтобы найти нужного человека в справочнике, содержащем номера миллиона человек, никогда не придется просматривать больше 20 записей.

Телефонный справочник отлично приспособлен для поиска абонента по имени, но не подходит для быстрого поиска, например по номеру телефона или по адресу.

Единственный способ получить эту информацию из телефонной книги – просмотреть каждую запись в ней, что потребует просмотреть в среднем N/2 записей.

Для эффективного поиска абонентов по номеру телефона нужен справочник, отсортированный по номерам телефонов (такие справочники еще называют «*обратными телефонными справочниками*»). Однако такой справочник удобен, только если известен номер телефона. Если вы решите найти в таком справочнике номер телефона конкретного абонента, то вам снова придется просмотреть каждую запись.

Этот пример наглядно иллюстрирует важное ограничение организации таблиц — таблицу можно организовать (упорядочить) только каким-то одним способом. Если необходимо, чтобы поиск выполнялся быстро и по номеру телефона и по имени абонента, потребуются две отдельные копии телефонной книги, каждая со своей организацией. А если понадобится возможность быстро находить номер телефона по известному адресу, потребуется третий экземпляр телефонного справочника, отсортированный по адресу. Это справедливо и в отношении таблиц в базе данных.

Чтобы иметь возможность эффективно находить в таблице записи с определенным значением некоторого поля, нужна версия таблицы, организованная по этому полю.

Механизмы баз данных это удовлетворяют эту потребность, поддерживая *индексы*.

Таблица может иметь один или несколько индексов, каждый из которых определен для отдельного поля. Каждый индекс действует подобно версии таблицы, организованной по соответствующему полю.

Индекс — это файл с индексными записями. Файл индекса имеет одну индексную запись для каждой записи в соответствующей таблице. Каждая индексная запись имеет два поля, которые хранят идентификатор соответствующей записи в таблице (значение первичного ключа) и значение индексированного поля в этой записи.

ID	Name	Address	Phone

Name	ID

Address	ID

Phone	ID

На поля в индексной записи можно условно ссылаться по именам `record_id` и `field_value`.

Движок БД организует записи в файле индекса по полю `field_value`.

Можно считать для простоты, что записи в индексе по полю `field_val` отсортированы. В этом случае для того, чтобы найти запись по значению `field_val`, нужно выполнить бинарный поиск и извлечь из поля `record_id` . первичный ключ искомой записи.

Аналогично можно найти все записи со значением `field_val`. Бинарный поиск выдаст `record_id` первой записи с атрибутом `field_val.` Поскольку в индексе все записи отсортированы по `field_val`, остается только последовательно читать следующие записи в индексе, пока получаемый `field_val` будет удовлетворять поставленному условию.

18, 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3	3	5	6	9	9	10	13	18	18	**18**	18	32	37	?
b0							b1				b2				e0
b0							b1				b2	b4	b3		e0

Насколько эффективно такое применение индексов?

В отсутствие индексов лучшее, что можно предпринять при обработке любого запроса, – выполнить последовательный поиск в таблице.

Если атрибут в таблице равномерно распределен и имеет вид «A = константа», справедливо правило — полезность индекса для поля A в таблице пропорциональна количеству уникальных значений в этом поле.

Согласно этому правилу, индекс наиболее полезен, когда индексируемое поле является первичным ключом таблицы, потому что каждая запись имеет свое уникальное значение ключа.

И наоборот, индекс будет бесполезен, если число уникальных значений в поле A меньше количества записей в блоке чтения.

Статические хеш-индексы

Статическое хеширование – это, пожалуй, самый простой способ реализации индексов.
Это не самая эффективная стратегия, но достаточно простая и понятная для реализации.

Статическое хеширование

Статический хеш-индекс использует фиксированное число N ячеек, пронумерованных от 0 до N–1. Индекс также использует хеш-функцию, отображающую значения в ячейки. По результатам хеширования поля `field_val` каждая индексная запись помещается в свою ячейку.

Статический хеш-индекс действует следующим образом:

- чтобы сохранить индексную запись, ее нужно поместить в ячейку, вычисленную хеш-функцией;

- чтобы найти индексную запись, нужно вычислить хеш-функцию ключа поиска и просмотреть соответствующую ячейку;

- чтобы удалить индексную запись, нужно сначала найти ее (как указано выше), а затем удалить из ячейки.

Стоимость поиска с помощью хеш-индекса обратно пропорциональна количеству ячеек. Если индекс содержит B блоков и N ячеек, то на каждую ячейку будет приходиться около B/N блоков, поэтому для поиска в ячейке потребуется обратиться к B/N блоков.

N = 17, B = 8 ячейка полностью в одном блоке -> гарантированно одно чтение

N = 17, B = 20 ячейка располагается в двух блоках -> может быть два чтения

Расширяемое хеширование

Стоимость поиска при использовании индексов на основе статического хеширования обратно пропорциональна количеству ячеек – чем больше ячеек используется, тем меньше блоков в каждой из них.

Наиболее оптимально, когда каждую ячейку приходится ровно один блок.

Если бы размер индекса никогда не изменялся, то рассчитать это идеальное количество ячеек легко. Но на практике индексы растут по мере добавления новых записей в базу данных.

Если исходить из текущего размера индекса, то впоследствии, при его увеличении, каждая ячейка будет содержать несколько блоков индекса.

Если выбрать большее количество ячеек, ориентируясь на потребности в будущем, то пустые и почти пустые в данный момент ячейки будут напрасно расходовать значительный объем дискового пространства до тех пор, пока индекс не вырастет и их не заполнит.

Эту проблему решает стратегия, известная как *расширяемое хеширование*.

Суть этой стратегии заключается в использовании достаточно большого количества ячеек, чтобы гарантировать, что каждая ячейка никогда не будет содержать более одного блока.

Но теперь несколько ячеек могут располагаться в одном блоке, что позволяет большому количеству ячеек совместно использовать меньшее количество блоков и тем самым не допустить напрасного расходования дискового пространства.

Совместное использование блоков ячейками обеспечивается с помощью двух файлов – файла ячеек и каталога ячеек.

Файл ячеек содержит блоки индекса, а каталог ячеек отображает ячейки в блоки.

Каталог можно рассматривать как массив целых чисел, по одному для каждой ячейки. Пусть это массив `Dir`. Тогда если индексная запись хешируется в ячейку b, то запись будет сохранена в блоке `Dir[b]` файла ячеек.

Допустим, что в блоке размещаются 3 ячейки, всего ячеек 8, хеш-функция h(x)=x mod 8, в индексе семь записей с идентификаторами 1, 2, 4, 5, 7, 8, 12.

Каталог ячеек: [0 1 2 1 0 1 2 1]

Файл ячеек: [(4, r4) (8, r8) (12, r12)] [(1, r1) (5, r5) (7, r7)] [(2, r2)]

Надо заметить, что этот подход не работает, когда имеется слишком много записей с одинаковым значением `field_val`. Поскольку эти записи всегда будут хешироваться в одну и ту же ячейку, стратегия хеширования никак не сможет распределить их по нескольким ячейкам. В этом случае ячейка будет занимать столько блоков, сколько потребуется для хранения этих записей.

Индексы на основе B-дерева

Предыдущие две стратегии индексирования были основаны на хешировании.

Индексы на основе B-дерева — подход на основе сортировки. Основная идея заключается в сортировке индексных записей по значениям `field_val`.

Индексный файл – это последовательность индексных записей с полями `field_val` и `record_id`. При работе с индексным файлом для нас важно иметь возможность как можно быстрее находить идентификаторы записей (`record_id`) по значениям индексированного поля (`field_val` .

Кроме поиска поддерживается вставка и удаление.