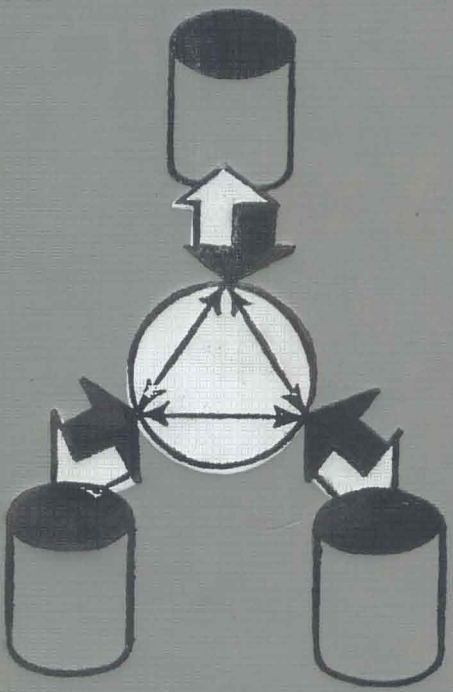
Т. ТИОРИ
ДЖ. ФРАЙ

ПРОЕКТИРОВАНИЕ СТРУКТУР БАЗ ДАННЫХ



ПРОЕКТИРОВАНИЕ СТРУКТУР БАЗ ДАННЫХ

DESIGN OF DATABASE STRUCTURES

Toby J. Teorey
James P. Fry

The
University
of
Michigan

Prentice-Hall, Inc.,
Englewood Cliffs 1982

Т. ТИОРИ
ДЖ. ФРАЙ

ПРОЕКТИРОВАНИЕ СТРУКТУР БАЗ ДАННЫХ

В ДВУХ КНИГАХ

2

Перевод с английского
Л. В. ОСИПОВОЙ,
канд. техн. наук М. Н. ПЕТУХОВА,
канд. техн. наук В. И. ЧУЧКИНА

под редакцией
канд. техн. наук В. И. СКВОРЦОВА

МОСКВА •МИР• 1985

ББК 32.973.2
Т 32
УДК 681.3

Тиори Т., Фрай Дж.

Т 32 Проектирование структур баз данных: В 2-х кн. Кн. 2. Пер. с англ. — М.: Мир. 1985. 320 с., ил.

Труд американских ученых посвящен проблеме проектирования баз данных. В русском переводе выпускается в 2-х книгах.

В книге 2 большое внимание уделено методам доступа и сравнительному анализу их эффективности, а также вопросам реорганизации баз данных. Рассматриваются архитектура систем управления распределенными базами данных и стратегии распределения данных.

Для специалистов в области вычислительной техники.

Т $\frac{2405000000\text{-}209}{041(01)\text{-}85}$ 166-85, ч. 1

ББК 32.973.2
6Ф7.3

Редакция литературы по информатике и электронике

Тоби Тиори, Джеймс Фрай

ПРОЕКТИРОВАНИЕ СТРУКТУР БАЗ ДАННЫХ

том 2

Научный редактор Т. Н. Шестакова. Младший редактор М. Ю. Григоренко. Художник Б. П. Груздев. Художественный редактор Н. М. Иванов. Технический редактор Н. И. Манохина. Корректор А. Я. Шехтер

НБ № 5011

Сдано в набор 05.07.84. Подписано к печати 08.01.85. Формат 60×90¹/₁₆. Объем 10,00. Бумага типографская № 2. Гарнитура литературная. Печать высокая. Усл. печ. л. 20,00. Усл. кр.-отт. 20,00. Уч.-изд. л. 20,49. Изд. № 20/3331. Тираж 28 000 экз. Зак. 250. Цена 1 р. 70 к.

ИЗДАТЕЛЬСТВО «МИР» 129820, ГСП, Москва, И-110, 1-й Рижский пер., 2.

Ленинградская типография № 2 головное предприятие ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 198052, г. Ленинград, Л-52, Измайловский проспект, 29.

Глава 12. Первичные методы доступа: последовательная обработка

12.1. Введение

Решения, которые принимаются на этапе физического проектирования, в отличие от решений на этапе проектирования реализации, должны учитывать вопросы, связанные с организацией путей доступа. На концептуальном уровне сущности и связи описываются в терминах ассоциаций данных без учета способа организации доступа. При проектировании СУБД-ориентированной логической структуры полученная ранее информационная структура усовершенствуется с тем, чтобы учесть различные системные ограничения и минимизировать длины путей доступа для всего множества приложений пользователей. Информационная структура с помощью достаточно простых преобразований превращается в СУБД-ориентированную логическую структуру базы данных, при этом сущности преобразуются непосредственно в записи, а связи между объектами формируют основу навигационных путей доступа. Конечно, такая организация работ по созданию баз данных серьезно ограничивает гибкость концептуального проектирования. Тем не менее введение в рассмотрение связей между сущностями создает некоторые отправные точки для изучения путей доступа, хотя необходимо учесть еще много других факторов, которые действительно позволяют спроектировать новые структуры базы данных, существенно отличающиеся от первоначальной концептуальной модели.

Проектирование логической структуры базы данных (или схемы) в основном базируется на минимизации длин путей доступа и, следовательно, тесно связано с конечной физической структурой. Физическая структура предполагает, что в схеме определен базис для навигационного или другого метода доступа; выбор конкретного варианта доступа производится в соответствии с принятыми критериями эффективности. Таким образом, принятые в качестве предварительной оценки относительной эффективности системы оценки количества обращений к логическим записям (LRA) и объема передаваемой информации преобразуются в оценки количества обращений к физическим блокам и в конечном итоге в оценки времени ввода-вывода, рассматриваемые как дополнительные меры относительной эффективности. Время отклика, включающее время обслуживания ввода-вывода и ряд других компонентов, может служить оценкой абсолютной эффективности, которая подлежит подтвержде-

нию в процессе контрольного эксперимента на реальных тестовых данных.

В этой главе будут рассмотрены способы оценки методов доступа, позволяющие рассчитывать ожидаемую относительную эффективность. В качестве средства описания многочисленных подходов, приведенных в литературе и иногда реализованных в действующих системах, используется простая модель последовательного и произвольного доступа к данным. Описываются способы оценки путей доступа при выполнении операций выборки и обновления данных, исследуются параметры, влияющие на выбор методов доступа, приводится классификация методов доступа. Одной из целей этого анализа является обеспечение достаточных условий для дальнейшей оценки методов доступа.

12.1.1. Методы доступа и их определение

Термин «метод доступа» будет определен в общем виде, т. е. будет дано определение, толкование которого не связано с каким-либо конкретным методом доступа, реализуемым СУБД или ОС. *Метод доступа* — это совокупность технических и программных средств, обеспечивающих возможность хранения и выборки данных, расположенных на физических устройствах (обычно это внешняя память). В методе доступа важны два компонента: структура памяти и механизм поиска. Структура памяти (как определено в разд. 9.1) задает ограничения на образование путей доступа к данным. *Механизм поиска* — это алгоритм, определяющий специфический путь доступа, который возможен в рамках заданной структуры памяти, и количество шагов вдоль этого пути для нахождения искомых данных.

Для иллюстрации этих определений рассмотрим последовательный файл. Структура памяти определяет возможный путь доступа как доступа в последовательном упорядоченном физическом файле, в то время как в качестве механизма поиска может быть использован последовательный поиск, бинарный поиск либо функция хеширования. Для каждого из этих механизмов поиска количество шагов (обращений к блокам) для одного и того же пути доступа существенно различается. Можно возразить, что фактически здесь не один, а по крайней мере три пути доступа, но приведенное определение удовлетворяет цели повышения согласованности и наглядности способов оценки эффективности. Доступ к последовательной структуре, отличный от последовательного, легче описать количеством шагов, требуемых для его осуществления. Доступ к произвольной структуре, такой, как дерево или сеть, часто не является последовательным в смысле физического упорядочения, но является по-

следовательным в смысле перехода от блока к блоку или от записи к записи по указателю «следующий».

Считается, что бинарный поиск или схема с хешированием адресов для физически последовательного размещения хранимых записей соответствуют произвольному доступу к блоку в пределах одного экстента базы данных. Эффективность методов доступа будет оцениваться в этой главе количеством обращений к физическим блокам при рабочей нагрузке, создаваемой приложениями пользователей. В гл. 9 показано, как по этим величинам можно рассчитать время ввода-вывода при условии, что размеры блоков и характеристики вычислительных средств известны.

12.1.2. Классификация методов доступа

Путь доступа в базе данных можно рассматривать как последовательность произвольных или последовательных обращений к хранимым записям. Различия в структуре памяти и в механизмах поиска для различных методов доступа проявляются в изменении порядка и длины последовательности указанных обращений. Вначале осуществляется обращение к блокам, а затем — к хранимым в них записям. Так как доступ к блокам требует десятков миллисекунд времени для ввода-вывода и микросекунд работы CPU для поиска внутри блока (при современной технологии), то в дальнейшем мы сосредоточим внимание на основных составляющих времени обработки: времени обращения к блоку и времени передачи. С другой стороны, с увеличением объема основной памяти все более важными становятся методы поиска внутри блоков. Они базируются на том же самом наборе методов, что и методы поиска, используемые в системном программном обеспечении для работы с данными, расположенными в основной памяти (таблицы символов и таблицы страниц). Мы рассмотрим весь диапазон методов доступа к внешней памяти, хотя те же концепции применимы к любому типу запоминающих устройств.

Классификация методов доступа, основанная на типе обрабатываемых приложений пользователей, была предложена в работе [278]. В соответствии с этой методикой в один класс объединяются те методы доступа, которые являются наиболее эффективными для обслуживания заданного типа приложений. Можно выделить три основные группы приложений:

1. ПОЛУЧИТЬ ВСЕ, ПОЛУЧИТЬ МНОГИЕ (GET ALL, GET MANY).

Этот класс приложений требует доступа к значительной части базы данных, обычно от 10 до 100 % записей. В эту группу чаще всего попадают: последовательная обработка, генерация

больших отчетов и пакетная обработка. Нижняя граница 10 % в большинстве случаев соответствует уровню, выше которого последовательный поиск *записей-целей* (т. е. записей, которые удовлетворяют поисковым условиям запроса) в базе данных обычно более эффективен, чем индексный.

2. ПОЛУЧИТЬ УНИКАЛЬНУЮ (GET UNIQUE).

Этот класс — противоположность первому. Доступ осуществляется только к одной требуемой записи, если таковая существует. В этот класс попадают: метод произвольного доступа и индексный метод, основанный на поиске по первичному ключу.

3. ПОЛУЧИТЬ НЕКОТОРЫЕ (GET SOME).

Третий класс приложений соответствует промежуточному случаю, когда количество обращений к базе данных для поиска требуемой записи находится в диапазоне от 0 до 10 % общего объема базы. Такого типа запросы представляют собой наиболее распространенный тип приложений, и для этого случая наибольший интерес представляет поиск по вторичному ключу.

Данная классификация позволяет проектировщику для заданного набора приложений пользователей быстро выбирать те методы, которые лучшим образом удовлетворяют требованиям

Таблица 12.1. **Классификация основных методов доступа**

Класс пользовательских приложений	Методы доступа
ПОЛУЧИТЬ ВСЕ, ПОЛУЧИТЬ МНОГИЕ (10—100 %)	Последовательный
	Физически последовательная организация (смежная память)
	Связанная последовательная организация (несмежная память)
ПОЛУЧИТЬ УНИКАЛЬНУЮ (одну или ни одной)	Прямой
	Произвольный (хеширование идентификатора)
	Индексно-произвольный (с полным индексом)
	Индексно-последовательный
	Бинарное дерево
	B-дерево
	Поиск по *TRIE*-структуре
ПОЛУЧИТЬ НЕКОТОРЫЕ (0—10%)	Мультисписковый
	Инвертированный
	Секционно-инвертированный
	Двусвязанное дерево

к времени ответа на запросы и ограничениям на объем памяти. Так как возможны и другие варианты классификации, будут проанализированы некоторые характеристики известных в настоящее время методов доступа, чтобы в дальнейшем разбить их на подклассы. Очевидно, существует большое количество методов доступа, которые не укладываются в граннцы трех вышеперечисленных основных категорнй. Нужно прнзнать, что многие методы доступа, реализованные в действующнх снстемах, представляют собой попытку найти компромиссное решение для некоторых типов пользовательских прнложений, наиболее характерных для конкретной снстемы. Такне методы легко идентифицируются в этой класснфикационной схеме. Табл. 12.1 обобщает обсуждаемую в данной главе основную классификацню. Следует обратить внимание на отсутствие системно-завнсимых методов доступа. Некоторые из них будут представлены в явном виде в рассматриваемых примерах, но большинство других может быть составлено из основных компонентов, приведенных в табл. 12.1.

12.2. Обработка данных при физически последовательной организации: ПОЛУЧИТЬ ВСЕ, ПОЛУЧИТЬ МНОГИЕ (GET ALL, GET MANY)

Наиболее эффективной для последовательной обработки большнх объемов данных является физнчески последовательная структура. Последовательное смежное размещение записей допускает фнзнческое блокирование, тем самым миннмизируя время доступа к данным. Обработка небольших объемов данных, особенно когда высока их степень изменчивости, часто наиболее эффективна при структуре хранения в виде последовательно соединенных участков (связанная последовательная структура). Каждый из этих вариантов будет рассмотрен для различных тнпов поиска и обновления данных, мы также получим выражение для оценкн некоторых часто встречающихся решенний.

Следует подчеркнуть, что физически последовательная и связанная последовательная структуры являются базовыми для очень большого класса методов доступа, таких, как индексно-последовательный, мультиспнсковый, инвертнрованный, и различных механизмов понска с использованнем деревьев. После анализа характеристик и ограничений последовательной организации будет легче перейти к рассмотрению более сложных структур, которые снитезируются из этих двух основных вариантов.

12.2.1. Поиск при физически последовательной организации данных

Физически последовательные структуры представляют собой простейшую для анализа организацию. Однако существует значительное количество параметров, которые важны не только для обработки последовательно хранимых данных, но и для анализа более сложных физических организаций:

- *Тип обработки.* Последовательная обработка, произвольные выборка и внесение изменений, пакетные поиск и внесение изменений, составление отчетов.
- *Упорядоченность записей данных.* Допускается пакетная обработка; есть возможность избежать сортировки, если порядок

Рис. 12.1. Физически последовательная организация данных.

данных в отчете соответствует порядку размещения записей в файле.

- *Коэффициент блокирования.* Влияет на эффективность последовательной обработки.
- *Размер файла* (количество записей).
- *Коэффициент загрузки.* Позволяет увеличивать объем базы данных без излишних переполнений и последующих перезагрузок.
- *Механизм поиска.*

Пример физически последовательной организации данных приведен на рис. 12.1.

Рассмотрим случай, когда физически последовательная структура содержит NR смежных записей. Они могут быть неупорядоченными или упорядоченными по возрастанию или убыванию значений первичного ключа. Предполагается, что записи имеют постоянный размер, содержат неключевые элементы и в них отсутствуют указатели.

Несблокированная последовательная организация (файл или подфайл) содержит NR блоков т. е. каждая запись расположена в отдельном блоке. В общем случае имеется NBLK = $\lceil NR/EBF \rceil$ блоков, где EBF — коэффициент полезного блокирования. Для файлов с записями переменной длины коэффициент блокирования не задается; записи объединяются в блоки, количество которых NBLK рассчитывается, исходя из характеристик длин записей, как показано в разд. 9.3.2. Если предположить, что ошибка, обусловленная неполным заполне-

нием блока, пренебрежимо мала, то среднее количество физических блоков, к которым осуществляется доступ при поиске произвольной записи, равно

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ, последоват. поиск)} = \frac{1 + \text{NBLK}}{2}\,\text{sba}, \qquad (12\text{-}1)$$

где sba означает, что рассматривается последовательный доступ к блокам. Аналогично rba означает произвольный доступ к блокам.

Выражение (12-1) справедливо как для упорядоченного, так и для неупорядоченного файла при условии, что файл содержит искомую запись. В противном случае для упорядоченного файла длина поиска остается той же самой, а для неупорядоченного файла она будет равна NBLK, т. е. требуется проверка каждой записи. Таким образом, неупорядоченный файл является неэффективным только при произвольной выборке в случае отсутствия в нем искомых записей.

Для того чтобы бинарный поиск уникальной записи в файле, содержащем NBLK блоков, был оправдан, требуется, чтобы этот файл был упорядочен и имел приемлемую длину. При бинарном поиске по мере выбора соответствующих блоков осуществляется их просмотр с целью поиска записи-цели. Средняя длина бинарного поиска в файле из NBLK блоков для случая $\text{NBLK} \geqslant 50$ равна [188]

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ, бинарн. поиск)} = \log_2 \text{NBLK} - 1 = \log_2 (\text{NBLK}/2)\,\text{rba}. \qquad (12\text{-}2)$$

Для $\text{NBLK} < 50$ более точное выражение приведено в работе [218]. Как было показано, в любом случае верхняя граница равняется $\lceil \log_2 \text{NBLK} \rceil + 1$. Заметим, что при бинарном поиске требуется произвольный доступ к блоку, причем количество обращений ограничено размером экстента файла. Кроме того, считается, что все блоки заполнены одинаково.

Если требуется выборка всех записей (ПОЛУЧИТЬ ВСЕ), то как для упорядоченных, так и для неупорядоченных файлов количество обращений одинаково:

$$\text{РВА (ПОЛУЧИТЬ ВСЕ)} = \text{NBLK}\,\text{sba}, \qquad (12\text{-}3)$$

где $\text{NBLK} = \lceil \text{NR/EBF} \rceil$ для записей фиксированной длины и постоянного значения коэффициента полезного блокирования. Выражение (12-3) справедливо для всех приложений пользователей, включая запросы, использующие для доступа вторичные ключи.

Последовательная обработка записей в пакетном режиме претерпевает некоторые изменения. Для того чтобы наиболее эффективно обработать в пакетном режиме RPBR записей, их помещают во вспомогательный файл в том же порядке, как они расположены в главном файле, и объединяют в один набор. Записи главного файла, ключи которых соответствуют ключам записей вспомогательного файла, сохраняются и обрабатываются во время операции слияния. Количество обращений, требуемое для обработки вспомогательного (tf) и главного файлов (mf), равно

$$\text{РВА (ПОЛУЧИТЬ ПАКЕТ)}_{tf} = \left\lceil \frac{\text{RPBR}}{\text{EBF}_{tf}} \right\rceil \quad \text{sba,} \quad (12\text{-}4)$$

$$\text{РВА (ПОЛУЧИТЬ ПАКЕТ)}_{mf} < \text{NBLK}_{mf} + \text{RPBR} \quad \text{sba,} \quad (12\text{-}5)$$

где первый член соответствует последовательному просмотру файла, а второй член отражает задержки рестарта после отыскания каждой записи-цели. Неравенство (12-5) является не очень конструктивным. Чтобы получить точное выражение для РВА (ПОЛУЧИТЬ ПАКЕТ)$_{mf}$, предположим, что эти RPBR записей распределены в основном файле равномерно. Для определения количества обращений к логическим записям файлов большой размерности докажем следующую теорему.

• **ТЕОРЕМА 12.1.** Средняя длина поиска множества k произвольно распределенных записей в файле, содержащем n записей, равна $kn/(k+1)$ записям.

Доказательство. Предположим, что n достаточно велико, что позволяет перейти от дискретной модели к непрерывной. Пусть $X_1, X_2, \ldots, X_k$ — независимые случайные переменные, определенные на интервале $[0, n]$. Необходимо найти среднее значение $E(a)$ величины a, где $a = \max(X_1, X_2, \ldots, X_k)$. Пусть

$$F(a) = P(X_1 \leqslant a \wedge X_2 \leqslant a \wedge \ldots \wedge X_k \leqslant a) =$$

$$= P(X_1 \leqslant a) \cdot P(X_2 \leqslant a) \cdot \ldots \cdot P(X_k \leqslant a) =$$

$$= \left(\frac{a}{n}\right)_1 \left(\frac{a}{n}\right)_2 \cdots \left(\frac{a}{n}\right)_k = \left(\frac{a}{n}\right)^k \text{ для комулятивной функции распределения.}$$

Тогда функция плотности будет

$$f(a) = \frac{dF(a)}{da} = \frac{d\left(\frac{a}{n}\right)^k}{da} = \frac{ka^{k-1}}{n^k}.$$

В конечном итоге среднее значение величины a равно

$$E(a) = \int_0^n af(a)\,da = \int_0^n \frac{aka^{k-1}}{n^k}\,da = \frac{k}{n^k}\int_0^n a^k\,da =$$

$$= \frac{k}{n^k}\,\frac{a^{k+1}}{k+1}\Bigg|_0^n = \frac{k}{n^k}\cdot\frac{n^{k+1}}{k+1} = \frac{kn}{k+1}.$$

Полагая, что NR достаточно велико, применим полученный результат к рассматриваемой задаче:

$$\text{LRA (ПОЛУЧИТЬ ПАКЕТ)}_{mf} = \frac{\text{NR} \times \text{RPBR}}{\text{RPBR} + 1}. \qquad (12\text{-}6)$$

Если $\text{RPBR} = k = 1$, то этот случай означает наличие в пакете одной записи и эквивалентен выборке произвольной записи. Для случая выборки произвольной записи уравнение (12-6) преобразуется к виду $\text{LRA}_{mf} = \text{NR}/2$. Для записей фиксированной длины можно перейти от средних оценок количества обращений к записям к среднему количеству обращений к блокам, используя следующее выражение:

$$\text{PBA (ПОЛУЧИТЬ ПАКЕТ)}_{mf} = \frac{\text{LRA}_{mf}}{\text{EBF}_{mf}} + \text{RPBR}\ \ \text{sba} =$$

$$= \frac{\text{NR} \times \text{RPBR}}{\text{EBF}_{mf}(\text{RPBR} + 1)} + \text{RPBR}\ \ \text{sba}. \qquad (12\text{-}7)$$

Общее количество обращений к блокам при пакетной выборке равно

$$\text{PBA (ПОЛУЧИТЬ ПАКЕТ)} = \text{PBA (ПОЛУЧИТЬ ПАКЕТ)}_{tf} +$$

$$+ \text{PBA (ПОЛУЧИТЬ ПАКЕТ)}_{mf} = \frac{\text{RPBR}}{\text{EBF}_{tf}} +$$

$$+ \frac{\text{NR} \times \text{RPBR}}{\text{EBF}_{mf}(\text{RPBR} + 1)} + \text{RPBR}\ \ \text{sba}. \qquad (12\text{-}8)$$

12.2.2. Внесение изменений при физически последовательной организации

Обновление последовательного файла требует создания нового главного файла. Следовательно, сначала должны быть прочитаны старый главный и некоторый вспомогательный файлы, а затем должен быть записан новый главный файл. Внесение изменений в произвольном порядке потребовало бы выполнения достаточно большого количества операций, поэтому обычно этот режим или просто запрещают, или строго ограничивают, как, например, в IMS HSAM. Пакетный режим внесения изменений в последовательный файл является обычной операцией, и для этого случая затраты на доступ вычисляются достаточно

Таблица 12.2. **Затраты на функционирование базы данных и количество обращений к физическим блокам (PBA_s) для физически последовательной организации** [1]

Операция	Упорядоченный файл [2]	Неупорядоченный файл [2]
ПОЛУЧИТЬ ВСЕ	$\lceil NR/EBF \rceil$ sba	$\lceil NR/EBF \rceil$ sba
ПОЛУЧИТЬ УНИКАЛЬНУЮ ПОСЛЕДОВАТЕЛЬНЫЙ (искомая запись найдена)	$\frac{1+NBLK}{2}$ sba	$\frac{1+NBLK}{2}$ sba
ПОСЛЕДОВАТЕЛЬНЫЙ (не найдена)	$\frac{1+NBLK}{2}$ sba	NBLK sba
БИНАРНЫЙ (найдена)	$\log_2 \frac{NBLK}{2}$ rba	N/A
БИНАРНЫЙ (не найдена)	$\lfloor \log_2 NBLK \rfloor + 1$ rba	N/A
ПРЯМОЙ (найдена)	1 rba	N/A
ПРЯМОЙ (не найдена)	1 rba	N/A
ПОЛУЧИТЬ СЛЕДУЮЩУЮ	1/EBF sba	1/EBF sba
ПОЛУЧИТЬ ПРЕДЫДУЩУЮ	1/EBF sba	1/EBF sba
ПОЛУЧИТЬ НЕКОТОРЫЕ (булевый запрос)	То же, что и при ПОЛУЧИТЬ ВСЕ	То же, что и при ПОЛУЧИТЬ ВСЕ
ПОЛУЧИТЬ ПАКЕТ	$\left\lceil \frac{RPBR}{EBF_{tf}} \right\rceil + \frac{NR \times RPBR}{EBF_{mf}(RPBR+1)} + RPBR$ sba	N/A
ИЗМЕНИТЬ ПАКЕТ	$NBLK_{mf\ (старый)} + NBLK_{tf} + \left\lceil \frac{NR + RPBI - RPBD}{EBF_{mf\ (старый)}} \right\rceil$ sba	N/A

[1] NR — количество записей в файле;
EBF — коэффициент полезного блокирования;
RPBR — количество записей, выбираемых из файла в пакетном режиме;
RPBI — количество записей, включаемых в файл в пакетном режиме;
RPBD — количество записей, удаляемых из файла в пакетном режиме;
NBLK — количество блоков в файле;
tf — вспомогательный файл;
mf — главный файл;
N/A — неприменима.

[2] Среднее количество обращений к блоку.

просто:

$$\text{PBA (ИЗМЕНИТЬ ПАКЕТ)} = \text{NBLK}_{\text{mf (старый)}} + \text{NBLK}_{\text{tf}} + \\ + \text{NBLK}_{\text{mf (новый)}} \text{ sba}, \quad (12\text{-}9)$$

где $\text{NBLK}_{\text{mf (новый)}} = \left\lceil \frac{\text{NR} + \text{RPBI} - \text{RPBD}}{\text{EBF}_{\text{mf (старый)}}} \right\rceil$, RPBI и RPBD — общее количество включаемых и удаляемых записей соответственно. Операции выборки и обновления физически последовательных файлов представлены в табл. 12.2.

Если при создании нового главного файла требуется контроль записи, то каждый из $\text{NBLK}_{\text{mf(новый)}}$ блоков должен быть считан повторно непосредственно после его записи. Это дополнительное считывание представляет собой последовательное обращение к блоку, так как блок считывается с той же самой дорожки (т. е. дополнительно вносятся только задержки, связанные с вращением диска). Если требуется учитывать время CPU и другие задержки, возникающие в период между записью и контрольным считыванием блока, то следует воспользоваться моделью доступа, приведенной в разд. 9.2, приняв $\text{PBA}_s = \text{NBLK}_{\text{mf (новый)}}$ С другой стороны, если эти задержки незначительны, то выполнение повторного считывания требует целого числа оборотов диска $k \times \text{ROT}$, где $k = \lceil \text{BKS/BPT} \rceil$ — количество дорожек, необходимых для размещения блока. Большинство систем ограничивают размер блока так, чтобы он не превышал размера дорожки, поэтому обычно $k = 1$.

12.2.3. Выбор параметров физически последовательной организации

Наиболее важными вопросами, которые нужно решить при проектировании структур физических баз данных, являются выбор надлежащего размера блока и выбор первичного ключа упорядочения данных. Каждая из этих проблем будет рассматриваться на примерах.

Размер блока

Эффективность последовательной обработки последовательно хранимых данных повышается с увеличением размера блока, так как при этом сокращается количество физических обращений к внешней памяти. Хотя с увеличением размера блока возрастает объем и соответственно время передачи данных, общее количество блоков уменьшается; это может привести к значительной экономии времени доступа к блокам.

Прямой или произвольный метод доступа к отдельным запи-

сям наиболее эффективен в случае несблокированных записей, так как при этом минимизируется количество выбираемых, но неиспользуемых данных. Рассмотрим случай, когда физически последовательный файл или подфайл обрабатывается последовательно с вероятностью P и к нему с вероятностью $1-P$ осуществляется прямой доступ. Для файла, содержащего NR записей, при LF = 1 и BOVHD = 0 имеем EBF = BF, и количество обращеиий к файлу при последовательной обработке блоков точно равно количеству блоков в файле:

$$\text{PBA} = \text{NBLK} = \left\lceil \frac{\text{NR}}{\text{EBF}} \right\rceil \text{ sba}. \qquad (12\text{-}10)$$

Количество обращений к блокам в случае прямого доступа к файлу равно 1. Объединяя эти два случая с учетом вероятностей их появления, получим среднее количество обращений к блокам:

$$\text{PBA} = P\left\lceil \frac{\text{NR}}{\text{EBF}} \right\rceil \text{ sba} + (1-P) \text{ rba}. \qquad (12\text{-}11)$$

Следовательно, среднее время ввода-вывода равно

$$\text{TIO} = P\left\lceil \frac{\text{NR}}{\text{EBF}} \right\rceil \text{TSBA} + (1-P)\,\text{TRBA}. \qquad (12\text{-}12)$$

Подставляя в (12-12) выражения (9-16) и (9-24) для случая разделенного диска (худший случай), получим

$$\text{TIO} = P\left\lceil \frac{\text{NR}}{\text{EBF}} \right\rceil \left[\text{SEEK(CPD)} + \frac{\text{ROT}}{2} + \frac{\text{BKS}}{\text{TR}}\right] + $$

$$+ (1-P)\left[\text{SEEK(CPD)} + \frac{\text{ROT}}{2} + \frac{\text{BKS}}{\text{TR}}\right]. \qquad (12\text{-}13)$$

Учитывая, что BKS = EBF×SRS, перепишем (12-13) в виде

$$\text{TIO} = P\left\lceil \frac{\text{NR}}{\text{EBF}} \right\rceil \times \left[\text{SEEK(CPD)} + \frac{\text{ROT}}{2} + \frac{\text{EBF} \times \text{SRS}}{\text{TR}}\right] + $$

$$+ (1-P)\left[\text{SEEK(CPD)} + \frac{\text{ROT}}{2} + \frac{\text{EBF} \times \text{SRS}}{\text{TR}}\right]. \qquad (12\text{-}14)$$

Для того чтобы найти величину EBF, которая минимизирует время ввода-вывода, воспользуемся максимальной оценкой количества блоков и перейдем к непрерывной аппроксимации выражения (12-14). Затем, взяв производную от TIO по EBF и

приравняв ее к нулю, получим

$$\mathrm{TIO} = \frac{P \times \mathrm{NR}\,[\mathrm{ROT}/2 + \mathrm{SEEK}\,(\mathrm{CPD})]}{\mathrm{EBF}} + \frac{P \times \mathrm{NR} \times \mathrm{EBF} \times \mathrm{SRS}}{\mathrm{EBF} \times \mathrm{TR}} +$$

$$+ \mathrm{SEEK}\,(\mathrm{CPD}) + \mathrm{ROT}/2 + \frac{\mathrm{EBF} \times \mathrm{SRS}}{\mathrm{TR}} - P \times \mathrm{SEEK}\,(\mathrm{CPD}) -$$

$$- P \times \mathrm{ROT}/2 - P \times \mathrm{EBF} \times \mathrm{SRS}/\mathrm{TR}, \qquad (12\text{-}15)$$

$$\frac{d\,\mathrm{TIO}}{d\,\mathrm{EBF}} = \frac{-P \times \mathrm{NR}\,[\mathrm{ROT}/2 + \mathrm{SEEK}\,(\mathrm{CPD})]}{\mathrm{EBF}^2} + \frac{\mathrm{SRS}}{\mathrm{TR}} - \frac{P \times \mathrm{SRS}}{\mathrm{TR}} = 0,$$

$$\frac{\mathrm{SRS}\,(1-P)}{\mathrm{TR}} = \frac{P \times \mathrm{NR}\,[\mathrm{ROT}/2 + \mathrm{SEEK}\,(\mathrm{CPD})]}{\mathrm{EBF}^2}\Bigg],$$

$$\mathrm{EBF} = \left[\frac{P \times \mathrm{NR}\,[\mathrm{ROT}/2 + \mathrm{SEEK}\,(\mathrm{CPD})]\,\mathrm{TR}}{\mathrm{SRS}\,(1-P)}\right]^{1/2}.$$

Вторая производная положительна, поэтому если существует положительный и конечный корень EBF, то он соответствует

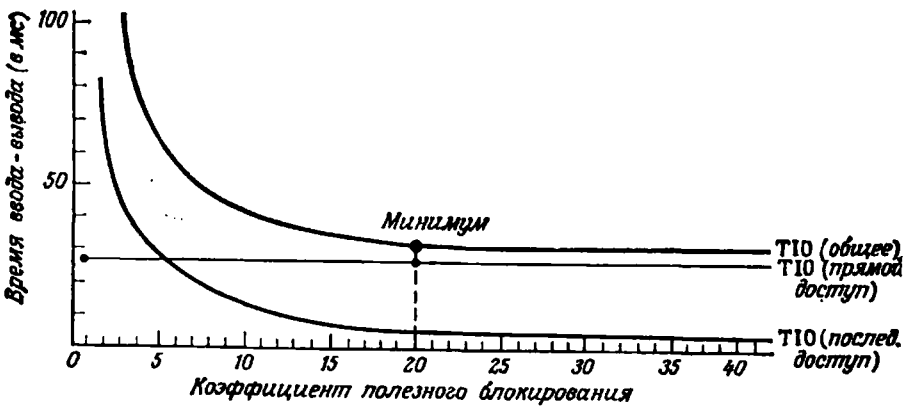


Рис. 12.2. Время обслуживания ввода вывода для последовательного файла.

минимуму TIO. Это справедливо для $0 < P < 1$. Дробное значение EBF должно быть округлено в сторону как увелнчения, так и уменьшения до ближайшего целого числа, и оба возможных значения нужно подставить в выражение (12-13) для TIO. Например, для $P = 10^{-4}$, NR $= 10^5$ записей, SEEK(CPD) $= 25$ мс, ROT $= 16{,}7$ мс, TR $= 1198$ байт/мс и SRS $= 100$ байт получим оптимальное значение коэффициента полезного блокирования

$$\mathrm{EBF} = \left[\frac{10^{-4} \times 10^5\,(8{,}35 + 25\ \text{мс}) \times 1198\ \text{байт/мс}}{100\ \text{байт} \times (1 - 10^{-4})}\right]^{1/2} = 19{,}99 \simeq 20.$$

В данном примере коэффициент полезного блокирования получился достаточно большим, несмотря на очень низкий про-

цент последовательно обрабатываемых приложений. Из-за того что для обработки последовательных приложений требуется достаточно большое время ввода-вывода, оно считается доминирующим фактором при обработке данных. Практически верхняя граница величины коэффициента блокирования определяется размером дорожки накопителя и ограничениями на величину буфера. Рис. 12.2 иллюстрирует изменение времени ввода-вывода для рассматриваемого примера. Из этого графика видно, что общее время ввода-вывода описывается достаточно пологой кривой для $EBF > 10$, которая начинает медленно расти с $EBF = 20$; таким образом, близкие к минимальным общие затраты будут иметь место для широкого диапазона изменения EBF. На графике хорошо виден выигрыш, который получается при последовательной обработке за счет выбора соответствующего значения коэффициента блокирования. Применение блокирования увеличивает также эффективность использования дорожек дисков и магнитных лент за счет уменьшения объема служебной информации, находящейся в прямой зависимости от количества блоков.

Выбор ключа упорядочения

Допустим, что данный файл имеет n возможных ключей для первичного упорядочения. Ключи упорядочения могут иметь либо уникальные, либо неуникальные значения для каждой записи. Мы хотим выбрать для упорядочения такой ключ, который позволяет минимизировать для всех приложений общие затраты ввода-вывода. Допускается, что могут потребоваться дополнительные затраты на пересортировку файла, прежде чем будет завершена обработка отдельных приложений или закончена печать отчетов. Если существует n способов упорядочения файла, то для файла, упорядоченного по ключу k, общие затраты на ввод-вывод можно записать как

$$\mathrm{COST}(k) = f_k \times \mathrm{CIO}_k + \sum_{\substack{j=1 \\ j \neq k}}^{n} f_j(\mathrm{CIO}_j + \mathrm{CSORT}_j), \quad (12\text{-}16)$$

где f_k — частота обработки приложений, использующих ключ k, т. е. не требуется сортировки записей; CIO_k — затраты на ввод-вывод при обработке приложений, использующих ключ k; f_j — частота обработки приложений, использующих ключ $j \neq k$, т. е. требуется время на дополнительную сортировку; CIO_j — затраты времени на ввод-вывод при обработке приложений, использующих ключ $j \neq k$; CSORT_j — затраты времени на сортировку файла по ключу j. Необходимо найти ключ k, который минимизирует общие затраты на ввод-вывод. Переписав выражение (12-16) так, чтобы включить член $f_k \times \mathrm{CIO}_k$ в общую

сумму, получим, что каждая составляющая этого выражения имеет общий постоянный член $\sum_{j=1}^{n} f_j \times \mathrm{CIO}_j$:

$$\mathrm{COST}(k) = \sum_{j=1}^{n} f_j \times \mathrm{CIO}_j + \sum_{\substack{j=1 \\ j \neq k}}^{n} f_j \times \mathrm{CSORT}_j. \quad (12\text{-}17)$$

Следовательно, общие затраты будут минимальны, если используется ключ k, который минимизирует сумму $\sum_{j=1, j \neq k}^{n} f_j \times \mathrm{CSORT}_j$, или, проще, ключ k, который максимизирует выражение $f_k \times \mathrm{CSORT}_k$. Если время сортировки постоянно, то общие затраты будут минимальны при выборе для упорядочения такого ключа k, для которого величина f_k наибольшая. В этом случае анализ значительно упрощается и интересно отметить, что для решения важна только частота использования приложений, а не затраты на их обработку.

В общем случае, когда время сортировки для различных вариантов первичного упорядочения различно, решение (если оно существует) сводится к нахождению максимума произведения $f_k \times \mathrm{CSORT}_k$. Очень часто можно утверждать, что если затраты на сортировку по ключу малы, то упорядочение по этому ключу не приводит к минимизации общих затрат.

Отметим, что для упрощения задачи предполагалось, что затраты на ввод-вывод для каждого типа приложений имеют одинаковый вес. На практике эти зависимости могут быть более сложными. Иногда может потребоваться, чтобы какое-то приложение было обработано за определенное время (например, из-за ограничения времени использования линии связи или других причин, связанных с наложением большого штрафа). Эти потери могут быть учтены в модели путем введения весового коэффициента WT_j перед сомножителем CSORT_j в выражении (12-17). Если сумма $\mathrm{CIO}_j + \mathrm{CSORT}_j$ больше, чем время, в течение которого может быть использована линия связи, то будет наложен штраф:

$$\mathrm{COST}(k) = \sum_{j=1}^{n} f_j \times \mathrm{CIO}_j + \sum_{\substack{j=1 \\ j \neq k}}^{n} f_j \times \mathrm{CSORT}_j \times WT_j, \quad (12\text{-}18)$$

где $WT_j = \begin{cases} 1, & \text{если } \mathrm{CIO}_j + \mathrm{CSORT}_j < \text{времени разрешенного простоя линии,} \\ \text{штраф} & \text{в противном случае.} \end{cases}$

Затраты на сортировку произвольно упорядоченных записей методом m-путевого слияния в первом приближении рассмотрены Кнутом [188]. Полная сортировка состоит из начальной

фазы и X_{min} фаз слияния, где X_{min} — минимальное значение X, которое удовлетворяет условию $m^X \geqslant$ NBLK. Если каждая фаза или прогон файла требуют считывания и записи каждого блока, то общее количество обращений к физическому блоку для всех фаз слияния равно

$$\text{PBA (SORT)} = 2 \times \text{NBLK}(1 + X_{min}). \qquad (12\text{-}19)$$

12.3. Обработка данных при связаниой последовательной организации

12.3.1. Поиск при связанной последовательной организации

Часто приходится выполнять последовательную обработку данных, которые расположены в физически несмежиых участках памяти. Связаниая последовательная организация даиных

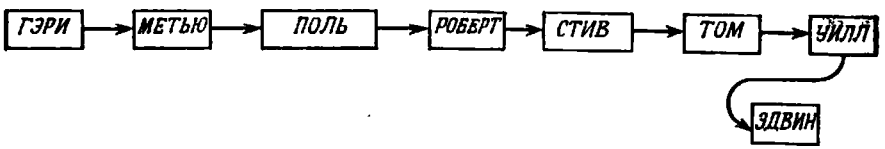


Рис. 12.3. Связанная последовательная организация данных.

позволяет осуществить динамическое распределение памяти в случае когда в одной и той же области памяти выполняются иесколько процессов или данные обладают большой степенью изменчивости. Логическое представление связанной последовательиой организации данных изображеио на рис. 12.3.

Основное различие между связанной последовательной и физически последовательной организациямн заключается в применеиии специальных указателей вместо смежного размещения данных в памяти. Это приводит к невозможности иснользования бинарного поиска, поскольку данные уже не упакованы плотно и отсутствует возможность вычисления промежуточиых адресов.

Каждая запись в связанном списке может оказаться расположенной в другом блоке по отиошению к предшествующей ей записи. При условии произвольного размещения записей вероятность того, что две такие записи окажутся в одном блоке, равна (EBF — 1)/(NR — 1), где EBF — 1 соответствует количеству остальных записей в данном блоке, а NR — 1 — общему количеству остальных записей в файле. В базе даниых с различными типами записей это выражение значительно сложнее. Предположим, что вероятность нахождения двух логически по-

следовательных записей в одном блоке мала и, следовательно, каждое обращение к записи в связанном последовательном списке требует произвольного доступа к блоку. Для последовательного доступа ко всем записям в файле имеем

$$\text{РВА (ПОЛУЧИТЬ ВСЕ)} = \text{NR rba}. \qquad (12\text{-}20)$$

При пакетной обработке RPBR записей требуется

$$\text{РВА (ПОЛУЧИТЬ ПАКЕТ)} = = \frac{\text{RPBR} \times \text{NR}}{\text{RPBR} + 1} \text{ rba} + \left\lceil \frac{\text{RPBR}}{\text{EBF}_{tf}} \right\rceil \text{ sba} \qquad (12\text{-}21)$$

при условии, что вспомогательный файл является физически последовательным. Поиск произвольной записи оценивается средним количеством обращений к блокам

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ)} = \frac{1 + \text{NR}}{2} \text{ rba}, \qquad (12\text{-}22)$$

когда файл упорядочен, и не зависит от существования в файле искомой записи. Если файл неупорядочен и искомая запись существует, то среднее количество обращений к блокам дается выражением (8-22). Если искомая запись не существует, то имеем

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ')} = \text{NR rba}. \qquad (12\text{-}23)$$

12.3.2. Внесение изменений при связанной последовательной организации данных

Внесение изменений в связанные списки детально изучено для случая использования внешней памяти и произвольного доступа. Рассмотрим простую, но полезную для иллюстрации данного подхода модель. Предположим, что список упорядочен; неупорядоченные списки будут рассмотрены в разд. 12.3.3.

Включение данных в связаный список производится после отыскания записи, которая будет являться последующей по отношению к включаемой записи. Средние затраты на поиск этой записи равны $(1 + \text{NR})/2$ произвольным обращениям к блокам. Если в блоке, содержащем требуемую последующую запись, нет свободного места, то возможно несколько различных вариантов действий. Во-первых, для новой записи может быть выделен новый пустой блок, что потребует выполнения одного обращения к блоку для помещения включаемой записи в новый блок и последовательного обращения для перезаписи блока, содержащего предыдущую запись. Во-вторых, может быть заранее создан блок переполнения; включение новой записи потребует обращения для чтения блока переполнения и последовательного обращения для его перезаписи плюс последовательное обращение для перезаписи предыдущей включаемой записи. В-третьих,

первый блок может быть расщеплен на два блока — это потребует записи двух блоков: одного произвольного обращения для записи нового блока и последовательного обращения для перезаписи старого блока. В этих трех случаях для включения новой записи в этот упорядоченный список после завершения операции поиска потребуется максимум три обращения к блокам.

Применение двусвязанного списка требует перезаписи блоков, содержащих вновь включаемую запись, а также предшествующую (PRIOR) и последующую (NEXT) записи. Это приводит к необходимости максимум четырех обращений к блокам:

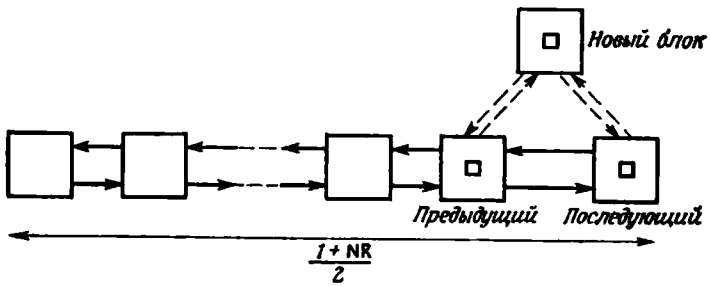


Рис. 12.4. Внесение нового блока в двусвязанный список.

одного произвольного обращения для чтения и последовательного обращения для записи блока переполнения, а также последовательных обращений к двум блокам для перезаписи предыдущей и последующей записей (см. рис. 12.4).

Удаление записи при связанной последовательной организации требует для нахождения искомой записи в среднем $(NR + 1)/2$ обращений к блокам. Если удаление сводится к установке флага удаления записи, то потребуется только одна перезапись блока, содержащего удаляемую запись. Если операция удаления состоит в изменении указателя в предшествующей записи, то после его изменения потребуется только повторная запись этого блока.

Удаление записи в двусвязанном списке дополнительно требует произвольного обращения для чтения и последовательного обращения для перезаписи блока, содержащего последующую запись с целью модификации (изменения) указателя на предшествующую запись. Считается, что если любой блок считывается и (после некоторой обработки в процессоре) сразу записывается, то имеет место последовательный доступ. Все другие доступы считаются произвольными.

Модификация записи требует выполнения только одной перезаписи блока, содержащего эту запись, после того, как к ней

уже осуществлен доступ. Если требуется проверка записи блока, то это вызывает повторное считывание любого, только что записанного блока, и такой доступ считается последовательным.

В заключение перечислим факторы, которые надо учитывать при внесении изменений в данные, представленные в виде связанной последовательной организации:

- одно- или двунаправленность связи;
- тип операции: включение, удаление, изменение;
- упорядоченность или неупорядоченность списка;
- наличие в блоках свободного пространства, упрощающего включение записей;
- способ реализации переполнения при включении записи;
- необходимость контроля записи блока;
- способ реализации удаления записей: установлением флага удаления или указателей.

• **Пример 12-1.** Дан упорядоченный по возрастанию первичного ключа связанный последовательный файл с двунаправленными указателями, содержащий 100 записей. Включение записей требует расщепления блока и контроля правильности его записей. Чему равно среднее количество произвольных и последовательных обращений при включении новой и изменении существующей записи? Сравнить эти затраты с затратами на первоначальное размещение конкретной записи.

$$\text{PBA}_1 = \text{среднее количество обращений к блокам с целью поиска следующей по величине ключа записи}$$

$$= \frac{100+1}{2} = 50{,}5 \quad \text{rba}.$$

Предполагается, что в первичной памяти выделена достаточная для хранения предшествующего блока буферная область. Тогда для того, чтобы установить указатели в предыдущей и последующих записях не потребуется дополнительного чтения блоков. После расщепления эти два блока переписываются во внешнюю память.

$\text{PBA}_2 =$ количество обращений к блоку для перезаписи после включения новой записи

$= 1 \quad \text{rba} + 1 \quad \text{sba},$

$\text{PBA}_3 =$ количество обращений к блокам для контроля записи

$= 2 \quad \text{sba}.$

Следовательно, среднее общее количество обращений к блокам для включения новой записи равно

$$\text{PBA (ВКЛЮЧИТЬ)} = \text{PBA}_2 + \text{PBA}_3 = 1 \quad \text{rba} + 3 \quad \text{sba}.$$

Предположим, что удаление записи сопровождается модификацией указателей, тогда имеем

PBA_4 = количество обращений к блокам для удаления записи
$= 1 \quad rba + 2 \quad sba$

для перезаписи блока, содержащего предыдущую запись, плюс произвольное обращение для чтения и последовательное обращение для перезаписи блока, содержащего последующую запись, указатель которой должен быть модифицирован. Если для верификации блоков после удаления записи требуется $PBA_5 = 2\,sba$ обращений, получим оценку полных затрат, связаных с удалением записи:

$$PBA\ (УДАЛИТЬ) = PBA_4 + PBA_5 = 1 \quad rba + 4 \quad sba.$$

Изменение записи требует одного обращения для перезаписи изменяемого блока плюс одно обращение для его считывания:

$$PBA\ (ИЗМЕНИТЬ) = 2 \quad sba.$$

Очевидно, что основные затраты приходятся на поиск, причем они зависят от размера файла. Затраты, связаные с внесением изменений, после завершения поиска от размера файла не зависят. □

12.3.3. Выбор параметров при связанной последовательной организации данных

В этом разделе обсуждаются две проблемы: выбор указателей и влияние коэффициента загрузки на обработку переполнения. Каждая проблема требует нахождения компромисса между временем обработки и объемом памяти, а также между затратами на поиск и на обновление данных.

Выбор указателей

В разд. 12.3.2 было рассмотрено использование одно- и двунаправленных указателей при внесении изменений в списки. Для того чтобы оценить эти два варианта, можно сравнить для каждого типа операций выборки и обновления данных затраты, связанные с обращениями к блокам. Кроме того, можно сравнить требования к объему памяти. Эти оценки объединены в табл. 12.3. Из таблицы следует, что однонаправленные связи более эффективны при включении записей и требуют меньше памяти, однако двунаправленные связи предпочтительны при выполнении операции получения предшествующей записи ПОЛУЧИТЬ ПРЕДШЕСТВУЮЩУЮ (GET PRIOR), которая является полезной при выполнении операций поиска записей в обратном направлении. Операция удаления, которая требует

Таблица 12.3. **Количество обращений к блоку при обработке файла и затраты на память (в байтах) для связанной упорядоченной организации (без подтверждения правильности записи)**[1]

Операция	Однонаправленная связанная	Двунаправленная связанная
ПОЛУЧИТЬ ВСЕ	NR rba	NR rba
ПОЛУЧИТЬ УНИКАЛЬНУЮ (неважно, найдется или нет)	(1 + NR)/2 rba	(1 + NR)/2 rba
ПОЛУЧИТЬ СЛЕДУЮЩУЮ	1 rba	1 rba
ПОЛУЧИТЬ ПРЕДЫДУЩУЮ	Макс. = NR/2 rba	Макс. = 1 rba
ПОЛУЧИТЬ НЕКОТОРЫЕ	NR rba	NR rba
ПОЛУЧИТЬ ПАКЕТ	$\frac{RPBR \times NR}{RPBR + 1}$ rba + $+\left\lceil \frac{RPBR}{EBF_{tf}} \right\rceil$ sba	То же, что и в однонаправленном случае
ИЗМЕНИТЬ ПАКЕТ	Затраты на поиск + +RPBI (индивид. затр. на включение) + +RPBD (индивид. затр. на удаление) + +RPBC (индивид. затр. на изменение)	Тот же формат, что и в однонаправленном случае
Изменение после завершения поиска		
ИЗМЕНИТЬ (неключевой элемент)	1 sba	1 sba
ВКЛЮЧИТЬ (в тот же блок)	1 sba	1 sba
ВКЛЮЧИТЬ (нужен новый блок)	Макс. = 1 rba + 2 sba	1 rba + 1 sba
УДАЛИТЬ (установка флага)	1 sba	1 sba
УДАЛИТЬ (изменение указателей)	Мин. = 1 sba, макс. = = NR/2 rba + 1 sba	1 rba + 2 sba
Область памятн для одной хранимой записи	RS + ROVHD + PS байт	RS + ROVHD + + 2 × PS байт

[1] NR — количество записей в стеке (файле);
RS — длина логнческой запнсн (в байтах);
ROVHD — длина служебного поля хранимой записи (в байтах);
PS — длина указателя (в байтах).

изменения указателя, выполняется эффективно при однонаправленной связи, если последующая запись находится в буфере, куда она была помещена в процессе поиска. Если же она не содержится в буфере, то поиск приходится начинать заново.

Большинство операций над файлами, представленными в виде неупорядочеиных списков, имеют характеристики, аналогичные приведенным в табл. 12.3. Исключения составляют операции ПОЛУЧИТЬ УНИКАЛЬНУЮ для случая, если искомая запись не найдена, и ВКЛЮЧИТЬ для случая, когда для включения записи требуется новый блок. Затраты на выполнение этих операций показаны в табл. 12.4. В первом случае, когда

Таблица 12.4. **Количество обращений к блоку (PBA) для связанной неупорядоченной структуры**

Операция	Однонаправленная связанная	Двунаправленная связанная
ПОЛУЧИТЬ УНИКАЛЬНУЮ (запись не найдена)	NR rba	NR rba
ВКЛЮЧИТЬ (нужен новый блок)	Поиск = 1 rba, макс. = 1 rba + 1 sba	Поиск = 1 rba, макс. = 1 rba + 1 sba

искомая запись не найдена, имеет место поиск в неупорядоченном списке, который занимает миого времени (так как просматривается весь список). Операция ВКЛЮЧИТЬ осуществляется для неупорядочеиных списков проще, так как новую запись можно внести в начало списка. При этом не требуется модификации указателей предыдущей записи и время поиска для включения записи минимально. С другой стороиы, для пакетной обработки необходим упорядоченный список.

Другим важным фактором для многих систем является выбор указателя, обеспечивающего прямой доступ из каждой выбраиной текущей записи к ее исходной записи. В CODASYL, например, это обеспечивается указателем OWNER (ВЛАДЕЛЕЦ) и командой GET OWNER (ПОЛУЧИТЬ ВЛАДЕЛЬЦА). В системе IMS эти возможиости обеспечиваются либо символьными, либо адресными указателями (см. разд. 9.1). Указатели на исходную запись иснользуются во многих приложениях баз данных, в которых поиск осуществляется «снизу вверх» от экземпляра порожденной записи определеиного типа к единственному экземпляру ее исходной записи. Эти операции типичны для нерархических и в особенности для сетевых баз данных. В реляционных базах данных связи данного типа моделируются иначе. Их можно установить в рамках одного отношения или

между отдельными отношениями. Связь между отношениями можно установить путем использования последовательности команд СОЕДИНЕНИЕ (JOIN), ВЫБОР (SELECTION) и ПРОЕКЦИЯ (PROJECTION) реляционного языка; связь в рамках одного отношения может быть установлена путем применения последовательности команд ВЫБОР и ПРОЕКЦИЯ.

Как в случае с другими вариантами указателей, более быстрый доступ к определенным (в данном случае к исходным) записям достигается за счет выделения в каждом экземпляре порожденной записи дополнительной памяти. К счастью, использование указателя на исходную запись серьезно не влияет на другие операции поиска или обновления. Выделение дополнительной области под указатели несколько увеличивает объем передаваемых данных, но операции ИЗМЕНИТЬ и ИСКЛЮЧИТЬ не требуют для поддержания этих указателей дополнительных обращений к блокам. При включении нового экземпляра порожденной записи соответствующая ей исходная запись уже должна быть размещена в памяти с тем, чтобы можно было указать ее адрес в указателе порожденной записи. Если доступ к порожденной записи был осуществлен через ее исходную запись, то обычно при этом последняя находится в состоянии «текущая» и ее положение сохраняется. С другой стороны, если доступ к экземпляру порожденной записи осуществляется через непосредственно предшествующий ей экземпляр записи, то для поиска экземпляра исходной записи потребуется дополнительная операция поиска типа ПОЛУЧИТЬ УНИКАЛЬНУЮ. В этом случае следует модернизировать программное обеспечение так, чтобы исключить двойной поиск.

Коэффициент загрузки

Разновидности последовательной организации файлов, такие, как индексно-последовательная организация, позволяют избежать дорогостоящих операций переполнения, если первоначально предусмотреть выделение в блоках свободного пространства, достаточного для дальнейшего роста объема базы данных. Для физически последовательных файлов избежать переполнения можно путем периодического пакетирования изменений и перезаписи всего файла. При использовании связанной последовательной организации переполнение не приводит к значительному ухудшению производительности. Это можно увидеть из табл. 12.3 и 12.4, рассматривая различие в количестве обращений к блокам при различных вариантах включения новых записей. Для неупорядоченных списков в худшем случае снижение производительности при включении записи возникает за счет выполнения одного дополнительного произвольного обращения к блоку. Для упорядоченных списков в худшем случае

потребуется дополнительно одно произвольное и два последовательных обращения к блокам. Затраты на поиск конкретной записи, если искомая запись существует, являются одинаковыми для всех вариантов упорядочения и типов связи.

Выделение при первоначальной загрузке свободного пространства в блоках позволяет сократить количество дополнительных обращений к ним; таким образом, имеет место противоречие *время — память*. Для заданного коэффициента загрузки LF требуемый объем памяти вычисляется следующим образом:

$$\mathrm{NBLK} = \left\lceil \frac{\mathrm{NR}}{\mathrm{EBF}} \right\rceil, \tag{9-38}$$

$$\mathrm{BLKSTOP} = \mathrm{BKS} \times \mathrm{NBLK}, \tag{9-39}$$

где EBF — коэффициент полезного блокирования, рассчитанный по формуле (9.36). При уменьшении значения LF значение коэффициента EBF уменьшается, а NBLK увеличивается, т. е. при увеличении свободного пространства для размещения базы данных требуется большее количество блоков. Потери от увеличения требуемого объема памяти должны быть оценены в сравнении с выигрышем, полученным от сокращения количества обращений к блокам для внесения новых записей. Точная связь между коэффициентом загрузки и средними затратами от переполнения достаточно сложна; она зависит от распределения ключевых значений включаемых записей и их количества. Детальный анализ методов обработки переполнения приведен в разд. 13.2.

12.3.4. Объем памяти для последовательных структур

Объм памяти для физически последовательных и связанных последовательных структур рассчитывается по одной и той же формуле. В обоих случаях он является функцией размера хранимой записи (SRS). Для различных вариантов последовательных структур, таких, как физически последовательные, связанные однонаправленные, связанные двунаправленные и т. п., размер хранимой записи может значительно изменяться. Получение оценок объема требуемой памяти было рассмотрено в гл. 9. Применим изложенный там подход для расчета объема памяти, необходимого для хранения физических блоков. Расчет затрат памяти в других единицах измерения, таких, как количество дорожек и цилиндров, был проиллюстрирован в разд. 9.3:

$$\mathrm{BLKSTOR} = \mathrm{NBLK} \times \mathrm{BKS} = \lceil \mathrm{NR/EBF} \rceil \times \mathrm{BKS} =$$

$$= \left\lceil \frac{\mathrm{NR}}{\left\lfloor \frac{(\mathrm{BKS} - \mathrm{BOVHD}) \times \mathrm{LF}}{\mathrm{SRS}} \right\rfloor} \right\rceil \times \mathrm{BKS} \text{ байт}, \tag{9-38, 9-39}$$

где SRS =
- RS + ROVHD байт — для физически последовательных структур,
- RS + ROVHD + PS байт — для однонаправленных связанных последовательных структур,
- RS + ROVHD + 2 × PS байт — для двунаправленных связанных последовательных структур.

12.4. Общие затраты на получение отчета

Генерация отчета — это специальный случай поиска (ПОЛУЧИТЬ ВСЕ), когда выходные данные часто должны быть структурированы в виде, существенно отличном от первоначальной схемы. Основные операции: поиск, реструктурирование, сортировка и отображение результатов — выполняются последовательно, как показано на рис. 12.5. Операции поиска и отображения отчета являются обязательными, а операции реструктурирования и сортировки необязательны и зависят от структуры исходной и структуры конкретной схем отчета.

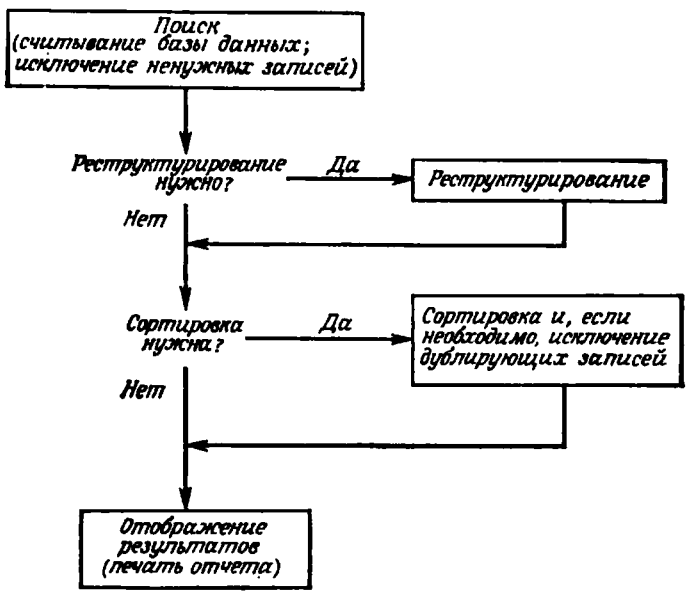


Рис. 12.5. Процесс получения отчета.

• **Пример 12.2.** Рассмотрим схему базы данных типа CODASYL, которая содержит данные о преподавателях, видах занятий и студентах (рис. 12.6, *а*). Предположим, что требуется получить следующие отчеты:

1. СПИСОК ВСЕХ СТУДЕНТОВ В АЛФАВИТНОМ ПОРЯДКЕ.
2. СПИСОК ВСЕХ ЗАНЯТИЙ ДЛЯ КАЖДОГО СТУДЕНТА С УКАЗАНИЕМ ПРЕПОДАВАТЕЛЯ.

Схема требуемого отчета приведена на рис. 12.6, *б*. Для того чтобы, исходя из схемы базы данных, составить схему отчета,

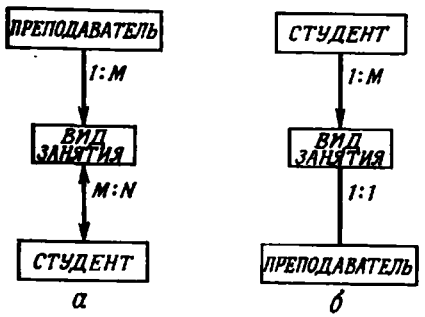


Рис. 12.6. Схема базы данных для генерации отчета.
а — схема базы данных; *б* — схема отчета.

должны быть выбраны все три типа записей. Первоначальная схема должна быть реструктурирована, чтобы соответствовать формату отчета, записи о студентах должны быть отсортированы по полному имени, и, наконец, отчет должен быть выведен на печать. Для того чтобы рассчитать время работы процессора и время ввода-вывода, проанализируем каждый шаг генерации отчета подробно.

Как обычно, для решения этой задачи выведем некоторые выражения для оценки времени ввода-вывода. Время работы процессора будет рассмотрено в разд. 16.4.

Поиск. В данном случае имеет место простой последовательный доступ, аналогичный уже изученному в этой главе. Допустим, что количество преподавателей 2500, количество занятий 5000 и количество студентов 35 000. Далее, предположим, что на каждом занятии присутствует в среднем 35 человек и что каждый студент посещает в день в среднем 5 занятий. Следовательно, имеется в общем $5 \times 35\,000 = 35 \times 5000 = 175\,000$ единичных актов посещений занятий студентами в день. На уровне схемы имеем:

$$\text{LRA (ПОЛУЧИТЬ ВСЕ)} = 2500I + 5000C + 175\,000S = 182\,500, \qquad (12\text{-}24)$$

где I — преподаватель, C — занятия и S — студент. На физическом уровне время ввода-вывода является функцией коэффи-

циента полезного блокирования:

$$\text{РВА (ПОЛУЧИТЬ ВСЕ)} = \left\lceil \frac{2500}{\text{EBF}(I)} \right\rceil + \left\lceil \frac{5000}{\text{EBF}(C)} \right\rceil + \left\lceil \frac{175\,000}{\text{EBF}(S)} \right\rceil \text{ sba.} \quad (12\text{-}25)$$

Для простоты предположим, что данные хранятся в иерархической последовательности. Это идеальный случай, тогда как в реальных задачах часто требуется осуществлять произвольный доступ к блокам. Если $\text{EBF}(I) = 20$, $\text{EBF}(C) = 50$, $\text{EBF}(S) = 10$, то уравнение (12-25) примет вид

$$\text{РВА (ПОЛУЧИТЬ ВСЕ)} = \left\lceil \frac{2500}{20} \right\rceil + \left\lceil \frac{5000}{50} \right\rceil + \left\lceil \frac{175\,000}{10} \right\rceil =$$
$$= 125 + 100 + 17\,500 = 17\,725 \text{ sba.}$$

Реструктурирование и сортировка. Эти операции тесно связаны между собой в том смысле, что алгоритм генерации отчета включает одновременно как реструктурирование, так и сортировку. В данном примере существует много различных вариантов преобразования исходной схемы в конкретную схему отчета; рассмотрим три из них.

1. Каждая найденная запись СТУДЕНТ должна быть записана в новую область в физической последовательности, необходимой для итогового отчета; при этом дублирующие записи удаляются. Предположим, что новые записи содержат комбинированную информацию о студентах, занятиях и преподавателях; этот новый вид записи обозначим буквой S'. Так как длина записи $S' > S$, можно положить $\text{EBF}(S') = 7$. Следует отметить, что этот подход рассматривает реструктурирование и сортировку как единую операцию с точки зрения обслуживания ввода-вывода.

Для реструктурирования и сортировки требуется 175 000 просмотров сортируемого файла, средняя длина которых равна 35 000/2 записям. Мы видим, что 20 % этих просмотров требуют внесения новых и сдвига оставшихся записей, 80 % просмотров (дубли записей СТУДЕНТ) требуют только включения новых данных о занятиях и преподавателях. Таким образом, имеем

$$\text{LRA (РЕСТРУКТ/СОРТ)} = 0{,}2 \times 175\,000\,(35\,000/2) +$$
$$+ 0{,}8 \times 175\,000\,(35\,000/4) = 1837{,}5 \times 10^6,$$

$$\text{РВА (РЕСТРУКТ/СОРТ)} = \left\lceil \frac{\text{LRA}}{\text{EBF}(S')} \right\rceil = \left\lceil \frac{1837{,}5 \times 10^6}{7} \right\rceil = 262{,}5 \times 10^6 \text{ sba.}$$

2. Каждая найденная запись СТУДЕНТ должна быть физически последовательно записана в новую область, при этом дуб-

лирующие записи удаляются, а требований к упорядоченности записей ие предъявляется. Затем требуется сортировка 35 000 записей СТУДЕНТ, помещеиных в эту область.

$$\text{LRA (ТОЛЬКО РЕСТРУКТ)} = 0{,}2 \times 175\,000\,(35\,000/4) + 0{,}8 \times 175\,000\,(35\,000/4)\,S' = 1531{,}4 \times 10^6 S',$$

$$\text{РВА (ТОЛЬКО РЕСТРУКТ)} = 1531{,}4 \times 10^6/7 = 218{,}75 \times 10^6 \text{ sba.}$$

Из уравиения (12-19) получаем оцеику количества обращений к физическим блокам для сортировки 35 000 записей (5000 блоков). Допустим, что сортировка осуществляется методом 6-путевого ($m = 6$) слияния:

$$m^X \geqslant \text{NBLK} = 5000, \quad X_{\min} = \lceil \log_6 5000 \rceil = 5,$$

$$\text{РВА (СОРТ)} = 2 \times \text{NBLK}(1 + X_{\min}) = 2 \times 5000 \times 6 = 60\,000 \text{ sba},$$

$$\text{РВА (РЕСТРУКТ/СОРТ)} = \text{РВА (ТОЛЬКО РЕСТРУКТ)} + \text{РВА (СОРТ)} = 218{,}81 \times 10^6 \text{ sba.}$$

Таким образом, в рассматриваемом вариаите затраты иа реструктурирование преобладают иад затратами на сортировку. Общие затраты меньше, чем для варианта 1.

3. Каждая найденная запись СТУДЕНТ по мере выборки должна быть физически последовательио записаиа в новую область, включая дублирующие записи. Затем выполняется сортировка 175 000 записей.

$$\text{LRA (ТОЛЬКО РЕСТРУКТ)} = 175\,000S',$$

$$\text{РВА (ТОЛЬКО РЕСТРУКТ)} = \left\lceil \frac{175\,000}{7} \right\rceil = 25\,000 \text{ sba.}$$

Для этого варианта имеем

$$m^X = \text{NBLK} = 25\,000, \quad X_{\min} = \lceil \log_6 25\,000 \rceil = 6,$$

$$\text{РВА (СОРТ)} = 2 \times \text{NBLK} \times (1 + X_{\min}) = 2 \times 25\,000 \times 7 = 350\,000 \text{ sba},$$

$$\text{РВА (РЕСТРУКТ/СОРТ)} = \text{РВА (ТОЛЬКО РЕСТРУКТ)} + \text{РВА (СОРТ)} = 25\,000 \text{ sba} + 350\,000 \text{ sba} = 375\,000 \text{ sba}.$$

Этот метод является несомненно иаиболее эффективным, и имеино он должен применяться в рассматриваемом примере.

Отображение результатов. Допустим, что S' записей уже подготовлено для выборки с целью получения заданного формата

отчета, тогда

$$\text{LRA (ОТОБРАЖЕНИЕ)} = 35\,000S',$$

$$\text{PBA (ОТОБРАЖЕНИЕ)} = \left\lceil \frac{35\,000}{7} \right\rceil = 5000 \quad \text{sba.}$$

Общие затраты на доступ к физическим блокам приведены в табл. 12.5. Из нее следует, что в данном примере и, возможно,

Таблица 12.5. **Затраты на генерацию отчета**

	LRA	PBA	
Выборка	$2500I + 5000C + 175\,000S$	17 725	sba
Реструктурирование	$175\,000S'$	25 000	sba
Сортировка	$7 \times 350\,000S'$	350 000	sba
Отображение результатов	$35\,000S'$	5 000	sba
Итого	$28\,425 \times 10^6$	397 725	sba

в других случаях первоначальные затраты на выборку данных составляют незначительную часть общих затрат на получение отчета. В этой главе рассмотрены хорошо известные выражения для оценки некоторых вариантов выборки, сортировки и отображения результатов. Выражения для общего случая оценки операций реструктурирования еще не известны, несмотря на активные усилия, предпринимаемые в настоящее время [73, 236, 311]. Однако большинство операций реструктурирования может быть легко выражено в виде последовательности операций поиска и обновления базы данных, и, таким образом, затраты на реструктурирование могут быть получены в виде суммы затрат на выполнение отдельных операций низкого уровня. □

Глава 13. Первичные методы доступа: произвольная обработка

Под *произвольной обработкой* принято понимать сочетание организации базы данных и способа выборки записи-цели приложениями пользователей, при котором следующая запись-цель оказывается расположенной физически произвольно по отношению к непосредственно предшествующей записи-цели. В одном случае данные, возможно, организованы (первоначально загружены) произвольным образом и никак не упорядочены. Вследствие этого последовательность команд ПОЛУЧИТЬ СЛЕДУЮЩУЮ (GET NEXT), основанная на упорядоченности данных по ключу, приводит к выполнению серии произвольных обращений к данным. В другом случае данные могут быть упорядочены и даже физически последовательны, но в последовательности команд выборки определены произвольно расположенные записи-цели, как, например, в случае простых незапланированных запросов. Само название команды выборки ПОЛУЧИТЬ УНИКАЛЬНУЮ (GET UNIQUE) подразумевает, что команда относится только к одиночной записи-цели. Если требуется получить несколько расположенных физически подряд записей, то их поиск можно организовать в виде двух команд: ПОЛУЧИТЬ УНИКАЛЬНУЮ (GET UNIQUE) для первой записи и ПОЛУЧИТЬ ВСЕ (GET ALL) для остальных записей.

В гл. 12 рассмотрены возможности применения последовательного и бинарного поиска для произвольного доступа к физически последовательным записям. В данной главе рассмотрено несколько альтернативных подходов. Показано, что ни один из подходов нельзя считать оптимальным для всех типов приложений базы данных, даже если ограничить их лишь операциями произвольного доступа; причина этого — в их различном влиянии на эффективность операций выборки, обновления или объема памяти.

13.1. Прямой доступ

Последовательный поиск одиночной записи в базе данных представляет собой один крайний случай механизмов поиска. Другим крайним случаем является прямой доступ к записи. Если проектировщик базы данных в состоянии предусмотреть в памяти для каждой записи место, определяемое уникальным значением ее первичного ключа, тогда можно построить про-

стую функцию преобразования ключа в адрес, обеспечивающую запомииаиие и выборку каждой записи в точности за один произвольный доступ к блоку. С этой целью каждой совокупности ключевых значений записей, расположеииых в одном и том же физическом блоке, можно присвоить отиосительный номер блока (относительный физический адрес). На самом деле данные хранятся в соответствии с порядком ключей, но при этом неявно им последовательно присвоеиы относительиые номера блоков.

На рис. 13.1 приведен простой пример, касающийся деятельности иекоторой компании с 30 отделениями; отделениям присвоеиы порядковые номера от 1 до 30. В каждом блоке хранится пять записей даиных об отделениях, так что всего требуется шесть блоков. Фуикция преобразования ключа в адрес:

$$\text{относительный адрес блока} = \left\lceil \frac{\text{НОМЕР-ОТДЕЛЕНИЯ}}{5} \right\rceil. \quad (13\text{-}1)$$

Например, запись данных об отделении номер 12 располагается в блоке с отиосительиым иомером 3. Такое абсолютное соответствие между ключом и отиосительиым адресом блока является основной отличительной чертой прямого метода доступа. Вследствие этого его использование наиболее целесообразно в тех случаях, когда можио управлять значениями ключей с целью минимизации потерь памяти. В рассмотрением примере полезное использование памяти составляет 100 %.

В случае реальных даиных не всегда возможны последовательные зиачения ключа, поэтому часто приходится строить более сложные функции преобразования. Предположим, например, что иекоторая компания изготовляет 2000 изделий с номерами соответственно от PB2000 до PB2999 и от QN500 до QN1499. В этом случае прямые адреса могут быть получеиы в соответствии со следующими правилами:

1. Если первые два зиака в номере изделия равны PB, то нужно из цифровой части иомера вычесть 1999.

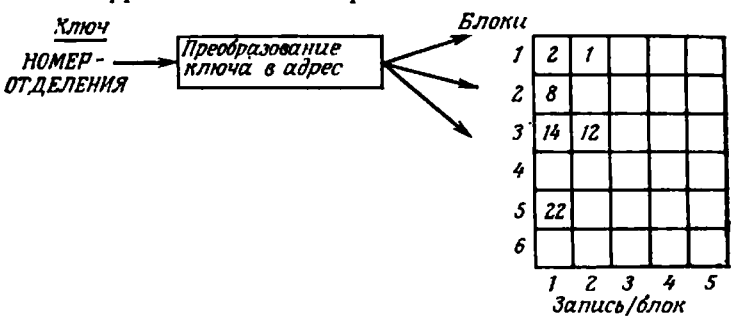


Рис. 13.1. Организация хранения и выборки записей даиных об отделеннях компании в методе прямого доступа.

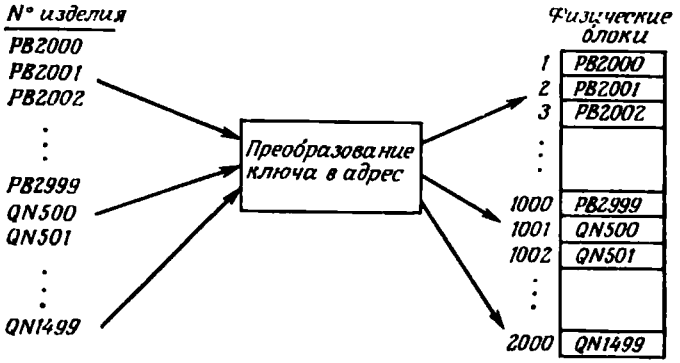


Рис. 13.2. Преобразование непоследовательных ключевых значений в методе прямого доступа.

2. Если первые два знака в номере изделия равны QN, то нужно к цифровой части номера прибавить 501.

В результате будут построены адреса в диапазоне от 1 до 2000, однозначно соответствующие номерам изделий. Этот случай показан на рис. 13.2.

До сих пор мы видели, что правильное применение прямого доступа оказывается чрезвычайно эффективным с точки зрения времени выборки данных и объема требуемой памяти. Любая операция выборки данных требует 1 rba (произвольного обращения к блоку), в то время как объем памяти зависит от количества возможных значений ключей. На выполнение преобразования ключа в адрес требуется минимальное время CPU, а простота операций обновления объясняется отсутствием ограничения на путь доступа (см. табл. 13.1).

Таблица 13.1. **Оценка операций базы данных (количество обращений к физическим блокам) для прямого доступа**

Операция	Прямой доступ
ПОЛУЧИТЬ ВСЕ, СЛЕДУЮЩУЮ, ПРЕДЫДУЩУЮ	Неприменима
ПОЛУЧИТЬ УНИКАЛЬНУЮ (неважно, найдена или не найдена)	1 rba
ПОЛУЧИТЬ НЕКОТОРЫЕ (булевый запрос)	Неприменима
После завершения поиска:	
ИЗМЕНИТЬ	1 sba
ВКЛЮЧИТЬ	1 sba
УДАЛИТЬ	1 sba

Если бы во всех приложениях существовала возможность управлять значениями ключей, то указанный метод доступа был бы распространен значительно шире, чем сейчас. Однако «неуправляемость» ключа является обычной проблемой, присущей большинству корпоративных баз данных. В качестве примера предположим, что в некоторой компании, насчитывающей 100 000 служащих, требуется в качестве первичного ключа записей данных о служащих использовать номер страхового полиса. Среди возможных 10^9 ключевых значений нельзя указать явные подмножества, которые можно было бы игнорировать; нельзя уменьшить объем требуемой памяти. Вследствие этого для хранения записей из каждых 10^4 будет использоваться только одно место в памяти, такое использование памяти крайне неэффективно.

Подобные же недостатки присущи использованию в качестве ключа полного имени служащего. Если под имя служащего отводится 20 знаков, то всего возможно 26^{20} ключевых значений [1]. Конечно, известно, что большая часть комбинаций букв не употребляется в качестве имени, но никто не возьмет на себя смелость выбрать и отбросить неупотребимые имена. Кроме того, неизбежны повторяющиеся имена, что не позволяет обеспечить уникальность адресов блоков, получаемых в результате преобразования ключей.

Объем памяти

$$\text{BLKSTOR} = \left\lceil \frac{\text{количество значений ключа}}{\text{EBF}} \right\rceil \times \text{BKS} \quad \text{байт,} \quad (13\text{-}2)$$

где каждое значение ключа соответствует одной выделяемой записи или фрагменту.

Адресация

Прежде чем завершить рассмотрение прямого доступа, необходимо пояснить смысл использованных понятий адресов. Значение первичного ключа, используемое в качестве входного параметра функции преобразования ключа в адрес, можно рассматривать как символический адрес. Значением функции преобразования является относительный физический адрес, который операционная система в свою очередь должна преобразовать в абсолютный физический адрес (НОМЕР-УСТРОЙСТВА, НОМЕР-ЦИЛИНДРА, НОМЕР-ДОРОЖКИ и т. д.) (см. рис. 13.3). Раздельное выполнение указанных преобразований повышает степень физической независимости данных, и, следовательно, изменение физического местоположения данных не предполагает использования программных средств адресации

[1] Количество букв в английском алфавите равно 26. — *Прим. ред.*

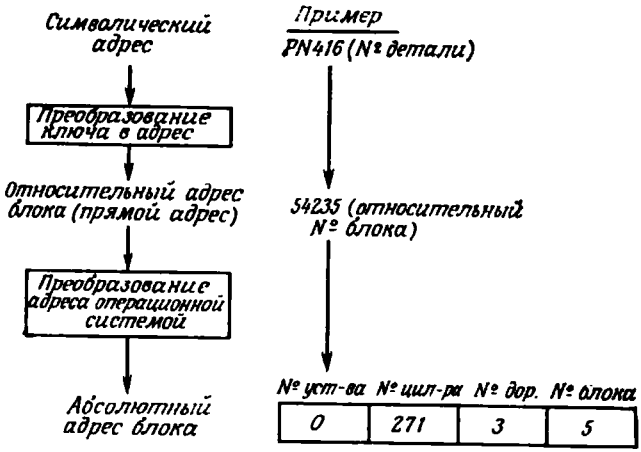


Рис. 13.3. Уровни адресации данных.

СУБД. Хранимые адреса, такие, как прямые указатели, часто представляют собой не абсолютные физические адреса, а относительные номера блоков (или страниц). Символические указатели представляют более высокий уровень адресации в сравнении с прямыми указателями, хотя тот и другой тип указателей может быть явно задан в записях.

В заключение отметим, что важными аспектами при проектировании файла прямого доступа являются:

• *Типы обработки.* Произвольная выборка (ПОЛУЧИТЬ УНИКАЛЬНУЮ) и обновление; физически последовательная выборка.

• *Функция преобразования ключа в адрес.* Влияет на время CPU.

Количество возможных значений ключа. Определяет размер базы данных.

• *Используемый уровень адресации данных.* Влияет на служебное использование CPU и степень независимости данных.

13.2. Хеширование идентификатора (произвольный доступ)

Метод хеширования идентификатора широко распространен как метод доступа, обеспечивающий быструю произвольную выборку и обновление записей. Было показано, что прямой метод доступа в сильной степени зависит от «управляемости» значений ключа и часто требует неоправданно больших служебных издержек памяти. Хеширование идентификатора обеспечивает эффективную выборку и обновление отдельных записей по за-

данному значению первичиого ключа. Платой за эту эффективность являются иарушеиие упорядоченности файла и потеря возможности выполиять пакетную обработку или генерацию отчетов, осиоваииую на упорядоченности записей по первичному ключу. Следует отметить, что допустим также метод хеширования вторичного ключа; все, о чем пойдет речь ниже, относится и к этому случаю.

Обсуждение метода хеширования идентификатора мы начнем с определения основных понятий. *Идентификатором* является атрибут, уникально определяющий каждый экземпляр некоторой сущности предметной области. В контексте базы данных или файла идентификатор представляет собой тип элемента данных, уникально идентифицирующий экземпляры коикретного типа записей (т. е. первичиый ключ). *Хешированием идентификатора* или просто *хешированием* называется метод доступа, обеспечивающий прямую адресацию данных путем преобразования значения ключа в относительный или абсолютный физический адрес. Функцию преобразования ключа часто называют также *функцией хеширования*. Альтернативный хешированию термин «произвольный (random) доступ» подсказывает, что для преобразоваиия, возможно, иеоднородиого множества значений ключа в однородное множество физических адресов используется иекоторый класс функций рандомизации. В прямом методе доступа рассматривается множество ключевых зиачений размером KS байт, определенных на словаре размером V, и для каждого из V^{KS} возможных ключевых значений требуется отдельный физический адрес [277]. Если в базе даиных в действительности используются только NR различных значений, то плотность идентификатора составляет NR/V^{KS}. В методе хешнрования предпринимается попытка построить равномерное отображение множества V^{KS} значений в множество $O(NR)$ физических адресов. В прямом методе доступа имеет место отображение 1 : 1, а в методе хеширования m : 1. При использовании функций рандомизации возможно преобразование двух или более значеиий ключа в одии и тот же физический адрес, так называемый *собственный* адрес. Такие ключи называют *синонимами*, а случай преобразования ключа в уже занятый собственный адрес называют *коллизией*. В прямом методе доступа синонимию ключей и коллизии избегают ценой больших служебных издержек памяти. При возникновении коллизии во время начальной загрузки базы данных или при включении в нее новой записи в процессе эксплуатации в методе хешироваиия приходится принимать решения о том, где хранить новую запись, содержащую ключ-синоиим.

На рис. 13.4 показан наиболее распространенный на практике способ организации хеширования. Все адресиое простран-

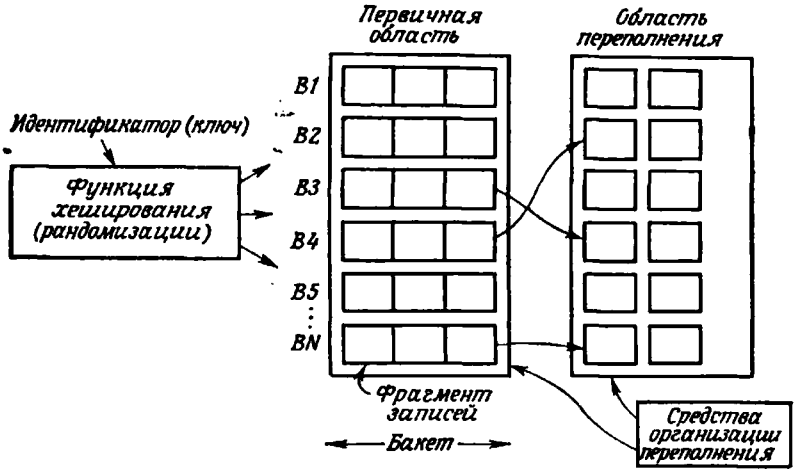


Рис. 13.4. Конфигурация метода хеширования (произвольного метода доступа).

ство, непосредственно доступное функции хеширования, делится на несколько областей фиксированного размера, называемых *бакетами*. В качестве бакета можно использовать цилиндр, дорожку, блок, страницу — любой участок памяти, адресуемый как одно целое. В свою очередь бакет может состоять из более мелких физических единиц данных, таких, как дорожка, блок (или страница) или хранимая запись. Наименьшая составная единица бакета, используемая при анализе производительности метода хеширования, называется *фрагментом* (хранимой) *записи данных* или *секцией*.

Процесс разрешения коллизий синонимов состоит из двух шагов. На первом шаге выполняется просмотр бакета с целью выявления в нем рядом с фрагментом первой записи свободного пространства для новой записи. При наличии свободного пространства просмотр прекращается. В противном случае должен быть осуществлен второй шаг — обработка переполнения. Условимся называть область памяти, содержащую бакеты, *первичной областью*, а оставшуюся часть доступной области памяти — *областью переполнения*. *Методы обработки переполнения* призваны обеспечить эффективное хранение записей переполнения. Оценка разных методов обработки переполнения представляет одну из основных задач данного раздела.

Прежде чем более подробно исследовать метод хеширования, отметим основные факторы, рассматриваемые при его проектировании:

Тип обработки. Только призвольная выборка и обиовление.

• *Распределение ключевых значений.* Влияет на распределение собственных адресов и количество синонимов.
• *Функция хеширования (рандомизации).* Влияет на распределение собственных адресов и количество синонимов.
• *Упорядоченность данных при начальной загрузке.* Влияет на общую производительность в случае использования метода открытой адресации; в случае метода цепочек влияния на производительность не оказывает.
• *Адресное пространство* (количество бакетов). Влияет на количество синонимов; позволяет также осуществлять изменение адресов для ключей, требующее модификации функции хеширования.
• *Размер бакета* (количество фрагментов записей). Обеспечивает гибкость обработки коллизий без использования области переполнения; по своему действию этот параметр подобен коэффициенту загрузки.
• *Коэффициент загрузки.* Оказывает влияние на вероятность возникновения переполнения.
• *Метод обработки переполнения.* Влияет на время обслуживания ввода-вывода для операций загрузки, выборки и обновления. Эти вопросы подробно рассматриваются ниже.

13.2.1. Функция хеширования

Наилучшей функцией хеширования является функция, отображающая NR значений ключа в точности в NR собственных адресов без синонимов. Теоретически существует NR! способов такого идеального отображения; однако, если учесть, что существует NR^{NR} способов присвоения NR ключам NR собственных адресов, вероятность этого идеального отображения ничтожна. Практический опыт подсказывает нам необходимость направить усилия на разработку удовлетворительных по производительности функций хеширования, преобразующих значения ключей в адреса, равномерно распределенные по всему адресному пространству.

В направлении конструирования эффективных функций хеширования была проведена большая исследовательская работа [188, 277, 279]. В качестве единиц измерения эффективности функции хеширования можно использовать такие характеристики, как время CPU на выполнение преобразования и время обслуживания ввода-вывода для доступа к бакету. Как правило, время CPU бывает мало в сравнении с временем ввода-вывода, требуемым для доступа к данным; необходимо подчеркнуть тот факт, что наиболее важным аспектом эффективности функции хеширования является ее способность рандомизировать значения ключа равномерно по всему адресному пространству. В про-

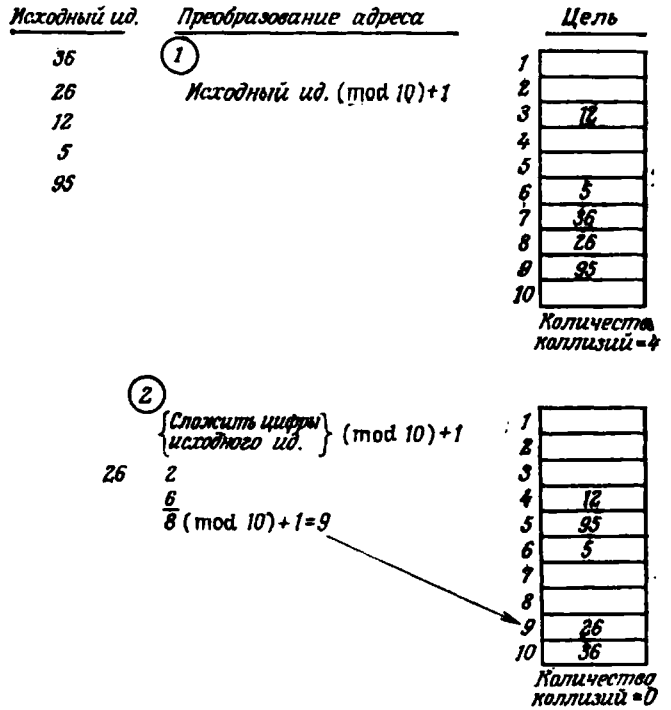


Рис. 13.5. Простейшая функция хеширования делением.

тивном случае часто имеют место синонимы (коллизии) и наблюдается спад производительности из-за необходимости выполнения поиска в области переполнения. К счастью, одна из простых функций преобразования, как было доказано, обеспечивает высокую производительность. Авторы работы [204] исследовали наиболее распространенные функции хеширования и доказали, что в силу зависимости распределения собственных адресов от распределения значений ключа (идентификаторов) ни один из методов преобразования нельзя назвать «наилучшим» с точки зрения рандомизации. Однако устойчивая производительность и простота метода хеширования делением и его вариантов позволяют рекомендовать их в качестве наилучших методов для обычных применений. Поясним на нескольких простых примерах понятие хеширования делением.

• **Пример 13-1.** Требуется для заданной последовательности значений ключа 36, 26, 12, 5, 95 и для адресного пространства из 10 бакетов построить функцию хеширования, не приводящую к коллизиям (см. рис. 13.5).

Если перенумеровать бакеты от 1 до 10 и применить простейшую функцию хеширования делением $f =$ идентификатор (mod 10) + 1, то окажется, что 36 и 26 являются синонимами, оба значения отображаются в бакет 7. Аналогично 5 и 95 являются синонимами и отображаются в бакет 6. Загрузка указанных пяти ключевых значений приводит по крайней мере к двум коллизиям. Однако если заменить функцию преобразования идентификатора на функцию «сумма цифр (mod 10) + 1»,

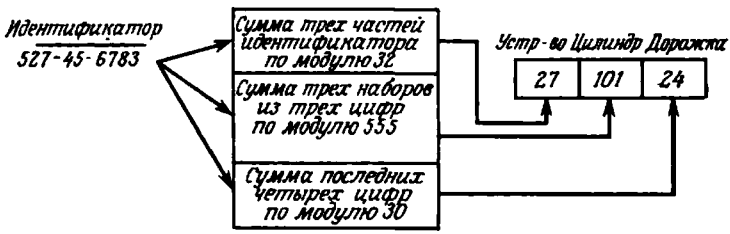


Рис. 13.6. Функция хеширования с абсолютными адресами.

то будут получены следующие пары вида «ключевое значение (собственный адрес)»: 36(10), 26(9), 12(4), 5(6) и 95(5). Данный подход представляет комбинацию методов хеширования сверткой (сложением цифр) и делением. Очевидно, что каждое значение ключа в этом случае отображается в уникальный адрес без коллизий. Хотя для некоторых последовательностей значений ключа, как, например, 36, 72, 27, 54, 63, эта функция может приводить к коллизии, в целом же для больших наборов ключевых значений она обеспечивает хорошую эффективность рандомизации. □

Заметим, что в рассмотренном примере в качестве физических адресов использованы относительные адреса блоков. В некоторых случаях, особенно когда хеширование выполняется низкоуровневыми программами операционной системы, целесообразно вычислять абсолютные адреса.

• **Пример 13.2.** Требуется построить функцию хеширования, которая отображает номер страхового полиса в адрес конкретной дорожки на диске. Предположим, что используются 32 НМД типа IBM 3350 и требуется равномерно распределить значения номера страхового полиса между ними.

Прежде всего отметим, что номер страхового полиса имеет вид ххх-хх-хххх, поэтому теоретически возможно 10^9 различных значений. Предположим, что к настоящему моменту задействованы только $0{,}2 \times 10^9$ номеров. Всю имеющуюся вторичную память можно представить в виде 32 устройств, $32 \times 555 = 17\,760$ цилиндров или $32 \times 555 \times 30 = 532\,800$ дорожек. Максимальная

емкость НМД 3350 составляет $10{,}14 \times 10^9$ байт, поэтому каждому из $0{,}2 \times 10^9$ элементов данных можно выделить порядка 50 байт. □

Использованный алгоритм представляет еще одну разновидность хеширования сверткой и делением, когда вычисляются различные суммы цифр и делятся на каждый из трех модулей: количество устройств, количество цилиндров и количество дорожек. На рис. 13.6 приведен пример применения указанного алгоритма к ключу 527-45-6783. Первая часть адреса равна $(527 + 45 + 6783) \bmod 32 = 27$, вторая часть адреса равна $(527 + 456 + 783) \bmod 555 = 101$ и, наконец, третья часть равна $(6 + 7 + 8 + 3) \bmod 30 = 24$. В проблеме подобного типа, когда невозможно описать адресное пространство как континуум посредством одной последовательности чисел, необходимо каждую часть адреса вычислять отдельно.

13.2.2. Методы обработки переполнения

Говорят, что имеет место переполнение, если требуется включить новую запись-синоним, а все фрагменты записей заданного бакета уже заполнены другими записями-синонимами. Выбор метода обработки переполнения представляет собой одно из важнейших проектных решений в произвольном методе доступа. Методику анализа этих способов можно также применять к проектированию других методов доступа, включая индексно-последовательный метод доступа. В работах [188, 279] рассмотрены основные методы обработки переполнения.

1. *Метод открытой адресации*

В этом методе запись переполнения заносится в первый неиспользованный («открытый») фрагмент записи следующего незаполненного бакета. Поиск незаполненного бакета осуществляется в блоках с последовательными адресами и завершается либо при обнаружении открытого фрагмента, либо бакета, с которого начинался поиск (т. е. имеет место цикл неуспешного поиска). Подобный вид поиска *называется линейным поиском или линейным опробированием.* Количество бакетов, которые приходится просмотреть, прежде чем запомнить запись, называется *смещением* записи. Когда группа записей преобразуется как синонимы, то наблюдается значительное увеличение смещения некоторых записей в сравнении с ожидаемым усредненным смещением. Аналогично при значении коэффициента загрузки, близком к 1,0, наблюдается быстрый рост среднего количества обращений к памяти для поиска записи. Следует отметить, что в данном методе обработки переполнения все записи переполнения помещаются в первичную область, а не в отдельную область переполнения.

2. *Метод нелинейного поиска*

Хотя метод открытой адресации, как правило, подразумевает применение линейного поиска, в действительности для поиска открытого фрагмента в первичной области можно использовать многие другие виды поиска. Во избежание проблем, связанных с кластеризацией и высокой плотностью записей при линейном поиске, были разработаны различные методы нелинейного (непоследовательного) доступа к бакетам. В методе рехеширования для поиска открытого фрагмента с целью включения новой записи или последующей ее выборки используется последовательность функций преобразования идентификатора [1]. В работе [223] первые исследования данного подхода связываются с В. А. Высотским. Еще один метод, известный под названием квадратичного поиска, был описан в работе [220]. В этом методе в случае возникновения коллизии для вычисления последовательных адресов используется квадратное уравнение вида $Ai^2 + Bi + C$ (по модулю, равному мощности адресного пространства), где i обозначает номер пробы, C — первоначальный собственный адрес начальной записи, а A и B — произвольные константы.

3. *Метод срастающихся цепочек*

В данном методе при начальной загрузке базы данных предусматривается специальная группа свободных бакетов с учетом возможности переполнения. В случае заполнения бакета A записями-синонимами запись переполнения помещается в незаполненный бакет (B), который связывается указателем с бакетом A. Все последующие записи переполнения бакета A помещаются в бакет B до тех пор, пока он не окажется заполненным. В последнем случае бакет B вычеркивается из списка свободного пространства. Затем в списке свободного пространства (в списке незаполненных бакетов первичной области) выполняется поиск нового бакета переполнения C. Хотя бакет B используется для записей переполнения бакета A, его адрес также может оказаться собственным адресом для других записей. При переполнении бакета B происходит сращивание цепочек синонимов бакетов A и B, и все последующие записи переполнения бакетов A и B будут помещаться в бакет C. Такая форма организации глобального переполнения характеризуется неоднородностью бакетов (или блоков) переполнения с точки зрения типов цепочек записей. В данном методе адреса бакетов переполнения при необходимости используются также в качестве собственных адресов записей, поэтому они по-прежнему рассматриваются как часть первичной области.

---

[1] Термин «rehashing» использован авторами в смысле, отличном от принятого в упоминаемой книге Кнута (см. [188], с. 619, 641). — *Прим. ред.*

4. *Метод раздельных цепочек*

В этом методе для каждого бакета первичной области выделяются локальные бакеты для записей переполнения. Это позволяет устранить перегруженность бакетов первичной области цепочкамн записей переполнения за счет возможных потерь памяти в тех бакетах переполнения, которые заполнены не полностью. Если впоследствии бакет, содержащий запись переполнения, потребуется для размещения записи, собственный адрес которой соответствует данному бакету, то для перемещения записи переполнения необходимы некоторые дополиительные служебные издержки. Подобных издержек можно избежать, если выделить все бакеты переполнения в отдельную область переполнения и запретить использованне их адресов в качестве собственных адресов запнсей. Определение отдельной области переполнения требует приняття проектных решений относительно использования бакетов, блоков или фрагментов записей.

На рис. 13.7 на примере простой базы данных проиллюстрированы основные методы обработки переполнения. По каждому из этих четырех методов записи переполнения размещаются поразному. Показанные на рисунке три характеристики операций выборки проясняют один важный момент: при оценке пронзводительности метода хеширования, исходя из количества обращений к записям, не учитывается влияние размеров бакета. Как известно, оценка производнтельности в виде количества обращений к блокам ближе к стандартной оценке производительности в виде времени обслуживания ввода-вывода, нежели оценка в виде количества обращений к записям. Например, в случае коэффициента блокирования первичной области, равного четырем, количество обращений к записям для включения иовой записи во всех четырех методах обработки переполнения останется прежним, но потребуются только два обращения к блокам. С другой стороны, взаимосвязь между количеством последовательных обращенн к блокам н размером блока для разных методов обработки переполнения различна. Вследствие этого невозможно дать сравнительную оценку производительности в виде времени обслуживания ввода-вывода только на основе колнчества обращений к записям, блокам нли бакетам. Достоверный сравнительный анализ может быть выполнен только в виде времени обслуживания ввода-вывода.

• **Пример 13-3.** Предположим, что для базы данных с рис. 13.7 время произвольного обращеиия к блоку (rba) на диске в среднем составляет 40 мс, а время последовательного обращения к блоку (sba) равно в среднем 10 мс. Требуется установить, в каком из методов обработкн переполнения время обслуживания ввода-вывода минимально для следующих случаев: 1) первичная область и область переполнеиия расположены на одиом

цилиндре; 2) каждый бакет является дорожкой на произвольном цилиндре, и область переполнения также расположена на произвольном цилиндре. Заметим, в что случае использования единственного цилиндра произвольное обращение эквивалентно по времени последовательному. Применив методику анализа из разд. 9.2 и используя оценки количества обращений рис. 13.7, получим оценки, приведенные в табл. 13.2. □

Таблица 13.2. **Оценка производительности методов обработки переполнения**

Метод обработки переполнения	Количество обращений к записям	Количество обращений к блокам	Количество обращений к бакетам	Время обслуживания ввода-вывода (1 случай), мс	Время обслуживания ввода-вывода (2 случай), мс
1. Открытая адресация	7	4	2	40	70
2. Нелинейный поиск	6	3	2	30	90
3. Срастающиеся цепочки	7	4	2	40	100
4. Раздельные цепочки	5	3	2	30	90

На рис. 13.8 графически представлены теоретические результаты, полученные Кнутом [188] и авторами работы [279] для некоторых ограниченных случаев. В качестве меры производительности для четырех основных методов обработки переполнения здесь использована оценка количества обращений к записям, а размер бакета ограничен одной записью. Записи выбираются с равной вероятностью. Хотя сравнительные значения количества обращений к записям нуждаются во внимательной оценке, наибольшего внимания заслуживает быстрый спад производительности методов открытой адресации и рехеширования при значениях коэффициента загрузки от 0,9 до 1,0. На рисунке показано, что при всех значениях коэффициента загрузки метод с раздельными цепочками устойчиво обладает довольно хорошей производительностью, в то время как методы открытой адресации и рехеширования при значениях коэффициента загрузки больше 0,9 дают низкую производительность. Вследствие того что в большинстве случаев каждое обращение, как правило, означает произвольный доступ к вторичной памяти, при значениях коэффициента загрузки больше 0,5 производительность метода рехеширования может оказаться значительно хуже в сравнении с методом открытой адресации. В табл. 13.4, 13.7 и 13.8 приведены экспериментальные результаты, полученные для более реальных размеров бакетов.

Помимо измерения времени обслуживания ввода-вывода в работе [279] показано, что программное обеспечение метода от-

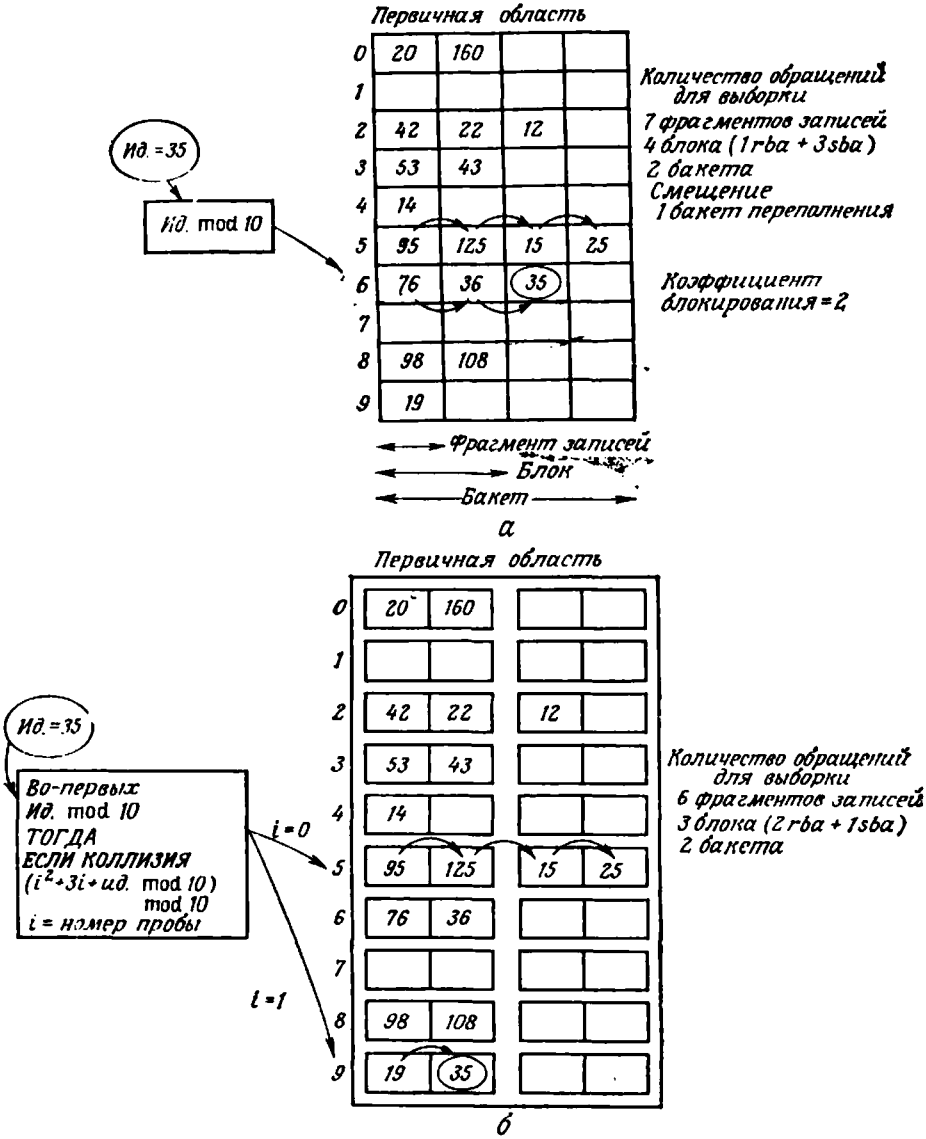
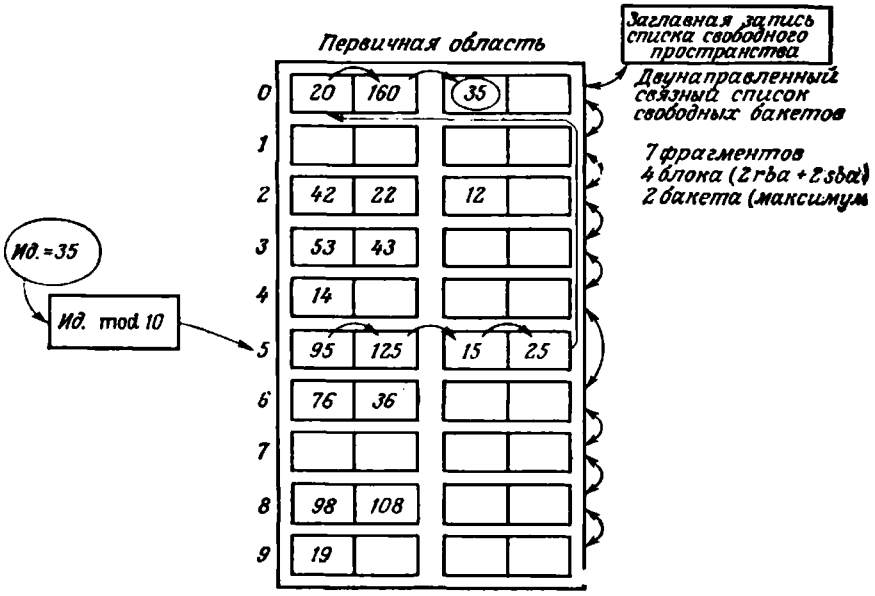
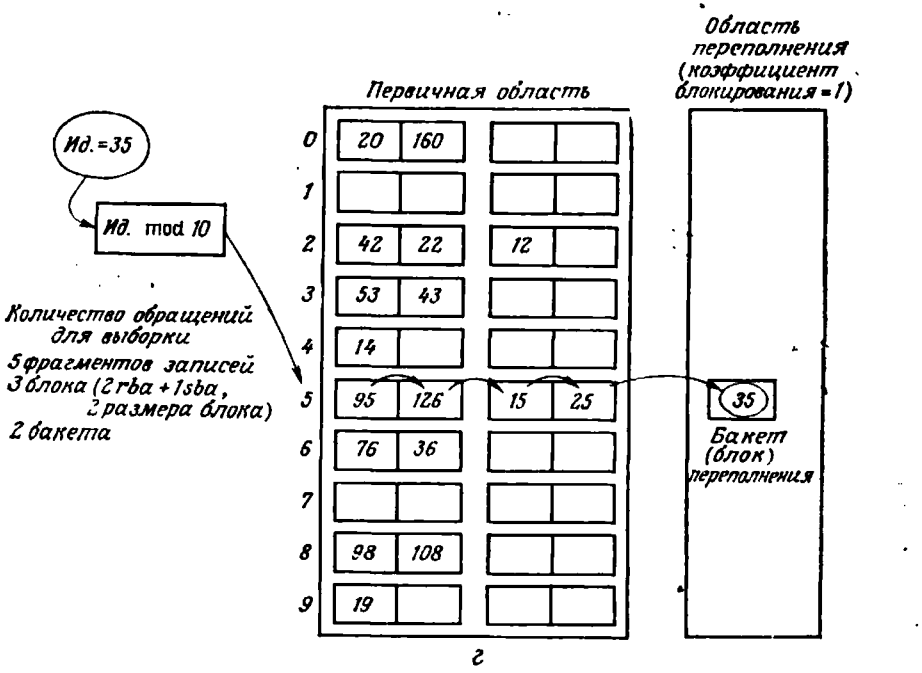


Рис. 13.7. Методы обработки переполнения для метода хеширования.
а — открытая адресация; *б* — нелинейный (квадратичный) поиск;



в



г

в — срастающиеся цепочки; *г* — раздельные цепочки.

крытой адресации менее сложно, т. к. в этом методе отсутствует необходимость хранения и обработки указателей. Благодаря этому уменьшаются затраты времени CPU на служебную обработку; влияние этого фактора на общую производительность системы может оказаться значительным. Авторы [279] также обнаружили, что, невзирая на отсутствие издержек памяти на указатели, в методе открытой адресации общий объем памяти необязательно минимален. В методе переполнения с цепочками в каждом бакете требуется только один указатель, что

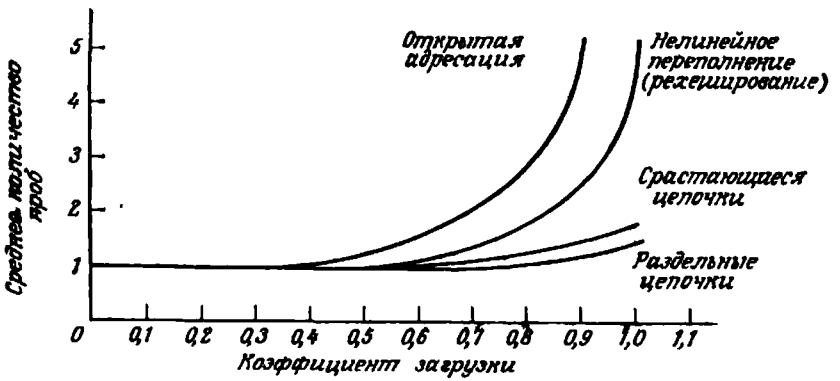


Рис. 13.8. Сравнение различных методов обработки переполнения для файла с большим количеством записей и размером бакета в одну запись [279].

составляет небольшие служебные издержки. К тому же коэффициент загрузки представляет более существенный показатель, а для обеспечения высокой производительности метода открытой адресации требуется относительно низкий коэффициент загрузки. Следовательно, для метода открытой адресации характерен компромисс между такими достоинствами, как простота реализации и экономия времени CPU, и такими недостатками, как увеличение времени обслуживания ввода-вывода и объема требуемой памяти. Относительная важность указанных компромиссных факторов, конечно, зависит от системных требований, видов обработки и вычислительного оборудования.

Для детальной оценки подобных компромиссов относительно производительности необходим более тщательный анализ методов обработки переполнения. За последние годы были разработаны аналитические и имитационные модели методов обработки переполнения; однако они значительно отличаются друг от друга по используемым исходным допущениям и критериям производительности. К тому же большинство из них слишком сложны, чтобы их здесь подробно изложить. Тем не менее можно рассмотреть основные полученные результаты и использовать эти

концепции при построении выражений оценки производительности для реалистичных конфигураций базы данных и критериев производительности, исследованных нами выше для всех методов доступа.

Если функция хеширования верно рандомизирует ключи, то это означает, что все ключи имеют равную вероятность отображения в любой собственный адрес (бакет):

$$P(\text{присвоенный бакет} = k) = 1/\text{NB} \quad \text{для} \quad 1 \leqslant k \leqslant \text{NB} \text{ бакетов.} \tag{13-3}$$

Такое предположение о характере рандомизации обладает двумя достоинствами; оно обеспечивает, во-первых, простоту анализа и, во-вторых, высокую производительность (т. е. большую вероятность получения записи-цели за одно обращение к блоку). В результате применения рандомизации получаем биномиальное распределение количества записей (X), отображаемых в один и тот же бакет:

$$P_{\text{NR, NB}}(X = r) = \binom{\text{NR}}{r} \left(\frac{1}{\text{NB}}\right)^r \left(1 - \frac{1}{\text{NB}}\right)^{\text{NR}-r}, \tag{13-4}$$

где NR обозначает количество записей, NB — количество бакетов. Для достаточно больших значений NR и NB, что, как правило, имеет место на практике, можно воспользоваться аппроксимацией Пуассона

$$P_{\text{RPB}}(X = r) = P(r) \simeq \frac{e^{-\text{RPB}}\text{RPB}^r}{r!}, \tag{13-5}$$

где RPB обозначает среднее количество записей на бакет (RPB = NR/NB). Это теоретическое распределение можно использовать для оценки вероятности переполнения и анализа различных методов обработки переполнения.

Используя метод обработки переполнения с раздельными цепочками, можно перейти к оценке среднего количества обращений к логическим записям, или количества проб, обеспечивающих успешный поиск записи в первичном бакете или в бакете переполнения. Во-первых, вероятность того, что бакет содержит не менее i записей, равна

$$\text{PRI} = \sum_{r=1}^{\infty} P(r) = \sum_{r=1}^{\text{NR}} P(r) \tag{13-6}$$

для достаточно большого NR, такого, что бесконечный ряд сходится с допустимым отклонением для некоторого $r < \text{NR}$. На практике этот ряд сходится довольно быстро, как показано ниже на примерах. Во-вторых, количество обращений к i-й записи в

бакете составляет

$$\mathrm{LRAB}_i = i \times \mathrm{PRI} = i \sum_{r=1}^{\mathrm{NR}} P(r), \tag{13-7}$$

а общее количество обращений к логическим записям для всех записей в бакете равно

$$\mathrm{LRAB} = \sum_{i=1}^{\mathrm{NR}} \mathrm{LRAB}_i = \sum_{i=1}^{\mathrm{NR}} \left[i \sum_{r=i}^{\mathrm{NR}} P(r) \right]. \tag{13-8}$$

В силу предположения о равномерности распределения NR записей по всем бакетам, среднее количество обращений к логическим записям для любого заданного бакета и любой записи равно

$$E[\mathrm{LRAB}] = \frac{\text{общее количество обращений к бакетам для всех записей}}{\text{среднее количество записей в бакете}} = \frac{\mathrm{LRAB}}{\mathrm{NR/NB}} = \frac{\mathrm{NB} \sum_{i=1}^{\mathrm{NR}} \left[i \sum_{r=i}^{\mathrm{NR}} P(r) \right]}{\mathrm{NR}} = \frac{\sum_{i=1}^{\mathrm{NR}} \left[i \sum_{r=i}^{\mathrm{NR}} P(r) \right]}{\mathrm{LF} \times \mathrm{RSPB}}. \tag{13-9}$$

где $\mathrm{LF} = \mathrm{NR}(\mathrm{NB} \times \mathrm{RSPB})$ и RSPB обозначает количество фрагментов записей на бакет.

• **Пример 13-4.** Пусть задана произвольная база данных большой размерности, такая, что $\mathrm{NR} = \mathrm{NB}$, $\mathrm{LF} = 1$, $\mathrm{RSPB} = 1$ и

Таблица 13.3. **Расчет производительности (количество проб) метода обработки переполнений с раздельными цепочками** [1]

r	$P(r)e^{\mathrm{RPB}}$	i	$\mathrm{PRI} \times e^{\mathrm{RPB}}$	PRI	LRAB_i
0	1,0	0	2,7183	1,0	0
1	1,0	1	1,7183	0,6321	0,6321
2	0,50	2	0,7183	0,2642	0,5284
3	0,1667	3	0,2183	0,0803	0,2409
4	0,0417	4	0,0516	0,0190	0,0760
5	0,0083	5	0,0099	0,0036	0,0180
6	0,0014	6	0,0016	0,0006	0,0036
7	0,0002	7	0,0002	0,00007	0,0005
8	0,000025	8	0,00003	0,00001	0,0001
9	0,0000027	9	0,000003	0,000001	0,00001
10	0,0000003	10	0,0000003	0,0000001	0,000001
					1,4996

[1] $\mathrm{LRAB} = 1{,}50$, $E[\mathrm{LRAB}] = 1{,}50/1{,}0 = 1{,}50$ обращения к бакетам.

RPB = 1. В табл. 13.3 представлен расчет среднего количества проб для произвольной записи. □

В предположении, что каждое обращение к переполнению означает отдельное обращение к бакету, получаем, что при RSPB = 1 каждая проба также представляет собой обращение к бакету. В случае RSPB > 1 для первых RSPB записей-синонимов, отображаемых в бакет, требуется только одно обращение к бакету. В общем случае бакет, в который отображаются RSPB + k записей, заполнен и имеет k записей или бакетов переполнения. Общее количество обращений к (RSPB + k)-й записи составляет $1 + k$ обращений к бакетам. Среднее количество обращений к бакетам для произвольной записи получается в результате подстановки в уравнения (13-7)—(13-9) вместо i функции количества обращений к бакетам для i-й записи BA_i:

$$E[BA] = \frac{\sum_{i=1}^{NR}\left[BA_i \sum_{r=i}^{NR} P(r)\right]}{LF \times RSPB}, \qquad (13\text{-}10)$$

$$\text{где } BA_i = \begin{cases} 1 & \text{для } i = RSPB, \\ 1 + i - RSPB & \text{для } i > RSPB. \end{cases}$$

Из уравнения (13-10) теперь легко получить оценки, представленные в табл. 13.4 и совпадающие с результатами, известными из других работ [188, 279].

Таблица 13.4. **Среднее количество обращений к бакетам для успешного поиска в методе обработки переполнения с раздельными цепочками**

	Коэффициент загрузки									
RSPB	0,2	0,5	0,7	0,9	1,0	1,1	1,5	2,0	3,0	5,0
1	1,10	1,25	1,35	1,45	1,50	1,55	1,75	2,00	2,50	3,50
2	1,02	1,13	1,24	1,36	1,43	1,50	1,82	2,25	3,17	5,10
3	1,01	1,08	1,18	1,32	1,40	1,49	1,90	2,50	3,83	6,70
4	1,00	1,05	1,14	1,29	1,38	1,48	1,98	2,75	4,50	8,30
5	1,00	1,04	1,12	1,27	1,37	1,48	2,07	3,00	5,17	9,90
10	1,00	1,01	1,06	1,21	1,33	1,50	2,49	4,25	8,50	17,90
20	1,00	1,00	1,02	1,15	1,31	1,55	3,33	6,75	15,17	33,90
50	1,00	1,00	1,00	1,08	1,29	1,71	5,83	14,25	35,17	81,90
100	1,00	1,00	1,00	1,04	1,28	1,97	10,00	26,75	68,50	161,90

Теперь распространим уравнения (13-9) и (13-10) на случай, когда или первичная область, или область переполнения, или

обе эти области организованы в блоки:

$$E[\mathrm{PBA}] = \frac{\sum_{i=1}^{\mathrm{NR}} \left[\mathrm{PBA}_i \sum_{r=i}^{\mathrm{NR}} P(r) \right]}{\mathrm{LF} \times \mathrm{RSPB}}, \qquad (13\text{-}11)$$

где PBA_i представляет собой дискретную функцию, значения которой приведены в табл. 13.5.

Таблица 13.5. **Количество обращений к физическим блокам при успешном поиске i-й записи бакета (случай выделенных ресурсов)**

i	PBA_i[1]
1	$1\ \mathrm{rba}_1$
2	$1\ \mathrm{rba}_1 + \left(\left\lceil \frac{i}{\mathrm{EBF}_1} \right\rceil - 1\right)$ sba
.	
.	
.	
RSPB	$1\ \mathrm{rba}_1 + \left(\left\lceil \frac{i}{\mathrm{EBF}_1} \right\rceil - 1\right)$ sba
RSPB + 1	$1\ \mathrm{rba}_1 + \left(\left\lceil \frac{\mathrm{RSPB}}{\mathrm{EBF}_1} \right\rceil - 1\right)$ $\mathrm{sba} + 1\ \mathrm{rba}_2$
RSPB + k	$1\ \mathrm{rba}_1 + \left(\left\lceil \frac{\mathrm{RSPB}}{\mathrm{EBF}_1} \right\rceil - 1\right)$ $\mathrm{sba} + \left\lceil \frac{i - \mathrm{RSPB}}{\mathrm{EBF}_2} \right\rceil \mathrm{rba}_2$

[1] rba_1 — произвольное обращение к бакету первичной области;
rba_2 — произвольное обращение к бакету переполнения от бакета первичной области или от некоторого другого бакета переполнения.

В случае если просмотрены все бакеты, связанные с данным бакетом первичной области, и запись-цель не обнаружена, поиск считается неуспешным. Так как первичная область имеет физически последовательную организацию, неуспешный поиск в ней завершается при обнаружении первого пустого фрагмента записей. Область переполнеиия имеет связанную последовательиую организацию; неуспешный поиск в области переполнения завершается при обнаружении последней записи — записи с нулевым значением указателя. Пусть $E[\mathrm{LRAB}']$ обозначает среднее количество проб в случае неуспешного поиска, тогда

$$E[\mathrm{LRAB}'] = \sum_{r=0}^{\mathrm{RSPB}-1} P(r)(r+1) + \sum_{r=\mathrm{RSPB}+1}^{\mathrm{NR}} P(r)\, r. \qquad (13\text{-}12)$$

Таблица 13.6. **Количество обращений к физическим блокам в случае успешного поиска для бакета с r записями-синонимами при использовании метода обработки переполнения с цепочками**

r	PBA_r
0	$1\ rba_1$
От 1 до RSPB − 1	$1\ rba_1 + \left(\left\lceil \frac{r+1}{EBF_1} \right\rceil - 1\right)\ sba$
RSPB + k для $k \geqslant 0$	$1\ rba_1 + \left(\left\lceil \frac{RSPB}{EBF_1} \right\rceil - 1\right)\ sba + \left\lceil \frac{r - RSPB}{EBF_2} \right\rceil\ rba_2$

Для расчета количества обращений к бакетам в случае неблокированных записей переполнения более полезно выражение вида

$$E[BA'] = \sum_{r=0}^{RSPB} P(r) + \sum_{r=RSPB+1}^{NR} P(r)(r - RSPB + 1), \quad (13\text{-}13)$$

а для расчета количества обращений к блокам в случае блокированной первичной области и (или) области переполнения — выражение вида

$$E[PBA'] = \sum_{r=0}^{NR} P(r)\, PBA_r, \quad (13\text{-}14)$$

где значения PBA_r определены в табл. 13.6. В результате применения уравнения (13-13) получаем значения в табл. 13.7, совпадающие со значениями в работе [188].

Таблица 13.7. **Среднее количество обращений к бакетам в случае неуспешного поиска для метода обработки переполнения с раздельными цепочками**

	Коэффициент загрузки									
RSPB	0,10	0,20	0,30	0,40	0,50	0,60	0,70	0,80	0,90	0,95
1	1,0048	1,0187	1,0408	1,0703	1,1065	1,1488	1,197	1,249	1,307	1,3
2	1,0012	1,0088	1,0269	1,0581	1,1036	1,1638	1,238	1,327	1,428	1,5
3	1,0003	1,0038	1,0162	1,0433	1,0898	1,1588	1,252	1,369	1,509	1,6
4	1,0001	1,0016	1,0095	1,0314	1,0751	1,1476	1,253	1,394	1,571	1,7
5	1,0000	1,0007	1,0056	1,0225	1,0619	1,1346	1,249	1,410	1,620	1,7
10	1,0000	1,0000	1,0004	1,0041	1,0222	1,0773	1,201	1,426	1,773	2,0
20	1,0000	1,0000	1,0000	1,0001	1,0028	1,0234	1,113	1,367	1,898	2,3
50	1,0000	1,0000	1,0000	1,0000	1,0000	1,0007	1,018	1,182	1,920	2,7

Только в методе обработки переполнения с цепочками используется область переполнения, полностью отделенная от пер-

вичной области. В других рассмотренных методах обработки переполнения в случае коллизии для запоминания новых записей используется первичная область. Следовательно, к этим методам неприменимы биномиальное распределение и аппроксимация Пуассона. Рассмотрим вкратце каждый из этих методов и, где можно, применим метод оценки в виде количества обращений к физическим блокам.

Открытая адресация

В работе [188] проводится анализ открытой адресации для случая «внутреннего» поиска (поиска в оперативной памяти):

$$E[\mathrm{LRA}] \simeq {}^1/_2\left(1 + \frac{1}{1-\mathrm{LF}}\right) \text{ в случае успешного поиска,} \quad (13\text{-}15)$$

$$E[\mathrm{LRA'}] \simeq {}^1/_2\left[1 + \left(\frac{1}{1-\mathrm{LF}}\right)^2\right] \text{ в случае неуспешного поиска.} \quad (13\text{-}16)$$

В том случае, когда имеется только один фрагмент записей в каждом бакете (RSPB = 1), для получения количества обращений к физическим блокам легко можно расширить уравнения (13-15) и (13-16):

$$E[\mathrm{PBA}] = E[\mathrm{LRA}]/\mathrm{EBF}_1 \text{ в случае успешного поиска,} \quad (13\text{-}17)$$

$$E[\mathrm{PBA'}] = E[\mathrm{LRA'}]/\mathrm{EBF}_1 \text{ в случае неуспешного поиска.} \quad (13\text{-}18)$$

Однако при RSPB > 1 каждый бакет может содержать несколько фрагментов записей, и выражения (13-17) и (13-18) неприменимы.

Среднее количество обращений к блокам для случая успешного линейного «внешнего» поиска (поиска во вторичной памяти) определяется так:

$$E[\mathrm{BA}] = 1 + t_{\mathrm{NB}}(\mathrm{LF}) + t_{2\mathrm{NB}}(\mathrm{LF}) + t_{3\mathrm{NB}}(\mathrm{LF}) + \ldots, \quad (13\text{-}19)$$

$$\text{где} \quad t_n(\mathrm{LF}) = e^{-n\mathrm{LF}} \sum_{n=0}^{\infty} \frac{(\mathrm{LF} \times n)^n}{(n+1)!}.$$

Этот ряд обладает быстрой сходимостью. Если в каждом бакете содержится одно и то же количество блоков (NBPB), то количество обращений к физическим блокам задается выражением

$$E[\mathrm{PBA}] = 1 \quad \mathrm{rba} + (E[\mathrm{BA}] - 1) \times \mathrm{NBPB} \quad \mathrm{sba} < E[\mathrm{PBA}] \leqslant$$
$$\leqslant 1 \quad \mathrm{rba} + E[\mathrm{BA}] \times \mathrm{NBPB}. \quad (13\text{-}20)$$

В результате применения уравнения (13-19) получаем значения, приведенные в табл. 13.8.

Таблица 13.8. **Среднее количество обращений к бакетам в случае успешного поиска для метода открытой адресации (метода линейных проб) [188]**

	Коэффициент загрузки									
RSPB	0,10	0,20	0,30	0,40	0,50	0,60	0,70	0,80	0,90	0,95
1	1,0556	1,1250	1,2143	1,3333	1,5000	1,7500	2,167	3,000	5,500	10,5
2	1,0062	1,0242	1,0553	1,1033	1,1767	1,2930	1,494	1,903	3,147	5,6
3	1,0009	1,0066	1,0201	1,0450	1,0872	1,1584	1,286	1,554	2,378	4,0
4	1,0001	1,0021	1,0085	1,0227	1,0497	1,0984	1,190	1,386	2,000	3,2
5	1,0000	1,0007	1,0039	1,0124	1,0307	1,0661	1,136	1,289	1,777	2,7
10	1,0000	1,0000	1,0001	1,0011	1,0047	1,0154	1,042	1,110	1,345	1,8
20	1,0000	1,0000	1,0000	1,0000	1,0003	1,0020	1,010	1,036	1,144	1,4
50	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,001	1,005	1,040	1,1

Метод нелинейного поиска

Данный метод был исследован только для случая RSPB = 1 [223, 277].

$E[\mathrm{LRA}] \simeq -1/\mathrm{LF} \times \log_e(1 - \mathrm{LF})$ в случае успешного поиска. (13-21)

В силу того, что каждая проба считается произвольным доступом к базе данных, $E[\mathrm{PBA}] = E[\mathrm{LRA}]$ rba.

Метод срастающихся цепочек

Данный метод также был исследован только для случая RSPB = 1 [188]:

$$E[\mathrm{LRA}] \simeq 1 + \frac{1}{8 \times \mathrm{LF}}(e^{2 \times \mathrm{LF}} - 1 - 2 \times \mathrm{LF}) + (1/4) \times \mathrm{LF}$$

в случае успешного поиска, (13-22)

$E[\mathrm{LRA}'] \simeq 1 + (1/4)(e^{2 \times \mathrm{LF}} - 1 - 2 \times \mathrm{LF})$ в случае неуспешного поиска. (13-23)

Подобно методу нелинейного поиска, здесь каждая проба также считается произвольным доступом, и $E[\mathrm{PBA}] = E[\mathrm{LRA}]$ и $E[\mathrm{PBA}'] = E[\mathrm{LRA}']$.

13.2.3. Характеристики производительности

Производительность выборки

В разд. 13.2.2 показано, что оценку операций произвольной выборки записей можно получить, рассматривая в качестве одиого пути доступа оба уровня организации данных: первичную организацию и организацию переполнения. Подставив в уравнение (13-11) уравнение (13-5), получим

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ, ключ найден)} = \frac{\sum_{i=1}^{NR}\left(\text{PBA}_i \sum_{r=i}^{NR} \frac{e^{-\text{RPB}}\text{RPB}^r}{r!}\right)}{\text{LF} \times \text{RSPB}}, \qquad (13\text{-}24)$$

где значения PBA_i заданы в табл. 13.5. Затем, применив уравнение (13-14), получим

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ', ключ не найден)} = \sum_{r=0}^{NR} \frac{e^{-\text{RPB}}\text{RPB}^r}{r!} \times \text{PBA}_r, \qquad (13\text{-}25)$$

где значения PBA_r заданы в табл. 13.6.

С методом хеширования непосредственно связана еще одна операция выборки: последовательная обработка первичной области и области переполнения. Поскольку данные не упорядочены, наиболее эффективным оказывается метод их последовательного просмотра:

$$\text{РВА (ПОЛУЧИТЬ ВСЕ)} = \text{NB} \times \left\lceil \frac{\text{RSPB}}{\text{EBF}_1} \right\rceil \text{sba} + \text{NBLKOV sba}, \qquad (13\text{-}26)$$

где NBLKOV обозначает количество блоков в области переполнения. В данном случае начальное произвольное обращение к бакету представляет несущественную часть общего поиска и игнорируется. Методом последовательного просмотра всех блоков можно также обрабатывать запросы булевого вида.

Для синонимов, соответствующих одному и тому же собственному адресу, необходимо предусмотреть команды типа ПОЛУЧИТЬ СЛЕДУЮЩУЮ:

$$\text{РВА (ПОЛУЧИТЬ СЛЕДУЮЩУЮ)} \simeq \simeq \frac{p}{\text{EBF}_1} \text{ sba} + (1-p) \times 1 \text{ rba}, \qquad (13\text{-}27)$$

где p обозначает долю синонимов, размещенных в первичной области, а $1-p$ соответственно долю синонимов в области пе-

реполнения:

$$p = \frac{\text{RPB} - \text{OVFLCH}}{\text{RPB}} = \frac{\text{RPB}_1}{\text{RPB}}; \qquad (13\text{-}28)$$

здесь OVFLCH обозначает среднюю длину цепочки переполнения и определяется по формуле

$$\text{OVFLCH} = \sum_{r=1}^{\text{NR}-\text{RSPB}} P(\text{RSPB} + r)\, r, \qquad (13\text{-}29)$$

а $P(\text{RSPB} + r)$ представляет функцию вероятности Пуассона для цепочки переполнения длиной r. $\text{RPB}_1 = \text{RPB} - \text{OVFLCH}$ означает среднее количество приписанных бакету в настоящий момент первичных записей данных.

Производительность обновления

В предыдущих разделах рассмотрены теоретические методы оценки операций обновления в случаях физически последовательной и связанной последовательной организаций памяти. Следовательно, оценка производительности обновления записей в методе хеширования не вызывает затруднений. Если операция обновления изменяет неключевое значение, то для перезаписи измененного блока требуется только одно дополнительное последовательное обращение к блоку (кроме предварительной операции выборки):

$$\text{РВА (ИЗМЕНИТЬ неключевое значение)} = 1 \quad \text{sba}. \qquad (13\text{-}30)$$

Аналогично при включении новой записи в первичную область помимо начальной операции выборки требуется выполнить только перезапись блока. P — вероятность этого действия. Включение записи в цепочку переполнения эквивалентно включению элемента в однонаправленный связанный список. Применяя к операции включения оценки из табл. 12.4, получаем

$$\text{РВА (ВКЛЮЧИТЬ)} = p \times 1 \quad \text{sba} + (1 - p)(1 \quad \text{rba} + 1 \quad \text{sba}). \qquad (13\text{-}31)$$

Подобным образом выполняется оценка операции удаления. В предположении, что удаление записи означает ее физическое удаление или установку в ней специального флажка, для удаления записи из первичной области требуется только перезапись одного блока. Для удаления записи из области переполнения достаточно изменить значение указателя в предшествующей записи. Если бы при этом предшествующая запись находилась в буфере, то в соответствии с результатами анализа из разд. 12.2 можно было бы использовать выражение вида

$$\text{РВА (УДАЛИТЬ)} = p \times 1 \quad \text{sba} + (1 - p) \times 1 \quad \text{sba} = 1 \quad \text{sba}. \qquad (13\text{-}32)$$

Операция обновления, включающая изменение значения ключа, означает удаление записи со старого места и включение ее на новое место в соответствии с новым значением ключа:

$$\text{РВА (ИЗМЕНИТЬ ключ)} = \text{РВА (УДАЛИТЬ)} + \text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ)} + \text{РВА (ВКЛЮЧИТЬ)}. \quad (13\text{-}33)$$

Общее количество обращений к блокам, требуемое для выборки и изменения значения ключа, равно РВА(ПОЛУЧИТЬ УНИКАЛЬНУЮ)+ РВА (ИЗМЕНИТЬ ключ).

Оценка памяти

Пространство вторичной памяти в произвольном методе доступа организовано в виде первичной области и области переполнения. В предположении, что EBF_1 и EBF_2 обозначают соответствующие коэффициенты блокирования и

$$EBF_i = \left\lfloor \frac{(BKS_i - BOVHD_i) \times LF_i}{SRS_i} \right\rfloor,$$

где $i = \begin{cases} 1 & \text{для первичной области,} \\ 2 & \text{для области переполнения,} \end{cases}$

получаем

$$\text{BLKSTOR (первичная область)} = (\text{количество бакетов}) \times (\text{количество блоков в бакете}) \times (\text{размер блока в байтах}) = NB\left\lceil \frac{RSPB}{EBF_1} \right\rceil \times BKS_1 \text{ байт}, \quad (13\text{-}34)$$

$$\text{BLKSTOR (область переполнения)} = NBLKOVFL \times BKS_2 \text{ байт}. \quad (13\text{-}35)$$

13.2.4. Таблица хеширования

В работе [233] для исследования принципа физической независимости базы данных в системах с виртуальной памятью использовано понятие таблицы хеширования. Было предложено ввести в организацию хеширования дополнительный уровень, на котором выполняется преобразование ключа в адрес таблицы, построенной из указателей действительных записей в первичной области данных. Достоинством введения дополнительного уровня является возможность выполнять загрузку файла данных в определенном порядке, продиктованном потребностями последовательной пакетной обработки (см. рис. 13.9). С другой стороны, вследствие независимости физического расположения записи данных от значения ключа (идентификатора) и его

адреса в таблице хеширования стала бы осуществимой физическая реорганизация файла данных без его полной перезагрузки. Кроме того, облегчилась бы работа с записями переменной длины, а также поиск записи по значениям нескольких идентификаторов. Недостатки использования таблицы хеширования очевидны; это дополнительная память для хранения таблицы хеширования (таблицы указателей записей), необходимость дополнительного доступа к блоку в операциях произвольной выборки и служебные издержки на обновление таблицы хеширования с целью обеспечения соответствующей упорядоченности «изменчивого» файла данных. Рассмотрим достоинства и недостатки применения таблицы хеширования более подробно. Прежде всего необходимо модифицировать построенную модель пути произвольного доступа (модель хеширования) и включить в нее таблицу хеширования (см. рис. 13.9). Предположим, что используется метод обработки переполнения с цепочками, согласующийся с предложенной в работе [233] моделью, хотя допустимы и другие методы, например метод открытой адресации. В табл. 13.9 собраны характеристики производительности операций ввода-вывода. Воспользуемся этой таблицей для сравнения производительности применения таблицы хеширования с основным вариантом метода хеширования, показанным на рис. 13.6. При использовании таблицы хеширования операция произвольной выборки во всех случаях более длительна, но благодаря упорядоченности файла данных повышается производительность пакетной обработки простых запросов и обновлений. В предположении, что после начальной загрузки операции вклю-

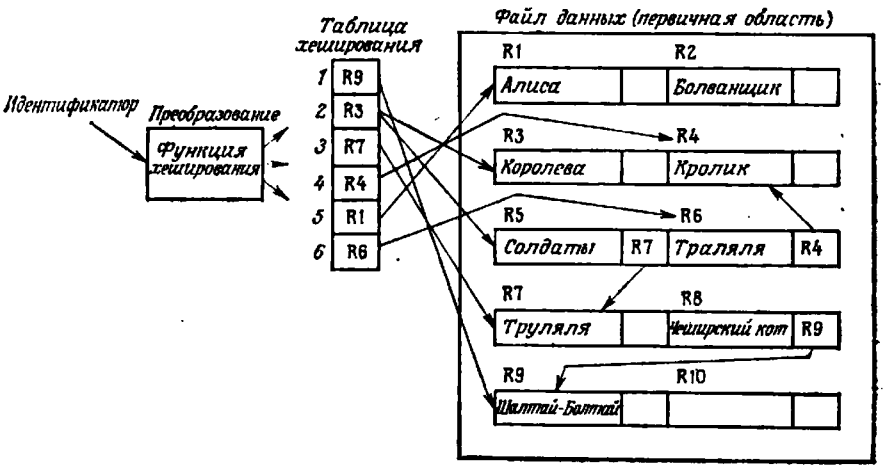


Рис. 13.9. Организация таблицы хеширования

Таблица 13.9. **Оценка операций базы данных (количество обращений к физическим блокам) для методов хеширования** [1]

Операция	Хеширование (произвольный доступ)	Доступ с использованием таблицы хеширования
ПОЛУЧИТЬ ВСЕ	$NB\left\lceil\frac{RSPB}{EBF_1}\right\rceil$ sba + + NBLKOV sba	Как для хеширования
ПОЛУЧИТЬ СЛЕДУЮЩУЮ	$p\times\frac{1}{EBF_1}$ sba + + (1 + p) rba	То же
ПОЛУЧИТЬ ПРЕДЫДУЩУЮ	Неприменима	Неприменима
ПОЛУЧИТЬ УНИКАЛЬНУЮ (ключ найден)	Уравнение (13-24)	ПОЛУЧИТЬ УНИКАЛЬНУЮ (хеширование) + 1 rba
ПОЛУЧИТЬ УНИКАЛЬНУЮ′ (ключ не найден)	Уравнение (13-25)	ПОЛУЧИТЬ УНИКАЛЬНУЮ (хеширование) + 1 rba
ПОЛУЧИТЬ RPBR записей (выборка пакета)	Неприменима	Как для физически последовательной организации (табл. 12.2)
ПОЛУЧИТЬ НЕКОТОРЫЕ (булевый запрос)	Как для операции ПОЛУЧИТЬ ВСЕ	Как для хеширования
После завершения поиска: ИЗМЕНИТЬ (неключевое значение)	1 sba	То же
ИЗМЕНИТЬ (ключевое значение)	ПОЛУЧИТЬ УНИКАЛЬНУЮ (ключ найден) + ВКЛЮЧИТЬ + УДАЛИТЬ	Возможна реорганизация файла данных и таблицы хеширования
ВКЛЮЧИТЬ запись	$p$sba + (1−p) (1 rba + + 1 sba)	Как для хеширования
УДАЛИТЬ запись	1 sba	То же
ОБНОВИТЬ ПАКЕТ (после выборки пакета)	Неприменима	Сумма оценок операций обновления для отдельных записей

[1] $p=\frac{RPB_1}{RPB_1+OVFLCH}=\frac{RPB_1}{RPB}$.

чения и удаления не затрагивают элементов таблицы хеширования, большая часть операций обновления в обоих методах доступа эквивалентна. В то же время, если требуется сохранить определенную физическую упорядоченность файла данных, изменение значения ключа в записи, доступ к которой осуществляется через таблицу хеширования, может повлечь за собой реорганизацию файла. Если же для файла данных предусмотрена возможность переполнения, то его реорганизацию можно отложить.

Применение таблицы хеширования вызывает увеличение объема требуемой памяти на величину, равную произведению размера указателя на количество элементов в таблице хеширования. Если же таблица хеширования организуется в виде блоков, то необходимо также учесть размеры соответствующих служебных полей.

13.3. Метод доступа с полным индексом (индексно-произвольный метод доступа)

Понятие индексирования нашло широкое применение в системах ручного доступа к данным, таких, как словарь или картотека каталога в библиотеке. В силу того, что это понятие предлагает в качестве альтернативы прямого доступа еще один подход к произвольной обработке данных, оно имеет непосредственное отношение к автоматизированным системам, которые призваны обеспечить эффективное хранение и выборку данных. Например, если данные хранятся физически последовательно, а доступ к ним осуществляется по командам типа ПОЛУЧИТЬ УНИКАЛЬНУЮ, то обычно оказывается более эффективно осуществлять поиск ключа-цели в файле, содержащем только значения первичных ключей, особенно в тех случаях, когда ключ по размеру значительно меньше самой записи данных. После того как в индексе обнаружено искомое значение ключа, доступ к записи можно осуществить с помощью указателя, который хранится в индексе рядом со значением ключа. В случае если в индексе не обнаружено искомое значение ключа, поиск завершается на уровне индекса. При этом не предполагается, что значения ключа как-либо упорядочены, хотя подобное упорядочение обеспечивает структуру, позволяющую применить к индексу бинарный поиск. Кроме того, допускается произвольный доступ к индексу посредством хеширования, как это реализовано в системе DMS-II [50].

Полный индекс представляет собой такую организацию файла, при которой для каждого конкретного экземпляра записи в файле или подфайле предусмотрен соответствующий элемент индекса. Этот элемент состоит из значения первичного ключа и

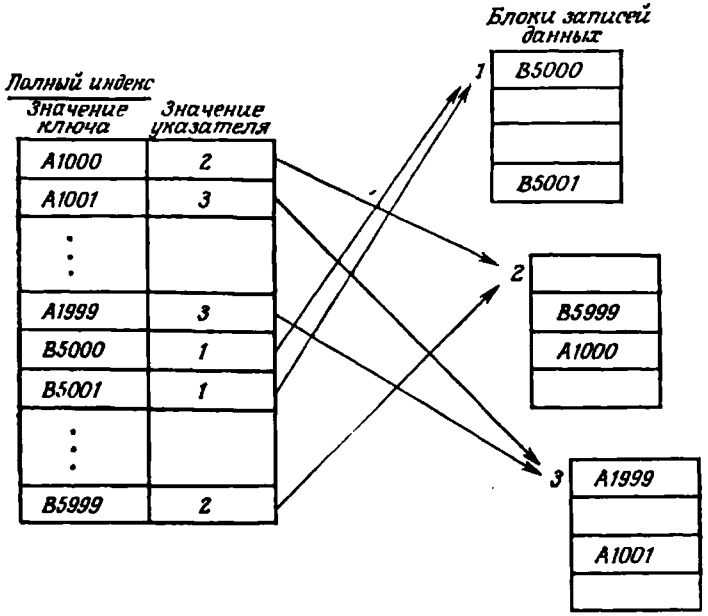


Рис. 13.10. Метод доступа с полным индексом.

указателя экземпляра записи, содержащей это значение. Обычно для ускорения поиска элементы индекса определенным образом упорядочиваются; при этом не требуется какое-либо упорядоченное или физически близкое размещение хранимых записей. Физически последовательное хранение записей представляет частный случай такой организации. На рис. 13.10 приведен пример элементов полного индекса. Термины «индексно-произвольный метод доступа» и «метод доступа с полным индексом» эквивалентны.

При организации файлов с полным индексом важны следующие моменты:

- *Тип обработки.* Произвольная выборка, произвольное обновление, последовательная обработка.
- *Коэффициент блокирования* (*индекса*, *данных*). Влияет на эффективность последовательной обработки и на время поиска индекса.
- *Коэффициент загрузки.* Обеспечивает возможность включения в индекс новых элементов без переполнения или переупорядочения индекса.
- *Размер файла* (*количество записей*). Влияет на объем требуемой памяти и размер индекса.
- *Упорядоченность записей данных и элементов индекса.*

Производительность выборки и обновления данных

Достоинством произвольного расположения записей данных является физическая независимость данных на уровне хранения записей и простота их обновления. Основной его недостаток проявляется в тех случаях, когда прикладная функция пользователя выдает команду ПОЛУЧИТЬ ВСЕ и при этом требуется, чтобы данные в отчете располагались в определенном порядке. В разд. 12.2 рассмотрены вопросы эффективной организации данных для команд типа ПОЛУЧИТЬ ВСЕ.

Произвольное расположение элементов индекса (каждый элемент можно рассматривать как «запись индекса») имеет те же достоинства и недостатки, что и для любого физически последовательного файла. Во-первых, в тех случаях, когда искомого ключа в индексе нет, приходится просматривать весь индекс. Во-вторых, возможен только последовательный поиск. С другой стороны, упорядоченность элементов индекса значительно повышает вероятность быстрого доступа для команд типа ПОЛУЧИТЬ УНИКАЛЬНУЮ и операций выборки пакета записей. К сожалению, полные индексы организованы плотно; для операций обновления, особенно для операций включения новых и изменения существующих ключевых значений, требуется выполнить чтение и перезапись всего индекса. Во избежание подобного переписывания индекса для операций удаления удаляемые ключевые значения можно просто пометить.

В табл. 13.10 приведены оценки основных операций для случаев упорядоченных и неупорядоченных индексов и записей базы данных. Вследствие того что полный индекс представляет собой частный случай физически последовательного файла, большую часть этих оценок можно получить из табл. 12.2.

Следует отметить, что общее количество обращений к блокам для одной операции базы данных равно сумме количеств обращений к блокам отдельно для индекса и для записей данных (т. е. сумма двух столбцов в табл. 13.10). Оценки для операций выборки получены на основании оценок производительности физически последовательной базы данных и применимы к полным индексам. Однако оценки для операций обновления нуждаются в дополнительном пояснении. Изменение неключевого значения выполняется посредством обращения к блоку, содержащему запись данных, собственно изменения значения, выполняемого в рабочей области оперативной памяти, и последующей записи измененного блока. В случае когда индекс упорядочен, изменение значения ключа выполняется еще более сложно. Независимо от того, вносится ли изменение в индекс или в блоки данных, оно выполняется посредством поиска и удаления старой записи (индекса или данных) и затем нового поиска и включе-

Таблица 13.10. **Оценка операций базы данных (количество обращений к физическим блокам) для организации с полным индексом** [1)]

Операция	Упорядоченный индекс	Неупорядоченные записи данных	Неупорядоченный индекс	Упорядоченные записи данных
ПОЛУЧИТЬ ВСЕ (без сортировки)	0	$\lceil NR/EBF_2 \rceil$ sba	0	$\lceil NR/EBF_2 \rceil$ sba
ПОЛУЧИТЬ СЛЕДУЮЩУЮ (по порядку)	$1/EBF_1$ sba	1 rba	$\left(\left\lceil \frac{NR}{EBF_1} \right\rceil + 1\right)/2$ sba	1 rba
ПОЛУЧИТЬ ПРЕДЫДУЩУЮ	$1/EBF_1$ sba	1 rba	Как для ПОЛУЧИТЬ СЛЕДУЮЩУЮ	1 rba
ПОЛУЧИТЬ УНИКАЛЬНУЮ (ключ найден)				
Последовательный поиск в индексе	$\frac{\left\lceil \frac{NR}{EBF_1} \right\rceil + 1}{2}$ sba	1 rba	Как для ПОЛУЧИТЬ СЛЕДУЮЩУЮ	1 rba
Бинарный поиск в индексе	$\log_2 \frac{NR}{2EBF_1}$ rba	1 rba	Неприменима	1 rba
Прямой поиск в индексе	1 rba	1 rba	Неприменима	1 rba
ПОЛУЧИТЬ УНИКАЛЬНУЮ' (ключ не найден)				
Последовательный поиск в индексе	$\frac{\left\lceil \frac{NR}{EBF_1} \right\rceil + 1}{2}$ sba	0	$\lceil NR/EBF_1 \rceil$ sba	0

Бинарный поиск в индексе	$\left\lfloor \log_2 \left(\frac{NR}{EBF_2} \right) \right\rfloor + 1$	0	Неприменима	0
Прямой поиск в индексе	1 rba	0	Неприменима	0
ПОЛУЧИТЬ НЕКОТОРЫЕ (булевый запрос)	0	$\lceil NR/EBF_2 \rceil$ sba	0	$\lceil NR/EBF_2 \rceil$ sba
После завершения поиска:				
ИЗМЕНИТЬ неключевое значение	0	1 sba	0	1 sba
ИЗМЕНИТЬ ключ	$2\left\lceil \frac{NR}{EBF_1} \right\rceil$ sba	1 sba	1 sba	$2\left\lceil \frac{NR}{EBF_2} \right\rceil$ sba
ВКЛЮЧИТЬ ключ и данные	$2\left\lceil \frac{NR}{EBF_1} \right\rceil$ sba	(1 rba + 1 sba)	(1 rba + 1 sba)	$2\left\lceil \frac{NR}{EBF_2} \right\rceil$ sba
УДАЛИТЬ ключ и данные	1 sba (если устанавливается флажок)	0	1 sba (если устанавливается флажок)	0

[1] EBF_1 — коэффициент полезного блокирования для записей индекса; EBF_2 — коэффициент полезного блокирования для записей данных; NR — количество записей в файле.

ния новой записи. Для этого требуется выполнить чтение и перезапись всего упорядоченного файла (индекса или данных).

Аналогично при включении новой записи данных подразумевается, что уже существует новое значение ключа, поэтому предварительно необходимо выполнить включение ключа. Для включения записи в упорядоченный (и плотный) файл требуется выполнить чтение и перезапись всего этого файла. В переписанном файле содержится на одну запись больше, чем в исходном, но предполагается, что одна запись не оказывает существенного влияния на производительность. Следовательно, для оценок производительности операций включения в упорядоченный файл (индекса или данных) и операций его обновления можно использовать одни и те же выражения. Для включения записи в неупорядоченный файл требуется (перед делением блока) выполнить последовательную запись старого блока, в котором теперь (в случае переполнения) содержится указатель нового блока переполнения, и затем выполнить произвольную запись блока. При отсутствии переполнения достаточно выполнить только последовательную запись старого блока. В разд. 13.2 показано, что существует много возможностей организации переполнения, каждую из которых необходимо проанализировать отдельно.

Операция удаления выполняется просто в предположении, что при удалении выполняется только пометка записей индекса. Удаление элемента индекса влечет за собой удаление записи данных, так как с ним удаляется указатель этой записи.

Оценка памяти

Оценка требуемого объема памяти для метода доступа с полным индексом равна сумме отдельных оценок для блоков индекса и блоков данных:

$$\begin{aligned}\text{BLKSTOR} &= \text{NBLK (индекс)} \times \text{BKS (индекс)} + \\ &+ \text{NBLK (данные)} \times \text{BKS (данные)} = \\ &= \left\lceil \frac{\text{NR}}{\left\lfloor \frac{(\text{BKS}_1 - \text{BOVHD}_1) \times \text{LF}_1}{\text{KS} + \text{PS}} \right\rfloor} \right\rceil \times \text{BKS}_1 + \\ &+ \left\lceil \frac{\text{NR}}{\left\lfloor \frac{(\text{BKS}_2 - \text{BOVHD}_2) \times \text{LF}_2}{\text{SRS}} \right\rfloor} \right\rceil \times \text{BKS}_2 \text{ байт,} \end{aligned} \qquad (13\text{-}36)$$

где подстрочный индекс 1 обозначает параметры файла индекса, а подстрочный индекс 2 — параметры файла данных.

13.4. Индексно-последовательный метод доступа

Прямой и произвольный методы доступа, а также метод доступа с полным индексом обеспечивают эффективные средства выборки данных для команд ПОЛУЧИТЬ УНИКАЛЬНУЮ

при некоторых ограничениях. Прямой метод доступа эффективен в тех случаях, когда ключи управляемы и есть возможность минимизации служебных издержек памяти. Произвольный метод доступа, как правило, более экономичен для конфигураций баз данных, требующих интенсивного произвольного доступа и обновления данных. Метод доступа с полным индексом обеспечивает эффективную выборку одиночных записей в случае неупорядоченного индекса, а также простоту операций обновления в случае неупорядоченных записей данных. Предположим, что ставится задача сконструировать такую организацию файла, которая обеспечивала бы и последовательную и произвольную обработку данных. Для решения этой задачи необходимо построить упорядоченный индекс и упорядоченные записи данных. Такую организацию файла можно реализовать с помощью метода полного индекса; однако в этом случае велики затраты на обработку ввода-вывода для операций обновления индекса и записей данных. Более того, в случае базы данных большой размерности может также оказаться чрезмерно велика оценка объема памяти, требуемой для полного индекса.

Индексно-последовательный метод доступа обеспечивает разумный компромисс между оценками затрат на выполнение операций выборки и обновления и объемом требуемой памяти. Этот метод эффективен для приложений с комбинированной последовательной и произвольной обработкой; в сравнении с методом полного индекса он позволяет уменьшить затраты на обновление и хранение индекса. Как показано ниже, он по-прежнему сохраняет неэффективность операций обновления в случае упорядоченных записей данных, которая присуща организации файла с полным индексом и физически последовательной организации, но в разрешении этой проблемы оказывает помощь применение новых методов обработки переполнения. В первую очередь рассмотрим возможность уменьшения объема памяти, требуемой для индекса, и времени последовательного доступа к данным за счет использования индекса блока. *Индекс блока* представляет собой упорядоченную таблицу значений первичных ключей, в которой каждый элемент содержит наибольшее значение ключа среди всех записей в указанном блоке. С каждым значением ключа в индексе связан указатель соответствующего блока. Структура индекса блока показана на рис. 13.11.

Использование упорядоченного индекса блока требует упорядоченности записей данных в файле. Благодаря этой упорядоченности можно в каждом элементе индекса указать границы соответствующего блока: значение ключа представляет верхнюю границу (т. е. наибольшее значение ключа в блоке), а указатель показывает нижнюю границу, так как задает адрес блока и тем

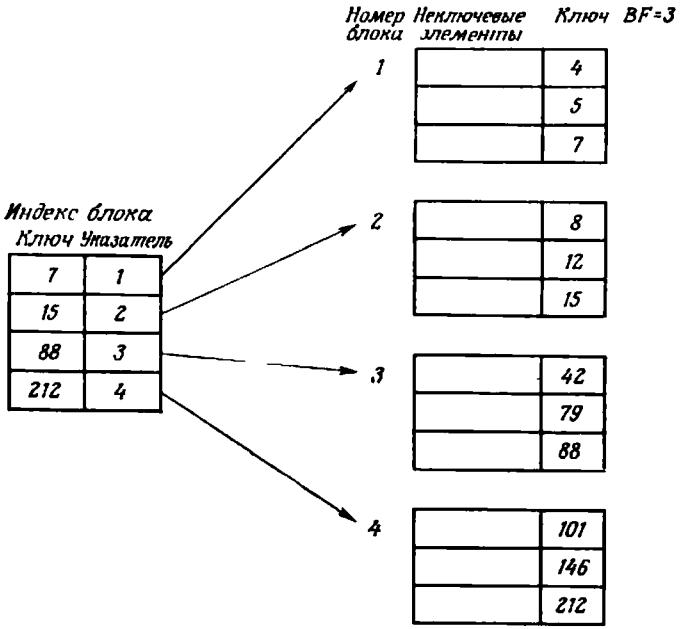


Рис. 13.11. Организация индекса блока (индексно-последовательная организация).

самым неявно задает адрес первой записи в блоке, содержащей наименьшее значение ключа.

Индексно-последовательный метод доступа строится на основе упорядоченного физически последовательного файла и иерархической структуры индексов блоков, каждый из которых упорядочен по значениям первичных ключей подобно записям в файле данных. Каждое значение ключа в индексе некоторого уровня j представляет собой наибольшее значение ключа в блоке индекса или блоке данных на уровне $j + 1$. На каждом уровне индекса и данных последовательный поиск выполняется до тех пор, пока не будет установлено местоположение искомой записи или ее отсутствие. Очевидно, что индексно-последовательный файл представляет собой обобщение понятия индекса блока, распространенное на несколько уровней. Благодаря этому обеспечивается быстрый доступ к записям баз данных и файлов большой размерности, в которых один индекс оказался бы слишком велик и не обеспечил бы эффективного поиска или обновления. Если бы база данных не менялась по размеру или если бы можно было включать новые записи в соответствующие блоки без использования переполнения, то можно было бы повысить

эффективность индексно-последовательного метода доступа за счет применения бинарного поиска для каждого блока индекса и данных. Для изменчивых файлов, как правило, требуются цепочки переполнения; отсюда потеря плотности и физической упорядоченности, препятствующая эффективному применению бинарного поиска. Вследствие этого в качестве стандартного механизма поиска для индексно-последовательных файлов используется последовательный поиск.

На рис. 13.12 приведен пример индексно-последовательной организации данных. В этом примере использованы три уровня индекса, один уровень для записей данных и один уровень для цепочек переполнения. Конкретная реализация индексно-последовательной организации может значительно отличаться от указанного примера по таким параметрам, как количество уровней индекса, длина поиска на уровень, тип поиска и метод обработки переполнения. Однако в основном архитектура индекса и его внутренняя структура согласуются с примером.

На рис. 13.12 представлена возможная реализация метода доступа ISAM фирмы IBM [169]. Уровень 1 соответствует главному индексу, который во время обработки файла может располагаться в оперативной памяти. На уровне 2 представлен индекс цилиндра, а на уровне 3 — индекс дорожки; и цилиндр, и дорожка могут содержать несколько блоков. На уровне 4 находятся записи данных, а записи переполнения занимают два последних уровня. На уровне индекса дорожки каждый указатель записи данных содержит адрес дорожки на соответствующем цилиндре.

В общем виде обработку переполнения можно осуществлять путем резервирования свободного пространства на уровне блока, дорожки, цилиндра или устройства. В каждом случае записи переполнения, как правило, организуются в виде цепочек и до реорганизации файла физически не переупорядочиваются. Конкретные детали обработки переполнения зависят от реализации.

Проектирование индексно-последовательного файла требует внимательного анализа следующих моментов:

- *Тип обработки.* Выборка, обновление, произвольные и пакетные операции, последовательная обработка.
- *Упорядоченность.* Ускоряет поиск, пакетную обработку; позволяет избежать сортировки в случае, если данные в отчете должны быть упорядочены подобно записям в файле.
- *Коэффициент блокирования (индекса и данных).* Влияет на эффективность последовательной обработки и объем памяти для хранения элементов индекса.
- *Размер файла (количество записей).* Влияет на объем памяти, необходимой для индексов.

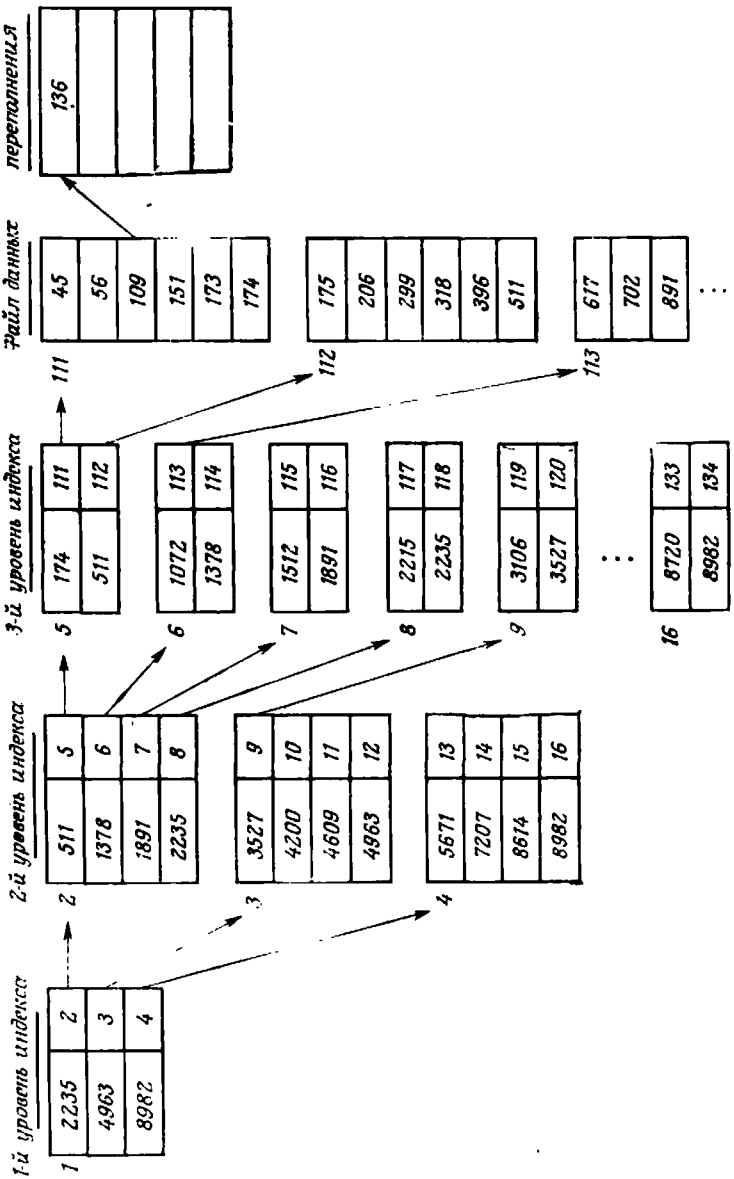


Рис. 13.12. Индексно-последовательный метод доступа с тремя уровнями индекса.

• *Коэффициент загрузки.* Обеспечивает возможность расширения файла без переполнения; влияет на требуемый объем памяти.
• *Уровни индекса.* Увеличение количества уровней индекса способствует уменьшению времени доступа к данным.
• *Размер индекса.* Влияет на время выборки и обновления элементов индекса.
• *Метод обработки переполнения.* Выбор подходящего метода препятствует снижению производительности операций выборки и обновления данных, уменьшает вероятность реорганизации файла.

13.4.1. Выборка данных из индексно-последовательного файла

Для доступа к данным индексно-последовательного файла выполняется последовательный просмотр элементов каждого уровня индекса, пока не будет обнаружено значение ключа, равное или большее искомого значения; в этот момент в соответствии со значением указателя осуществляется переход на следующий уровень индекса. На уровне данных поиск продолжается до тех пор, пока не будет найдена запись-цель или обнаружено ее отсутствие или же выявлена необходимость поиска среди записей переполнения. Расстояние между блоками данных и соответствующими им записями переполнения, а также способ хранения записей переполнения зависят от используемого метода обработки переполнения. Как правило, можно подобрать соответствующие значения для коэффициентов загрузки блока, дорожки и цилиндра или же некоторую их комбинацию. Считается, что внутри блоков новые записи располагаются подряд, друг за другом. Однако внутри области переполнения дорожки и области переполнения цилиндра записи обычно организованы в цепочки и не сблокированы. Следовательно, каждая запись связана с некоторой другой записью, но, поскольку обычно это происходит в пределах одного и того же цилиндра, для доступа к данным (rba) не требуется перемещение считывающей головки на диске. Возможно применение различных методов обработки переполнения, поэтому следует помнить, что для конкретного метода может потребоваться видоизменение рассматриваемой упрощенной модели.

Для представленного на рис. 13.12 примера конфигурации индекса и в предположении отсутствия переполнения оценка средней длины пути доступа для выборки произвольной записи равна сумме количеств обращений к блокам, включающей начальное обращение к уровню индекса или данных и последу-

ющий поиск внутри уровня:

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ, нет переполнения)} = n\,\text{rba} + \sum_{i=1}^{n}\left(\left\lceil \frac{(1+\text{NR}_i)}{\overline{\text{EBF}_i}} \right\rceil - 1\right)\text{sba}, \quad (13\text{-}37)$$

где $n = \text{NIL} + 1$, NIL обозначает количество уровней в индексе, а NR_i — количество записей (даниых нли элементов индекса) на уровне i. Первое слагаемое представляет количество обращений к блокам, соответствующих начальному обращению к каждому уровню индекса н данных; для получения общей оценки времени обслуживания ввода-вывода каждое из этих начальных обращений может потребовать отдельной оценки времени выполнения «rba». Второе слагаемое соответствует оцеике последовательного просмотра блоков на каждом уровне, выполняемого после начального обращения к уровню.

Для оценки пути доступа с учетом переполнения воспользуемся применительно к индексно-последовательному файлу моделью обработки переполнения с раздельными цепочками, построенной ранее для метода хеширования; при этом предположим, что каждый блок является блоком, имеющим собственный адрес, и записи переполнения для каждого блока организованы в отдельную цепочку. Для применения оценки, используемой в методе хеширования, прежде всего определим NR_0 как (начальное) количество записей в базе данных во время начальной загрузки. Обозначим через NR размер базы данных в текущий момент, а через $\text{NR} - \text{NR}_0$ количество включенных новых записей. Условимся, что в NR учтены также удаленные записи; в случае физического удаления записей NR соответственно уменьшено, в случае «пометки» удаленных записей NR неизменно.

На рис. 13.13 представлена конфигурация переполнения. После начальной загрузки произвольно включаемые записи в конечном счете заполняют первичный блок данных и вызывают переполнение. Первичный блок подобен бакету с количеством фрагментов записей, равным $\text{BF}_n - \text{EBF}_n$, а цепочка переполнения функционально эквивалентна цепочке переполнения в методе хеширования. Так как $\text{NR} - \text{NR}_0$ обозначает общее количество новых записей в базе данных, то, применив уравнение (13-11) к индексно-последовательной конфигурации, получим оценку среднего количества обращений к блокам переполнения:

$$E\,[\text{OVPBA}] = \frac{\sum_{i=1}^{\text{NR}-\text{NR}_0}\left[\text{PBA}_i \sum_{r=i}^{\text{NR}-\text{NR}_0} P\,(r)\right]}{\text{RPB}}, \quad (13\text{-}38)$$

где $\mathrm{RPB} = \dfrac{\mathrm{NR} - \mathrm{NR}_0}{\left\lceil \dfrac{\mathrm{NR}_0}{\mathrm{EBF}_n} \right\rceil}$ и обозначает среднее количество новых записей на блок. $P(r)$ определено уравнением (13-5), а

$$\mathrm{PBA}_i = \begin{cases} 0, & \text{если } 1 \leqslant i \leqslant \mathrm{BF}_n - \mathrm{EBF}_n, \\ k\,\mathrm{rba}, & \text{если } i = \mathrm{BF}_n - \mathrm{EBF}_n + k \text{ для } k \geqslant 0. \end{cases}$$

До тех пор пока записи помещаются в свободную область блока данных, дополнительные обращения к блокам не требуются. После того как произошло переполнение, каждое следующее

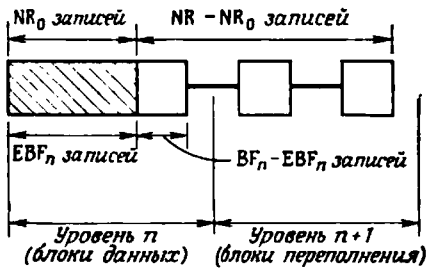


Рис. 13.13. Основная конфигурация переполнения в индексно-последовательном методе доступа.

новое обращение к блоку представляет собой произвольное обращение к области переполнения.

Теперь для получения общей оценки произвольного доступа в индексно-последовательной организации объединим уравнения (13-37) и (13-38):

$$\mathrm{PBA}\ (\text{ПОЛУЧИТЬ УНИКАЛЬНУЮ, ключ найден}) = n\,\mathrm{rba} + \sum_{i=1}^{n}\left(\left\lceil \frac{(1+\mathrm{NR}_i)}{\mathrm{EBF}_i} \right\rceil - 1\right)\mathrm{sba} + \frac{\sum_{i=1}^{\mathrm{NR}-\mathrm{NR}_0}\left[\mathrm{PBA}_i \sum_{r=i}^{\mathrm{NR}-\mathrm{NR}_0} P(r)\right]}{\mathrm{RPB}}. \quad (13\text{-}39)$$

Уравнение (13-39) относится к случаю успешного поиска. В случае неуспешного поиска кроме просмотра всех записей в блоке выполняется также просмотр всех записей переполнения:

$$\mathrm{PBA}\ (\text{ПОЛУЧИТЬ УНИКАЛЬНУЮ}', \text{ключ не найден}) = n\,\mathrm{rba} + \sum_{i=1}^{n}\left(\left\lceil \frac{(1+\mathrm{NR}_i)}{\mathrm{EBF}_i} \right\rceil - 1\right)\mathrm{sba} + \sum_{i=0}^{\mathrm{NR}-\mathrm{NR}_0} P(i)\,\mathrm{PBA}_i, \quad (13\text{-}40)$$

$$\text{где } \mathrm{PBA}_i = \begin{cases} 0, & \text{если } i \leqslant \mathrm{BF}_n - \mathrm{EBF}_n, \\ k, & \text{если } i = \mathrm{BF}_n - \mathrm{EBF}_n + k \text{ для } k \geqslant 0. \end{cases}$$

При выполнении команды ПОЛУЧИТЬ ВСЕ осуществляется прямой доступ к записям, минуя уровни индекса. Пренебрегая в оценке начальным доступом и эпизодическими доступами к новым цилиндрам, получаем

$$\text{PBA (ПОЛУЧИТЬ ВСЕ)} = \left\lceil \frac{\mathrm{NR}}{\mathrm{EBF}_n} \right\rceil \mathrm{sba} + \\ + \left\lceil \frac{\mathrm{NR}}{\mathrm{EBF}_n} \right\rceil E\,[\mathrm{OVPBA}]\,\mathrm{rba}. \quad (13\text{-}41)$$

Для выполнения команды типа ПОЛУЧИТЬ НЕКОТОРЫЕ также требуется полный просмотр файла, оценка которого дана в уравнении (13-41).

Высокой эффективностью отличается выборка пакета записей, основанная на упорядоченности файла транзакций, особенно в случаях незначительного переполнения. Подобно операциям произвольной выборки для поиска каждой записи-цели в пакете требуется минимум n обращений к блокам, однако при этом некоторые, а возможно, и все блоки уже находятся в буфере в результате выполнения предыдущего доступа. Следовательно, фактическое время ввода-вывода для индексно-последовательной выборки пакета записей может оказаться намного меньше, чем для функционально эквивалентной совокупности операций произвольной выборки (каждая из них довольно длительна по времени, поэтому вероятность того, что данные все еще находятся в буфере, незначительна), и, возможно, меньше, чем время выборки пакета из физически последовательного файла без индексов. Для пакетов большой размерности, может быть, лучше выполнять просто последовательный просмотр файла данных, пренебрегая поиском в индексе.

Нам хотелось бы оценить среднее количество обращений к индексу и блокам данных для выполнения выборки пакета, содержащего RPBR записей из общего количества NR записей, имеющих индексно-последовательную организацию. Предположим, что имеются два уровня индекса: индекс цилиндра и индекс дорожки. Обозначим общее количество элементов (записей) индекса цилиндра как NRC, а общее количество элементов (записей) индекса дорожки как NRT. Для определения количества блоков данных воспользуемся соотношением $\mathrm{NBLK} = \lceil \mathrm{NR}/\mathrm{EBF} \rceil$.

В интересной статье [347] приведено аналитическое выражение, позволяющее для операций выборки пакета специфицировать количество блоков, просматриваемых на каждом уровне индекса и данных. В предположении, что задано произвольное

распределение пакетных запросов, требуется оценить общее количество просмотренных блоков индекса и даниых. В худшем случае потребовалось бы RPBR обращений к блокам на уровне данных. Для простоты используем в теореме 13.1 следующие обозначения: $n = \text{NR}$, $m = \text{NBLK}$ и $k = \text{RPBR}$. Теорема касается проблемы производительности выборки без возвращения; применительно к пакетной обработке такое предположение естественно. Для оценки произвольной выборки с возвращением можно получить более простые выражения.

• **Теорема 13.1.** [347] Пусть заданы r записей, сгруппированных в b блоков ($1 \leqslant b \leqslant r$); каждый блок содержит r/b записей. Если произвольно выбрать k запнсей из r записей, то среднее количество блоков «успеха» (блоков, содержащих запись-цель) определяется по формуле:

$$\text{PBA (блоки успеха)} = b\left(1 - \prod_{i=1}^{k} \frac{rd - i + 1}{r - i + 1}\right), \text{ где } d = 1 - 1/b. \tag{13-42}$$

Доказательство. Пусть X есть случайная переменная, соответствующая количеству блоков успеха, и пусть I_j есть случайная переменная, такая, что $I_j = 1$, когда из j-го блока выбрана хотя бы одна запись, в противном случае $I_j = 0$. В j-м блоке размещены $p = r/b$ записей, а $r - p$ записей находятся вие j-го блока. Вероятность того, что из блока j не выбрана ни одна запись, равна C_k^{r-p}/C_k^r, нли C_k^{rd}/C_k^r, где $d = 1 - 1/b$. Отсюда следует, что математическое ожидание I_j равно $E[I_j] = 1 - C_k^{rd}/C_k^r$. Следовательно, математическое ожидание количества блоков успеха равно $E[X] = \sum_{j=1}^{b} E[I_j] = b(1 - C_k^{rd}/C_k^r)$. Используя тождество $C_y^x = x!/y!\,(x - y)!$, получим

$$E[X] = b\left[1 - \frac{(rd)!\,(r-k)!}{r!\,(rd-k)!}\right] = b\left(1 - \prod_{i=1}^{k} \frac{rd - i + 1}{r - i + 1}\right).$$

Следствие. Если $k > r - r/b$ или $b = 1$, то все m блоков являются блоками успеха. □

Теперь применим теорему 13.1 к анализу ожидаемого повышения производительности операций выборки пакета из RPBR записей в сравнении с совокупностью отдельных произвольных

выборок:

$$\text{PBA (выборка пакета, индексно-последовательная организация)} = \text{PBA (индекс цилиндра)} + \text{PBA (индекс дорожки)} + \\ + \text{PBA (записи данных)} + \text{PBA (переполнение)} + \\ + \text{PBA (файл транзакций)}. \quad (13\text{-}43)$$

Индекс цилиндра представляет собой индекс первого уровня. Он содержит

$$\text{NBLKC} = \lceil \text{NCYL}/\text{EBF}_1 \rceil \text{ блоков}. \quad (13\text{-}44)$$

Если в пакет включено RPBR произвольных записей, то среднее количество блоков, просмотренных в индексе цилиндра для обработки такого пакета, составляет

$$\left\lceil \frac{\text{NCYL}}{\text{EBF}_1} \times \frac{\text{RPBR}}{\text{RPBR}+1} \right\rceil.$$

Произвольная выборка одной записи включает в себя произвольный доступ к первому блоку индекса цилиндра и затем последовательный просмотр остальных блоков приблизительно в половине индекса. В силу того, что каждое обращение к записи данных не зависит от предыдущего обращения, указанная процедура повторяется RPBR раз. Иное дело при выборке пакета. Вследствие того, что, как правило, по крайней мере несколько последних из просмотренных блоков индекса цилиндра хранятся в буфере, а пакет упорядочен в соответствии с физическими адресами блоков, обращение к каждому блоку выполняется не более одного раза. Если индекс цилиндра и блоки данных размещены на разных устройствах (т. е. случай выделенного устройства), то каждое обращение к блоку, за исключением первого, представляет собой обыкновенный последовательный доступ. Если же индекс цилиндра и блоки данных размещены на одном и том же устройстве (т. е. случай разделяемого устройства), то каждое обращение к блоку представляет собой произвольный доступ. На это отличие указано в гл. 9, поэтому условимся рассматривать случай выделенного устройства и получим следующую оценку общего количества обращений к блокам индекса цилиндра для всего пакета:

$$\text{PBA (ПОЛУЧИТЬ ПАКЕТ, индекс цилиндра)} = 1\,\text{rba} + \\ + \left(\left\lceil \frac{\text{NCYL}}{\text{EBF}_1} \times \frac{\text{RPBR}}{\text{RPBR}+1} \right\rceil - 1 \right) \text{sba}. \quad (13\text{-}45)$$

Индекс дорожки представляет собой индекс второго уровня. Каждый индекс дорожки содержит элементы, описывающие дорожки данных для соответствующего цилиндра, а также, возможно, выделенные дорожки переполнения. Имеется

NCYL индексов дорожки, по одному для каждого элемента индекса цилиндра. Каждый индекс дорожки состоит из $\lceil (\text{NTRK}/\text{NCYL})/\text{EBF}_2 \rceil$ блоков. Следовательно, общее количество блоков во всех NCYL индексах дорожки равно

$$\text{NBLKT} = \left\lceil \frac{\text{NTRK}/\text{NCYL}}{\text{EBF}_2} \right\rceil \times \text{NCYL}. \qquad (13\text{-}46)$$

В соответствии с теоремой 13.1 среднее колнчество просмотренных блоков индекса дорожки, необходимое для выборки произвольных RPBR записей (без возвращения), составляет

$$\text{NBLKT}\left(1 - \prod_{i=1}^{\text{RPBR}} \frac{\text{NTRK} \times d_1 - i + 1}{\text{NTRK} - i + 1}\right),$$

где $d_1 = (\text{NBLKT} - 1)/\text{NBLKT}$.

В силу того что пакет упорядочен в соответствии с физическими адресами блоков, доступ к каждому блоку осуществляется только один раз; кроме того, предполагается, что каждый индекс дорожки находится в буфере до тех пор, пока не будут просмотрены все описываемые им записи. Каждый доступ к блоку расценивается как произвольный, так как он по определению не является последовательным, за исключением случая, когда пакет по размеру соизмерим с файлом. Однако расстояние установки считывающих головок при произвольном доступе к последовательным записям пакета невелико благодаря его упорядоченности, а время обслуживания ввода-вывода значительно меньше, чем в случае произвольного доступа в пределах всего устройства. Среднее расстояние установки считывающих головок при произвольном доступе к записям пакета составляет NCYL индексов дорожки/(RPBR − 1) цилиндров. Используя это значение и методы расчета из разд. 9.2, можно вычислить среднее время обслуживания ввода-вывода. Однако сейчас мы ограничимся только количеством обращений к физическим блокам.

$$\text{PBA (ПОЛУЧИТЬ ПАКЕТ, индексы дорожки)} = 1\,\text{rba} + \left\lceil \text{NBLKT} \times \left(1 - \prod_{i=1}^{\text{RPBR}} \frac{\text{NTRK} \times d_1 - i + 1}{\text{NTRK} - i + 1}\right) \right\rceil - 1\,\text{rba}_2, \qquad (13\text{-}47)$$

где $d_1 = \dfrac{\left\lceil \frac{\text{NTRK}}{\text{EBF}_2} \right\rceil - 1}{\left\lceil \frac{\text{NTRK}}{\text{EBF}_2} \right\rceil}$, а rba_2 обозначает произвольный доступ с другим средним расстоянием установки считывающих головок (отличным от CPD или NCYL), которое также необходимо учесть.

Применяя теорему 13.1 еще раз, можно вычислить среднее количество обращений к блокам для RPBR записей данных в пакете:

$$\text{PBA (ПОЛУЧИТЬ ПАКЕТ, записи данных)} = 1\,\text{rba} + \left\lceil \text{NBLK} \times \left(1 - \prod_{i=1}^{\text{RPBR}} \frac{\text{NR} \times d_2 - i + 1}{\text{NR} - i + 1}\right)\right\rceil - 1\,\text{rba}_3, \quad (13\text{-}48)$$

где $d_2 = (\text{NBLK} - 1)/\text{NBLK}$, а rba_3 обозначает ограниченый произвольный доступ, для которого среднее расстояние установки считывающих головок составляет NCYL записей данных/(RPBR — 1).

Далее выполняется доступ к цепочкам переполиения. В предположении, что среднее количество обращений к блокам переполнения равно E[OVPBA] и что, как правило, каждая следующая запись в цепочке физически расположена не рядом с записью, предшествующей ей в цепочке, получаем:

$$\text{PBA (ПОЛУЧИТЬ ПАКЕТ, переполнение)} = \left\lceil \text{NBLK} \times \left(1 - \prod_{i=1}^{\text{RPBR}} \frac{\text{NR} \times d_2 - i + 1}{\text{NR} - i + 1}\right) \times E[\text{OVPBA}] \right\rceil \text{rba}_4. \quad (13\text{-}49)$$

Наконец, воспользуемся (12.4) оценкой количества обращений к файлу транзакций из разд. 12.2: $\text{PBA}_{\text{tf}} = \left\lceil \frac{\text{RPBR}}{\text{EBF}_{\text{tf}}} \right\rceil \text{sba}$.

Общее количество обращений к физическим блокам, обеспечиваюших выборку пакета записей из индексно-последовательного файла, снабженного индексами цилиндра и дорожки, равно сумме уравиений (13-45), (13-47)—(13-49) и (12.4).

В отличие от выборки пакета произвольная выборка использует произвольное обращение к каждому уровню индекса и данных, сопровождаемое последовательным поиском записи-цели на каждом уровне:

$$\text{PBA (ПОЛУЧИТЬ УНИКАЛЬНУЮ, индекс цилиндра)} = \left[1\,\text{rba} + \left(\left\lceil \frac{(1 + \text{NCYL})/\text{EBF}_1}{2} \right\rceil - 1\right)\text{sba}\right] \times \text{RPBR}. \quad (13\text{-}50)$$

Поиск на уровне индекса дорожки подобен поиску на уровне индекса цилиндра, за исключением случая, когда индекс дорожки ограничен одной дорожкой. Следовательно, для доступа к произвольному элементу индекса дорожки требуется одно

произвольное обращение к блоку:

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ, индекс дорожки)} = \\ = [1\ \text{rba}] \times \text{RPBR} = \text{RPBR rba}. \qquad (13\text{-}51)$$

Далее, в силу того, что элементы индекса дорожки содержат отдельные указатели дорожек данных, для доступа к записи данных также достаточно одного произвольного обращения к блоку:

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ, запись данных)} = \\ = [1\ \text{rba}] \times \text{RPBR} = \text{RPBR rba}, \qquad (13\text{-}52)$$

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ, переполнение)} = \\ = \lceil \text{RPBR} \times E[\text{OVPBA}] \rceil \text{rba}_1. \qquad (13\text{-}53)$$

Общее количество обращений к физическим блокам для произвольной выборки RPBR записей из индексно-последовательного файла, снабженного индексами дорожки и цилиндра, равно сумме выражений (13-50)—(13-53) и (12-4).

• **Пример 13-5.** Пусть дан файл из 10^6 записей фиксированного размера (100 байт каждая); размер блока равен 1000 байт и коэффициент полезного блокирования составляет 10 (при коэффициенте загрузки LF = 1 и BOVHD = 0). Предположим, что размер блока в индексе равен размеру блока данных, а коэффициент блокирования индекса равен 100. Пусть в индексе цилиндра имеется NCYL = 100 элементов (записей), а в индексах дорожек NTRK = 10 000 элементов (записей), т. е. в каждом индексе дорожки имеется NTRK/NCYL = 100 элементов; на каждой дорожке данных имеется NR/NTRK элементов и в индексе цилиндра имеется NCYL = 100 элементов. Предположим также, что файл только что был загружен и переполнение еще отсутствует. Пусть RPBR обозначает количество выбираемых из файла записей (пакет). Требуется сравнить производительности произвольной выборки и выборки пакета для следующих значений RPBR: $\text{RPBR} = 10^j$ для $j = 0, 1, \ldots, 6$. Поскольку оценка количества обращений к файлу транзакций одинакова для обоих случаев, ею здесь можно пренебречь. Требуется также определить, что изменилось бы, если бы вместо индексно-последовательной пакетной обработки использовалась последовательная пакетная обработка.

Решение. $\text{NBLK} = \lceil \text{NR}/\text{EBF}_3 \rceil = 10^6/10 = 10^5$.

ПОЛУЧИТЬ УНИКАЛЬНУЮ (индексно-последовательная обработка):

$$\begin{aligned}\text{PBA} &= \text{PBA (индекс цилиндра)} + \text{PBA (индекс дорожки)} + \\ &+ \text{PBA (запись данных)} + \text{PBA (переполнение)} = \\ &= \left[1\ \text{rba} + \left(\left\lceil \frac{(1+10^2)/10^2}{2} \right\rceil - 1\right)\text{sba}\right] \times \text{RPBR} + \text{RPBR rba} + \\ &+ \text{RPBR rba} + 0\ \text{rba} = 3 \times \text{RPBR rba}.\end{aligned}$$

ПОЛУЧИТЬ ПАКЕТ (индексно-последовательная обработка):

$$\begin{aligned}\text{PBA} &= \text{PBA (индекс цилиндра)} + \text{PBA (индекс дорожки)} + \\ &+ \text{PBA (записи данных)} + \text{PBA (переполнение)} = \\ &= \left[1\ \text{rba} + \left(\left\lceil \frac{10^2}{10^2} \times \frac{\text{RPBR}}{\text{RPBR}+1} \right\rceil - 1\right)\text{sba}\right] + \\ &+ \left[1\ \text{rba} + \left(\left\lceil 10^2\left(1 - \prod_{i=1}^{\text{RPBR}} \frac{10^4\,(0{,}99) - i + 1}{10^4 - i + 1}\right)\right\rceil - 1\right)\text{rba}_2\right] + \\ &+ \left[1\ \text{rba} + \left(\left\lceil 10^5\left(1 - \prod_{i=1}^{\text{RPBR}} \frac{10^6\,(0{,}99999) - i + 1}{10^6 - i + 1}\right)\right\rceil - 1\right)\text{rba}_3\right] + \\ &+ \left[\left\lceil 10^5\left(1 - \prod_{i=1}^{\text{RPBR}} \frac{10^6\,(0{,}99999) - i + 1}{10^6 - i + 1}\right) \times \frac{0}{2}\right\rceil\right] = \\ &= 3\ \text{rba} + \left(\left\lceil 10^2\left(1 - \prod_{i=1}^{\text{RPBR}} \frac{10^4\,(0{,}99) - i + 1}{10^4 - i + 1}\right)\right\rceil - 1\right)\text{rba}_2 + \\ &+ \left(\left\lceil 10^5\left(1 - \prod_{i=1}^{\text{RPBR}} \frac{10^6\,(0{,}99999) - i + 1}{10^6 - i + 1}\right)\right\rceil - 1\right)\text{rba}_3.\end{aligned}$$

Для случая RPBR = 1 оценка в обоих методах выборки одинакова. В каждом методе требуется ровно три произвольных обращения к блокам одного и того же вида.

ПОЛУЧИТЬ ПАКЕТ (физически последовательная обработка): пренебрегая оценкой для файла транзакций, воспользуемся уравнением (12-8):

$$\begin{aligned}\text{PBA} &= \left\lceil \frac{\text{RPBR}}{\text{RPBR}+1} \times \frac{\text{NR}}{\text{EBF}_3} \right\rceil \text{sba} + \text{RPBR sba} = \\ &= \left\lceil \frac{\text{RPBR}}{\text{RPBR}+1} \times 10^5 \right\rceil \text{sba} + \text{RPBR sba}.\end{aligned}$$

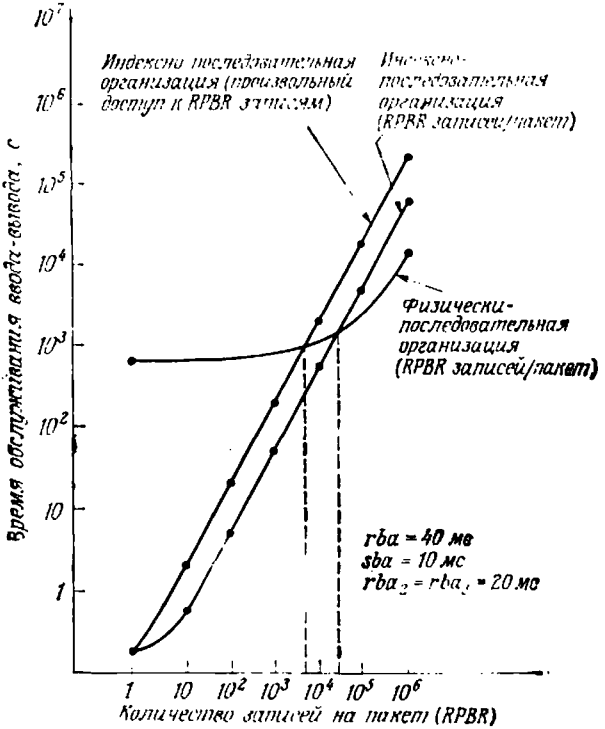


Рис. 13.14. Производительности произвольной и пакетной обработки при индексно-последовательной организации.

Теперь представим графически зависимость указанных оценок от значения RPBR. Для блока размером 1000 байт можно использовать следующие значения: $rba = 40$ мс, $rba_2 = rba_3 = 20$ мс и $sba = 10$ мс.

На рис. 13.14 показано, что в случае индексно-последовательной организации производительность произвольной обработки всегда ниже, чем пакетной обработки. В то же время производительность пакетной обработки в случае индексно-последовательной организации оказывается выше производительности простой последовательной пакетной обработки (без индексов) для пакетов размером меньше чем примерно 20 000 записей или 2 % файла. Для пакетов большего размера использование индексов становится неоправданным. Индексно-последовательная организация становится менее эффективной, чем последовательная организация, использующая только файл данных. □

13.4.2. Обновление индексно-последовательного файла

Метод оценки выполнения операции обновления индексно-последовательного файла очевиден. Самый простой случай изменения записи представляет модификация в ней неключевых данных. Подобное изменение выполняется путем выборки записи-цели, ее изменения в рабочей области пользователя и последующего занесения измененной записи вместе с остальной частью блока во вторичную память. Помимо операции выборки требуется только одна операция последовательной записи блока. Эта процедура не отличается от процедуры обновления для метода с полным индексом.

Модификация ключевого значения в записи представляет более сложный случай. Если изменение незначительно и не вызывает перемещения записи данных в другой блок, то достаточно только простой операции перезаписи блока. При более существенном изменении ключевого значения приходится удалить запись данных со старого места и включить ее на новое место в другой блок. Каждой из подопераций удаления и включения должен предшествовать отдельный поиск. Затем необходимо проверить все уровни индекса на предмет выявления возможного воздействия удаляемого и нового включаемого ключевых значений на хранимые в нем значения ключей блоков. Если такое воздействие имеет место, то необходимо произвести модификацию индекса. Для оценки этой операции обозначим PRMV — вероятность перемещения записи данных в новый блок, $PRDEL_i$ — вероятность наличия старого ключевого значения в индексе уровня i и $PRINS_i$ — вероятность включения нового ключевого значения в индекс уровня i.

Ниже приводятся оценки указанных подопераций в виде количества обращений к блокам.

1. *Поиск записи-цели*

$$PBA_1 = n\,\text{rba} + \sum_{i=1}^{n}\left(\left\lceil\frac{(1+NR_i)/2}{EBF_i}\right\rceil - 1\right)\text{sba} + E\,[\text{OVPBA}]\,\text{rba}. \tag{13-54}$$

2. *Изменение записи данных*

$$PBA_2 = PRMV \times (\text{удаление и повторное включение записи данных})\,\text{pba} + (1 - PRMV) \times 1\,\text{sba} = PRMV \times \times (PBA_3 + PBA_4)\,\text{pba} + (1 - PRMV)\,\text{sba}, \tag{13-55}$$

где pba обозначает просто обращение к физическому блоку в случае, когда неизвестно, произвольное это обращение или последовательное.

3. *Удаление записи данных* (после завершения поиска пометить запись как удаленную и перезаписать)

$$\mathrm{PBA}_3 = 1\ \mathrm{sba}. \tag{13-56}$$

4. *Включение записи данных* (после завершения поиска)

$$\mathrm{PBA}_4 = \mathrm{OVFLPR} \times (1\ \mathrm{rba} + 2\ \mathrm{sba}) + (1 - \mathrm{OVFLPR}) \times 1\ \mathrm{sba}. \tag{13-57}$$

Если происходит переполненне, то необходимо прочитать и перезаписать блок переполнения, а в переполненном блоке данных установить соответствующее значение указателя и перезаписать блок. При отсутствии переполнения достаточно перезаписать блок данных, включающий новую запись.

Простейший метод определения вероятности переполнения для заданного бакета (OVFLPR) состоит в том, чтобы сначала определить вероятность отсутствия переполнения 1 — OVFLPR. Она получается в результате суммирования вероятностей наличия в бакете x записей для всех $x \leqslant \mathrm{RSPB}$:

$$1 - \mathrm{OVFLPR} = \sum_{x=1}^{\mathrm{RSPB}} P(x). \tag{13-58}$$

Перестановкой членов получаем

$$\mathrm{OVFLPR} = 1 - \sum_{x=1}^{\mathrm{RSPB}} P(x) = 1 - \sum_{x=1}^{\mathrm{RSPB}} \frac{e^{-\mathrm{RPB}} \times \mathrm{RPB}^x}{x!}. \tag{13-59}$$

5. *Доступ ко всем уровням индекса* с целью возможного удаления или включения ключевых значений

$$\mathrm{PBA}_5 = \mathrm{NIL}\ \mathrm{rba} + \sum_{i=1}^{\mathrm{NIL}} \left\lceil \frac{(1 + \mathrm{NR}_i)/2}{\mathrm{EBF}_i} - 1 \right\rceil \mathrm{sba}. \tag{13-60}$$

6. *Включение записи в индекс*

$$\mathrm{PBA}_6 = \sum_{i=1}^{\mathrm{NIL}} \mathrm{PRINS}_i\ \mathrm{sba}. \tag{13-61}$$

Для изменения записи индекса необходимо выбрать соответствующий блок индекса, внести в него изменение и перезаписать.

7. *Удаление записи из индекса*

$$\mathrm{PBA}_7 = \sum_{i=1}^{\mathrm{NIL}} \mathrm{PRDEL}_i\ \mathrm{sba}. \tag{13-62}$$

Общая оценка процедуры изменения значения ключа:

$$\mathrm{PBA}\ (\text{ИЗМЕНИТЬ ключ}) = \mathrm{PBA}_1 + \mathrm{PBA}_2 + \mathrm{PBA}_5 + \mathrm{PBA}_6 + \mathrm{PBA}_7. \tag{13-63}$$

Для операции включения необходимо включить запись данных и, возможно, значение ключа в один или более индекс, если значение ключа является наибольшим в блоке:

$$\text{РВА (ВКЛЮЧИТЬ)} = \text{РВА}_1 + \text{РВА}_4 + \text{РВА}_5 + \text{РВА}_6. \quad (13\text{-}64)$$

Для выполнения операции удаления необходимо пометить удаляемую запись данных и перезаписать включающий ее блок. Кроме того, если значение ключа удаляемой записи входит в один или несколько индексов, его необходимо заменить на ключевое значение, наибольшее среди оставшихся в блоке:

$$\text{РВА (УДАЛИТЬ)} = \text{РВА}_1 + \text{РВА}_3 + \text{РВА}_5 + \text{РВА}_7. \quad (13\text{-}65)$$

Благодаря возможности использовать индексы для доступа к записи-цели без просмотра всего файла пакетные обновления, как правило, являются более эффективными применительно к индексно-последовательной организации, нежели к физически последовательной организации. Компонент поиска эквивалентен выборке пакета, оцениваемой уравнением (13-48), а компонент обновления эквивалентен совокупности операций обновления в произвольной обработке, оцениваемым уравнениями (13-55)—(13-65). Вследствие близости последовательных записей-целей в упорядоченном пакете, уменьшается время поиска, что является безусловным достоинством пакетной обработки.

13.4.3. Объем памяти для индексно-последовательной организации

Пространство памяти в индексно-последовательной организации занято блоками индекса (уровни от 1 до NIL), блоками данных (уровень $n = \text{NIL} + 1$) и блоками переполнения (уровень $n + 1$):

$$\text{BLKSTOR} = \sum_{i=1}^{\text{NIL}} \left\lceil \frac{\text{NR}_i}{\left\lfloor \frac{(\text{BKS}_i - \text{BOVHD}_i) \times \text{LF}_i}{\text{KS} + \text{PS}} \right\rfloor} \right\rceil \times \text{BKS}_i +$$

$$+ \left\lceil \frac{\text{NR}_n}{\left\lfloor \frac{(\text{BKS}_n - \text{BOVHD}_n) \times \text{LF}_n}{\text{SRS}} \right\rfloor} \right\rceil \times \text{BKS}_n +$$

$$+ \left\lceil \frac{E[\text{OVPBA}]}{\left\lfloor \frac{(\text{BKS}_{n+1} - \text{BOVHD}_{n+1})}{\text{SRS}} \right\rfloor} \right\rceil \times \text{BKS}_{n+1} \text{ байт}, \quad (13\text{-}66)$$

где NR_i обозначает количество элементов индекса уровня i.

13.4.4. Сравнительный анализ индексно-последовательной организации

Индексно-последовательная организация в сравнении с методом полного индекса

Предположим, что рассматривается вариант метода доступа с полным индексом, где индекс упорядочен, а файл данных нет. Теперь можно применить введенный выше метод анализа для сравнения производительности индексно-последовательной организации с производительностью метода полного индекса. При отсутствии переполнения операции типа ПОЛУЧИТЬ ВСЕ в обоих методах доступа имеют одинаковую производительность. Если же требуются данные, упорядоченные по значениям первичного ключа, то в случае индексно-последовательного файла дополнительная сортировка данных не нужна. В случае неупорядоченного варианта метода полного индекса всегда требуется последующая сортировка данных в соответствии со спецификациями отчетов.

В методе полного индекса не предусмотрена обработка переполнения; вместо этого всякий раз при включении нового элемента данных выполняется переупорядочение индекса. Операции типа ПОЛУЧИТЬ УНИКАЛЬНУЮ весьма эффективны в обоих методах, хотя реализация их различна вследствие различия в реализации механизма поиска. В обоих методах доступа также относительно просто выполняется изменение неключевых значений ценой одной операции перезаписи блока.

Вследствие физически последовательного размещения записей операции типа ПОЛУЧИТЬ СЛЕДУЮЩУЮ и ПОЛУЧИТЬ ПРЕДЫДУЩУЮ выполняются гораздо эффективнее в индексно-последовательной организации. Кроме того, благодаря упорядоченности файла данных индексно-последовательный метод является единственным методом, в котором эффективно реализуется пакетная обработка. При условии, что при удалении записей выполняется только специальная пометка элементов индекса, операции удаления в методе полного индекса оказываются очень простыми; однако добавления и изменения ключевых значений в обоих методах трудоемки по причине объемного обновления индекса. Благодаря использованию индекса блока объем памяти меньше в индексно-последовательном методе.

• **Пример 13-6.** Пусть дан файл из 10^6 записей размером 100 байт каждая, и пусть размер ключа равен 6 байт, размер указателя — 4 байт и размер блока (BKS) — 1000 байт. Предположим для простоты, что коэффициент загрузки равен 1, а BOVHD = 0. Требуется сравнить объемы памяти, необходимые для организации с полным индексом и для индексно-последовательной организации, в предположении, что использованы два

уровня индекса, содержащие соответственно 10^2 и 10^4 элементов. Предположим, что записи переполнения отсутствуют. При условии, что в единицу времени выполняются одна операция произвольной выборки, одна операция произвольного включения и одна операция произвольного удаления, требуется определить разницу в общем времени обслуживания ввода-вывода для двух методов доступа. При этом предполагается, что в методе полного индекса для его 10^6 элементов индекса используется бинарный поиск.

$$\text{BLKSTOR (полный индекс)} = \text{BLKSTOR (индекс)} +$$

$$+ \text{BLKSTOR (данные)} = \left\lceil \frac{10^6}{\left\lfloor \frac{1000}{10} \right\rfloor} \right\rceil \times 1000 + \left\lceil \frac{10^6}{\left\lfloor \frac{1000}{100} \right\rfloor} \right\rceil \times$$

$$\times 1000 = 10^7 + 10^8 = 110 \times 10^6 \text{ байт.}$$

Метод полного индекса: $NR_1 = 10^6$, $NR_2 = 1$.
Индексно-последовательный метод: $NR_1 = 10^2$, $NR_2 = 10^2$, $NR_3 = 10^2$.

$$\text{BLKSTOR (индексно-послед.)} = \left\lceil \frac{100}{\left\lfloor \frac{1000}{10} \right\rfloor} \right\rceil \times 1000 +$$

$$+ \left\lceil \frac{10\,000}{\left\lfloor \frac{1000}{10} \right\rfloor} \right\rceil \times 1000 + \left\lceil \frac{10^6}{\left\lfloor \frac{1000}{100} \right\rfloor} \right\rceil \times (1 + 0) \times 1000 =$$

$$= 10^3 + 10^5 + 10^8 = 100{,}101 \times 10^6 \text{ байт (это на 9\% меньше}$$

в сравнении с оценкой для метода полного индекса).

$$\text{PBA (полный индекс)} = \text{PBA (выборка)} \times 3 +$$

$$+ \text{PBA (включение)} + \text{PBA (удаление)} = [\log_2(5000)] \times$$

$$\times 3\,\text{rba} + 2 \times 10^4\,\text{sba} + 1\,\text{rba} + 1\,\text{sba} + 1\,\text{sba} = 12{,}29 \times 3 +$$

$$+ 1\,\text{rba} + 2 \times 10^4 + 2\,\text{sba} = 37{,}87\,\text{rba} + 20\,002\,\text{sba}$$

$$\text{PBA (индексно-послед.)} = \text{PBA (выборка)} \times 3 +$$

$$+ \text{PBA (включение)} + \text{PBA (удаление)} =$$

$$= \left(3\,\text{rba} + \left(\left\lceil \frac{(1 + 10^2)/2}{100} \right\rceil - 1\right)\text{sba} + \left(\left\lceil \frac{(1 + 10^2)/2}{100} \right\rceil - 1\right)\text{sba} + 0\right) \times$$

$$\times 3 + 1\,\text{sba} + 2\,\text{rba} + \left(\left\lceil \frac{(1 + 10^2)/2}{100} \right\rceil - 1\right)\text{sba} +$$

$$+ \left(\left\lceil \frac{(1 + 10^2)/2}{100} \right\rceil - 1\right)\text{sba} + 0{,}11\,\text{sba} + 1\,\text{sba} + 2\,\text{rba} +$$

$$+ \left(\left\lceil \frac{(1 + 10^2)/2}{100} \right\rceil - 1\right)\text{sba} + \left(\left\lceil \frac{(1 + 10^2)/2}{100} \right\rceil - 1\right)\text{sba} +$$

$$+ 0{,}11,\,\text{sba} = (3\,\text{rba}) \times 3 + 1\,\text{sba} + 2\,\text{rba} + 0{,}11\,\text{sba} +$$

$$+ 1\,\text{sba} + 2\,\text{rba} + 0{,}11\,\text{sba} = 13\,\text{rba} + 2{,}22\,\text{sba},$$

где $\text{PRINS}_i = 1/\text{EBF}_{i+1}$, $\text{PRDEL}_i = 1/\text{EBF}_{i+1}$.

Для накопителя на магнитных дисках, характеристики которого приведены в гл. 9 (устройство типа IBM 3350), получаем:

$\text{rba} = 25 + 8{,}35 + 16{,}7/16 = 34{,}4$ мс;

$\text{sba} = 8{,}35 + 16{,}7/16 = 9{,}4$ мс;

$\text{TIO (полный индекс)} = 37{,}87 \times 36{,}4 + 20\,002 \times 9{,}4 =$

$= 189{,}4 \times 10^3$ мс;

$\text{TIO (индексно-послед.)} = 13 \times 36{,}4 + 2{,}22 \times 9{,}4 = 0{,}49 \times 10^3$ мс.

Для рассмотренной рабочей нагрузки гораздо более эффективное обслуживание обеспечивает индексно-последовательная организация. В методе полного индекса наблюдается увеличение времени обслуживания вследствие неэффективной по времени операции включения в индекс новых элементов. Несмотря на применение бинарного поиска в полном индексе, операция выборки также оказывается значительно быстрее в индексно-последовательном методе. В заключение следует отметить, что к тому же для индексно-последовательного метода требуется меньший объем памяти. Хотя в случае организации с полным индексом сами операции удаления оказываются более быстрыми, но общее время поиска и удаления больше, чем для индексно-последовательной организации. Следовательно, можно утверждать, что для файлов большой размерности, включающих около миллиона записей, нет подлинных компромиссов между двумя указанными методами доступа. При заданной рабочей нагрузке и характеристиках базы данных индексно-последовательная организация имеет явное преимущество в сравнении с организацией полного индекса. □

Оптимальная конфигурация индекса

Проблема выбора оптимального количества уровней индекса и размера индекса на каждом уровне — это сложная проблема, в значительной степени зависящая от соотношения операций выборки, включения, удаления и изменения файла. Рассмотрим оценку количества обращений к блокам для операций выборки и построим простую модель. Пусть NR обозначает общее количество записей в файле, EBF_2 — коэффициент полезного блокирования данных, а EBF_1 — коэффициент полезного блокирования одного уровня индекса. Следовательно, $\text{NBLK} = \lceil \text{NR}/\text{EBF}_2 \rceil$ и $\text{NBLKI} = \lceil \text{NRI}/\text{EBF}_1 \rceil$, где NBLKI обозначает количество требуемых блоков индекса, а NRI — количество элементов в индексе блока. По определению файла NRI = NBLK (т. е. в ин-

дексе имеется один элемент для каждого блока файла данных). Средняя длина пути доступа для операции выборки (исключая переполнение) в виде количества обращений к блокам задается уравнением

$$\text{РВА (выборка)} = 1\,\text{rba} + \left(\frac{1 + \text{NBLKI}}{2} - 1\right)\text{sba} + 1\,\text{rba}. \quad (13\text{-}67)$$

Минимальное значение эта функция принимает при NBLKI= = 1. В общем случае для организации индекса с n уровнями время выборки стаиовится минимальным, когда размер блока на каждом уровне максимален. Следовательно, рекомендуется в качестве размера блока индекса использовать самый большой размер, технически допустимый в вычислительной системе; это может быть четверть, половина или целая дорожка на диске. Кроме того, рекомендуется использовать блоки данных большого размера, так как это обеспечивает более эффективную последовательную обработку и уменьшает число требуемых элементов индекса.

Теперь ставится задача по заданному максимальному размеру блока данных BKS и максимальному размеру блока индекса IBKS найти оптимальное количество уровней индекса. Известно, что если количество уровней индекса равно n, то для получения блока данных требуется как минимум $n + 1$ произвольных обращений к блокам, по одному для каждого из n уровней индекса и одно обращение к уровню записей данных. В тех случаях, когда файл индекса или данных состоит из нескольких блоков, на каждом уровне требуется также ряд дополнительных последовательных обращений к блокам. Если при этом на каком-то уровне требуется свыше 30 последовательных обращений к блокам, то рекомендуется добавить уровень в индекс. Плата за дополнительный уровень составит 1 rba, а экономлено будет на j-ом уровне 30 sba. Известно, что для размера блока в пределах 2^{10}—2^{12} байт и накопителя на магнитных дисках типа IBM 3350 соотношение оценок rba и sba составляет приблизительно 4 : 1. В конечном счете всякий раз, когда некоторый уровень индекса занимает несколько блоков, с точки зрения уменьшения времени ввода-вывода оправдано введение индекса верхнего уровня (главного индекса), размещаемого в оперативной памяти. Платой за главный индекс явится дополнительный объем постоянно занятой оперативной памяти. Введение дополнительных уровней не влечет за собой увеличения стоимости операций обновления по причине уменьшения в них компонента поиска и минимизации количества обращений к блокам при обновлении индекса блока. В справедливости этого утверждения можно убедиться, применив полученные выше оценки операций обновления и выборки к различным конфигурациям индекса.

Глава 14. Первичные методы доступа: деревья поиска и произвольная обработка

Применение методов доступа, основанных на использовании явных древовидных структур, к поиску элементов в таблицах и списках в оперативной памяти известно в течение многих лет. Однако в последнее время область их применения расширена произвольной выборкой данных из вторичной памяти; таким образом, эти методы стали конкурентами классических методов доступа, таких, как индексно-последовательный метод и хеширование. В данной главе исследуются три основных типа древовидных структур и некоторые их разновидности: бинарное дерево поиска, *B*-дерево и *TRIE*-структура. В качестве механизма поиска для всех этих структур, как правило, используется связанный последовательный метод; поэтому их отличительной чертой, заслуживающей анализа и оценки, является тип структуры.

14.1. Бинарные деревья поиска

Бинарные деревья поиска представляют важный класс физических структур баз данных, обеспечивающих разумное компромиссное решение для прикладных функций как с произвольной, так и с последовательной выборкой данных, а также для большого по объему обновления данных. В целях улучшения обработки отчетов и пакетных транзакций можно организовать записи данных в виде упорядоченного связанного последовательного файла. Наличие дополнительного указателя в каждой хранимой записи облегчает быстрый поиск при выполнении операций произвольной выборки, которые наиболее часто используются с такой организацией. Операции включения и удаления по завершении поиска выполняются достаточно быстро благодаря тому, что изменяются только значения указателей и не требуется выполнять перераспределение памяти. *Бинарное дерево* T_{NR} представляет собой упорядоченную тройку (T_l, R, T_r), где R обозначает корневую вершину, а T_l и T_r — соответственно левое и правое поддеревья вершины R. T_l и T_r содержат соответственно l и r вершин. Следует отметить, что $l \geqslant 0$, $r \geqslant 0$ и $l + r = NR - 1$. *Бинарное дерево поиска* на множестве имен (идентификаторов) NR есть бинарное дерево T_{NR}, в котором каждая вершина помечена отдельным именем и расположена в соответствии с лексикографическим порядком имен, то есть для любой вершины i имена вершин в его левом поддереве лексикографически предше-

ствуют имени вершины i и именам вершин в его правом поддереве. Механизм поиска для такого дерева вначале выполняет просмотр корневой вершины R и сравнение ключа поиска k со значением ключа в корневой вершине k_R. В случае $k = k_R$ поиск завершается успешно. В случае $k < k_R$ (в лексикографическом смысле) поиск продолжается соответственно в левом поддереве. В случае $k > k_R$ (в лексикографическом смысле) поиск продолжается в правом поддереве. Если алгоритм достигает «листа» i и $k \neq k_i$, то поиск заканчивается неуспешно. *Листом* называется вершина бинарного дерева поиска, у которой нет поддеревьев (то есть конечная вершина).

В каждой вершине бинарного дерева поиска хранятся: значение вершины, левый указатель и правый указатель. Значение вершины состоит из значения полного первичного ключа и, возможно, значений нескольких неключевых элементов данных; каждая вершина эквивалентна записи. Механизм включения новой вершины использует сначала механизм поиска с целью найти вершину (запись), содержащую значение ключа поиска. Если поиск заканчивается неуспешно на листе j, новая вершина включается в дерево в качестве левого или правого поддерева листа j в зависимости от того, соответственно меньше или больше значение (имя) ключа, чем значение k_j. При таком алгоритме длина пути до идентификатора зависит от порядка, в котором хранятся идентификаторы; вследствие этого форма полученного дерева может изменяться от линейного списка (рис. 14.1, *а*) до полностью сбалансированного дерева (рис. 14.1, *б*). Обычно форма дерева представляет нечто среднее между указанными экстремальными случаями (рис. 14.1, *в*).

Для некоторого заданного множества значений идентификаторов NR средняя длина пути до идентификатора изменяется в диапазоне от $(1 + \mathrm{NR})/2$ для линейного списка до приблизительно $\log_2(\mathrm{NR} + 1) - 2$ для сбалансированного дерева [277]. В работе [151] показано, что при отсутствии упорядоченности средняя длина пути составляет $1{,}4 \log_2 \mathrm{NR}$ для неуспешного поиска и $1{,}4 \log_2(\mathrm{NR} - 1)$ для успешного поиска. Кроме того, в этой работе проанализированы операции обновления и объем памяти.

Понятие «баланса» применительно к бинарному дереву поиска представляет важный момент при проектировании древовидных структур, эффективных с точки зрения операций выборки и обновления. Прежде чем рассмотреть это понятие, необходимо ввести некоторые дополнительные определения. *Уровень* вершины или листа i, обозначенный L_i, определяется длиной пути от корневой вершины T_{NR} до вершины i. Корневая вершина по определению имеет уровень 0. *Высота* дерева определяется как максимальный уровень всех его вершин. Дерево называется *сбалансированным*, если разница уровней любых двух листьев

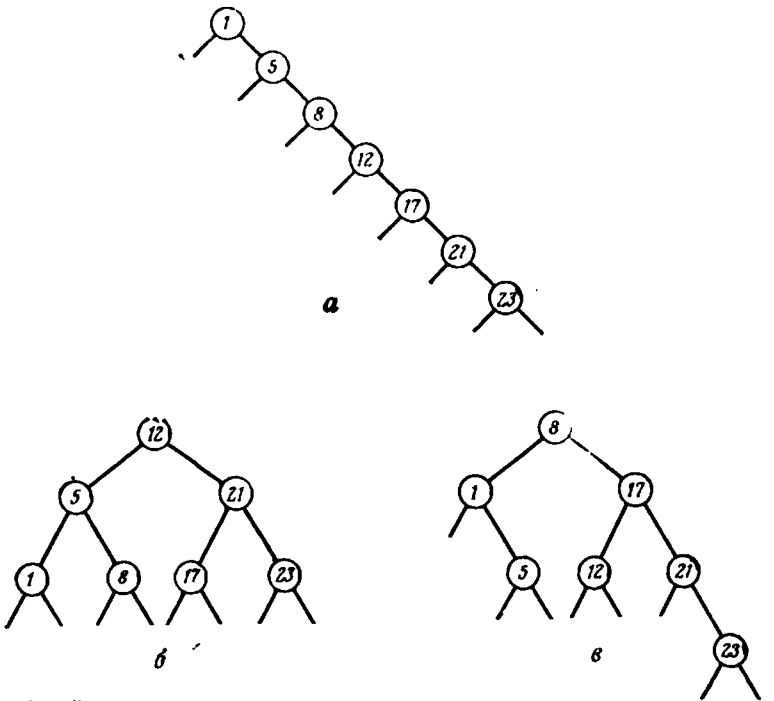


Рис. 14.1. Бинарные деревья поиска.

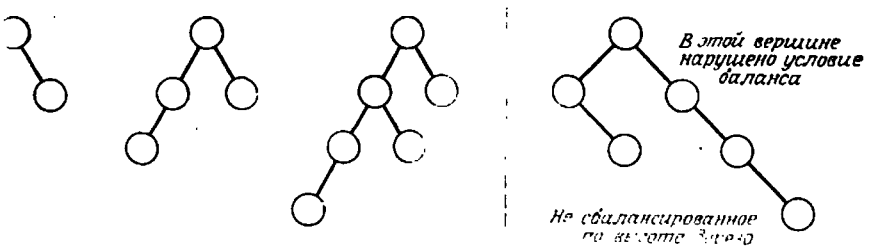


Рис. 14.2. Деревья, сбалансированные по высоте (AVL-деревья) [240] и не сбалансированные по высоте.

(конечных вершин) не превышает единицы. В тех случаях, когда равновероятно обращение к любой из вершин, использование сбалансированного дерева минимизирует среднюю длину доступа. По этой причине важно научиться конструировать сбалансированные деревья.

Для изменчивых баз данных в работах [3, 117] предложена и исследована еще одна возможная организация бинарного де-

рева поиска, известная под названием *AVL-дерева*. Деревья такого вида называют еще «сбалансированными по высоте» деревьями. Дерево T_{NR} называется сбалансированным по высоте (*AVL*-деревом), если для любой вершины i разница по высоте ее левого и правого поддеревьев не превышает единицы. На рис. 14.2 приведены несколько примеров таких деревьев. Это менее строгое определение баланса, чем определение полностью

Таблица 14.1. **Сравнение длины поиска (количества проб) для бинарных деревьев поиска с NR вершинами**[1]

	Линейное [1]	Несбалансированное [2]	Сбалансированное [3]	*AVL* [4]
Наименьшая длина поиска	1	1	1	1
Средняя длина поиска	$(1+NR)/2$	$1{,}4\log_2(NR-1)$ (в случае успешного поиска) $1{,}4\log_2 NR$ (в случае неуспешного поиска)	$\log_2(NR+1)-2$	$\log_2(NR+1)$
Наибольшая длина поиска	NR	Не определено	$\log_2(NR+1)$	$1{,}44\log_2(NR+1)$

[1] Линейный (связанный) список; наихудший случай несбалансированного бинарного дерева поиска.

[2] Несбалансированное дерево; включение новых записей в произвольном порядке.

[3] Полностью сбалансированное дерево поиска; длина пути от корня до любых двух листьев различается не более чем на единицу.

[4] Сбалансированное по высоте дерево (*AVL*-дерево); подчиненные одной вершине поддеревья различаются по высоте не более чем на единицу.

сбалансированного бинарного дерева поиска. Действительно, для m-уровневого *AVL*-дерева максимальная разница длин любых двух путей составляет $m/2$. Однако средняя длина пути успешного поиска приблизительно равна $\log_2(NR+1)$. Это несущественно больше, чем оптимальная средняя длина поиска для полностью сбалансированного бинарного дерева поиска. В табл. 14.1 приведены сравнительные оценки производительности в виде длин путей доступа (обращений к логическим записям) для разных видов деревьев.

В целях повышения эффективности операций выборки и обновления предложено много других видов сбалансированных деревьев; среди них можно назвать «сбалансированные по весу»[1] деревья и обобщение понятия сбалансированного по высоте дерева с разницей длины пути между поддеревьями $d > 1$. Обзор работ, касающихся бинарных деревьев поиска, можно найти в [188, 240].

Основные моменты, на которые следует обратить внимание при проектировании бинарных деревьев поиска:

• *Изменчивость данных.* Сравнительная оценка важности операций обновления и выборки влияет на вид требуемого баланса дерева.

• *Распределение частот обращений к идентификаторам записей.* Влияет на вид требуемого баланса дерева.

• *Последовательность включения новой записи.* Влияет на степень балансa, на время выборки.

• *Уровень запоминающего устройства.* Файл во вторичной памяти, возможно, выиграл бы от организации записей в кластеры (например, верхние вершины в непрерывной области памяти).

14.1.1. Производительность выборки

На рис. 14.1 показаны пути доступа в бинарных деревьях поиска. На каждом уровне происходит доступ только к одной вершине, причем для случая вторичной памяти (если не оговорено особо) подразумевается произвольный доступ. Если бы при реализации вершины определенным образом были организованы в кластеры, потребовалось бы дополнительное исследование с целью определения «веса» (в виде расстояния или времени) каждого произвольного доступа. Используя для оценки произвольного поиска средние значения количества проб из табл. 14.1, получим:

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ)} = \begin{cases} 1{,}4\log_2(\text{NR}-1)\,\text{rba} & \text{для несбалансированного дерева,} \\ \log_2(\text{NR}+1)-2\,\text{rba} & \text{для сбалансированного дерева,} \\ \log_2(\text{NR}+1)\,\text{rba} & \text{для } AVL\text{-дерева.} \end{cases} \qquad (14\text{-}1)$$

[1] В данном типе деревьев вместо высоты дерева рассматривается вес, равный количеству листьев дерева. Сбалансированным по весу деревом называется дерево, вес вершины которого удовлетворяет условию $\sqrt{2}-1 < \frac{\text{вес левого поддерева}}{\text{вес правого поддерева}} < \sqrt{2}+1$. — *Прим. ред.*

В случае неуспешного поиска имеем:

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ')} = \begin{cases} = 1{,}4 \log_2 \text{NR rba} & \text{для несбалансированного дерева,} \\ \leqslant \log_2 (\text{NR} + 1)\,\text{rba} & \text{для сбалансированного дерева,} \\ \leqslant 1{,}44 \log_2 (\text{NR} + 1)\,\text{rba} & \text{для } AVL\text{-дерева.} \end{cases} \quad (14\text{-}2)$$

Для операций ПОЛУЧИТЬ СЛЕДУЮЩУЮ и ПОЛУЧИТЬ ПРЕДЫДУЩУЮ требуется одно или несколько произвольных обращений к блокам в соответствии со значениями указателей на правое и левое поддеревья. В худшем случае, возможно, придется просмотреть дерево по всей высоте. Для операции ПОЛУЧИТЬ ВСЕ требуется выполнить обход всего дерева; эта операция применима как к последовательной обработке, так и к обработке булевых запросов:

$$\text{РВА (ПОЛУЧИТЬ ВСЕ)} = \text{NR rba}. \quad (14\text{-}3)$$

В целях повышения эффективности при реализации бинарных деревьев поиска, как правило, используется коэффициент блокирования; записи группируются в блоки в соответствии с естественным порядком их включения. Такой способ организации записей называется *последовательным распределением* [234, 240]. При так называемом *групповом распределении* предпринимается попытка расположить новую вершину (запись) поблизости от соответствующей исходной вершины (записи). Если содержащий исходную вершину блок (или страница) заполнен, то новой вершине выделяется новый блок (страница). Результаты имитационного моделирования показали, что при использовании схемы группового распределения наблюдается уменьшение количества обращений к блокам. В то же время вследствие небольшой заполненности блоков, выделяемых для подчиненных записей в случае, когда их нельзя разместить рядом с соответствующими исходными записями (то есть в одном блоке), при групповом распределении для хранения данных требуется бо́льшая память. Можно добиться некоторого уменьшения объема памяти за счет использования незаполненных блоков для хранения записей переполнения.

В лучшем случае для группового распределения, когда в каждом блоке размещаются EBF записей, вместо $\log_2 \text{EBF}$ обращений к записям требуется одно обращение к блоку. Следовательно, в лучшем случае:

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ, групповое распределение)} = \frac{\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ)}}{\log_2 \text{EBF}}\,\text{rba}. \quad (14\text{-}4)$$

Аналогично можно определить лучший случай для неуспешного поиска. В то же время для неупорядоченной последовательности обработки более эффективна схема последовательного распределения. В лучшем случае:

$$\text{РВА (ПОЛУЧИТЬ ВСЕ, неупорядоченная обработка, последовательное распределение)} = \left\lceil \frac{NR}{EBF} \right\rceil \text{sba}. \quad (14\text{-}5)$$

Упорядоченную последовательную обработку можно выполнять двумя способами: 1) посредством упорядоченной последовательности произвольных обращений к вершинам, уравнение (14-5); 2) посредством последовательных обращений к вершинам в порядке их расположения в дереве, уравнение (14-3), с последующей сортировкой записей; выбирается метод доступа, оценка которого меньше. В работе [234] получена оценка количества обращений к блокам для случая несбалансированного дерева и последовательного распределения:

$$\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ, несбаланс. дерево, последовательное распределение)} = 0{,}75 + \frac{1{,}5 \times EBF}{NR} + 1{,}4 \log_2 \left(\frac{NR}{2 \times EBF} \right). \quad (14\text{-}6)$$

14.1.2. Производительность обновления

Выполняемые после завершения поиска операции обновления во всех конфигурациях бинарных деревьев поиска, приведенных в табл. 14.1, имеют одинаковые характеристики. Простейший вид обновления представляет изменение неключевого значения в вершине. Оценка этой операции совпадает с оценкой для связанной последовательной организации:

$$\text{РВА (ИЗМЕНИТЬ неключевое значение)} = 1 \text{ sba}. \quad (14\text{-}7)$$

Изменение ключевого значения в записи бинарного дерева поиска, как правило, влечет за собой перераспределение вершин дерева. Если же изменение так незначительно, что перераспределения вершин не требуется, то его оценка совпадает с оценкой изменения неключевого значения. В том случае, когда одно ключевое значение заменяется на другое произвольное ключевое значение, операция изменения эквивалентна удалению старого ключевого значения и последующему включению в дерево поиска нового ключевого значения. Давайте оценим эти операции.

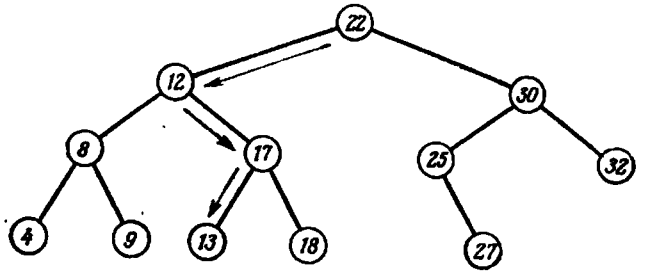
Перед нами стоит задача получить оценки операций удаления и включения произвольной записи для разных конфигура-

ций бинарных деревьев поиска. Конфигурация несбалансированного дерева, в которую записи включаются в естественном (произвольном) порядке их поступления, интересна как иллюстрация случая нижней оценки операции обновления, имеющей место при отсутствии необходимости сохранения баланса дерева. Мы увидим также, что процедура обновления полностью сбалансированного дерева весьма трудоемка; только структура *AVL*-дерева обеспечивает разумный компромисс производительности между операциями выборки и обновления.

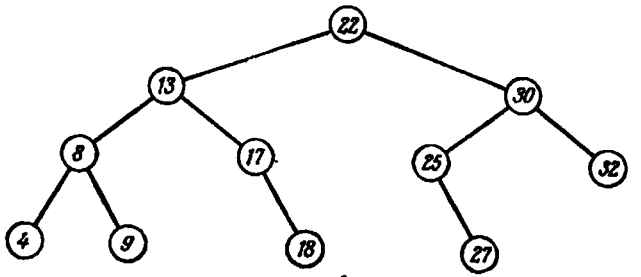
В конфигурации несбалансированного дерева по определению эффективно сочетаются операции поиска и включения записи. Поиск ведется до обнаружения конечной вершины, то есть имеет место случай так называемого «неуспешного» поиска вершины, содержащей значение, равное включаемому ключевому значению. Конечная вершина становится исходной для новой вершины, а новая вершина включается в дерево в качестве ее левого или правого поддерева в зависимости от того, соответственно меньше или больше (в лексикографическом смысле) новый ключ в сравнении с ключом исходной вершины. Если в блоке, содержащем исходную вершину, есть место для новой записи и стратегия организации записей в блоки допускает возможность такого включения, оценка операции включения равна одному последовательному обращению к блоку исходной записи. Во всех других случаях для включения новой порожденной записи требуется дополнительное произвольное обращение к блоку:

$$1\,\mathrm{sba} \leqslant \text{РВА (ВКЛЮЧИТЬ, несбаланс.)} \leqslant 1\,\mathrm{sba} + 1\,\mathrm{rba}. \quad (14\text{-}8)$$

При условии, что операция удаления записи сводится к установке в ней специального флажка, ее реализация для несбалансированных деревьев очень проста. Для этого потребуется только одно последовательное обращение (sba) для перезаписи блока, доступ к которому осуществлен в результате так называемого «успешного» поиска. Однако чаще всего удаление записи осуществляется посредством изменения значений соответствующих указателей, а для этого в целях сохранения структуры бинарного дерева требуется определенным образом переместить вершины, порожденные удаляемой вершиной. В работе [188] предложен алгоритм такого способа удаления записи. На рис. 14.3 на примере удаления вершины 12 продемонстрировано действие этого алгоритма. Если бы у вершины 12 не было поддеревьев, то для ее удаления достаточно было бы установить в нуль левый указатель вершины 22. Если бы у вершины 12 отсутствовало только одно из поддеревьев (левое или правое), то потребовалось бы переместить второе поддерево на место удаляемой вершины; для этого достаточно установить ле-



а



б

Рис. 14.3. Удаление вершины из несбалансированного бинарного дерева поиска:

а — до удаления вершины 12; *б* — после удаления вершины 12.

вый указатель вершины 22 на корневую вершину этого поддерева.

На рис. 14.3, *а* показан наиболее трудный случай, когда у удаляемой вершины есть и левое и правое поддеревья. В этом случае поиск ведется в правом поддереве удаляемой вершины; выполняется рекурсивный проход по его левым указателям до обнаружения конечной вершины. Эта вершина (вершина 13) имеет наименьшее значение ключа в поддереве; при этом оно, конечно, больше любого ключевого значения левого поддерева удаляемой вершины. Кроме того, по определению у нее нет поддеревьев; благодаря этому перераспределение вершин в дереве сводится к минимуму. Конечная вершина (вершина 13) удаляется с ее текущей позиции и включается вместо исходной удаляемой вершины (вершины 12). Теперь вершина 13 указывает на вершины 8 и 17, вершина 22 указывает на вершину 13, а левый указатель вершины 17 установлен в нуль. Если бы у вершины 17 не было левого поддерева, то она сама встала бы на

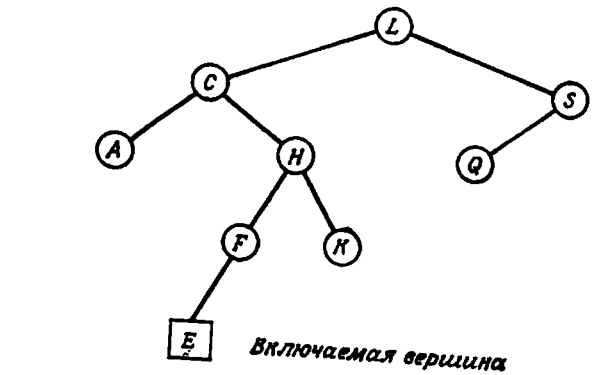
место вершины 12. На рис. 14.3, *б* показано расположение вершин после завершения операции удаления.

Оценка операции удаления изменяется от простейшего случая, когда достаточно изменить значение левого указателя вершины 22 (1 sba) до наиболее сложного случая, когда требуется удалить корень и приходится просмотреть вершины по всей высоте правого поддерева. Следовательно, можно указать следующие нижнюю и верхнюю оценки операции удаления:

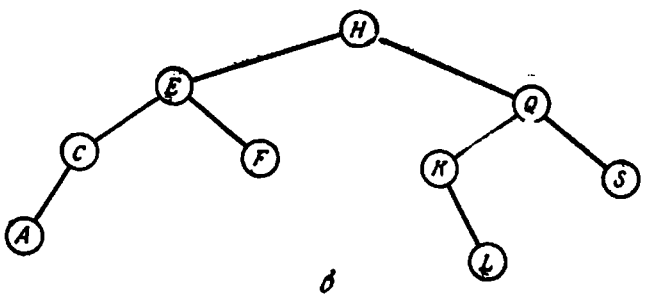
$$1\,\text{sba} \leqslant \text{РВА (УДАЛИТЬ, несбаланс.)} \leqslant 1{,}4\log_2(\text{NR} - 1) - \\ - 1\,\text{rba} + 1\,\text{sba}. \qquad (14\text{-}9)$$

Из-за того что для полностью сбалансированного дерева необходимо после каждой операции включения и удаления оставить неизменными условия баланса, оценка этой конфигурации оказывается сложнее случая несбалансированного дерева. В *AVL*-дереве условие баланса менее сильное, следовательно, потребуется меньше преобразований вершин, и каждое преобразование, как правило, занимает меньше времени в сравнении с полностью сбалансированным деревом. В качестве примера на рис. 14.4 приведено бинарное дерево поиска, сбалансированное по высоте и являющееся одновременно полностью сбалансированным деревом. В результате включения вершины *E* получается новое дерево, по-прежнему сбалансированное по высоте, но, поскольку оно не полностью сбалансировано, так как длина пути от *L* к *E* равна 4 и есть другие пути длины 2, условие полностью сбалансированного дерева нарушено. В случае дерева, сбалансированного по высоте, не потребовалось бы никаких дополнительных преобразований. В противоположность этому преобразование дерева с целью восстановления полного баланса изменяет местоположение каждой его вершины. Это весомый аргумент в пользу применения *AVL*-дерева в промышленных системах, для которых характерна большая изменчивость данных.

Остановимся более подробно на операции обновления *AVL*-дерева и процедуре восстановления баланса по высоте. На рис. 14.5 приведен пример операции включения, после которой для восстановления баланса требуется стандартное преобразование, известное под названием *поворота*. Исходное поддерево сбалансировано по высоте. Символы в промежуточных вершинах обозначают коды условия, характеризующие состояние баланса до операции включения. Символы «/», «—» и «\» в вершине указывают соответственно, что ее левое поддерево выше правого, левое и правое поддерева равны по высоте, правое поддерево выше левого. После включения вершины *E* произошло нарушение условия баланса по высоте в вершине *M*, и для его восстановления необходимо преобразование дерева.

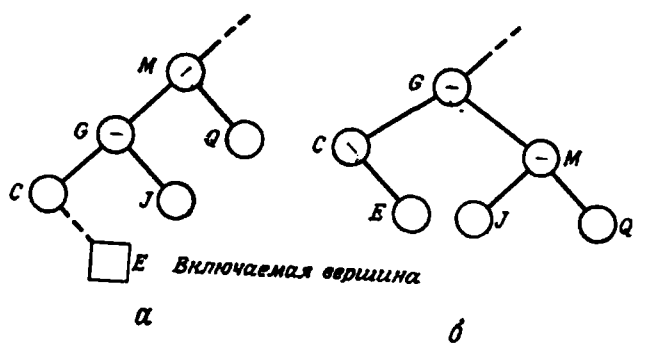


а



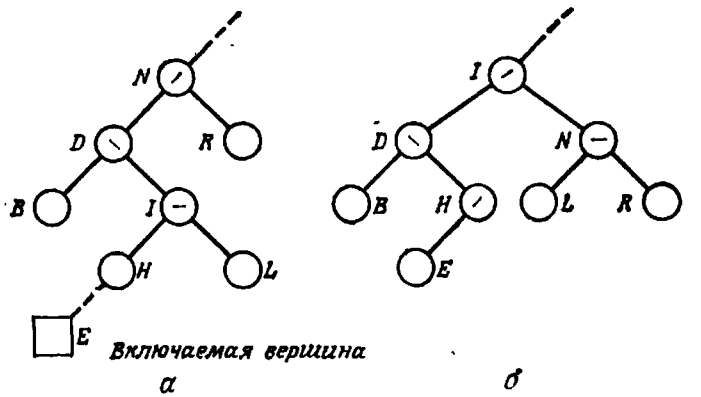
б

Рис. 14.4. Восстановление полного баланса *B*-дерева после включения новой вершины:
а — исходное состояние до включения вершины *E*; *б* — после включения вершины и восстановления баланса.



а *б*

Рис. 14.5. Восстановление баланса дерева по высоте после включения новой вершины методом поворота:
а — исходное состояние; *б* — после поворота.



а *б*

Рис. 14.6. Восстановление баланса по высоте методом двойного поворота: *а* — исходное состояние; *б* — после двойного поворота.

При повороте сохраняются связи между *C*, *E* и *G*, а также между *M* и *Q*. Фактически выполняется поворот вершины *M* отиосительно вершины *G*; по определеиию *G* может иметь только два поддерева, поэтому вершииа *J* стаиовится левым поддеревом вершиины *M*, благодаря чему сохраняется упорядоченность вершии *G*, *J* и *M*.

На рис. 14.6 приведен пример преобразования, известного под названием *двойного поворота*. Включение вершины *E* в поддерево *N* вызывает нарушение баланса по высоте в вершине *D*. При преобразовании миогие связи вершин остаются неизменными. Однако требуются два поворота: поворот вершиины *I* относительно вершины *D*, а затем еще один поворот относительно вершины *N*. В результате первого поворота левое поддерево *E*, *H* вершиины *I* становится правым поддеревом вершины *D*. В результате второго поворота вершины *I* относительно вершины *N* вершина *L* становится левым поддеревом вершиины *N*.

Включеиие новой записи в дерево, сбалаисированное по высоте, вызывает (самое большее) одинариый или двойной поворот. В [188] дан подробный анализ операций обновления сбалаисированиого по высоте дерева исходя из требуемого количества обращений к логическим записям. Кроме того, предложен алгоритм поиска и включения, выполиен математический анализ его сложности и построена таблица вероятностей одинарного или двойного поворота как функции длины пути от вершины, в которой нарушено условие баланса, до включаемой вершиины.

Ввиду сложности анализа операции включения для *AVL*-дерева мы ограничимся только определением нижией и верхней оценок количества обращений к физическим блокам. На рис. 14.7 приведен пример нижней оценки, когда не нужно преобразова-

ний для восстановления балаиса и достаточно устаиовить значение указателя в исходиой вершине. Если при этом обе вершины (исходная и новая подчиненная) расположены в одиом блоке, требуется только одна операция последовательной перезаписи блока. Если же они расположены в разных блоках, то необходимо выполнить операцию последовательной перезаписи исходного блока и операцию произвольной записи нового подчинеииого блока.

Нижняя оценка:

$$\text{РВА (ВКЛЮЧИТЬ, } AVL\text{-дерево)} = \begin{cases} 1\,\text{sba}, & \text{если исходная и новая подчиненная вершины расположены в одном блоке,} \\ 1\,\text{sba} + 1\,\text{rba}, & \text{если в разных блоках.} \end{cases} \qquad (14\text{-}10)$$

Верхнюю оценку получить сложиее. В худшем случае выполняется двойной поворот, затрагивающий три вершииы и включающий обмен двумя поддеревьями между исходными вершинами (см. рис. 14.6). Указанная процедура включает следующие шаги:

1. После завершения поиска места включения необходимо выполнить (самое большее) одну операцию произвольной записи для включения новой записи даиных и затем операцию последовательной записи для обновленных указателей.
2. В предположении, что для хранения адресов вершин, просмотренных в поиске места включения новой вершины, используется стек, для перемещения вершины при повороте не требуется дополнительный поиск во вторичной памятн. При выполнении алгоритма восстаиовлеиия баланса адресá этих вершин берутся из стека.
3. Необходимо измеиить значения указателей в трех вершинах, участвующих в повороте, и в исходной вершине преобразуемого поддерева. Если в худшем случае все эти вершины расположены в разных произвольных блоках, то требуется по четыре операции произвольного чтения и последовательиой перезаписи.

Верхняя оценка:

$$\text{РВА (ВКЛЮЧИТЬ, } AVL\text{-дерево)} = \begin{cases} 1\,\text{sba}, & \text{если все обновляемые вершины размещены в одном блоке,} \\ 5\,\text{sba} + 5\,\text{rba}, & \text{если они расположены в разных произвольных блоках.} \end{cases} \qquad (14\text{-}11)$$

Таблица 14.2. **Оценка операций базы данных (количества обращений к физическим блокам) для бинарных деревьев поиска** [1)]

	Линейное	Несбалансированное	Сбалансированное	AVL
ПОЛУЧИТЬ ВСЕ	←	{ВО : NR rba НО : NR/EBF sba}		→
ПОЛУЧИТЬ СЛЕДУЮЩУЮ ПОЛУЧИТЬ ПРЕДЫДУЩУЮ	←	{ВО : $\log_2$ NR rba НО : 0}		→
		ВО:	ВО:	ВО:
ПОЛУЧИТЬ УНИКАЛЬНУЮ (ключ найден)	(1 + NR/2)	1,4 $\log_2$ (NR − 1) rba	$\log_2$ (NR + 1) − 2 rba	$\log_2$ (NR + 1) rba
	←	$\left\{\text{НО} : \frac{\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ)}}{\log_2 \text{EBF}} \text{ rba}\right\}$		→
ПОЛУЧИТЬ УНИКАЛЬНУЮ′ (ключ не найден)	(1 + NR)/2	1,4 $\log_2$ NR rba	$\log_2$ (NR + 1) rba	1,44 $\log_2$ (NR + 1) rba
	←	$\left\{\text{НО} : \frac{\text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ′)}}{\log_2 \text{EBF}} \text{ rba}\right\}$		→
ПОЛУЧИТЬ RPBR записей (пакет)	←	как для связанной последовательной организации (см. табл. 12.3)		→

ПОЛУЧИТЬ НЕКОТОРЫЕ (булевый запрос)	как для ПОЛУЧИТЬ ВСЕ			
После завершения поиска:				
ИЗМЕНИТЬ неключевое значение	← 1 sba →			
ИЗМЕНИТЬ ключ	← ПОЛУЧИТЬ УНИКАЛЬНУЮ + УДАЛИТЬ + ПОЛУЧИТЬ УНИКАЛЬНУЮ + ВКЛЮЧИТЬ →			
ВКЛЮЧИТЬ запись				
НО:	1 sba	1 sba	1 sba	1 sba
ВО:	1 rba + 2 sba	1 rba + 1 sba	0 (NR) rba	5 rba + 5 sba
УДАЛИТЬ запись				
НО:	1 sba	1 sba	1 sba	1 sba
ВО:	1 sba + NR/2 rba	1,4 $\log_2$ (NR − 1) − − 1 rba + 1 sba	0 (NR) rba	$\log_2$ NR (3 rba + 3 sba)
ОБНОВИТЬ ПАКЕТ	← Сумма отдельных оценок обновления RPBR записей →			

[1]) ВО — верхняя оценка, НО — нижняя оценка.

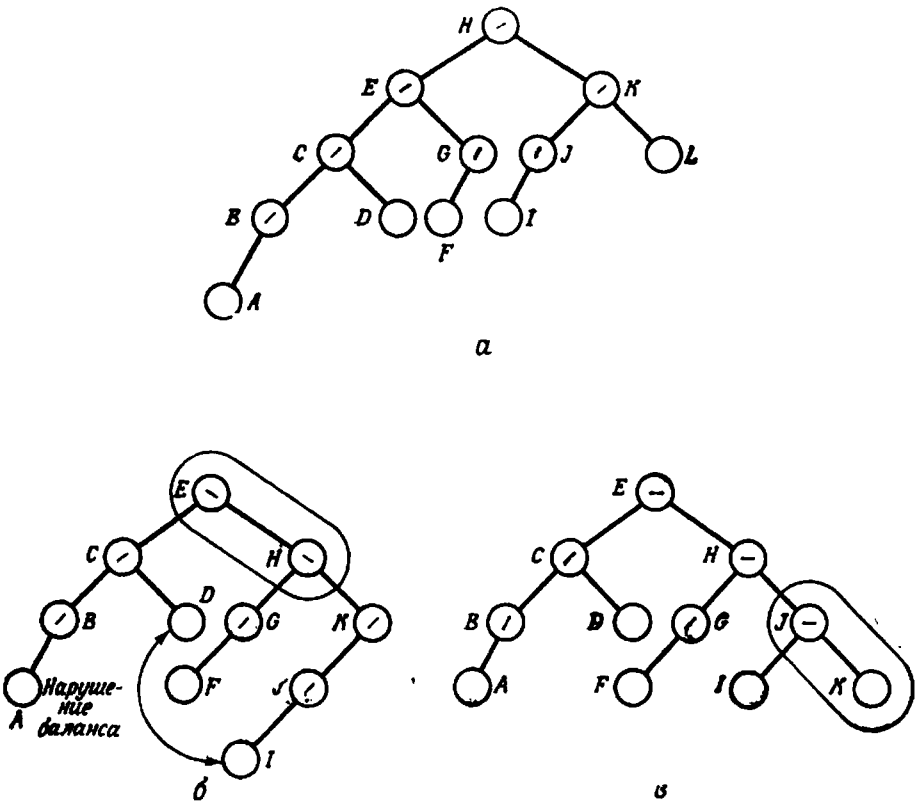


Рис. 14.7. Удаление записи из *AVL*-дерева:
а — до удаления вершины *L*; *б* — первый поворот (*E*, *H*); *в* — второй поворот (*J*, *K*).

Удаление записи из *AVL*-дерева может потребовать до $\log_2 NR$ преобразований. Эта операция значительно сложнее операции включения; с математической и алгоритмической точек зрения она изучена очень мало. Для пояснения проблемы надо заметить, что для удаления вершины *L* из дерева на рис. 14.7 требуется выполнить два одинарных поворота отдельно для пар вершин *E*, *H* и *J*, *K*. Нижняя оценка проста: при удалении вершины-листа и отсутствии нарушения баланса достаточно выполнить одну операцию последовательной перезаписи блока, содержащего исходную вершину. При расчете верхней оценки необходимо учесть $\log_2 NR$ поворотов, три вершины для каждого поворота (две вершины, участвующие в повороте, и исходную вершину преобразуемого поддерева) и тот факт, что все эти вершины, возможно, расположены в разных

блоках и требуют отдельных операций чтения и перезаписи.

$$1\,\text{sba} \leqslant \text{РВА (УДАЛИТЬ, } AVL\text{-дерево)} \leqslant$$
$$\leqslant \log_2 \text{NR}\,(3\,\text{sba} + 3\,\text{rba}). \qquad (14\text{-}12)$$

В табл. 14.2 приведены характеристики производительности бинарных деревьев поиска.

14.1.3. Объем памяти

Объем памяти, требуемой для бинарного дерева поиска, зависит от организации его хранения: несблокированные записи, сблокированные записи с последовательным распределением блоков или сблокированные записи с групповым распределением блоков. Вначале определим размер хранимой записи, предположив, что записи в дереве однородны по формату и размеру:

$$\text{SRS} = \text{KS} + \text{NKS} + 2 \times \text{PS} \text{ байт.} \qquad (14\text{-}13)$$

В случае несблокированной организации базы данных объем требуемой памяти равен

$$\text{BLKSTOR} = \text{SRS} \times \text{NR} \text{ байт.} \qquad (14\text{-}14)$$

В случае сблокированной организации базы данных и последовательного распределения блоков объем памяти составляет

$$\text{BLKSTOR} = \left\lceil \frac{\text{NR}}{\text{EBF}} \right\rceil \times \text{BKS} \text{ байт.} \qquad (14\text{-}15)$$

Реализация группового метода распределения блоков допускает много вариантов, обеспечивающих различную степень плотности размещения записей в блоках. Очевидно, что нижняя оценка определяется выражением (14-15). Грубая верхняя оценка равна NR блокам. Однако эту оценку можно уменьшить, используя тот факт, что в худшем случае для каждого заполненного блока существует одна запись «переполнения», помещаемая в отдельный блок. Так что общее количество записей в двух блоках равно EBF + 1 или в среднем на блок (EBF + 1)/2. Вследствие этого оценки объема требуемой памяти в случае группового распределения блоков таковы:

$$\left\lceil \frac{\text{NR}}{\text{EBF}} \right\rceil \times \text{BKS} \leqslant \text{BLKSTOR} \leqslant \left\lceil \frac{\text{NR}}{(\text{EBF} + 1)/2} \right\rceil \times \text{BKS} \text{ байт.} \qquad (14\text{-}16)$$

14.2. *B*-дерево

Структура *B*-дерева представляет широко распространенную структуру организации и управления индексами большой размерности. Подобно бинарному дереву поиска, эта структура

обеспечивает высокую производительность операций произвольных выборки и обновления, сохраняя при этом возможность эпизодической последовательной обработки без предварительной реорганизации данных. *B*-дерево относится к категории так называемых многоходовых деревьев, в которых допускается более двух ветвей, исходящих из одной вершины. Индексно-последовательная организация также является одной из разновидностей миогоходового дерева: каждый индекс можно рассматривать как вершину дерева с исходящей из нее ветвью для каждого ключевого значения (блока), хранимого в индексе. *B*-дерево представляет обобщение этого понятия и допускает различные размеры вершин, количество исходящих ветвей и количество уровней вершин. Каждая вершина *B*-дерева состоит из совокупности значений первичного ключа, указателей индекса и ассоциированных данных. Указатели индекса используются для перехода на следующий, более низкий уровень вершии в *B*-дереве. «Хранимые» в вершине ассоциированные данные фактически представляют собой совокупность указателей данных и служат для определения физического местоположения данных, ключевые значения которых храиятся в этой вершине индекса. Хотя записи данных можно было бы разместить непосредственио в вершинах индекса, но для баз данных большой размерности такой способ оказался бы неэффектнвным. Обычно *B*-деревья используют только для организации индекса; записи данных располагаются в отдельной области, для которой предусмотрена возможность произвольного доступа.

В начале 70-х годов были проведены серьезные исследования структуры *B*-дерева [20, 21, 89]. Эта структура оказалась очень перспективной для организации эффективного хранения во вторичной памяти индексов большой размерности и обеспечила быструю произвольную выборку записей. Фирма IBM использовала некоторые принципы *B*-дерева при реализации метода доступа VSAM [89, 166, 167, 179, 331]. В последние годы этой структуре также уделялось много внимания, а способы ее применения продолжают разрабатываться и сегодня.

Основные достоинства *B*-дерева

1. Во всех случаях полезное использование пространства вторичной памяти составляет свыше 50 %. Обеспечивается динамическое распределение и использование вторичной памяти; с ростом степени полезного использования памяти не происходит снижения качества обслуживания.
2. Произвольный доступ к записи реализуется посредством малого количества подопераций (обращений к физическим блокам) и по эффективности сопоставим с методами хеширования и многоуровневого индекса.

3. В среднем достаточно эффективно реализуются операции включения и удаления записей; при этом сохраняется лексикографический (естественный) порядок ключей с целью последовательной обработки, а также соответствующий баланс дерева для обеспечения быстрой произвольной выборки.
4. Неизменная упорядоченность по ключу обеспечивает возможность эффективной пакетной обработки.

Осиовной недостаток *B*-деревьев состонт в отсутствии для них средств выборки данных по вторичному ключу. Однако можио надеяться, что в будущем вторичные методы доступа включат обслуживание структуры *B*-дерева как часть своей общей схемы наравне с компонентами поиска в индексе.

Определение

B-дерево представляет собой обобщение понятия бинарного дерева: в нем из каждой вершины могут выходить две и более ветвей. *B*-дерево *степени k* обладает следующими свойствами:
1. Все пути от корневой вершины до вершин-листьев имеют одинаковую длину h, называемую также высотой *B*-дерева (то есть h есть количество вершии от корня до листа включительно).
2. Для каждой вершины, за исключением корня и листьев, количество подчиненных ей вершин должно быть не меньше $k+1$ и не больше $2k+1$.
3. Для корневой вершины количество подчиненных ей вершин должно быть не меиьше двух и не больше $2k+1$.
4. В каждой вершине, за исключением корня, количество ключей должно быть не меньше k и не больше $2k$. В корне допускается только один ключ. В общем случае любая неконечная (ветвящаяся) вершина с j ключами должна иметь $j+1$ подчиненых вершин.

На основании этих свойств легко видеть, что *B*-дерево степени 1, в котором из каждой вершины выходят в точности две ветви, представляет собой полностью сбалансированное бинарное дерево поиска. Вследствие переменного количества ключей в вершинах *B*-деревья оказываются более гибкими в сравнении с бинарными деревьями. Так, многие операции включения и удаления можно выполнять без последующего преобразования дерева. Однако в некоторых случаях преобразование требуется (см. разд. 14.2.2).

Физическая организация ветвящейся вершины *B*-дерева подобна физически последовательной структуре, как показано на рис. 14.8. При реализации каждый блок (или страница), как правило, содержит только одну вершину; если вообще имеет смысл использовать понятие коэффициента блокирования, оно здесь представлено в виде максимального количества ключевых элементов: NKEY $= 2k$. Количество действительных ключевых

элементов, показанных на рис. 14.8, равно $NK \leqslant 2k$; следовательно, NKEY — NK ключевых позиций и соответствующих им указателей свободны. Ключевые значения обозначены через k_i, указатели индекса — p_i, а соответствующие данные (или указатели данных) — α_i. В данном разделе предполагается, что α_i обозначает указатель записи данных, расположенной в файле произвольного доступа.

На рис. 14.9, *а* представлена совокупность вершин, составляющих *B*-дерево степени 1. В явном виде показаны действительные ключевые значения, а указатели ветвей изображены

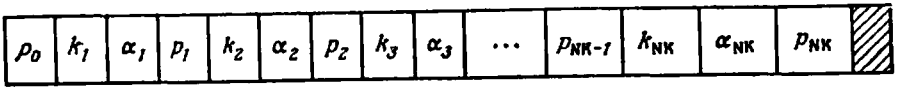


Рис. 14.8. Структура ветвящейся вершины *B*-дерева с NK ключами.

стрелками, обозначающими направление путей доступа. Ассоциированные данные, присущие вершинам *B*-дерева, на рисунке не показаны. На рис. 14.9, *б* и *в* приведены примеры *B*-деревьев соответственно степеней 2 и 3. Надо заметить, что полезное использование вершин варьируется от k до $2k$ ключей, а все листья располагаются в дереве на одинаковом уровне. Любое *B*-дерево всегда остается полностью сбалансированным по высоте. Отметим также тот факт, что независимо от степени дерева корень может иметь самое меньшее один ключ.

В сравнении с бинарным деревом поиска, изображенным на рис. 14.3, можно отметить уменьшение высоты *B*-деревьев, представленных на рис. 14.9, особенно по мере возрастания степени *B*-дерева. Следовательно, при условии размещения каждой вершины в отдельном блоке или на отдельной странице *B*-дерево обеспечивает потенциальное ускорение операции выборки по сравнению с бинарным деревом поиска. За тем исключением, что для каждой вершины допускается несколько исходящих ветвей, механизм поиска для *B*-дерева аналогичен механизму поиска в бинарном дереве поиска. Выполняется просмотр упорядоченных ключей в вершине с целью поиска хранимого ключа (индекса), равного или большего искомого ключа. Если найден ключ, равный искомому, поиск считается успешным и для определения местоположения записи данных используется значение указателя данных. Если в позиции i найден ключ, больший искомого, то осуществляется переход на вершину следующего уровня в соответствии со значением предыдущего указателя индекса p_{i-1} и повторяется просмотр выбранной вершины. Если же все хранимые в вершине ключи меньше иск мого, то осуществля-

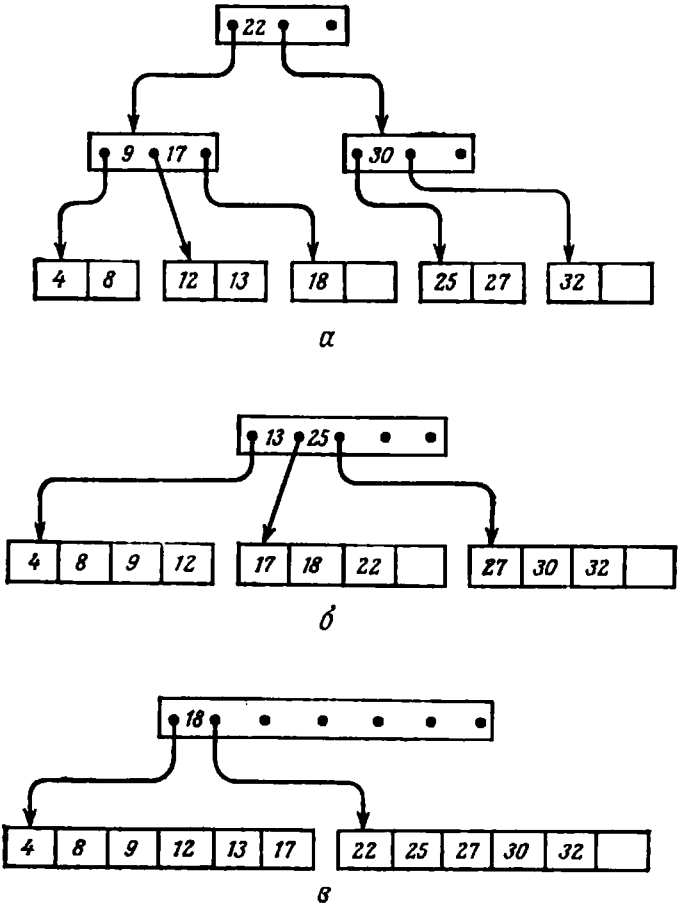


Рис. 14.9. Возможные конфигурации *B*-дерева для файла с 12 ключами: *а* — *B*-дерево степени 1 ($h = 3$); *б* — *B*-дерево степени 2 ($h = 2$); *в* — *B*-дерево степени 3 ($h = 2$).

ется переход на следующий уровень в соответствии со значением самого правого указателя индекса (p_{NK}). Если искомый ключ не обнаружен в вершине-листе, поиск завершается безрезультатно. На рис. 14.9 поиск ключа 25 потребует в случае *а* трех обращений к блокам, в случае *б* — одного обращения к блоку и в случае *в* — двух обращений к блокам.

При проектировании *B*-дерева индекса необходимо учесть следующие факторы:

• *Тип обработки.* Наиболее эффективны произвольные выборка и обновление; допускаются последовательная и пакетная обработки; обработка булевых запросов медленная.

• *Изменчивость данных.* Структура *B*-дерева и правила баланса хорошо приспособлены к интенсивному обновлению данных.
• *Степень B-дерева.* Оказывает значительное влияние на время поиска; обеспечивает средство управления полезным использованием блока (вершины) и эффективностью включения новых записей данных.
• *Размер файла.* *B*-дерево обеспечивает эффективную структуру для файлов большой размерности.
• *Фиксированная или переменная длина ключа.*
• *Размер указателей.*
• *Методы обработки переполнения при обновлении.*

Указанные вопросы анализируются ниже.

14.2.1. Производительность выборки

Большое разнообразие конфигураций, реализующих *B*-дерево, затрудняет расчет средней длины пути доступа. Напомним читателю, что каждая вершина в *B*-дереве степени k может содержать от k ко $2k$ ключей включительно; эти колебания оказывают сильное влияние на форму дерева. К тому же форма дерева постоянно меняется вследствие операций включения и удаления записей, поэтому сама оценка может значительно ме-

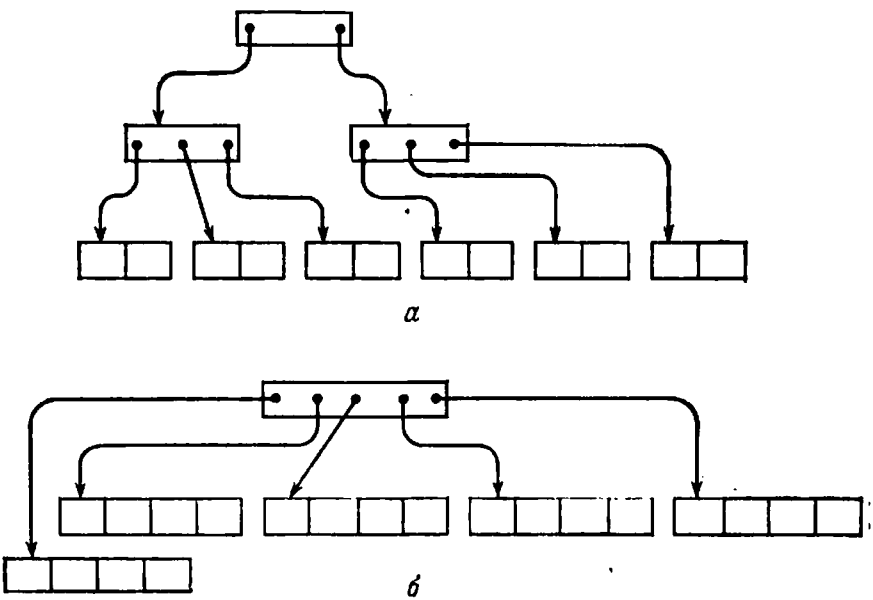


Рис. 14.10. Характеристики *B*-дерева степени $k = 2$:
а — максимальная высота; *б* — минимальная высота.

няться со временем. При таких условиях наиболее стабильной мерой производительности являются верхняя и нижняя оценки длины пути доступа. Кроме того, многоходовая архитектура *B*-дерева приводит к тому, что большинство ключей располагается на уровне h. Вследствие этого средняя длина пути становится близка максимальной длине пути. Вначале определим верхнюю и нижнюю оценки высоты *B*-дерева. Это позволит затем определить верхнюю и нижнюю оценки производительности выборки.

Высота *B*-дерева: верхняя оценка

На рис. 14.10, *а* приведен худший случай конфигурации, которая имеет максимальную высоту *B*-дерева степени k с NR ключами для $k = 2$. В этом худшем случае корневая вершина содержит только один ключ, а каждая ветвящаяся вершина содержит только k ключей. В табл. 14.3 приведены данные о ко-

Таблица 14.3. **Характеристики *B*-дерева максимальной высоты**

Уровень	Количество вершин	Количество ключей
1	1	1
2	2	$2k$
3	$2(k+1)$	$(2k)(k+1)$
4	$2(k+1)^2$	$(2k)(k+1)^2$
.	.	.
.	.	.
.	.	.
h	$2(k+1)^{h-2}$	$(2k)(k+1)^{h-2}$

личестве вершин и ключей для каждого уровня. При высоте дерева h в каждой вершине на уровне h находится не менее k ключей, и *минимальная сумма* ключей на уровнях с 1 по h меньше или равна NR:

$$
\begin{aligned}
&1 + 2k + (2k)(k+1) + (2k)(k+1)^2 + \ldots + \\
&\qquad + (2k)(k+1)^{h-2} \leqslant \mathrm{NR}, \\
&1 + (2k)[1 + (k+1) + (k+1)^2 + \ldots + (k+1)^{h-2}] \leqslant \mathrm{NR}, \\
&1 + (2k)\left[\frac{(k+1)^{h-1} - 1}{(k+1) - 1}\right] \leqslant \mathrm{NR}, \\
&1 + 2[(k+1)^{h-1} - 1] \leqslant \mathrm{NR}.
\end{aligned}
\tag{14-17}
$$

Перестановкой членов получаем

$$(k+1)^{h-1} \leqslant \frac{NR+1}{2}.$$

Логарифмируя обе стороны неравенства по основанию $k+1$, получаем

$$h \leqslant \log_{k+1}\left(\frac{NR+1}{2}\right)+1 \text{ и } h - \text{целое число.} \quad (14\text{-}18)$$

$$\text{Следовательно, } h \leqslant \left\lfloor \log_{k+1}\left(\frac{NR+1}{2}\right)\right\rfloor + 1. \quad (14\text{-}19)$$

Высота *B*-дерева: нижняя оценка

На рис. 14.10, *б* показан пример *B*-дерева минимальной высоты, а его характеристики приведены в табл. 14.4. В этом случае каждая вершина, включая и корневую вершину, содержит максимальное количество ключей, равное $2k$. При высоте дерева h максимальная сумма ключей на уровнях с 1 по h больше или равна общему количеству хранимых ключей NR:

Таблица 14.4. **Характеристики *B*-дерева минимальной высоты**

Уровень	Количество вершин	Количество ключей
1	1	$2k$
2	$2k+1$	$(2k)(2k+1)$
3	$(2k+1)^2$	$(2k)(2k+1)^2$
4	$(2k+1)^3$	$(2k)(2k+1)^3$
.	.	.
.	.	.
.	.	.
h	$(2k+1)^{h-1}$	$(2k)(2k+1)^{h-1}$

$$\begin{gathered}
2k+(2k)(2k+1)+(2k)(2k+1)^2+\ldots+ \\
+(2k)(2k+1)^{h-1} \geqslant NR, \\
2k[1+(2k+1)+(2k+1)^2+\ldots+(2k+1)^{h-1}] \geqslant NR, \\
2k\left[\frac{(2k+1)^h-1}{(2k+1)-1}\right] \geqslant NR, \\
(2k+1)^h \geqslant NR+1,
\end{gathered} \quad (14\text{-}20)$$

$$h \geqslant \log_{2k+1}(NR+1) \text{ и } h - \text{целое число.} \quad (14\text{-}21)$$

$$\text{Следовательно, } h \geqslant \lceil \log_{2k+1}(NR+1) \rceil. \quad (14\text{-}22)$$

Например, в случае NR = 17 верхняя и нижняя оценки высоты B-дерева степени 2 таковы:

$$\lceil \log_5 (17+1) \rceil \leqslant h \leqslant \left\lfloor \log_3 \left(\frac{17+1}{2} \right) \right\rfloor + 1 \quad \text{и} \quad 2 \leqslant h \leqslant 3.$$

В случае NR = 24 имеем оценки:

$$\lceil \log_5 (24+1) \rceil \leqslant h \leqslant \left\lfloor \log_3 \left(\frac{24+1}{2} \right) \right\rfloor + 1 \quad \text{и} \quad 2 \leqslant h \leqslant 3.$$

Справедливость оценок для каждого из этих случаев можно неформально проверить. Для NR = 17 пример на рис. 14.10, *б* показывает, что h должно быть не меньше 2, а пример на рис. 14.10, *а*, что h должно быть не больше 3 (при 100 %-ном полезном использовании структуры). Для NR = 24 при 100 %-ном полезном использовании структуры все еще можно разместить все ключи в дереве с $h = 2$ уровнями (рис. 14.10, *б*), а при полезном использовании меньше 100 % потребуется h не больше 3 (рис. 14.10, *а*). Из табл. 14.3 легко видеть, что минимальное количество ключей, для которых в качестве верхней оценки требуется $h = 4$ уровней, равно $(2k)(k+1)^2 = 36$ для $k = 2$. Заметим, что в AVL-дереве средняя длина поиска h равняется $\log_2(24+1) = 4{,}65$. Очевидно, что даже в хорошо сбалансированном бинарном дереве поиска время произвольной выборки оказывается больше, чем в B-дереве.

Производительность выборки составляет h обращений на пути к вершине, содержащей адрес записи данных, и еще одно обращение к самой записи данных. Однако в случае неуспешного поиска последний не выполняется. В предположении, что каждая вершина B-дерева занимает физический блок, можно получить следующую верхнюю оценку производительности выборки:

РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ) =

$$= h + 1 \leqslant \left\lfloor \log_{k+1} \left(\frac{\text{NR}+1}{2} \right) \right\rfloor + 2 \text{ rba}. \qquad (14\text{-}23)$$

Нижняя оценка имеет место, если искомый ключ обнаружен среди ключей корневой вершины. В этом случае требуются два произвольных обращения к блоку: один к корневой вершине (поиск в индексе) и один к блоку записи данных. В случае неуспешного поиска верхняя и нижняя оценки эквивалентны:

РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ') =

$$= h \leqslant \left\lfloor \log_{k+1} \left(\frac{\text{NR}+1}{2} \right) \right\rfloor + 1 \text{ rba}. \qquad (14\text{-}24)$$

При последовательной обработке B-дерева, так же как при последовательной обработке бинарного дерева поиска, исполь-

зуется алгоритм обратного обхода дерева, при котором сначала выполняется обход левого поддерева, затем корня, а затем правого поддерева. Если из кориевой вершины поддерева выходит несколько путей, обход поддеревьев выполняется слева направо с промежуточным посещением корня, так что лексикографический порядок сохраняется. На практике целесообразно было бы хранить корневую вершину в буфере, так что для ее выборки оказалось бы достаточно одного обращения к блоку. При таком подходе потребовалось бы предусмотреть область буфера, достаточно большую для размещения $h+1$ вершин или блоков. В B-дереве максимальной высоты каждая вершина содержит только k ключей (за исключением корня, содержащего один ключ). В B-дереве минимальной высоты каждая вершина содержит $2k$ ключей. Следовательно, верхняя и нижняя оценки последовательной обработки NR произвольно расположенных записей данных равны сумме количеств обращений к вершинам индекса и к записям (блокам) данных:

$$\frac{\text{NR}}{2k}+\text{NR}\leqslant \text{РВА(ПОЛУЧИТЬ ВСЕ)}\leqslant\frac{\text{NR}}{k}+\text{NR rba.} \quad (14\text{-}25)$$

Блокирование записей данных позволило бы значительно уменьшить эти оценки.

Операция ПОЛУЧИТЬ СЛЕДУЮЩУЮ (а также операция ПОЛУЧИТЬ ПРЕДЫДУЩУЮ) эквивалентна каждому из NR компонентов операции ПОЛУЧИТЬ ВСЕ:

$$\text{РВА(ПОЛУЧИТЬ СЛЕДУЮЩУЮ)}=$$
$$=\text{РВА(ПОЛУЧИТЬ ВСЕ)/NR.} \quad (14\text{-}26)$$

В случае лексикографической упорядоченности файла транзакций для B-дерева допустимы также операции выборки пакета. В худшем случае для выборки пакета RPBR записей требуется выполнить обращения к RPBR блокам данных, а также обращения ко всем вершинам B-дерева:

$$\text{РВА(выборка пакета RPBR записей)}=$$
$$=\text{РВА(ПОЛУЧИТЬ ВСЕ)}-(\text{NR}-\text{RPBR})\,\text{rba.} \quad (14\text{-}27)$$

14.2.2. Производительность обновления

Вследствие необходимости соблюдения определенного условия баланса многие характеристики обновления B-дерева и бинарного дерева поиска совпадают. В худшем случае восстановление баланса после операций включения и удаления может затронуть вершины по всей высоте дерева. Как обычно, простейший случай обновления представляет изменение неключевого элемента в записи, выполняемое после того, как изменяемая запись выбрана механизмом поиска. Для занесения обнов-

ленного блока записей в базу данных требуется только одна простая операция перезаписи блока:

$$\text{РБА (ИЗМЕНИТЬ неключевое значение)} = 1\ \text{sba}. \qquad (14\text{-}28)$$

В случае если изменение значения ключевого элемента не вызывает перестановки ключевых значений в вершинах B-дерева, его выполнение потребует одной операции перезаписи блока данных (sba) и еще одной операции перезаписи блока индекса (sba). Максимальная оценка, как правило, имеет место при замене значения ключа на произвольное новое значение; при этом необходимо удалить старый элемент индекса, затем найти соответствующее место и включить новый элемент индекса и, наконец, перезаписать измененный блок данных:

$$\begin{aligned} 2\,\text{sba} \leqslant\ & \text{РВА (ИЗМЕНИТЬ ключ)} \leqslant \\ \leqslant\ & \text{РВА (УДАЛИТЬ элемент индекса)} + \\ & + \text{РВА (ПОЛУЧИТЬ УНИКАЛЬНУЮ)} + \\ & + \text{РВА (ВКЛЮЧИТЬ элемент индекса)} + \\ & + \text{РВА (перезаписать блок данных)}. \end{aligned} \qquad (14\text{-}29)$$

Теперь оценим операции удаления и включения.

По операции ВКЛЮЧИТЬ выполняется поиск соответствующего блока индекса, включение в него нового значения ключа, перезапись измененного блока индекса, выделение пространства памяти для нового блока данных и занесение записи данных на выделенное место. Вначале проанализируем операции, касающиеся блока данных. Выделение памяти для нового блока данных является функцией системного программного обеспечения и требует только внутренней обработки. Для занесения новой записи данных в качестве блока в файл требуется одно произвольное обращение к блоку (rba). Если при этом записи данных организованы в блоки, то необходимо прочитать соответствующий блок (rba), в рабочей области оперативной памяти включить в него новую запись данных и затем повторно записать блок (sba). Следовательно, если записи данных сблокированы, операция включения требует дополнительно 1 sba.

Операция включения элемента индекса в B-дерево значительно сложнее из-за того, что по завершении поиска возможно несколько вариантов. В силу того, что новый элемент индекса можно включить только в вершину-лист, операция поиска в любом случае состоит из h подопераций. По завершении поиска возможны четыре случая:

• **Случай 1.** Простейший случай, когда в вершине-листе есть место для нового ключевого значения. Помимо первоначального обращения к вершине требуется операция последовательной

записи (sba) для возврата обновленной вершины на соответствующую физическую позицию.

• **Случай 2.** Вершина-лист содержит $2k$ ключей. В этом случае ее необходимо разделить на две вершины, каждая из которых содержит k ключей. После включения нового ключа всего ключей станет $2k + 1$, но один из них должен быть включен в вершину, исходную по отношению к разделяемой вершине. В предположении, что исходная вершина неполна, получаем следующие подоперации:

а. Включение ключа в исходную вершину (она находится в буфере) и ее перезапись: 1 sba.

б. Перезапись разделенной вершины: 1 sba.

в. Создание новой вершины с k ключами и ее запись: 1 rba.

• **Случай 3.** Исходная вершина тоже полна. Следовательно, ее также необходимо разделить на две; в худшем случае процесс деления вершин может распространиться на все вершины вплоть до корня; при этом высота дерева увеличится на единицу. В худшем случае будут выполнены подоперация 2а для нового корня и подоперации 2б и 2в h раз каждая. На рис. 14.11 приведены примеры случая 1 (ключ 7), случая 2 (ключ 37) и случая 3 (ключ 57).

• **Случай 4.** Вместо разделения заполненной вершины с целью включения нового ключа можно проанализировать правую подобную ей вершину на предмет наличия в ней свободного места для возможной перегруппировки ключей с участием заполненной вершины. Если в ней свободны не меньше двух ключевых позиций, то соответствующее количество ключей из заполненной вершины можно сдвинуть в исходную вершину, вытеснив из нее такое же количество ключей в правую подобную вершину; при этом сохраняются и лексикографическая упорядоченность вершин, и приблизительно одинаковая степень полезного использования подобных вершин. Данный подход иллюстрируется на рис. 14.12. Очевидно, что для включения одного ключа требуются три подоперации:

а. Перемещение нового значения в исходную вершину (она находится в буфере) и перезапись соответствующего блока: 1 sba.

б. Перезапись «переполненной» вершины: 1 sba.

в. Доступ к правой подобной вершине, включение в нее ключевого значения и перезапись: 1 rba + 1 sba.

Фактически оценка метода перераспределения «переполненной» вершины больше, чем метода разделения вершины, при условии, что разделение не распространяется вверх по дереву. С другой стороны, сочетание обоих методов могло бы повысить эффективность операции включения, позволяя по возможности избегать излишнего разделения вершин. В работе [188] описан

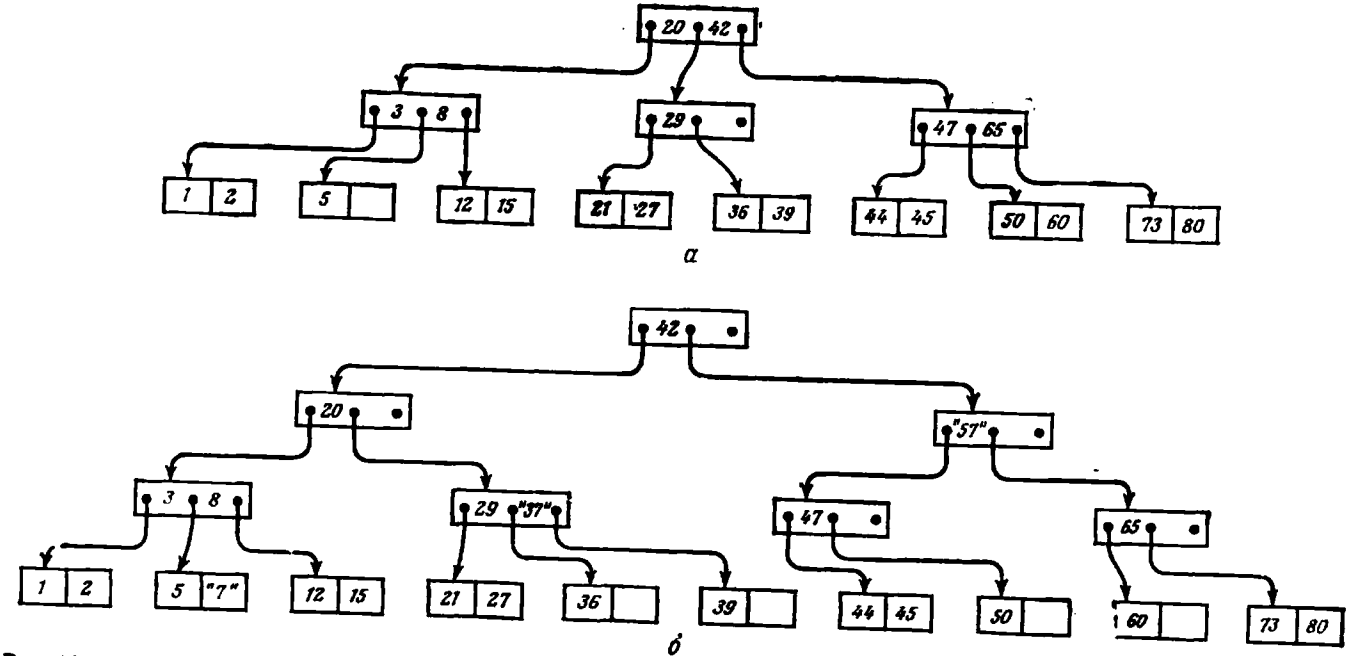


Рис. 14.11. Включение ключей в *B*-дерево с возможным делением вершин (случаи 1—3):
а — до включения ключей 7, 37, 57; *б* — после включения и деления вершин вплоть до корня (вследствие включения ключа 57).

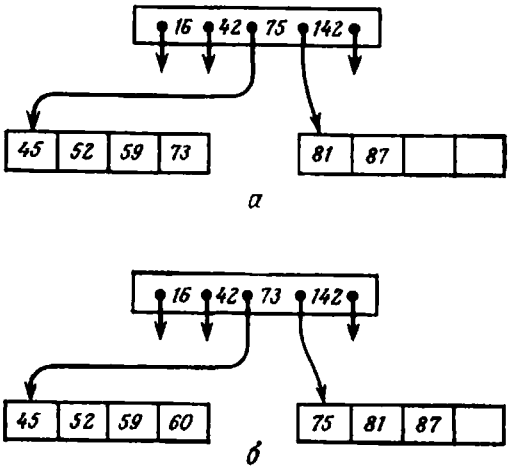


Рис. 14.12. Переполнение *B*-дерева вследствие включения ключа — способ перераспределения ключей (случай 4):

а — до включения ключа 60; *б* — после включения и перераспределения ключей.

вариант *B*-дерева, который иногда называют *B**-деревом (не следует путать с понятием *B**-дерева из разд. 14.2.4); в нем каждая вершина может быть заполнена как минимум почти на две трети; если помимо переполненной вершины полна и правая подобная ей вершина, то из этих двух вершин строятся три вершины. Как следствие повышается степень полезного использования памяти и уменьшается средняя высота дерева. При этом также, возможно (но необязательно), улучшается производительность обновления.

Подводя итог вышеизложенному, можно привести следующие оценки операции включения:

$$\text{РВА (ВКЛЮЧИТЬ)} = \text{РВА (ВКЛЮЧИТЬ ключ в индекс)} + \text{РВА (ВКЛЮЧИТЬ запись данных)}, \quad (14\text{-}30)$$

$$\text{РВА (ВКЛЮЧИТЬ ключ в индекс)} = \begin{cases} 1\,\text{sba} & \text{для случая 1: тот же блок,} \\ 1\,\text{rba} + 2\,\text{sba} & \text{для случая 2: разделение одной вершины,} \\ 1\,\text{sba} + h(1\,\text{rba} + 1\,\text{sba}) & \text{для случая 3: разделение } h \text{ вершин,} \\ 1\,\text{rba} + 3\,\text{sba} & \text{для случая 4: перераспределение переполненной вершины,} \end{cases} \quad (14\text{-}31)$$

$$\text{РВА (ВКЛЮЧИТЬ запись данных)} = \begin{cases} 1\,\text{sba}, & \text{если записи не сблокированы,} \\ 1\,\text{rba} + 1\,\text{sba}, & \text{если записи сблокированы.} \end{cases} \quad (14.32)$$

Необходимость удаления ключа из *B*-дерева индекса возникает вследствие удаления записи из области данных. Как и в случае операции включения, необходимо рассмотреть несколько возможных ситуаций. В случае несблокированных записей данных удаление записи является следствием удаления ключа из индекса; следовательно, не требуется операций ввода-вывода. При этом, возможно, потребуется некоторое время CPU на перераспределение освобожденного блока данных в свободную

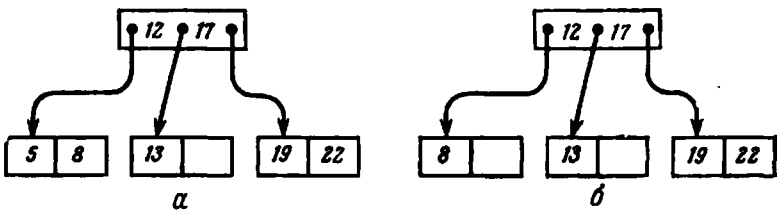


Рис. 14.13. Удаление ключа из вершины-листа *B*-дерева (случай 1):
а — до удаления вершины 5; *б* — после удаления.

память. Если же записи данных организованы в блоки, то операция удаления вызывает выполнение одной операции последовательной перезаписи этого блока.

Удалению ключа предшествует успешный поиск вершины, содержащей этот ключ. После завершения поиска возможны 3 случая:

• **Случай 1.** В простейшем случае ключ удаляется из вершины-листа и в ней по-прежнему остается не меньше k ключей (то есть отсутствует так называемое антипереполнение). После удаления ключа требуется простая операция перезаписи блока. На рис. 14.13 приведен пример этого случая.

• **Случай 2.** Более сложный случай представляет удаление ключа из ветвящейся вершины. Даже при отсутствии антипереполнения его необходимо заменить каким-то другим ключом в целях сохранения правого поддерева удаляемого ключа. Он замещается ключом из левой вершины-листа правого поддерева удаляемого ключа. (Надо заметить, что эта же процедура применяется при удалении ключа из несбалансированного бинарного дерева поиска.) Для этого при удалении ключа из кориевой вершины приходится просмотреть максимум h вершин. Как только найдена вершина-лист, левый ключ из нее удаляется и включается в новую вершину. Если удаление ключа из вершины-листа не вызывает антипереполнения, операция заканчивается. Итак, указаиная процедура включает в себя:

а. Поиск левой вершины-листа в правом поддереве: (минимум) 1 rba; (максимум) h rba.

б. Удаление ключа из вершины-листа (в случае отсутствия антипереполиения): 1 sba (для случая антипереполнения см. случай 3).

в. Удаление ключа из ветвящейся вершины с заменой его новым ключом, выбранным из вершины-листа.

На рис. 14.14 приведен пример случая 2.

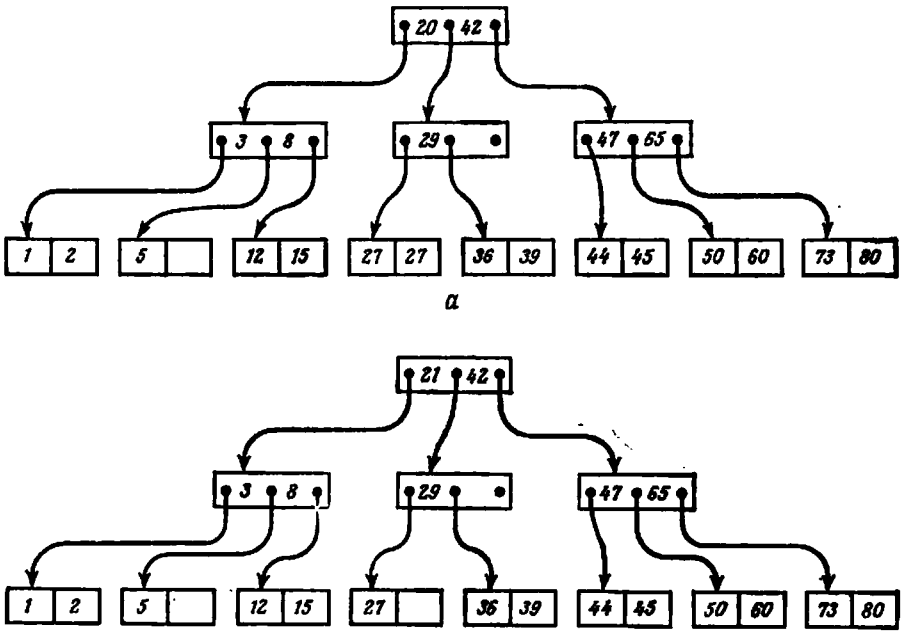


Рис. 14.14. Удаленне ключа из ветвящейся вершины *B*-дерева (случай 2): *а* — до удаления ветвящейся вершины 20; *б* — после удаления вершины 20 и замены ее вершиной-листом 21.

• **Случай 3.** Если удаление ключа из вершины-листа вызывает антипереполнение (в вершине остается меньше k ключей), недостающий ключ можно занять из левой или правой соседней (подобной) вершины. При этом в целях получения сбалансированного распределения ключей между двумя соседними вершинами и без дополнительных операций ввода-вывода можно занять несколько ключей. Последнее требует наличия в двух вершнах не менее $2k$ ключей. Если ключей меньше $2k$, то выполняется *сцепление* вершин. Это означает, что все ключи переносятся в одну из вершин, а другая вершина из дерева удаляется. Операция сцепления по своему действию противоположна операции деления вершин. Она подразумевает также пересылку в оставшуюся подобную вершину ключа из исходной вершины,

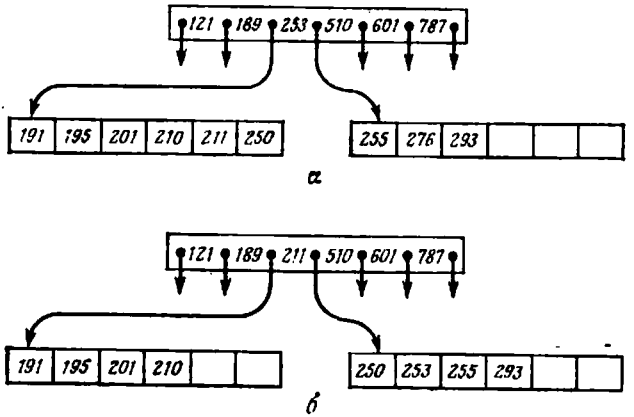


Рис. 14.15. Антипереполнение *B*-дерева вследствие удаления ключа из вершины-листа способом перераспределения ключей (случай 3а):

а — до удаления ключа 276; *б* — после удаления ключа 276 и перераспределения ключей в подобных вершинах.

разделяющего две соседние подобные вершины. К сожалению, удаление ключа из исходной вершины может вызвать в ней антипереполнение, и в худшем случае оно может распространиться вверх до корневой вершины. Итак, возможные операции для случая антипереполнения представляют:

а. Перераспределение ключей в подобных вершинах. Для этого требуется выполнить операцию чтения правой подобной вершины, операции последовательной перезаписи обеих подобных вершин и операцию перезаписи исходной вершины: **1 rba + 3 sba.**

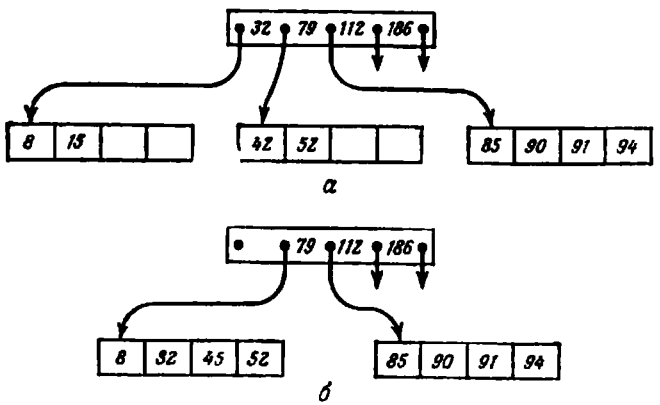


Рис. 14.16. Способ сцепления вершин *B*-дерева при удалении ключа из вершины-листа (случай 3б):

а — до удаления ключа 15; *б* — после удаления ключа 15 и сцепления подобных вершин.

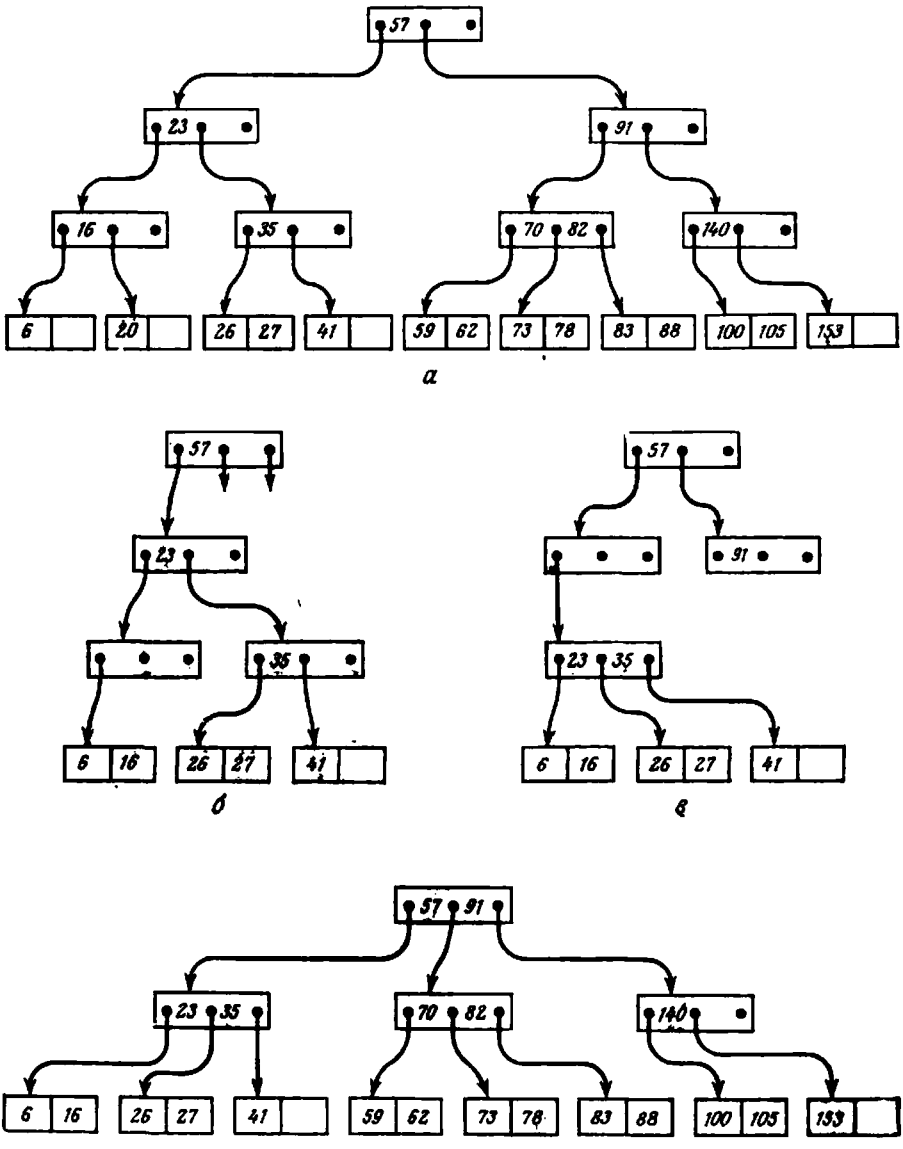


Рис. 14.17. Последовательность сцеплений вершин B-дерева (включая корень) при удалении ключа из вершины-листа (случай 3в):

а — до удаления ключа 20; *б* — после первого сцепления ключей 6 и 16; *в* — после второго сцепления ключей 23 и 25; *г* — структура B-дерева после третьего сцепления ключей 57 и 91 и организации новой корневой вершины.

б. Сцепление вершин на одном уровне. Для этого требуется выполнить операцию чтения второй подобной вершины, операцию перезаписи исходной вершины и операцию перезаписи оставшейся подобной вершины: 1 rba + 2 sba.

в. В худшем случае процесс сцепления вершин распространяется вверх до корня, то есть имеет место сцепление на $h-1$ уровнях. На каждом уровне требуется выполнить свою операцию чтения правой подобной вершины и перезаписи оставшейся вершины. Поскольку исходная вершина становится одной из подобных вершин для операции сцепления на следующем уровне, операция ее перезаписи не нужна: $(h-1)\times(1\ \text{rba}+1\ \text{sba})$. На рис. 14.15—14.17 приведены примеры для каждого из трех случаев антипереполнения: соответственно для случаев 3а, 3б и 3в.

$$\text{РВА (УДАЛИТЬ)} = \text{РВА (УДАЛИТЬ ключ из индекса)} + \text{РВА (УДАЛИТЬ запись данных)}, \quad (14\text{-}33)$$

$$\text{РВА (УДАЛИТЬ ключ из индекса)} = \begin{cases} 1\ \text{sba} & \text{для случая 1: вершина-лист, нет антипереполнения,} \\ 2\ \text{sba} + \text{поиск замещающего ключа (от 1 rba до } h\ \text{rba)} & \text{для случая 2: ветвящаяся вершина, нет антипереполнения,} \\ 1\ \text{rba} + 3\ \text{sba} & \text{для случая 3а: перераспределение вершин-листьев,} \\ 1\ \text{rba} + 2\ \text{sba} & \text{для случая 3б: сцепление вершин-листьев,} \\ (h-1)(1\ \text{rba} + 1\ \text{sba}) & \text{для случая 3в: сцепление вершин-листьев, распространенное до корня (худший случай),} \end{cases} \quad (14\text{-}34)$$

$$\text{РВА (УДАЛИТЬ запись данных)} = \begin{cases} 0, & \text{если записи не сблокированы,} \\ 1\ \text{sba}, & \text{если записи сблокированы.} \end{cases} \quad (14\text{-}35)$$

14.2.3. Объем памяти

Файлы *B*-дерева включают блоки ветвящихся вершин (BNODES), блоки вершин-листьев (LNODES) и блоки данных. Для каждого из трех компонентов можно использовать свой

размер блока и внутренний формат:

$$BKS_1(BNODES) = (2k) \times KS + [(2k+1) + (2k)] \times PS + BOVHD_1, \quad (14\text{-}36)$$

$$BKS_2(LNODES) = (2k) \times KS + (2k) \times PS + BOVHD_2, \quad (14\text{-}37)$$

$$BKS_3(\text{блоки данных}) = KS + NKS + BOVHD_3, \quad (14\text{-}38)$$

где $BOVHD_i$ содержит флажок, обозначающий соответственно вершины-листья, ветвящиеся вершины и блоки данных.

Для заданного общего количества ключей NR и B-дерева степени k оценка количества вершин листьев такова:

$$2(k+1)^{h_1-2} \leqslant LNODES \leqslant (2k+1)^{h_2-1}, \quad (14\text{-}39)$$

где $h_1 = \lceil \log_{2k+1}(NR+1) \rceil$ и $h_2 = \left\lfloor \log_{k+1}\left(\frac{NR+1}{2}\right) \right\rfloor + 1$

в соответствии с уравнениями (14-19) и (14-22). Эти оценки приведены в последней строке, соответствующей уровню h в табл. 14.3 и 14.4. Количество ветвящихся вершин можно оценить, используя соотношение

$$\frac{NR}{2k} \leqslant LNODES + BNODES \leqslant \frac{NR}{k} \quad (14\text{-}40)$$

и справедливое для B-дерева неравенство LNODES > BNODES. Следовательно, оценка общего объема памяти равна

$$BLKSTOR = BKS_1 \times BNODES + BKS_2 \times LNODES + BKS_3 \times NR \text{ байт}. \quad (14\text{-}41)$$

14.2.4. B-дерево*

B^*-дерево (или B+-дерево) представляет наиболее распространенный вариант B-дерева, характеризующийся тем, что ключи и ассоциированные данные расположены в вершинах-листьях, а доступ к ним осуществляется через B-дерево индекса. В B^*-дереве в качестве начальных разделителей в индексе используется подмножество ключей вершин-листьев, организованных в виде B-дерева. Таким образом в организации B^*-дерева наблюдается избыточное хранение некоторых ключей, но размещение всех ключей и записей данных в вершинах-листьях, объединенных в связанное последовательное упорядоченное множество, обеспечивает возможность более быстрой последовательной обработки. Основные сравнительные характернстики B^*-дерева и B-дерева приведены ниже [23, 89, 335]:

1. В обоих случаях загрузка индекса выполняется снизу вверх. Первые ключи загружаются на уровень h (в начальный момент $h = 1$), и после того, как происходит деление вершины, ключ-

разделитель помещается в вершину на уровень $h-1$; процесс повторяется, пока не будут загружены все начальные ключи. Ограничение на допустимое колнчество ключей в вершине B-дерева от k до $2k$ также относится н к B^*-деревьям; исключение в обоих случаях составляет корень.

2. Вершины-листья в B-дереве состоят из $k \leqslant \mathrm{NK} \leqslant 2k$ ключей и ассоциированных данных или указателей данных. Вершины-листья в B^*-дереве состоят из $k \leqslant \mathrm{NK} \leqslant 2k$ ключей, ассоциированных данных и одного указателя на следующую подобиую вершину-лист. При этом можно использовать двунаправленные указатели. Следовательно, форматы и размеры вершин-листьев в двух рассматриваемых физических организациях могут быть различны.

3. По существу, формат ветвящейся вершины в B-дереве и в B^*-дереве одинаков. Отличие B^*-дерева с точки зрения начальной загрузки (и операций включеиия) состоит в том, что все ключи хранятся в вершинах-листьях и, кроме того, часть их дублируется в индексе. В индексе содержится достаточное для обеспечения быстрого доступа к вершинам-листьям колнчество значений разделителей. На рис. 14.18 приведен пример поэтапной загрузки B-дерева и B^*-дерева. Если происходит деление вершины, то в обоих случаях в исходную вершину пересылается значение среднего ключа. Кроме этого, в B^*-дереве его копия помещается в левую часть правого листа исходной вершины. Процедура деления ветвящихся вершин в обеих организациях выполняется одинаково.

4. Операция выборки в B^*-дереве всегда занимает h доступов до вершины-листа. Если в вершине индекса обнаружено искомое значение ключа, осуществляется доступ к правому поддереву и поиск продолжается до вершины-листа. Наоборот, при обнаружении искомого ключа в B-дереве поиск может завершиться в середнне дерева.

5. Для операции удаления из B^*-дерева достаточно удалнть значение ключа из вершины-листа. Все копии этого ключа в индексе могут оставаться там, пока это полезно процессу выборки. Как следствие в качестве разделителей индекса в B^*-дереве могут выступать неключевые значения. С другой стороны, любой ключ может быть удален из исходиой (ветвящейся) вершины индекса в результате любой операции удаления, потребовавшей перераспределения или сцепления вершин. На рис. 14.19 приведены примеры возможных сценариев операции удаления.

Проанализируем производительность B^*-деревьев в сравнении с B-деревьями. Из-за отсутствия в ветвящихся вершинах ассоциированных данных (указателей данных) формат B^*-дерева оказывается несколько проще, чем формат B-дерева. Это

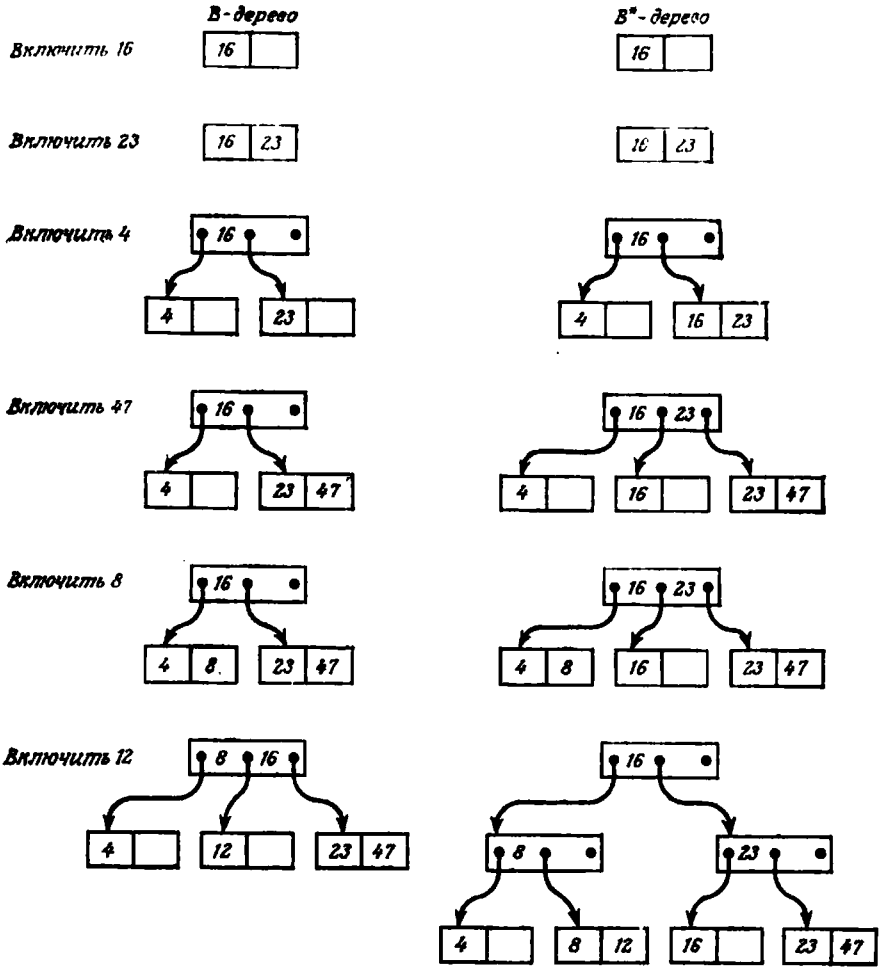


Рис. 14.18. Пример последовательности загрузки (операций включения) *B*-дерева и *B**-дерева степени 1.

означает, что при одном и том же физическом размере ветвящаяся вершина *B**-дерева может содержать больше ключей, чем вершина *B*-дерева. Указанное обстоятельство уравновешивает отрицательный эффект от избыточных ключей в индексе (см. рис. 14.18).

Перечисленные незначительные различия в двух организациях не оказывают особого влияния на производительность произвольной выборки, к тому же они уравновешивают друг друга.

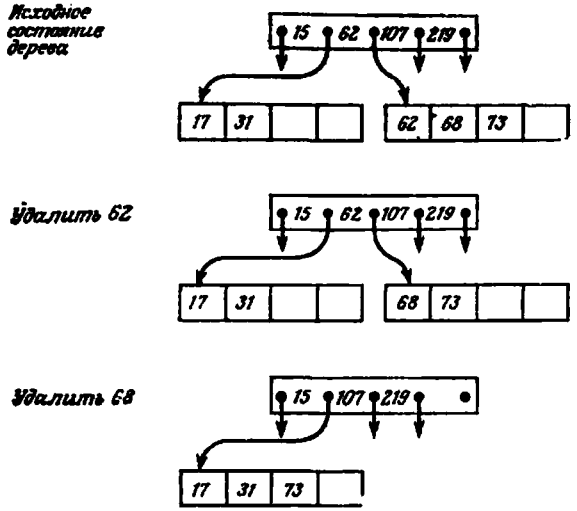


Рис. 14.19. Последовательность удаления ключей (и записи данных) из B^*-дерева степени 2.

С другой стороны, в случае неравномерного распределения операций выборки, при котором 80 % операций выборки адресованы к 20 % ключей в окрестности корня дерева, было бы лучше использовать организацию B-дерева. Для операции ПОЛУЧИТЬ СЛЕДУЮЩУЮ (а также ПОЛУЧИТЬ ПРЕДЫДУЩУЮ при использовании двунаправленного указателя) в случае B^*-дерева требуется одно произвольное обращение к упорядоченному множеству. Для операции ПОЛУЧИТЬ ВСЕ требуется одно обращение к каждой из вершин-листьев, число которых ограничено значениями из выражения (14-39). Эту операцию более эффективно можно реализовать в B^*-дереве, используя упорядоченное множество.

Производительность операций обновления B^*-дерева и B-дерева практически одинакова, за исключением некоторых случаев операций включения и удаления. В табл. 14.5 приведены сводные данные о производительности различных операций для B^*-дерева и для B-дерева. Обсудим вкратце различия в операциях обновления.

Изменить. К обеим организациям применимы одни и те же выражения: (14-30) и (14-31). Однако длина пути поиска записи данных в B^*-дереве в точности равна h вершинам, тогда как в B-дереве поиск может быть любой длины.

Включить. В B^*-дереве после деления вершины все ключи размещаются в вершинах-листьях, но это не влияет на количество

Таблица 14.5. **Оценка операций базы данных (количества обращений к физическим блокам) для *B*-дерева и *B**-дерева степени *k***

Операция	*B*-дерево	*B**-дерево
ПОЛУЧИТЬ УНИКАЛЬНУЮ	ВО: $\left\lfloor \log_{k+1}\left(\frac{NR+1}{2}\right)\right\rfloor +$ $+ 2$ rba НО: 2 rba	$\left\lfloor \log_{k+1}\left(\frac{NR+1}{2}\right)\right\rfloor +$ $+ 1$ rba
ПОЛУЧИТЬ УНИКАЛЬНУЮ′	$\left\lfloor \log_{k+1}\left(\frac{NR+1}{2}\right)\right\rfloor +$ $+ 1$ rba	Как для ПОЛУЧИТЬ УНИКАЛЬНУЮ
ПОЛУЧИТЬ СЛЕДУЮЩУЮ ПОЛУЧИТЬ ПРЕДЫДУЩУЮ	РВА (ПОЛУЧИТЬ ВСЕ)/ /NR	1 rba
ПОЛУЧИТЬ ВСЕ	ВО: NR $(1 + 1/k)$ rba НО: NR$(1 + 1/2k)$ rba	ВО: NR/k rba НО: NR/$2k$ rba
ПОЛУЧИТЬ ПАКЕТ (RPBR записей)	РВА (ПОЛУЧИТЬ ВСЕ)— (NR — RPBR) rba	Как для связанной последовательной организации с блокированием
После завершения поиска:		
ИЗМЕНИТЬ неключевое значение	1 sba	Как для *B*-дерева
ИЗМЕНИТЬ ключ	ВО: УДАЛИТЬ индекс + ПОЛУЧИТЬ УНИКАЛЬНУЮ + + ВКЛЮЧИТЬ индекс + перезаписать блок данных НО: 2 sba	ВО: как для *B*-дерева НО: 1 sba
ВКЛЮЧИТЬ	ВО: 2 sba + h (1 rba + + 1 sba) + 1 rba НО: 2 sba	ВО: 1 sba + h (1 rba + + 1 sba) НО: 1 sba
УДАЛИТЬ	ВО: 1 sba + $(h-1) \times$ $\times$ (1 rba + 1 sba) НО: 1 sba	ВО: $(h-1)$ (1 rba + + 1 sba) НО: 1 sba
ОБНОВИТЬ ПАКЕТ	Сумма оценок отдельных обновлений	Как для *B*-дерева

физических операций ввода-вывода (обращений к блокам), требуемых для включения ключа в индекс. Однако в B^*-дереве ключ и ассоциированные данные включаются в один и тот же блок, поэтому достаточно одной операции перезаписи блока.

Удалить. В B^*-дереве ключи удаляются только из вершин-листьев, поэтому здесь неприменим случай 2 удаления ключа из B-дерева. В остальном с точки зрения количества обращений

Таблица 14.6. **Сравнительная оценка производительности B-деревьев и B^*-деревьев степени 10**

	B-дерево		B^*-дерево	
	Нижняя оценка	Верхняя оценка	Нижняя оценка	Верхняя оценка
ПОЛУЧИТЬ УНИКАЛЬНУЮ	2 rba	9 rba	8 rba	8 rba
ПОЛУЧИТЬ ВСЕ	$1{,}05 \times 10^8$ rba	$1{,}1 \times 10^8$ rba	$0{,}05 \times 10^8$ rba	$0{,}1 \times 10^8$ rba
ВКЛЮЧИТЬ	2 sba	9 rba + 10 sba	1 sba	8 rba + 9 sba
УДАЛИТЬ	1 sba	7 rba + 8 sba	1 sba	7 rba + 7 sba

к физическим блокам операции удаления ключа эквивалентны. Однако, как и в случае операции включения, выполняется совместное удаление ключа и ассоциированных данных из вершин-листьев, поэтому удаление данных не вызывает дополнительной операции записи блока, тогда как в B-дереве такая операция необходима (см. выражение (14-37)).

• **Пример 14-1.** Пусть база данных состоит из 10^8 записей; каждая запись идентифицируется уникальным значением ключа. Требуется сравнить оценки производительности операций произвольной выборки, последовательной обработки, включения и удаления для B-дерева и B^*-дерева степени 10 ($k = 10$). Требуется также графически изобразить оценки произвольной выборки для $k = 1$, 10, 100, 1000. Очевидно, что можно использовать уравнения из табл. 14.5. Полученные результаты приведены в табл. 14.6 и графически представлены на рис. 14.20.

$$h_1 = \log_{21}(10^8 + 1) = 7, \quad h_2 = \log_{11}\left[\frac{10^8 + 1}{2} + 1\right] = 8;$$

это нижняя и верхняя оценки высоты B-дерева.

Поскольку коэффициент блокирования для вершин B-дерева составляет $2k$, на рис. 14.20 можно видеть, что оптимальное значение k должно быть по величине близко NR/2. По мере возрастания k время поиска уменьшается. Однако, как и в случае физически последовательного файла, верхняя граница ко-

эффициента блокирования, как правило, зависит от других факторов, таких, как физические характеристики запоминающего устройства (например, размер дорожки), установленный в системе размер буфера или стандартное соглашение, принятое на

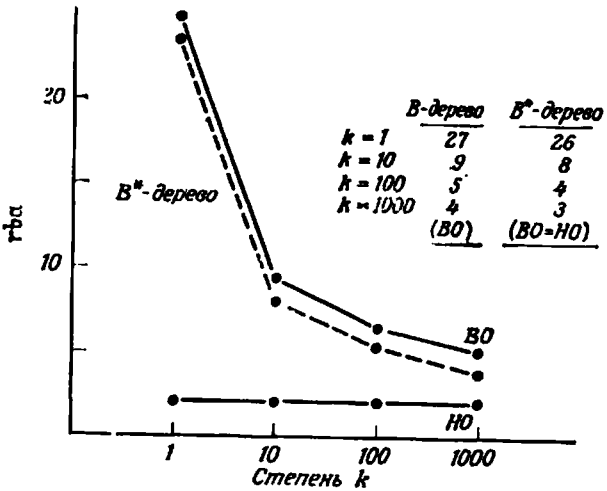


Рис. 14.20. Сравнительные оценки производительности произвольной выборки для *B*-деревьев и *B**-деревьев.

конкретной вычислительной установке. В мультипрограммной системе размер буфера, как правило, ограничен, благодаря чему исключается возможность захвата физических ресурсов вычислительной системы со стороны одного приложения. □

14.2.5. B-дерево с префиксом*

В работе [23] описаны две разновидности *B*-деревьев: *B**-дерево с простым префиксом и *B**-дерево с префиксом; их достоинством является тот факт, что в индексе *B**-дерева не требуется хранить ключи полной длины. Понятие *B*-дерева с простым префиксом* представляет расширение понятия *B**-дерева в том аспекте, что с целью экономии объема памяти и уменьшения времени выборки в индексе хранятся лишь уникальные префиксы ключей. *В B*-дереве с префиксом* вообще отсутствует явное хранение префиксов ключей; последние динамически строятся в процессе поиска. Остановимся на понятии *B**-дерева с простым префиксом, для простоты условимся называть его *SPB*-деревом.*

Основные характеристики *SPB**-дерева совпадают с характеристиками *B**-дерева (см. разд. 14.2.4), за тем исключением,

что индекс заменяется B-деревом разделителей переменной длины. В зависимости от значений соседних (упорядоченных) ключей в записях данных размер разделителя может изменяться от одного знака до полного значения ключа.

На рис. 14.21 приведен пример выбора соответствующих разделителей для файла клиентов, где в качестве ключей вершин-листьев используются фамилии клиентов. Если бы для включения новой фамилии клиента, скажем Jung, потребовалось деление вершины, в обычном B^*-дереве произошло бы перераспределение вершин с ключами Johns и Johnson, при котором ключ Johnson был бы продублирован в исходной вершине индекса.

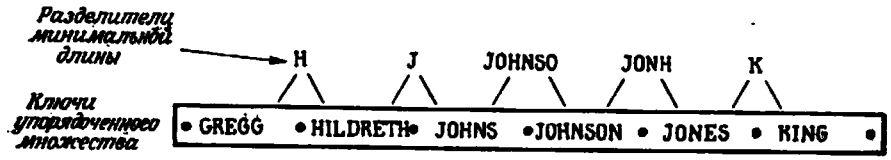


Рис. 14.21. Пример разделителей вершины-листа для SPB*-дерева степени 3.

Если эта же точка деления вершины используется в SPB^*-дереве, выбирается самый короткий префикс между Johns и Johnson — Johnso. Однако при смещении точки деления на один элемент в любую сторону префикс бы уменьшился, а при смещении влево был бы получен префикс J минимальной длины 1. Для SPB^*-дерева вводится понятие *интервала деления* σ_l для вершин-листьев (и σ_b для ветвящихся вершин), специфицирующего количество байт или ключей в обе стороны от середины вершины, любой из которых можно принять за точку деления. В пределах интервала точка деления выбирается так, чтобы длина разделителя вершины была минимальна. С ростом σ_l и σ_b возрастает степень свободы при выборе разделителя и, следовательно, уменьшается размер разделителя, хотя при этом могут быть получены вершины с полезным использованием менее 50 % и увеличен объем требуемой памяти. Заметив это противоречие, авторы работы [23] проанализировали зависимость производительности SPB^*-дерева от значений σ_l и σ_b в сравнении с B^*-деревом. Кроме того, они провели расчет средней длины разделителей в SPB^*-дереве и вывели закономерность уменьшения высоты B^*-дерева в связи с уменьшением размера индекса.

Неформальная сравнительная оценка основных операций базы данных для SPB^*-дерева и B^*-дерева позволяет утверждать, что производительность произвольной выборки в SPB^*-дереве оказывается выше за счет вероятного уменьшения высоты дерева с уменьшением длины разделителей. Вследствие того что

хранение упорядоченного множества данных в обоих случаях организовано одинаково, производительность последовательной обработки (как, например, операций ПОЛУЧИТЬ СЛЕДУЮЩУЮ и ПОЛУЧИТЬ ПРЕДЫДУЩУЮ) также одинакова. Операция включения в обоих случаях реализуется одной и той же процедурой, за тем исключением, что в SPB^*-дереве в качестве исходной вершины выбирается минимальный разделитель; процедура удаления также одна и та же для обеих организаций. Конечно, возможно, что в некоторых случаях операции включения и удаления позволили бы заменить больший по размеру разделитель на меньший, однако авторы указанной работы не рекомендуют прилагать к этому дополнительные усилия, так как улучшение производительности, скорее всего, будет незначительным. В заключение следует отметить, что, вероятно, наибольшее достоинство SPB^*-дерева представляет увеличение производительности произвольной выборки в случае баз данных большой размерности.

В современной технической литературе предложено множество гибридных древовидных организаций, сочетающих в себе лучшие качества метода хеширования, бинарных деревьев, B-деревьев и многочисленных их вариантов. Среди них можно назвать хеш-деревья [86, 277], расширенный метод хеширования [115], виртуальное B-дерево [20], бинарное B-дерево [21] и дерево 2-3 [348]. Для подробного ознакомления с ними рекомендуем читателю обратиться к указанным работам.

14.3. TRIE-структуры

Бóльшая часть рассмотренных выше способов индексирования баз данных или файлов большой размерности основывалась на хранении полных ключевых значений в вершинах разных уровней дерева поиска. Очевидное исключение из этого правила представляют варианты B^*-дерева. Еще одно исключение представляет класс методов доступа, известных под названием *TRIE* (сравните с «try» — попытка); в них значение первичного ключа рассматривается как упорядоченная последовательность (или строка) знаков из алфавита размером V. *TRIE*-структура представляет собой структуру многоходового дерева, в котором из каждой ветвящейся вершины выходят в точности V ветвей, по одной для каждого возможного знака в позиции строки ключа. Если строка ключа состоит из KS знаков (байтов), то поиск соответствующей записи данных с целью выявления ее наличия или отсутствия в файле требует просмотра KS вершин. Если запись данных в файле присутствует, то для доступа к самой записи требуется выполнить еще одно обращение к блоку. В действительности записям дан-

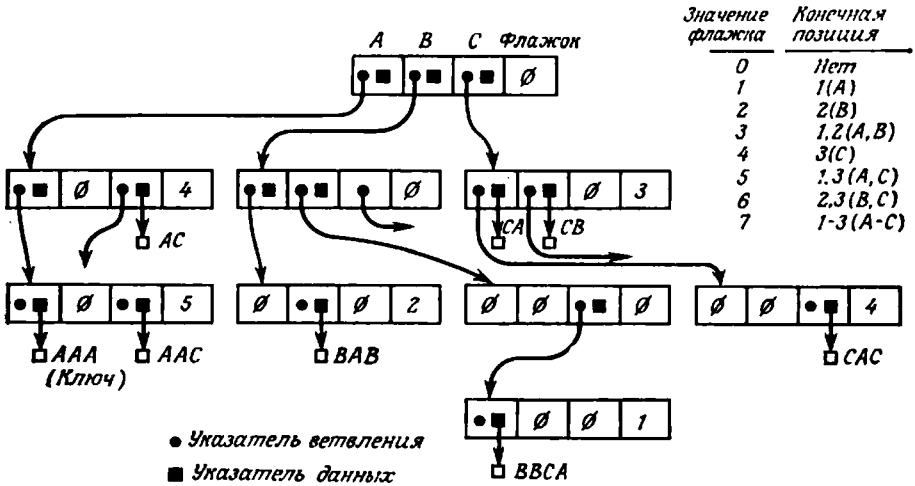


Рис. 14.22. Пример TRIE-структуры индекса для значений ключа переменной длины (NR = 8).

ных в этой структуре соответствуют листья. Достоинством *TRIE*-структуры является возможность быстрого произвольного доступа к данным ценой дополнительных служебных издержек памяти.

Наиболее известные из ранних вариантов *TRIE*-структуры были предложены в работах [119, 104]. Само название «*TRIE*» происходит от слова retrieval (выборка). Вариант *TRIE*-структуры, предложенный в [119], состоит из нескольких уровней вершин, каждая из которых содержит V полей указателей и поле флажка; пример такой структуры приведен на рис. 14.22. Некоторая i-я позиция в ключе может иметь V возможных значений, каждому значению соответствует определенная позиция в определенной вершине на i-м уровне структуры дерева. Такой локальный способ адресации представляет разновидность прямой адресации, при которой значение символа неявно определяет позицию в вершине, где может быть найдено значение соответствующего указателя. Значение указателя определяет местоположение вершины на следующем уровне, в которой представлены возможные значения $i+1$ символов в ключе, и т. д. Следовательно, для ключа из KS знаков выполняется просмотр KS вершин, по одной на каждом из KS уровней дерева. Вследствие того что ключи могут быть переменной длины, *TRIE*-структура, как правило, не сбалансирована; это означает следующее: если KS_j обозначает длину ключа j, то для разных j вершины-листья в *TRIE*-структуре располагаются на разных KS_j уровнях.

Предусмотренный в каждой вершине *TRIE*-структуры флажок предназначается для идентификации последней вершины на пути, соответствующем значению ключа. На практике, как правило, во флажке указывается значение *k*, обозначающее *k*-ю позицию в вершине, соответствующую последнему знаку в полном ключе. В этом случае для доступа к записи данных используется *k*-й указатель данных, а для перехода к вершине индекса на следующем уровне — *k*-й указатель ветвления. Благодаря этому, с одной стороны, обеспечивается «усеченность» дерева для одной комбинации знаков, представляющей полное ключевое значение, а с другой стороны, обеспечивается возможность представить в дереве более длинные ключевые значения, префикс которых совпадает с этой комбинацией знаков. Если на одной вершине заканчивается несколько ключевых значений, то во флажок можно поместить число, которое расшифровывается в соответствующие (несколько) номера позиций в вершине для последних знаков полных ключевых значений. На рис. 14.22 приведен пример использования *TRIE*-структуры индекса.

Хотя в каждой вершине предусматривается место для *V* возможных значений, многие из комбинаций значений в качестве ключевых идентификаторов не используются. Вследствие этого возможны большие потери памяти. Если конкретная позиция в вершине не используется, ее указатели устанавливаются в нуль (0), отражая тот факт, что поддерево, соответствующее данному префиксу, пусто. Благодаря этому «усечение» дерева выполняется при первой же возможности и не требуется выделять память для пустого поддерева.

Рассмотренный вариант *TRIE*-структуры, в котором каждая ветвящаяся вершина предусматривает соответствующий указатель ветвления для каждого знака, является представителем целого семейства конфигураций цифровых деревьев поиска. Можно предусмотреть соответствующий указатель ветвления для каждой комбинации знаков из двух, трех и так далее до KS позиций ключа. В случае KS позиций первичного ключа каждая ветвящаяся вершина соответствует полному ключевому значению и, следовательно, получен вариант *TRIE*-структуры с прямым доступом к данным. Однако напомним читателю, что в большинстве случаев прямой доступ к базе данных влечет за собой огромные служебные издержки памяти, а последовательный поиск для приложений с произвольной выборкой выполняется чрезвычайно долго. Подобные компромиссы, касающиеся объема памяти и производительности произвольной выборки, характерны и для вариантов *TRIE*-структуры, использующих комбинации двух и более знаков; так же как в случае *TRIE*-структуры с одним знаком, их можно подробно проанализировать. Здесь же достаточно отметить быстрое возрастание

максимального количества ветвящихся вершин с возрастанием количества уровней, однако в случае *TRIE*-структуры с одним знаком скорость возрастания наименьшая. В табл. 14.7 приве-

Таблица 14.7. **Количество ветвящихся вершин на уровнях различных вариантов *TRIE*-структуры**

Уровень	1-й знак/вершина	2-й знак/вершина	3-й знак/вершина	*KS*-знак/вершина
1	1	1	1	1
2	V	V^2	V^3	V^{KS}
3	V^2	V^4	V^6	—
4	V^3	V^6	V^9	—
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
i	V^{i-1}	V^{2i-2}	V^{3i-3}	—
Всего вершин	$\frac{V^{KS}-1}{V-1}$	$\frac{V^{KS}-1}{V^2-1}$	$\frac{V^{KS}-1}{V^3-1}$	$V^{KS}+1$

дены данные о максимальном числе вершин для каждого уровня *TRIE*-структуры.

Основные моменты, заслуживающие внимания при проектировании *TRIE*-структур, представляют:

- *Тип обработки.* Наиболее целесообразны произвольная выборка и обновление.
- *Изменчивость данных.* Влияет на выбор варианта *TRIE*-структуры.
- *Размер словаря V.* Влияет на выбор варианта *TRIE*-структуры, определяет общий размер дерева.
- *Длина строки ключевого значения KS.* Определяет длину пути для операции выборки.
- *Размер указателя.* Влияет на размер вершины, на коэффициент блокирования.

14.3.1. Производительность выборки

При использовании *TRIE*-структуры высокой производительностью отличается произвольная выборка. В каждом случае операция представляет последовательность произвольных обращений к блокам от корневой вершины до вершины-листа, содержащей требуемую запись. В случае успешного поиска всегда приходится выполнить доступ к $KS + 1$ вершинам, тогда как неуспешный поиск может закончиться ранее на любой вершине с нулевым значением указателя или же при обнаружении конца

искомого значения ключа, когда поиск в *TRIE*-структуре еще не завершен. Мы предполагаем, что каждая вершина *TRIE*-структуры расположена в отдельном блоке, но в промышленных реализациях с целью уменьшения количества требуемых обращений к блокам, как правило, вершины группируются в блоки. Обычно для *TRIE*-структуры, как для бинарных деревьев поиска, используется способ группового распределения вершин, при котором новые вершины помещаются в кластеры поблизости от соответствующих исходных вершин. В лучшем случае все вершины индекса объединяются в один блок, а данные — в другой блок. Однако такой способ применим только к индексам малой размерности.

$$\text{РВА(ПОЛУЧИТЬ УНИКАЛЬНУЮ, } TRIE\text{-структура, несблокированные записи)} = \text{KS} + 1\,\text{rba}, \tag{14-42}$$

$$\text{РВА(ПОЛУЧИТЬ УНИКАЛЬНУЮ, } TRIE\text{-структура, групповое распределение)} = \begin{cases} \text{KS} + 1\,\text{rba (максимум)}, \\ 2\,\text{rba (минимум)}, \end{cases} \tag{14-43}$$

$$\text{РВА(ПОЛУЧИТЬ УНИКАЛЬНУЮ', } TRIE\text{-структура)} = \begin{cases} \text{KS rba (максимум)}, \\ 1\,\text{rba (минимум)}. \end{cases} \tag{14-44}$$

Последовательная обработка выполняется так же, как в случае любого неупорядоченного файла данных, такого, как индексный произвольный файл:

$$\text{РВА(ПОЛУЧИТЬ ВСЕ, } TRIE\text{-структура)} = \left\lceil \frac{\text{NR}}{\text{EBF}_2} \right\rceil \text{sba}, \tag{14-45}$$

где EBF_2 обозначает коэффициент полезного блокирования для неупорядоченного файла данных. Вследствие произвольного расположения записей данных такие операции, как ПОЛУЧИТЬ СЛЕДУЮЩУЮ и ПОЛУЧИТЬ ПРЕДЫДУЩУЮ, к *TRIE*-структуре неприменимы. Кроме того, из-за неупорядоченности файла данных неприменима также и пакетная обработка.

14.3.2. Производительность обновления

Оценка стандартных операций обновления применительно к *TRIE*-структуре не вызывает затруднений. В предположении, что время поиска учтено отдельно, простейшая операция — изменение неключевого значения — вызывает выполнение только

операции перезаписи измененного блока данных:

$$\text{РВА(ИЗМЕНИТЬ неключевое значение, } TRIE\text{-структура)} = = 1\,\text{sba}. \quad (14\text{-}46)$$

Изменение значения ключа выполняется значительно сложнее. *В TRIE*-структуре это означает, что необходимо проанализировать весь путь доступа к данным с целью выявления требуемых в нем изменений в ответ на новое значение ключа. Независимо от того, потребуется ли изменить значения каких-либо указателей, просмотр всего пути от корня до листа (до записи данных) выполняется дважды: один раз с целью удаления пути старого ключа и еще один раз с целью включения пути нового ключа:

$$\begin{aligned}\text{РВА(ИЗМЕНИТЬ ключ, } TRIE\text{-структура)} = \\ = \text{РВА(УДАЛИТЬ старый ключ)} + \\ + \text{РВА(ПОЛУЧИТЬ УНИКАЛЬНУЮ)} + \\ + \text{РВА(ВКЛЮЧИТЬ новый ключ).} \quad (14\text{-}47)\end{aligned}$$

Уравнение (14-47) представляет оценку производительности изменения ключевого значения в общем виде. Оценим каждый его компонент отдельно.

В целях определения возможности использования существующих указателей или выделения памяти для новых указателей при включении нового ключа и записи данных в *TRIE*-структуру вначале требуется выполнить обращения к KS вершинам индекса. В любом случае необходим доступ ко всем KS вершинам; оценка этих обращений учтена в оценке начального поиска позиции для записи. После установления пути доступа выполняется включение новой записи в существующий блок (с последующей его перезаписью) или создается новый блок.

$$\text{РВА(ВКЛЮЧИТЬ, } TRIE\text{-структура)} = \begin{cases} 1\,\text{rba} + 1\,\text{sba} & \text{включение в существующий блок,} \\ 1\,\text{rba} & \text{создание нового блока.} \end{cases} \quad (14\text{-}48)$$

Подобным образом выполняются также операции удаления записей из *TRIE*-структуры. Если записи данных организованы в блоки, то необходимо выполнить перезапись измененного блока; в противном случае перезапись данных не требуется. Дополнительно независимо от того, блокированы записи или нет, для завершенности операции удаления необходимо изменить значение указателя ветвления в соответствующей ветвящейся вершине индекса на уровне KS. В предположении, что

вершины индекса находятся в буферах, потребуется одна операция перезаписи блока, содержащего ветвящуюся вершину. В случае если указанная вершина не содержит других указателей на записи данных в *TRIE*-структуре, ее следует удалить. Указанная процедура может распространиться вверх до корня, в результате чего *TRIE*-структура возвращается к некоторому предыдущему состоянию. В худшем случае из структуры будут удалены KS ветвящихся вершин; при удалении каждой вершины выполняется изменение значения соответствующего указателя в исходной для нее вершине и последующая перезапись исходной вершины.

$$\text{РВА (УДАЛИТЬ, } TRIE\text{-структура)} = \begin{cases} 1\,\text{sba} & \text{минимум, если записи не сблокированы и вершины из индекса не удаляются,} \\ 2\,\text{sba,} & \text{если записи сблокированы и вершины из индекса не удаляются,} \\ \text{KS sba,} & \text{если записи не сблокированы и из индекса удаляется KS вершин,} \\ \text{KS} + 1\,\text{sba} & \text{максимум, если записи сблокированы и из индекса удаляется KS вершин.} \end{cases} \quad (14\text{-}49)$$

Напомним читателю, что количество начальных обращений к вершинам индекса и записи данных учитываются при оценке предшествующей операции ПОЛУЧИТЬ УНИКАЛЬНУЮ.

14.3.3. Объем памяти

В табл. 14.7 приведена оценка общего количества ветвящихся вершин в индексе *TRIE*-структуры с одним знаком: $\text{IBNODES} = (V^{\text{KS}} - 1)/(V - 1)$.

Количество записей данных равно NR.

$$\text{SRS} = V \times \text{PS} + 1 \text{ байт}, \quad (14\text{-}50)$$

$$\text{EBF}_i = \left\lfloor \frac{(\text{BKS}_i - \text{BOVHD}_i) \times \text{LF}_i}{\text{SRS}} \right\rfloor, \quad (9\text{-}36)$$

$$\text{BLKSTOR}(TRIE\text{-структура}) = \left\lceil \frac{\text{IBNODES}}{\text{EBF}_1} \right\rceil \times \text{BKS}_1 + \left\lceil \frac{\text{NR}}{\text{EBF}_2} \right\rceil \times \text{BKS}_2 \text{ байт}, \quad (14\text{-}51)$$

где EBF_1 и EBF_2 обозначают соответственно коэффициенты полезного блокирования ветвящихся вершин индекса и неупорядоченной области данных.

14.3.4. Сравнение производительности TRIE-структуры с B-деревом

Основная особенность *TRIE*-структуры в сравнении с другими древовидными структурами состоит в обеспечении быстрой произвольной выборки за счет дополнительных служебных издержек памяти. Рассмотрим на примере практические аспекты применения *TRIE*-структуры.

• **Пример 14.2.** Пусть база данных содержит 10^8 записей. Все записи имеют одинаковый формат и размер, равный 100 байт. Каждая запись имеет уникальный ключ, состоящий из девяти десятичных цифр 0—9; указанные ключевые значения можно использовать для построения *TRIE*-индекса. В предположении, что размер блока равен 1000 байт, $BOVHD_i = 0$ и $LF_i = 1$ для всех i, требуется сравнить производительность произвольной выборки с производительностью *B*-дерева, представленной на рис. 14.20 и в табл. 14.6.

Применяя к оценке памяти уравнения (14-50) и (14-51), получаем:

$$SRS\,(TRIE\text{-структура}) = 10 \times 4 + 1 = 41 \text{ байт},$$

$$EBF_1\,(TRIE\text{-структура}) = \lfloor 1000/41 \rfloor = 24,$$

$$IBNODES = \frac{10^9 - 1}{10 - 1} = 11\,111\,111,$$

$$BLKSTOR\,(TRIE\text{-структура}) = \left\lceil \frac{11\,111\,111}{24} \right\rceil \times 1000 + \frac{10^8}{10} \times$$

$$\times 10^3 \text{ байт} = 462{,}9 \times 10^6 \text{ байт (индекс)} + 10^{10} \text{ байт (данные)} =$$

$$= 10{,}463 \times 10^9 \text{ байт}.$$

Наличие большого количества вершин в индексе позволяет предположить, что в общие блоки объединяются только вершины первых двух уровней *TRIE*-индекса. Следовательно, оценка общего количества обращений составляет $KS = 9$ rba. Это приблизительно совпадает с оценкой количества произвольных обращений для *B*-дерева степени 10. Уравнения (14-36)—(14.41) позволяют получить оценку объема памяти для *B*-дерева индекса:

$$BKS_1\,(BNODES) = 20 \times 9 + 41 \times 4 + 0 = 344 \text{ байт},$$

$$BKS_2\,(LNODES) = 20 \times 9 + 20 \times 4 + 0 = 260 \text{ байт},$$

$$BKS_3\,(\text{блоки данных}) = 100 \text{ байт (при } BF = 1).$$

В лучшем случае, когда преобладают вершины-листья и полезное использование памяти максимально, получаем

$$\text{BLKSTOR } (B\text{-дерево}) = 260 \text{ байт} \times 10^8/20 \text{ вершин} + 100 \times \times 10^8 \text{ байт} = 13 \times 10^8 + 100 \times 10^8 = 11{,}3 \times 10^9 \text{ байт}.$$

Таким образом, в этом случае оценка памяти для *TRIE*-структуры оказалась лучше. Однако следует обратить внимание на тот факт, что в случае использования значений ключа с полным набором знаков алфавита в *TRIE*-структуре (то есть в *SRS*) потребовался бы значительно больший объем памяти, а размер *B*-дерева при этом не изменился бы. В результате производительность *B*-дерева (операции выборки и объем памяти) оказалась бы лучше в сравнении с *TRIE*-структурой. □

Глава 15. Вторичные методы доступа

15.1. Введение

Возрастает количество приложений баз данных, предполагающих использование операций типа ПОЛУЧИТЬ НЕКОТОРЫЕ, в которых критерием отбора экземпляров записей служат значения одного или нескольких атрибутов. *Незапланированным запросом* называется запрос, точная спецификация которого (формат, значение ключа и размер записи-цели) заранее не известна и, следовательно, при проектировании базы данных не может быть явно учтена. Тем не менее из анализа требований можно извлечь разумные предположения относительно ожидаемых типов запросов. Выше мы ознакомились с эффективными способами организации выборки и обновления данных в первичных методах доступа, использующих только первичный ключ. Теперь рассмотрим так называемые *вторичные методы доступа* (ВМД) — совокупность способов и средств, предназначенных для организации эффективного доступа ко всем записям-целям, значения вторичных ключей в которых удовлетворяют заданному в запросе множеству условий.

В общем виде запрос представляет собой совокупность булевых функций, построенных с помощью логических операторов конъюнкции (*И*) и дизъюнкции (*ИЛИ*). Ниже в примере приводятся основные понятия, полезные при анализе вторичных методов доступа [58]. В целях иллюстрации компонентов запроса используется упрощенный вариант языка SEQUEL [64].

Пример запроса 15-1

ВЫБРАТЬ ИМЯ_СЛУЖАЩЕГО, АДРЕС_СЛУЖАЩЕГО ИЗ СЛУЖАЩИЙ	операционная часть; получить значения атрибутов типа ИМЯ_СЛУЖАЩЕГО и АДРЕС — СЛУЖАЩЕГО из записи типа СЛУЖАЩИЙ
ГДЕ (ВОЗРАСТ = 21 — 59 *И* ГОРОД = 'ДЕТРОЙТ' *И* ДОЛЖНОСТЬ = 'ПРОГРАММИСТ') *ИЛИ* (ВОЗРАСТ < 65 *И* ДОЛЖНОСТЬ = = 'ПРОГРАММИСТ-АНАЛИТИК')	квалификационная часть. □

1. *Атомное условие (АУ)*

Атомное условие представляет основное условие квалификации запроса. Оно имеет вид: ИМЯ $\left\{\begin{matrix}<\\=\\>\end{matrix}\right\}\left\{\begin{matrix}\text{ИМЯ}\\ \\ \text{ЗНАЧЕНИЕ}\end{matrix}\right\}$, где ИМЯ определяет тип элемента данных (или домен в реляционной базе данных), а ЗНАЧЕНИЕ задает значение элемента данных. Разрешается использовать составные реляционные операторы.

2. *Условие элемента (УЭ)*

Условие элемента представляет собой дизъюнкцию атомных условий, $АУ_1$ *ИЛИ* $АУ_2$ *ИЛИ* ... *ИЛИ* $АУ_p$, таких, что все $АУ_i$ относятся к одному и тому же типу элемента данных, но содержат разные его значения. Кроме того, под условием элемента понимается также совокупность значений элемента данных в *спецификации области значений*. Например, область значений первого условия элемента в примере запроса 15-1 ВОЗРАСТ-21—59 состоит из 39 значений или 39 атомных условий с разными значениями элемента. Условие элемента вида ВОЗРАСТ $<$ 65 по определению представляет собой единственное атомное условие, хотя при этом подразумевается область значений. Ниже в тексте в качестве отдельных атомных условий будут рассматриваться спецификации областей значений вида ЗНАЧЕНИЕ-ЗНАЧЕНИЕ. Процесс выборки записей, содержащих значения из определенной области значений, называют *поиском области значений*.

3. *Условие записи (УЗ)*

Условие записи представляет собой конъюнкцию условий элемента, $УЭ_1$ *И* $УЭ_2$ *И* ... *И* $УЭ_q$, таких, что каждое $УЭ_j$ представляет отдельный тип элемента данных (или домен). В примере запроса 15-1 в первые скобки заключено одно условие записи, построенное из трех условий элемента; во вторые скобки заключено второе условие записи, построенное из двух условий элемента.

4. *Условие запроса (УЗР)*

Условие запроса представляет собой дизъюнкцию условий записи, $УЗ_1$ *ИЛИ* $УЗ_2$ *ИЛИ* ... *ИЛИ* $УЗ_r$, таких, что все они относятся к одному и тому же типу записей. Квалификационная часть запроса 15-1 представляет собой пример условия запроса, построенного из двух условий записи.

Условимся называть определенные выше запросы булевого вида *конъюнктивными запросами*. Использование введенных стандартных понятий при анализе вторичных методов доступа позволит нам лучше понять и оценить затраты, требуемые для удовлетворения сложных условий запроса. В данной главе мы ознакомимся с основными методами доступа, предназначенными

для обработки запросов: мультисписковой структурой, инвертированным файлом и двусвязанным деревом. Для иллюстрации основных различий в структурах хранения мультисписконого и инвертнрованного файлов вводится понятие гибридной секционной структуры. В то же время важно отметить, что инвертнрованный файл и его варианты являются представителями широкого класса механизмов выборки по вторичному ключу, которые включают первичные методы доступа, схемы ассоциативного поиска и специальные схемы кодирования.

15.2. Мультисписковый файл

Выполнение обобщенной операции запроса влечет за собой выборку переменного количества записей-целей. Поскольку обычно данные не упорядочены по вторичному ключу, физическое расположение записей-целей, выбираемых в соответствии со значением вторичного ключа, между собой никак не связано. Вследствие этого механизмы поиска и структуры хранения, используемые для доступа по первичным ключам, оказываются обычно крайне неэффективными. В случае если для удовлетворения запроса применяется последовательный поиск в базе данных, среднее расстояние поиска для произвольно расположенных записей-целей составляет всю базу данных, т. е. приходится выполнять доступ к каждой записи и проверять ее на соответствие каждому условию записи в запросе. К счастью, для обработки запросов можно успешно использовать сочетание индексирования и связаиного последовательного метода доступа.

Мультисписковая структура, или файл, представляет собой совокупность связанных последовательных структур (связанных списков), содержащих экстенты хранимых записей, таких, что каждый список связывает записи с одинаковым значением конкретного типа элемента данных; кроме того, в целях обеспечения быстрого доступа к записям, связанным между собой значением вторичного ключа, списки индексируются. Мультисписковый файл является предшественником инвертированиого файла; для доступа к списку записей с одинаковым значением вторичного ключа в обеих организациях можно использовать многоуровневый индекс. Однако в общем случае мультисписковый файл определяется или описывается без индексной структуры.

На рис. 15.1 приведен пример мультисписковой организации с двухуровневым индексом, предназначенной для удовлетворения запроса из примера 15-1. На уровне 1 индекса перечислены типы элементов данных, относящиеся к одному типу записей в файле или подфайле. Для каждого имени типа элемента предусмотрен специальный указатель, значение которого определяет местоположение соответствующей области на уровне 2 ин-

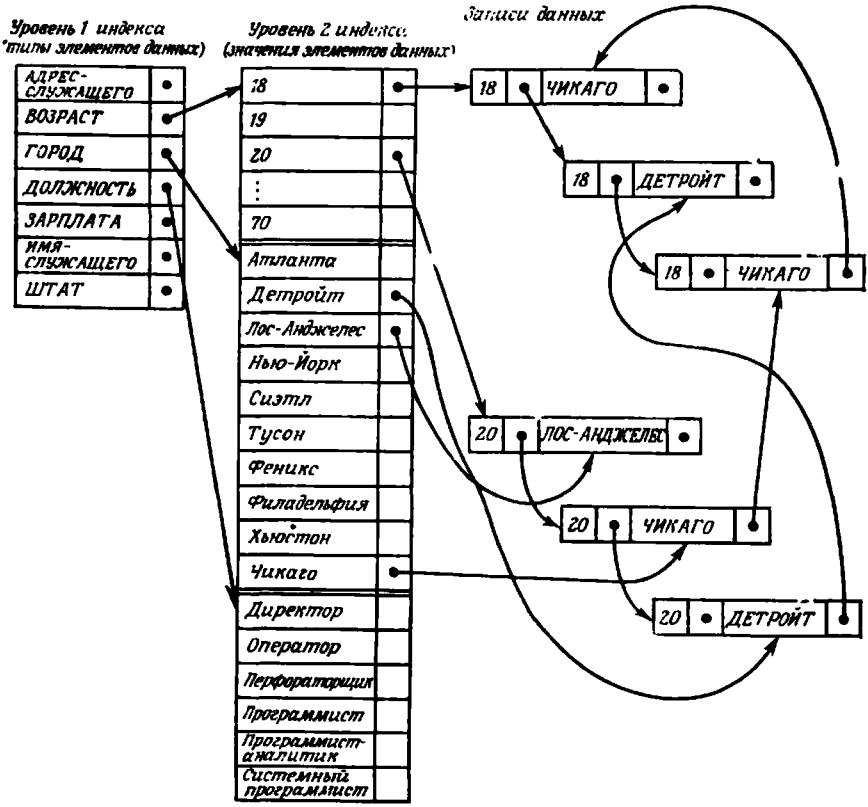


Рис. 15.1. Мультисписковая структура с двухуровневым индексом.

декса: в этой области перечислены значения конкретного типа элемента данных. На уровне 1 можно использовать также и другие варианты индекса, например индекс блока для блоков индекса уровня 2, или любую другую разновидность, допустимую для индексно-последовательного файла [57]. Каждый элемент индекса на уровне 2, содержащий активное значение элемента данных определенного типа, снабжен указателем. Этот указатель указывает на заголовок списка записей данных, каждая из которых содержит соответствующее данному указателю значение элемента данных заданного типа. Заметим, что каждая запись в файле может участвовать в нескольких связанных списках; максимальное количество списков, в которых участвует одна запись, равно количеству типов элементов данных в данном типе записи; отсюда происходит термин «мультисписок». На самом деле некоторые типы элементов данных в записи мо-

ДОЛЖНОСТЬ	ГОРОД	ВОЗРАСТ	ПРОЧИЕ
Программист	Атланта	20	...
Программист	Атланта	21	...
Программист	Атланта	23	...
Программист	Детройт	19	...
Программист	Детройт	21	...*
Программист	Детройт	22	...*
Программист-аналитик	Нью-Йорк	68	...
Программист-аналитик	Филадельфия	22	...*
Программист-аналитик	Филадельфия	22	...*
Системный программист	Тусон	57	...

Рис. 15.2. Подфайл, упорядоченный по значениям вторичных ключей ДОЛЖНОСТЬ, ГОРОД, ВОЗРАСТ для запроса примера 15-1.

* *Запись-цель запроса*

гут быть не определены (то есть имеют нулевое значение), и, следовательно, запись может участвовать в количестве списков, меньшем, чем максимально допустимое.

Используемый механизм поиска в мультисписковой структуре зависит от конкретной системы. Возможен последовательный поиск в индексе; если индекс упорядочен, то можно было бы использовать бинарный поиск. В силу того, что все записи данных, организованные в список, представляют записи-цели, на уровне записей данных всегда используется последовательный поиск. Из-за несмежного физического расположения записей, для того чтобы минимизировать время произвольного доступа, их можно не группировать в блоки. Однако для других приложений с целью минимизации времени последовательного доступа может оказаться целесообразным использовать блоки данных. Выбор мультисписковой организации данных для некоторого приложения оправдан лишь тогда, когда общее время поиска записей-целей в мультисписковой структуре оказывается меньше, чем общее время сортировки подфайла по указанным в запросе вторичным ключам и последующего поиска в упорядоченном подфайле множества подряд расположенных записей, уловлетворяющих условиям записи в запросе. На рис. 15.2 показана структура подфайла, упорядоченного в соответствии с условиями запроса из примера 15-1. Вследствие того что этот запрос построен из двух условий записи и нескольких атомных условий, приходится перебирать большое количество множеств

расположенных подряд записей данных. В разд. 15.3.4 на примере запроса приведены сравнительные оценки последовательного поиска, мультисписконого поиска и поиска в инвертированном файле.

В мультисписковой организации стратегия поиска записей данных, удовлетворяющих условию записи, содержащему несколько условий элемента, включает в себя прежде всего подсчет количества записей в каждом списке и занесение данного значения в соответствующий элемент индекса второго уровня. Затем применяется стратегия минимизации поиска, включающая определение местоположения каждого элемента индекса второго уровня, соответствующего условию элемента в запросе, определение самого короткого списка, исходя из значений счетчиков в соответствующих элементах индекса, и поиск записей данных в этом списке. Каждая запись в списке проверяется на удовлетворение каждому из условий элемента в условии записи и в зависимости от результатов проверки либо запоминается (если является записью-целью для всего условия записи), либо пропускается.

В случае запроса, построенного из нескольких условий записи, стратегия поиска включает определение совокупности записей-целей отдельно для каждого условия записи и последующее объединение полученных списков в один список, упорядоченный так, чтобы наилучшим образом представить результаты. В процессе объединения можно устранить повторяющиеся записи, то есть записи, которые одновременно удовлетворяют нескольким условиям записи в запросе.

Поиск области значений для нескольких атомных условий в условии элемента можно выполнять несколькими способами. В случае заранее известных запросов наиболее эффективным оказался бы способ представления каждой области значений для условия элемента в виде отдельного элемента 2-го уровня индекса, т. е. индекса значений элементов данных. Однако для незапланированных запросов этот способ оказывается крайне неэффективным, если исходить из объема памяти, требуемого для отображения всех теоретически возможных областей значений. Более эффективный способ по требуемому объему памяти, но более медленный по времени доступа включает в себя отдельный поиск каждого значения из спецификации области значений в индексе значений элементов данных. Тогда для поиска области значений приходится просмотреть все списки записей данных. При условии, что область значений невелика, а индекс значений элементов данных упорядочен, применение указанного способа не вызывает существенного увеличения количества операций ввода-вывода; однако для осуществления поиска нескольких элементов в индексе необходимы дополни-

тельные затраты, связанные с выполнением соответствующих программ. К счастью, за последнее время разработан ряд более эффективных альтернативных методов поиска области значений в особенности для статичных файлов. В этих методах с целью повышения эффективностн предпринята попытка уменьшить протяженность поиска за счет применения списков записей, упорядоченных по значениям вторичных ключей, инвертированных индексов (см. разд. 15.3), а также вариантов структур бинарных деревьев и соответствующих механизмов поиска. В качестве меры производительности обычно используются время загрузки индекса и данных, объем требуемой памяти и время обработки запроса.

Подводя итог обсуждению мультисписковой организации, перечислим основные факторы, на которые необходимо обратить внимание при ее проектировании:

- *Тип обработки.* Наиболее эффективны обобщенные запросы н обновления.
- *Объем памяти, необходимый для хранения индексов.*
- *Затраты на сортировку* (в качестве альтернативного способа обработки запроса).
- *Сложность запроса.* Влияет на время доступа к записям-целям в мультисписковом файле.
- *Упорядоченность индекса.* Влияет на выбор механизма поиска в индексе и на оценку операций обновления.
- *Стратегия поиска записи, удовлетворяющей нескольким условиям элемента.*
- *Размер блока в индексе.*

15.2.1. Производительность выборки при обработке запросов

Вторичные методы доступа специально предназначены для выборки записей по значениям вторичных ключей (т. е. для операции типа ПОЛУЧИТЬ НЕКОТОРЫЕ). Теоретически можно было бы выполнять другие операцни, такие, как ПОЛУЧИТЬ УНИКАЛЬНУЮ или ПОЛУЧИТЬ ВСЕ, но используемая для них структура хранения соответствует первичной области данных в мультисписке. Если эту область считать непрерывной в памяти, ее можно организовать в виде физически последовательной или связанной последовательной структуры и применить анализ оценки производительности последовательной и произвольной выборкн из разд. 12.2. В данном разделе предметом рассмотрения является обобщенный запрос ПОЛУЧИТЬ НЕКОТОРЫЕ.

Предположим, что индексы упорядочены, а поиск последовательный. Кроме того, предположим, что индекс уровня 1 невелик, и будем считать, что он содержится в одном физическом

блоке. В простейшем случае запрос состоит из одного атомного условия и требует доступа к одиому списку записей. Для такого запроса каждая запись в списке представляет запись-цель.

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, 1АУ/УЭ)} =$$

$$= \text{РВА}_1(\text{поиск в индексе}_1) + \text{РВА}_2(\text{поиск в индексе}_2) +$$

$$+ \text{РВА}_3(\text{поиск записи данных}), \quad (15\text{-}1)$$

$$\text{где } \text{РВА}_1 = 1\,\text{rba}, \quad (15\text{-}2)$$

$$\text{РВА}_2 = 1\,\text{rba}\left(\left\lceil \frac{\text{NIV}+1}{2\,\text{EBF}_2}\right\rceil - 1\right)\text{sba}, \quad (15\text{-}3)$$

$$\text{РВА}_3 = \text{NRIV}\,\text{rba}. \quad (15\text{-}4)$$

Для небольшого количества типов элементов, являющихся вторичными ключами, NIV обозначает среднее количество значений элемента данных каждого типа, а NRIV — среднее количество записей-целей, соответствующих одному значению элемента данных. В случае если заданы несколько (последовательных) атомных условий или спецификация области значений в одном условии элемеита, то поиск в иидексе$_2$ оказывается длиннее и требует доступа ко всем спискам возможных записей-целей.

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, } p \text{ АУ/УЭ)} = [\text{РВА}_1] + [1\,\text{rba} +$$

$$+ \left(\left\lceil \frac{(\text{NIV}-p+1)+1}{2\times \text{EBF}_2}\right\rceil - 1\right)\text{sba} + \left\lceil \frac{p-1}{\text{EBF}_2}\right\rceil \text{sba}\Big] + \left[\sum_{i=1}^{p} \text{NRIV}_i\,\text{rba}\right] \quad (15\text{-}5)$$

для $p \geqslant 1$. Член в первых скобках представляет оценку одного обращения к индексу$_1$. Член во вторых скобках, условимся обозначать его РВА$_{2A}$, содержит оцеику первоначального обращения к иидексу$_2$, последующих поиска в иидексе$_2$ первого значения элемента-цели и обращений к остальиым $p-1$ зиачениям. Сумма в третьих скобках представляет оценку общего количества обращений к p спискам записей-целей. В выражении (15-5) подразумевается, что область значений в условии элемента специфицирована как множество последовательных значений. В противном случае поиск в индексе$_2$ мог бы потребовать просмотра всего индекса. Другие компоненты общей оценки поиска остались без изменеиий.

Наличие в запросе нескольких условий элемента еще больше усложняет анализ операции выборки. Если задано q условий элемента и в каждом из них одно атомное условие, потребуется выполнить один поиск в индексе$_1$ и q поисков в индексе$_2$ и, кроме того, обращение к самому короткому виртуальному списку

записей.

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, } q \text{ УЭ/УЗ, 1АУ/УЭ)} = \text{РВА}_1 + \\ + \sum_{j=1}^{q} \text{РВА}_{2j} + \min_j (\text{NRIV}_j)\ \text{rba}, \qquad (15\text{-}6)$$

для $q \geqslant 1$, где РВА_{2j} обозначает количество обращений к физическим блокам при поиске в индексе$_2$, соответствующем j-му условию элемеита в запросе, NRIV_j обозначает количество записей, соответствующих j-му условию элемента. Аналогично в случае запроса из q условий элемеита с p атомными условиями в каждом потребуется выполнить, как и прежде, один поиск в индексе$_1$, но q поисков в индексе$_2$, каждый из которых длиннее, чем для случая одного атомного условия.

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, } q \text{ УЭ/УЗ, } p \text{ АУ/УЭ)} = \\ = \text{РВА}_1 + \sum_{j=1}^{q} \text{РВА}_{2Aj} + \left[\min_j \left(\sum_{i=1}^{p} \text{NRIV}_{ij}\right) \text{rba}\right], \qquad (15\text{-}7)$$

где $q \geqslant 1$ и $p \geqslant 1$.

В случае иескольких условий записи в запросе каждое из них анализируется отдельно, а общая оцеика получается как сумма оценок иезависимых операций. NTR_j обозначает количество записей-целей для j-го условия записи; предполагается также, что в целях устраиения повторяющихся записей выполияется сортировка.

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, } r \text{ УЗ/УЗР)} = \\ = \sum_{j=1}^{r} \text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, } q \text{ УЭ/УЗ, } p \text{ АУ/УЭ)}_j + \\ + \text{РВА}\left[\text{SORT}\left(\sum_{j=1}^{r} \text{NTR}_j\right)\right] \qquad (15\text{-}8)$$

для $q \geqslant 1$, $p \geqslant 1$ и $r \geqslant 1$.

15.2.2. Производительность обобщенного обновления

В мультиспискоиой организации операции обновления данных предшествует доступ к записям-целям. Для оценки производительности обиовлеиия необходимо зиать количество обновляемых записей-целей NTR. Изменение значения первичного ключа не затрагивает вторичиые индексы; это же справедливо в отношении изменения значеиия любого элемеита, не являющегося вторичным ключом. Следовательио, после изменения записи-цели (доступ к ией выполнен предварительно) для заверше-

ния транзакции достаточно выполнить ее перезапись.

$$\text{PBA(ИЗМЕНИТЬ неключевое значение)} = 1\,\text{sba}. \quad (15\text{-}9)$$

Изменение значения вторичного ключа вызывает, во-первых, удаление записи из одного пути доступа, возможно сопровождаемое удалением одного элемента из индекса$_2$, в случае когда

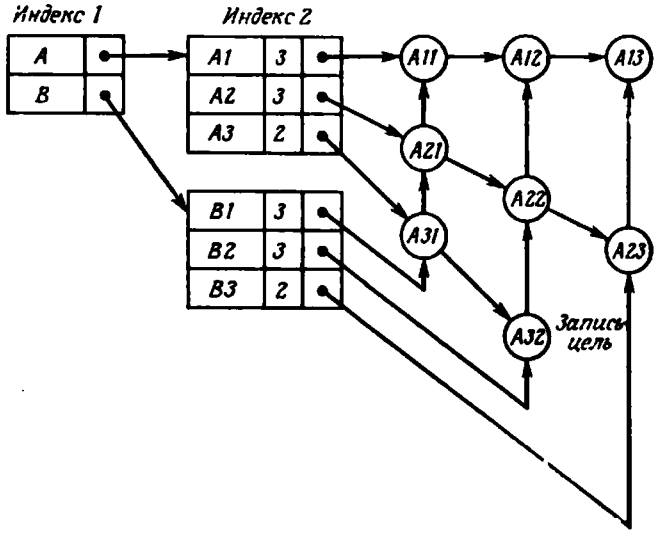


Рис. 15.3. Изменение значения ключа записи в мультисписковом файле.

в списке не остается других записей, и, во-вторых, включение ее в новый путь доступа, возможно сопровождаемое созданием нового элемента в индексе$_2$. При этом не требуется физическое перемещение записи данных, но значения соответствующих указателей в индексе должны быть изменены. Можно считать, что для управления плотным, упорядоченным индексом при выполнении операций включения и удаления требуется выполнить чтение и перезапись всего фрагмента индекса для подгруппы индекса, подчиненной одному элементу предыдущего уровня индекса. Таким фрагментом в индексе$_1$ мультисписка является список из NIT элементов. Каждый фрагмент индекса$_2$ включает NIV элемеитов.

$$\text{PBA(УДАЛИТЬ запись данных)} = \text{PBA (перезаписать значение указателя СЛЕДУЮЩАЯ в предыдущей записи)} + \text{PRDEL} \times$$
$$\times\,\text{PBA (прочитать и перезаписать индекс}_2) =$$
$$= 1\,\text{sba} + \text{PRDEL} \times \left\lceil \frac{\text{NIV}}{\text{EBF}_2} \right\rceil \times (2\,\text{sba}), \quad (15\text{-}10)$$

где PRDEL обозначает вероятность того, что запись является последней в списке записей-целей, т. е. определяет вероятность обновления индекса$_2$.

$$\text{РВА (ВКЛЮЧИТЬ запись данных)} =$$
$$= \text{РВА (записать новую запись данных и указатель)} +$$
$$+ \text{РВА (прочитать и перезаписать индекс}_2) = 1\,\text{rba} +$$
$$+ \text{PRINS} \times \left\lceil \frac{\text{NIV}}{\text{EBF}_2} \right\rceil \times (2\,\text{sba}), \qquad (15\text{-}11)$$

где PRINS обозначает вероятность того, что эта запись является первой в списке записей-целей, т. е. определяет вероятность обновления индекса$_2$.

$$\text{РВА (повторно ВКЛЮЧИТЬ запись данных с измененным значением вторичного ключа)} = \text{РВА (перезаписать измененную запись-цель с новым значением указателя СЛЕДУЮЩАЯ)} +$$
$$+ \text{PRINS} \times \text{РВА (прочитать и перезаписать индекс}_2) = 1\,\text{sba} +$$
$$+ \text{PRINS} \times \left\lceil \frac{\text{NIV}}{\text{EBF}_2} \right\rceil \times (2\,\text{sba}). \qquad (15\text{-}12)$$

$$\text{РВА (ИЗМЕНИТЬ вторичный ключ, NTR записей-целей)} =$$
$$= \text{NTR} \times \text{РВА (УДАЛИТЬ запись данных)} +$$
$$+ \text{РВА (найти новый элемент индекса}_2) +$$
$$+ \text{РВА (повторно ВКЛЮЧИТЬ измененную запись данных).} \qquad (15\text{-}13)$$

На рис. 15.3 приведен пример изменения значения вторичного ключа в записи данных.

15.2.3. Объем памяти

Легко видеть, что общий объем памяти, требуемой для мультисписковой организации, равен сумме количества блоков индекса типов элементов (индекса$_1$), количества блоков индекса значений элементов (индекса$_2$) и количества блоков записей данных. Выше введено предположение, что индекс$_1$ состоит из одного блока.

$$\text{BLKSTOR (мультисписковый файл)} = \text{BKS}_1 + \sum_{i=1}^{\text{NIT}} \left\lceil \frac{\text{NIV}_i}{\text{EBF}_2} \right\rceil \times$$
$$\times \text{BKS}_2 + \left(\sum_{i=1}^{\text{NIT}} \sum_{j=1}^{\text{NIV}} \text{NRIV}_{ij} \right) \times \text{SRS байт.} \qquad (15\text{-}14)$$

В силу того, что количество экземпляров записей, соответствующих значению элемента данных, переменно так же, как количество значений элемента данных каждого типа, в приведенном выражении вместо средних значений использованы суммы. Если же известны только средние значения, NIV и NRIV, получаем

$$\text{BLKSTOR (мультисписковый файл)} = \text{BKS}_1 + \text{NIT} \times$$
$$\times \left\lceil \frac{\text{NIV}}{\text{EBF}_2} \right\rceil \times \text{BKS}_2 + \text{NIT} \times \text{NIV} \times \text{NRIV} \times \text{SRS байт}, \quad (15\text{-}15)$$

где SRS определяется с учетом задания NIT указателей в каждой хранимой записи.

15.2.4. Секционный мультисписок

Одной из отрицательных черт, присущих мультисписковой структуре, является вероятность длинных списков записей-целей и, как следствие, увеличение количества произвольных обращений, требуемых для удовлетворения запроса. Если существует возможность физической кластеризации записей в списке записей-целей, то может оказаться полезным сгруппировать записи данных в так называемые секции; отсюда происходит термин «секционный». На рис. 15.4 показано, что секции могут быть неоднородными, то есть в одну секцию могут входить записи из разных списков записей-целей. При обращении к секции в процессе обработки некоторого списка записей необходимо опознать и пропустить записи из других списков, так называемые «ошибочные результаты поиска». Это действие является составной частью поиска и требует дополнительного использования CPU. В общем случае *ошибочным результатом поиска* называется любая запись, которая по результатам поиска в таблице индекса определена как удовлетворяющая запросу, но на самом деле не является записью-целью запроса. Для обнаружения ошибочных результатов поиска обычно после доступа к записи выполняется ее *тестирование на удовлетворение условию запроса.* Ошибочные результаты поиска часто появляются в тех способах вторичного доступа, которые в целях уменьшения общего времени ввода-вывода осуществляют выборку нескольких записей вместе с действительной записью-целью.

Секционный мультисписок представляет вариант мультисписковой организации, в котором указатели используются не для связи записей, а для связи физических блоков, в каждом из которых содержится по крайней мере одна запись-цель. Одним из достоинств этой структуры является потенциальное уменьшение объема памяти, занимаемой указателями; это воз-

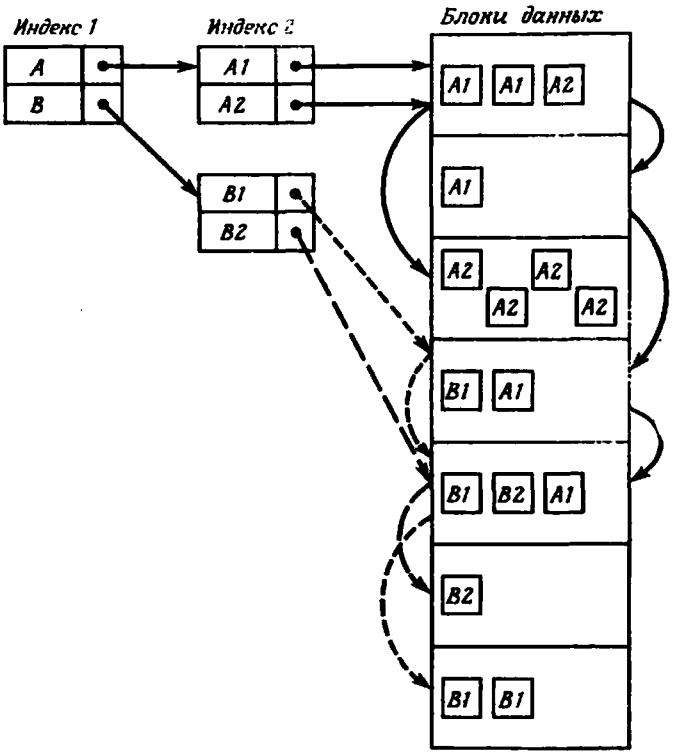


Рис. 15.4. Секционная мультисписковая организация.

можно, если в блоках будут размещаться записи, являющиеся однородными или близкими к однородным в смысле списков записей-целей; в противном случае приходится сохранить указатели записей. Более очевидное достоинство секционного мультисписка представляет умеиьшение количества обращений к физическим блокам при обращении ко всему списку записей. Когда блоки содержат полностью одиородные записи-цели, тогда затраты ввода-вывода будут наименьшими. Осиовным недостатком секциоииого мультисписка является дополнительное служебное использование CPU с целью обеспечения желаемой степени однородности блоков и проверкн каждого блока на наличие ошибочных результатов поиска. Этот недостаток может оказаться весьма существенным, поэтому выбору секционной мультисписковой структуры должен предшествовать подробный анализ альтернативных методов и сравнительная оценка их производительности.

15.3. Инвертированный файл

Выше было показано, что индексная мультисписковая структура позволяет эффективным образом обрабатывать простые запросы, касающиеся вторичных ключей. Однако, как видно из выражений (15-5)—(15-8), по мере усложнения запросов за счет ввода в них сложных булевых функций возрастают и коэффициенты, специфицирующие требуемое количество обращений к физическим блокам. В частности, мультисписковая структура теряет эффективность по мере возрастания количества условий элементов в запросе. Например, для запроса из примера 15-1 может случиться, что самый короткий список содержит 5000 записей и лишь 200 из них, или 4 %, действительно являются записями-целями всего запроса. Следовательно, большая часть обработки запроса тратится на выявление ошибочных результатов поиска. Более эффективная схема поиска может быть основана на использовании вместо списков записей массивов указателей. Количество указателей совпадает с количеством записей в списке, но при размещении они группируются в массивы указателей, называемые списками указателей доступа. *Список указателей доступа* представляет собой физически последовательный (линейный) список указателей записей, содержащих идентичное значение ключевого элемента.

Инвертированным файлом называют такую организацию или структуру файла, которая обеспечивает быстрый поиск для запросов общего вида, включающих спецификацию значений вторичного ключа. Он состоит из многоуровневого индекса и набора списков указателей доступа, обеспечивающих доступ к записям данных в соответствии с определенным критерием ключевого значения. В первичных методах доступа всегда осуществляется доступ к записи в соответствии со значением первичного ключа и лишь затем анализируются значения вторичных ключей или неключевых элементов. Инвертированные файлы (и вообще все вторичные методы доступа) вызывают модификацию указанного процесса; в них для определения местоположения класса записей данных используются значения вторичных ключей.

В наиболее распространенном варианте инвертированного файла используются двухуровневый индекс (как в мультисписковом файле) и набор списков указателей доступа, как показано на примере рис. 15.5. Записи данных, возможно, организованы в блоки (например, в соответствии с требованиями первичного метода доступа или в целях оптимизации операции чтения всего файла). Для заранее известных запросов допустима кластеризация записей, однако для динамичных запросов это исключено.

В целях лучшего понимания основных вопросов проектирования инвертированного файла необходимо ввести некоторые

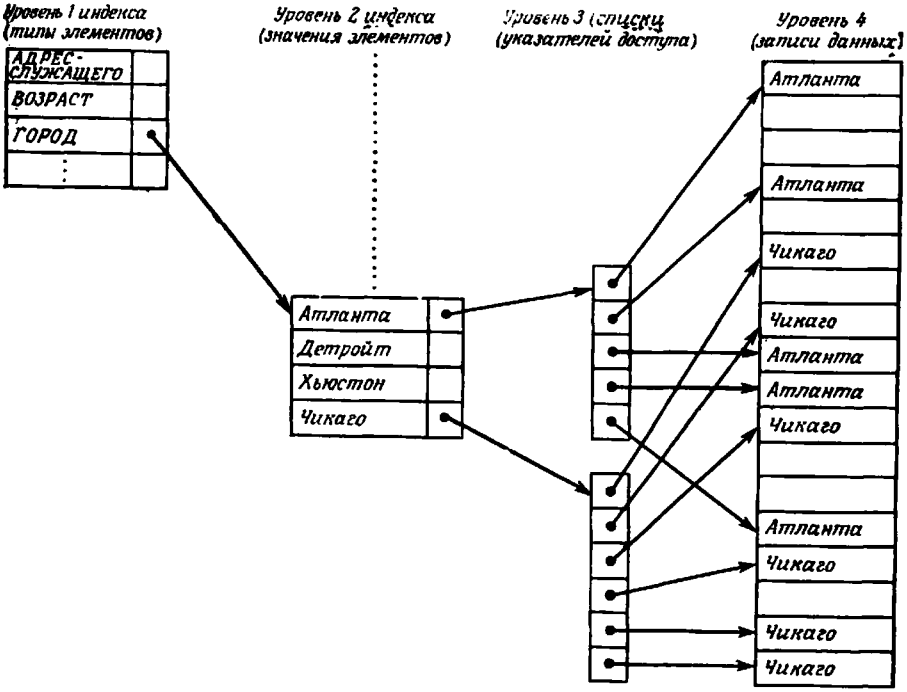


Рис. 15.5. Структура инвертированного файла с двухуровневым индексом для запроса из примера 15-1.

дополнительные понятия. Модель пути доступа не проясняет различие между понятиями частично и полностью инвертированных файлов. *Полностью инвертированный файл* инвертируется по каждому ключевому типу элемента данных, включая и первичные ключи (т. е. NSK = NIT, где NIT обозначает количество атрибутов или типов элементов данных в записи). Индекс первичного ключа представляет не что иное, как структуру с полным индексом (см. разд. 13.3). Во многих СУБД, таких, как ADABAS и Система 2000, предусмотрены средства организации полностью инвертированных файлов. Благодаря этому обеспечивается быстрый доступ к данным в соответствии со значением первичного или вторичного ключей или же их комбинации. Платой за такую полную степень свободы служат большие издержки памяти на организацию индекса. Есть надежда, что по мере удешевления памяти значимость этих служебных издержек будет ослабевать. *Частично инвертированный файл* инвертируется по выборочному числу ключевых типов элемен-

тов данных (т. е. NSK < NIT). Надо заметить, что вообще

$$NIT = NSK + NPK + NNK, \qquad (15\text{-}16)$$

где NSK обозначает количество вторичных ключей, NPK — количество первичных ключей, а NNK — количество неключевых элементов данных.

До сих пор речь шла об инвертированных файлах, индексы которых строились из типов и значений одиночных ключевых элементов. Такой способ индексирования принято называть способом с *отдельными индексами*. (В технической литературе они известны также как одноатрибутные или одноключевые индексы.) Кроме этого, можно использовать *составные индексы*, элементы которых соответствуют значениям сцепленных ключевых элементов различных типов. (Такие индексы принято называть многоатрибутными или многоключевыми индексами.) Возможность составного индексирования предусмотрена, например, в составе IMS и в других системах. В гл. 16 рассмотрены сравнительные оценки времени и объема памяти для случаев отдельных и составных индексов.

Ниже перечислены основные вопросы проектирования инвертированного файла:

- Тип обработки (наиболее приемлемы сложные конъюнктивные запросы, обобщенное обновление записей-целей).
- Сложность запросов.
- Количество уровней индекса.
- Объем памяти для индексов.
- Упорядоченность индекса, используемые механизмы поиска и коэффициент блокирования.
- Полная или частичная инвертированность.
- Выбор вторичного индекса.
- Отдельные или составные индексы.
- Упорядоченность списка указателей доступа, коэффициент блокирования.

15.3.1. Производительность выборки при обработке запросов

Анализ производительности выборки для инвертированных файлов рассматривается на примере операции ПОЛУЧИТЬ НЕКОТОРЫЕ и незапланированных конъюнктивных запросов. Для других операций целесообразно использовать первичные методы доступа. Как в случае мультисписконого файла, механизм поиска в инвертированном файле, особенно на уровнях индексов, зависит от конкретной системы. Если индексы упорядочены, можно использовать средства ускоренного поиска: бинарный поиск, бинарное дерево поиска, многоуровневые индексы и т. д. Поскольку для любого атомного условия в запросе необходим доступ ко всем элементам (указателям) списка указателей до-

ступа, его обработка всегда выполняется последовательно. Так как указатели списка доступа однозначно адресуют записи данных, то к последним производится прямой доступ.

Стратегия поиска для условий записей, содержащих несколько условий элементов, состоит в том, чтобы для каждого условия элемента найти с помощью индекса соответствующий список указателей доступа. Затем из этих списков за один проход строится обобщенный список указателей доступа, представляющий конъюнкцию (пересечение) исходных списков. Операция слияния списков указателей доступа может быть выполнена только в том случае, если они упорядочены по физическим адресам, специфицируемым этими указателями. Наконец, с помощью построенного списка осуществляется доступ к записям данных. Для инвертированного файла процесс поиска оказывается значительно более эффективным, чем для мультисписковой структуры (см. пример запроса 15-2).

Стратегия поиска в случае нескольких условий записей состоит в отдельной обработке каждого условия записи, упорядочении полученных записей-целей и, если это желательно, в их слиянии с целью удаления повторяющихся записей. По существу, этот же процесс используется для мультисписковых файлов.

Несколько атомных условий в запросе обрабатываются по отдельности; полученные в результате списки указателей доступа объединяются, прежде чем перейти к новому условию элемента. Иерархическое пошаговое слияние списков указателей доступа позволяет избежать их повторной сортировки и сохранить упорядоченность, необходимую для эффективного доступа к записям-целям, удовлетворяющим нескольким условиям элемента.

Приведенный ниже анализ производительности предполагает упорядоченность всех индексов и списков указателей доступа. В простейшем случае запрос состоит из одного атомного условия:

$$\begin{aligned}\text{PBA}(\text{ПОЛУЧИТЬ НЕКОТОРЫЕ, 1 АУ/УЭ}) = \\ = \text{PBA}_1(\text{поиск в индексе}_1) + \text{PBA}_2(\text{поиск в индексе}_2) + \\ + \text{PBA}_3(\text{поиск в списке указателей доступа}) + \\ + \text{PBA}_4(\text{доступ к записи-цели}),\end{aligned} \tag{15-17}$$

где $\text{PBA}_1 = 1\,\text{rba}$ (один блок, содержащий индекс$_1$), (15-18)

$$\text{PBA}_2 = 1\,\text{rba} + \left(\left\lceil \frac{\text{NIV} + 1}{2 \times \text{EBF}_2} \right\rceil - 1\right)\text{sba}, \tag{15-19}$$

$$\text{PBA}_3 = 1\,\text{rba} + \left(\left\lceil \frac{\text{NRIV}}{\text{EBF}_3} \right\rceil - 1\right)\text{sba}, \tag{15-20}$$

$$\text{PBA}_4 = \text{NRIV}\,\text{rba}, \tag{15-21}$$

а подстрочные индексы обозначают уровни поиска в представлении пути доступа. Для простоты в нижеследующих выражениях воспользуемся обозначениями оценок поиска, такими, как PBA_1, PBA_2 и т. д.

$$PBA(\text{ПОЛУЧИТЬ НЕКОТОРЫЕ, } p \text{ АУ/УЭ}) = [PBA_1] + \\ + \left[1\,\text{rba} + \left(\left\lceil \frac{(NIV - p + 1) + 1}{2 \times EBF_i} \right\rceil - 1\right)\text{sba} + \left\lceil \frac{p-1}{EBF_2} \right\rceil \text{sba}\right] + \\ + \left[p\,\text{rba} + \left(\sum_{i=1}^{p} \left\lceil \frac{NRIV_i}{EBF_3} \right\rceil - p\right)\text{sba}\right] + \left[\sum_{i=1}^{p} NRIV_i\right]\text{rba}. \quad (15\text{-}22)$$

Член во вторых скобах (условимся обозначать его PBA_{2A}) представляет оценку доступа к p последовательным значениям элемента данных в индексе$_2$. Член в третьих скобах (PBA_{3A}) представляет оценку начального и последующих обращений к нескольким (p) спискам указателей доступа, а член в четвертых скобках обозначает общее количество обращений к записям данных. Предполагается, что записи данных не сблокированы.

Для нескольких (q) условий элемента, каждое из которых содержит одно атомное условие, требуется выполнить почти полный просмотр индекса$_1$, поиск в q фрагментах индекса$_2$, q обращений к спискам указателей доступа, их слияние и обращение к NTR записям-целям, удовлетворяющим всем условиям элемента в условии записи.

$$PBA(\text{ПОЛУЧИТЬ НЕКОТОРЫЕ, } q \text{ УЭ/УЗ, 1 АУ/УЭ}) = \\ = PBA_1 + \sum_{j=1}^{q} PBA_{2j} + \sum_{j=1}^{q} PBA_{3j} + [\text{NTR rba}], \quad (15\text{-}23)$$

где PBA_{2j} обозначает колнчество обращений к физическому блоку для поиска в индексе$_2$, вызванного j-м условием элемента. Аналогично определяется PBA_{3j} для поиска в списке указателей доступа, вызванного j-м условием элемента.

Пусть f_A и f_B обозначают долю записей, содержащих заданное значение элементов данных типов A и B соответственно, и значения элементов независимы; тогда $f_A \times f_B$ представляет долю записей, содержащих одновременно заданные значения элементов данных типа A и B. Это же справедливо для двух, трех и более типов элементов данных. Следовательно, в предположении независимости значенний элементов данных можно на основе известных значений параметров, определяющих каждое условие элемента в условии записи, рассчитать значение NTR. В случае зависимости значений элементов данных указанных типов данный способ оценки NTR для условия записи неприменим; в этом

случае оценка значения NTR должна быть получена на основе эмпирических данных для известных запросов.

Для случая нескольких условий элемента и нескольких атомных условий оценка количества обращений равна

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, } q \text{ УЭ/УЗ, } p \text{ АУ/УЭ)} =$$

$$= \text{РВА}_1 + \sum_{j=1}^{q} \text{РВА}_{2A_j} + \sum_{j=1}^{q} \text{РВА}_{3A_j} + [\text{NTR rba}]. \qquad (15\text{-}24)$$

В силу того, что для нескольких условий записи множество записей-целей получается в результате объединения множеств записей-целей отдельных условий записи, оценка количества обращений для такого вида запроса равна сумме оценок для запросов с одним условием записи:

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, } r \text{ УЗ/УЗР)} =$$

$$= \sum_{j=1}^{r} \text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, } q \text{ УЭ/УЗ, } p \text{ АУ/УЭ)}_j +$$

$$+ \text{РВА}\left(\text{SORT}\left(\sum_{j=1}^{r} \text{NTR}_j\right)\right) \qquad (15\text{-}25)$$

для $r \geqslant 1$, $q \geqslant 1$ и $p \geqslant 1$.

15.3.2. Производительность обобщенного обновления

Как в случае мультисписконого файла, обновление инвертированного файла выполняется только после предварительной выборки записей-целей. Предполагается также, что в обеих структурах функциональное назначение двух основных индексов одинаково. Пусть по-прежнему NTR обозначает количество обновляемых записей-целей. При условии, что во вторичных индексах не представлены первичные ключи, изменение их значения или значения любого другого элемента данных — невторичного ключа — является простой операцией, требующей одной перезаписи измененной записи данных:

$$\text{РВА (ИЗМЕНИТЬ неключевое значение)} = 1\,\text{sba}. \qquad (15\text{-}26)$$

Изменение значения вторичного ключа, представленного в индексе$_2$, вызывает перезапись соответствующих записей-целей, удаление списка указателей доступа, адресующего эти записи, и его включение в структуру в качестве подчиненного другому ключевому значению в индексе$_2$. Физическое расположение записей данных остается неизменным. Если старое ключевое значение больше не требуется (вероятность этого случая PRDEL), выполняется удаление соответствующего элемента из индекса$_2$. Аналогично если новое ключевое значение еще не представлено в индексе$_2$ (вероятность этого случая PRINS), его необходимо

включить в индекс$_2$. Предполагается, что любая операция удаления или включения в индекс$_2$ вызывает выполнение чтения и перезаписи всего фрагмента индекса в целях сохранения его плотности и упорядоченности. Поскольку оценка операции выборки выполняется отдельно, операции чтения и перезаписи требуют только последовательных обращений к блокам.

PBA(ИЗМЕНИТЬ вторичный ключ, NTR записей) =
= NTR × (PBA(перезаписать измененную запись данных)) +
+ PBA(найти новое место для списка указателей доступа) +
+ 2 × PBA(прочитать и перезаписать список указателей доступа) + (PRDEL + PRINS) × PBA(прочитать и перезаписать индекс$_2$). (15-27)

Если значение NTR определяет все множество записей, содержащих заданное ключевое значение (т. е. NTR = NRIV), уравнение (15-27) можно упростить, заметив, что PRDEL=PRINS= = 1 и PBA (прочитать. и перезаписать список указателей доступа) = 0. В этом случае выполняется изменение всех записей, поэтому список указателей доступа остается неизменным; перемещается только соответствующий ему указатель в индексе$_2$. Однако это перемещение указателя уже учтено в оценке операции чтения и перезаписи фрагментов индекса$_2$ для старого и нового ключевых значений.

PBA(УДАЛИТЬ запись данных) = PBA(прочитать и перезаписать список указателей доступа) + PRDEL × PBA(прочитать и перезаписать индекс$_2$) = $\left\lceil \frac{\text{NRIV}}{\text{EBF}_3} \right\rceil$ × (2 sba) + PRDEL × × $\left\lceil \frac{\text{NIV}}{\text{EBF}_2} \right\rceil$ × (2 sba). (15-28)

PBA(ВКЛЮЧИТЬ запись данных) = PBA(записать новую запись данных) + PBA(прочитать и перезаписать список указателей доступа) + PRINS × PBA(прочитать и перезаписать индекс$_2$) = 1 rba + $\left\lceil \frac{\text{NRIV}}{\text{EBF}_3} \right\rceil$ × (2 sba) + PRINS × × $\left\lceil \frac{\text{NIV}}{\text{EBF}_2} \right\rceil$ × (2 sba). (15-29)

15.3.3. Объем памяти

Требования к памяти со стороны инвертированного файла включают: блок индекса типов элементов данных (индекс$_1$), блоки индекса значений элементов данных (все фрагменты ин-

декса$_2$), блоки списков указателей доступа и блоки данных:

$$\text{BLKSTOR (инвертированный файл)} = \text{BKS}_1 + \\ + \sum_{i=1}^{\text{NIT}} \left\lceil \frac{\text{NIV}_i}{\text{EBF}_2} \right\rceil \times \text{BKS}_2 + \sum_{i=1}^{\text{NIT}} \sum_{j=1}^{\text{NIV}} \left\lceil \frac{\text{NRIV}_{ij}}{\text{EBF}_3} \right\rceil \times \text{BKS}_3 + \\ + \sum_{i=1}^{\text{NIT}} \sum_{j=1}^{\text{NIV}} \text{NRIV}_{ij} \times \text{SRS байт.} \qquad (15\text{-}30)$$

Если известны только средние значения NIV и NRIV, получаем

$$\text{BLKSTOR (инвертированный файл)} = \text{BKS}_1 + \text{NIT} \times \\ \times \left\lceil \frac{\text{NIV}}{\text{EBF}_2} \right\rceil \times \text{BKS}_2 + \text{NIT} \times \text{NIV} \times \left\lceil \frac{\text{NRIV}}{\text{EBF}_3} \right\rceil \times \text{BKS}_3 + \\ + \text{NIT} \times \text{NIV} \times \text{NRIV} \times \text{SRS байт.} \qquad (15\text{-}31)$$

15.3.4. Сравнительная оценка производительности вторичных методов доступа

Лучше всего проиллюстрировать сравнительный анализ производительности инвертированного и мультисписконого файлов на примере конкретного запроса. Кроме того, полезно сравнить производительность указанных методов доступа с производительностью искусственно навязанного подхода последовательной обработки.

Пример запроса 15-2. Требуется сравнить оценки операцнй выборки и объема памяти для заданного условия запроса и указанных ниже спецификаций:

ВЫБРАТЬ ИМЯ, АДРЕС, ОБРАЗОВАНИЕ, ПОСЛУЖНОЙ-СПИСОК ИЗ ПЕДАГОГ
ГДЕ ГОРОД = ДЕТРОЙТ *И* ДОЛЖНОСТЬ = УЧИТЕЛЬ *И* КАТЕГОРИЯ = 6 *ИЛИ* 7 *ИЛИ* 8

NR = 10^6
SRS = 200 байт
BKS = 4 000 байт (индексы и данные)
BOVHD = 0 для всех блоков
LF = 1 для всех блоков
IS = 6 байт (в среднем для всех типов элементов данных)
PS = 4 байт
TSBA = 11,7 мс
TRBA = 36,7 мс
} спецнфикации устройства IBM 3350

NIT = 20
NIV (ГОРОД) = 100
NIV (ДОЛЖНОСТЬ) = 5
NIV (КАТЕГОРИЯ) = 13

NRIV(УЭ$_1$) = 10 000
NRIV(УЭ$_2$) = 750 000
NRIV(УЭ$_3$) = 150 000
NTR(УЗ) = 500 записей-целей

Физически последовательный файл и последовательный поиск

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ)} = \text{РВА(ПОЛУЧИТЬ ВСЕ)} = \left\lceil \frac{\text{LRA}}{\text{EBF}} \right\rceil = \left\lceil \frac{10^6}{20} \right\rceil = \frac{10^5}{2}\ \text{sba},$$

$$\text{TIO} = \frac{10^5}{2} \times \text{TSBA} = \frac{10^5}{2} \times 11{,}7\ \text{мс} = \underline{585\ \text{с.}}$$

Мультисписковый файл

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, 3 УЭ/УЗ, } p \text{ АУ/УЭ)} =$$

$$= \text{PBA}_1 + \sum_{j=1}^{3} \text{PBA}_{2A_j} + \min_j \left(\sum_{i=1}^{2} \text{NRIV}_{ij}\ \text{rba} \right) =$$

$$= [1\ \text{rba}] + \left[\left(1\ \text{rba} + \left(\left\lceil \frac{100-1+2}{2\times 400} \right\rceil - 1 \right) \text{sba} + 0 \right) + \right.$$

$$+ \left(1\ \text{rba} + \left(\left\lceil \frac{5-1+2}{2\times 400} \right\rceil - 1 \right) \text{sba} + 0 \right) +$$

$$\left. + \left(1\ \text{rba} + \left(\left\lceil \frac{13-1+2}{2\times 400} \right\rceil - 1 \right) \text{sba} + \left\lceil \frac{2}{400} \right\rceil \text{sba} \right) \right] +$$

$$+ [10\,000\ \text{rba}] = 10\,004\ \text{rba} + 1\ \text{sba},$$

$$\text{TIO} = 10\,004 \times 36{,}7\ \text{мс} + 1 \times 11{,}7\ \text{мс} = \underline{367\ \text{с.}}$$

Инвертированный файл

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, 3 УЭ/УЗ, } p \text{ АУ/УЭ)} =$$

$$= \text{PBA}_1 + \sum_{j=1}^{3} \text{PBA}_{2Aj} + \sum_{j=1}^{3} \text{PBA}_{3Aj} + \text{NTR rba} = [1\ \text{rba}] +$$

$$+ [4\ \text{rba} + 1\ \text{sba}] + \left[\left\{ 1\ \text{rba} + \left(\left\lceil \frac{10k}{1k} \right\rceil - 1 \right) \text{sba} \right\} + \right.$$

$$\left. + \left\{ 1\ \text{rba} + \left(\left\lceil \frac{750k}{1k} \right\rceil - 1 \right) \text{sba} \right\} + \left\{ 1\ \text{rba} + \left(\left\lceil \frac{150k}{1k} \right\rceil - 1 \right) \text{sba} \right\} \right] +$$

$$+ 500\ \text{rba} = 1\ \text{rba} + 4\ \text{rba} + 1\ \text{sba} + 1\ \text{rba} + 9\ \text{sba} + 1\ \text{rba} +$$

$$+ 749\ \text{sba} + 1\ \text{rba} + 149\ \text{sba} + 500\ \text{rba} = 508\ \text{rba} + 908\ \text{sba},$$

$$\text{TIO} = 508 \times 36{,}7\ \text{мс} + 908 \times 11{,}7\ \text{мс} = \underline{29\ \text{с.}}$$

По оценкам времени обработки запроса, инвертированный файл оказывается на порядок эффективнее мультисписка; для данного конкретного запроса оба вторичных метода доступа по производительности значительно превосходят последовательную обработку. Ниже приведена схема распределения требуемых объемов памяти для трех организаций файла:

	Индекс$_1$, К	Индекс$_2$, К	Списки указателей доступа, Мбайт	Записи данных, Мбайт	Общий объем памяти, Мбайт
Физически последовательный файл	0	0	0	200,0	200,000
Мультисписковый файл	4	4	0	212,0	212,008
Инвертированный файл	4	4	12	200,0	212,008

Хотя для инвертированного файла требуется на 6 % больше памяти, при этом, однако, его время ввода-вывода в 12 раз меньше в сравнении с мультисписковым файлом и почти в 20 раз меньше в сравнении с физически последовательной обработкой. Явное превосходство инвертированного файла в данном случае объясняется наличием небольшого количества записей-целей запроса. Если бы их было 30 000, то время ввода-вывода для инвертированного файла составило бы приблизительно 1112 с, что намного больше времени последовательной обработки (которое не зависит от количества записей-целей). Следовательно, для больших совокупностей записей-целей метод последовательной обработки оказывается более эффективным. Точку пересечения производительностей двух методов доступа определяет приблизительно 1,5 % базы данных. □

15.3.5. Секционный инвертированный файл

Секционная инвертированная организация представляет вариант организации инвертированного файла, в котором каждый список указателей доступа состоит из указателей физических блоков, содержащих по крайней мере одну запись данных со значением ключа, соответствующим списку. Основанием для такого проектного решения служит тот факт, что при любой естественной физической кластеризации записей данных с одинаковыми значениями вторичных ключей многие блоки будут состоять из нескольких подобных записей. В связи с этим средняя длина списка указателей доступа значительно уменьшится, и, следовательно, уменьшится среднее время ввода-вывода при обслуживании запроса. По порядку величины уменьшение времени ввода-вывода могло бы оказаться равнозначным использованию коэффициента блокирования данных, при котором блоки данных совершенно однородны в смысле наличия в них записей с одинаковым значением ключа (т. е. не содержат ошибочных результатов поиска). При разумном выборе размера блока время ввода-вывода для записей данных также умень-

шится. Для списков указателей доступа большой длины секционная инвертированная организация кажется весьма перспективной в качестве способа повышения эффективности обработки запроса. В худшем случае в каждом блоке находится одна запись-цель, то есть список указателей доступа совпадает со списком в инвертированной структуре; можно предположить, что если коэффициент блокирования для секционной инвертированной организации больше, то и время передачи блока данных окажется больше.

В обеих организациях требуется дополнительное время CPU на выполнение проверки соответствия записей данных всем условиям запроса; однако в случае секционной инвертированной организации наблюдается уменьшение времени CPU за счет меньшего процента ошибочных результатов поиска. При анализе производительности секционного инвертированного файла необходимо принять во внимание возможность кластернзацин записей данных и связанные с ней служебные издержки CPU. Указанные факторы зависят от конкретной системы, что особенно существенно в тех случаях, когда требуется оценить эффективиость программного обеспечения.

На рис. 15.6 приведен пример секционной инвертированной структуры. Из-за отсутствия указателей в блоках данных ее реализация оказывается проще в сравнении с секционным мультисписком (см. рис. 15.4), в котором блоки данных содержат переменные количества указателей в зависимости от их содержимого. В секционном мультисписке на обработку блоков и управление указателями требуется дополнительное использование CPU. В секционной инвертированной структуре также предусмотрены переменные количества указателей, но онн вынесены в списки указателей доступа.

Оценка производительности выборки:

$$\text{PBA (ПОЛУЧИТЬ НЕКОТОРЫЕ, 1 АУ/УЭ)} = \text{PBA}_1 + \text{PBA}_2 + \\ + \text{PBA}_{3C} + \text{PBA}_{4C}, \qquad (15\text{-}32)$$

где PBA_1 и PBA_2 обозначают оценки затрат на поиск в индексе, определенные для инвертированного файла соответственно в выражениях (15-18) и (15-19). PBA_{3C} обозначает оценку затрат на поиск в списке указателей доступа секционной инвертированной организации, а PBA_{4C} — оценку затрат на поиск записи данных:

$$\text{PBA}_{3C} = 1 \text{ rba} + \left(\left\lceil \frac{\text{NBIV}}{\text{EBF}_3} \right\rceil - 1 \right) \text{sba}, \qquad (15\text{-}33)$$

где NBIV обозначает среднее количество блоков, содержащих записи данных с конкретным значением ключевого элемента. В общем случае NBIV является переменной величиной; при

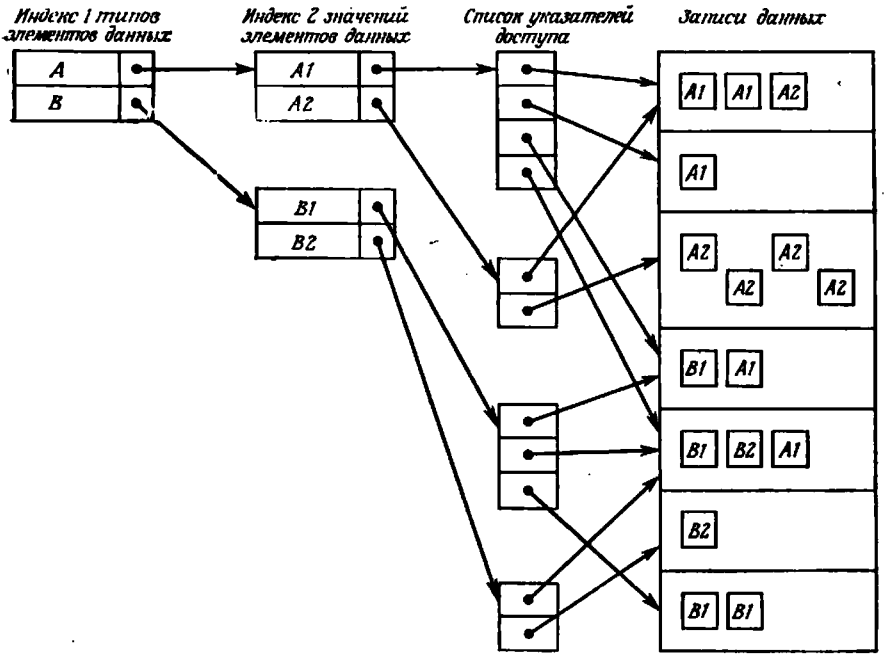


Рис. 15.6. Секционная инвертированная организация для двух типов элементов данных.

расчете отдельных оценок для разных типов элементов данных можно использовать обозначение $NBIV_i$. На рис. 15.6 значения $NBIV_i$ для четырех элементов данных равны соответственно 4, 2, 3 и 2.

$$PBA_{4C} = NBIV\ rba. \qquad (15\text{-}34)$$

Заметим, что при NBIV = NRIV производительность секционной инвертированной организации совпадает с производительностью инвертированной организации, не считая того, что секционная инвертированная организация имеет больший размер блока данных. Напомним читателю, что размер блока учитывается при преобразовании количества обращений к физическим блокам во время обслуживания ввода-вывода. Аналогично оценка запроса с несколькими атомными условиями для секционной инвертированной структуры получается в результате подстановки NBIV вместо NRIV в выражение (15-22). Выражения (15-23) и (15-24) не изменяются. Оценки операций обновления получаются в результате подстановки NBIV вместо NRIV

в уравнения (15-26)—(15-29). Оценка объема памяти равна

$$\text{BLKSTOR (секц. инв. файл)} = \text{BKS}_1 + \\ + \sum_{i=1}^{\text{NIT}} \left\lceil \frac{\text{NIV}_i}{\text{EBF}_2} \right\rceil \times \text{BKS}_2 + \sum_{i=1}^{\text{NIT}} \sum_{j=1}^{\text{NIV}} \left\lceil \frac{\text{NBIV}_{ij}}{\text{EBF}_3} \right\rceil \times \text{BKS}_3 + \\ + \text{NBLK}_4 \times \text{BKS}_4 \text{ байт,} \qquad (15\text{-}35)$$

где NBLK_4 обозначает количество блоков данных в файле или подфайле:

$$\text{NBLK}_4 = \left\lceil \frac{\text{NR}}{\text{EBF}_4} \right\rceil. \qquad (15\text{-}36)$$

Если известны лишь средние значения NIV и NBIV, получаем

$$\text{BLKSTOR (секц. инв. файл)} = \text{BKS}_1 + \text{NIT} \times \left\lceil \frac{\text{NIV}}{\text{EBF}_2} \right\rceil \times \text{BKS}_2 + \\ + \text{NIV} \times \text{NIT} \times \left\lceil \frac{\text{NBIV}}{\text{EBF}_3} \right\rceil \times \text{BKS}_3 + \text{NBLK}_4 \times \text{BKS}_4 \text{ байт.} \qquad (15\text{-}37)$$

Пример запроса 15-2 (продолжение). Продолжим расчет времени ввода-вывода для обработки запроса 15-2 в случае использования секционной инвертированной структуры:

$$\text{NBLK}_4 = \lceil 10^6/20 \rceil = 50\,000.$$

Предположим, что в каждом блоке из 20 записей данных содержится 95 % ошибочных результатов поиска и 5 % записей-целей. Тогда в каждом блоке есть одна запись-цель и всего NBIV = = 500 блоков, содержащих записи-цели.

$$\text{PBA (ПОЛУЧИТЬ НЕКОТОРЫЕ, 3 УЭ/УЗ, } p \text{ АУ/УЭ)} = \\ = \text{PBA}_1 + \sum_{j=1}^{3} \text{PBA}_{2Aj} + \sum_{j=1}^{3} \text{PBA}_{3ACj} + \text{NTR rba} = \\ = [1\text{ rba}] + [4\text{ rba} + 1\text{ sba}] + \left\{1\text{ rba} + \left(\left\lceil \frac{10k/20}{1k} \right\rceil - 1\right)\text{sba}\right\} + \\ + \left\{1\text{ rba} + \left(\left\lceil \frac{750k/20}{1k} \right\rceil - 1\right)\text{sba}\right\} + \\ + \left\{1\text{ rba} + \left(\left\lceil \frac{150k/20}{1k} \right\rceil - 1\right)\text{sba}\right\} + 500\text{ rba} = 1\text{ rba} + 4\text{ rba} + \\ + 1\text{ sba} + 1\text{ rba} + 0\text{ sba} + 1\text{ rba} + 37\text{ sba} + 1\text{ rba} + 7\text{ sba} + \\ + 500\text{ rba} = 508\text{ rba} + 45\text{ sba,}$$

$$\text{TIO} = 508 \times 36{,}7 \text{ мс} + 45 \times 11{,}7 \text{ мс} = \underline{19 \text{ с.}}$$

Производительность обработки запроса 15-2 при использовании секционной инвертированной структуры оказывается на

50 % лучше производительности инвертированного файла, которая при предыдущем сравнении оказалась лучшей среди других организаций. Однако служебные издержки CPU для секционной инвертированной организации могут оказаться большими и нуждаются в дополнительной оценке.

$$\text{BLKSTOR} = 4\text{K}(\text{индекс}_1) + 4\text{K}(\text{индекс}_2) + (1 + 75 + 15) \times$$
$$\times 4\text{K}(\text{списки указателей доступа}) +$$
$$+ 200\ \text{Мбайт}\ (\text{для блоков записей}) = 4\text{K} + 4\text{K} +$$
$$+ 91 \times 4\text{K} + 200\ \text{Мбайт} = 200{,}372\ \text{Мбайт}.$$

Списки указателей доступа в секционной инвертированной структуре короче, что обеспечивает экономию 11,632 Мбайт памяти.

15.4. Двусвязанное дерево

Последний из рассматриваемых вторичных методов доступа общего назначения представляет двусвязанное дерево. Его структура отличается от инвертированной и мультисписковой структур подобно тому, как *TRIE*-структура отличается от многоуровневой организации индекса. *Двусвязанное дерево* представляет вариант организации инвертированного файла, предусматривающий в иерархической структуре индекса отдельный уровень для каждого ключевого типа элемента данных; в качестве элементов индекса на каждом уровне представлены значения соответствующего ключевого типа элемента данных, а в блоках записей данных значения ключевых элементов данных, используемые для индексации, не содержатся. Последнее обстоятельство приводит к значительному уменьшению объема памяти для блоков данных в сравнении с инвертированным и мультисписковым файлами.

На рис. 15.7 приведен пример возможного варианта двусвязанного дерева для запроса из примера 15-1. В соответствии с тремя типами элементов данных, используемых для запоминания и выборки записей данных, предусмотрены три уровня индекса: ДОЛЖНОСТЬ, ГОРОД И ВОЗРАСТ. В индексе нижнего уровня представлены указатели блоков данных, или заголовков списков блоков данных, состоящих только из записей-целей, т. е. записей, значения ключевых элементов которых удовлетворяют критерию пути в дереве от индекса до блоков данных. Используя введенную выше терминологию, можно назвать блоки однородными в смысле содержащихся в них записей. Для согласованности с вышеизложенной методикой анализа предполагается, что все элементы индекса соответствуют непустым блокам данных (блокам-целям). Это означает, что если в ре-

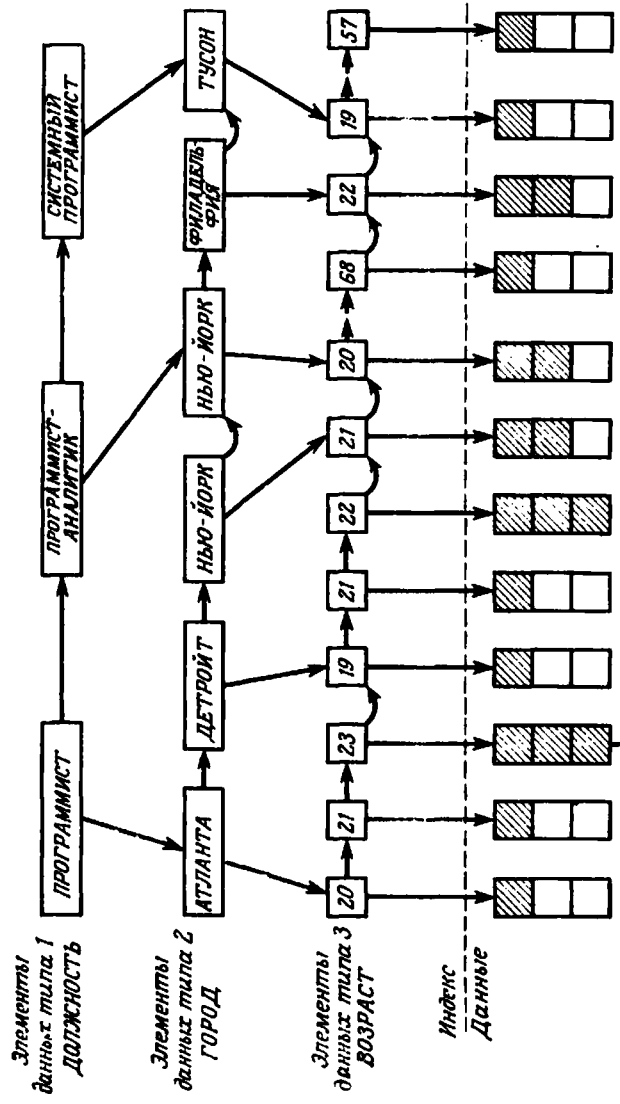


Рис. 15.7. Структура двусвязанного дерева для запроса из примера 15-1.

зультате выполнения операции удаления блок данных становится пуст, из индекса удаляются все элементы, которые в конечном счете ведут к пустым блокам. Предполагается, что реорганизация индекса выполняется динамически.

Для любого обобщенного запроса множество записей-целей в двусвязанном дереве может быть пустым в случае запроса, на который нет ответа, может находиться в одном блоке или списке блоков в случае запроса из одного атомного условия или же размещаться в нескольких блоках в случае запроса из нескольких атомных условий, условий элемента и (или) условий записи. На рис. 15.7 показано, что в результате обработки рассматриваемого запроса будет обнаружено не менее восьми записей-целей, расположенных в четырех блоках. В двусвязанном дереве отсутствует дублирование записей данных в представлении пути доступа, т. е. не существует виртуальных записей. Однако в случае непустого пересечения множеств записей-целей при обработке запроса из нескольких условий записи по-прежнему может потребоваться последующая сортировка. Например, в результате повторного доступа к левому блоку дерева на рис. 15.7 при обработке запроса (ДОЛЖНОСТЬ = ПРОГРАММИСТ $\wedge$ ГОРОД = АТЛАНТА) $\vee$ (ВОЗРАСТ = 20) будут получены повторяющиеся записи-цели.

В примере на рис. 15.7 количество значений элемента данных типа 1, NIV_1 постоянно. Однако на уровне $j \geqslant 2$ NIV_i является переменной функцией от значения исходного элемента на уровне $i - 1$. Для любого фрагмента индекса значение NIV_i в точности равно количеству различных значений элемента данных типа i, содержащихся в записях данных к настоящему моменту и подчиненных совокупности уникальных значений элементов данных в дереве. Например, в первом (левом) фрагменте индекса уровня 2 на рис. 15.7 содержатся три вершины вследствие того, что имеются три различных значения элемента типа 2 в записях, содержащих указанное (левое) значение элемента типа 1 (на уровне 1). Длина поиска на уровне данных NR_{NIL+1} равна количеству записей-целей, содержащих последовательность значений элементов данных, совпадающую со значением в иерархии вершин от корня до блока или списка блоков данных. Для реального запроса множество записей-целей состояло бы из одной или нескольких групп записей-целей соответственно для каждой последовательности значений элементов данных.

В работах [56, 58, 59, 263] авторы провели исследование вопросов стратегии конструирования структур индекса, оптимальных с точки зрения объема памяти и эффективности времени доступа. Основное правило минимизации объема памяти индекса состоит в упорядочении ключевых типов элементов по количеству значений NIV_i и их размещении на уровнях 1 - - NIL

дерева в порядке возрастания значений NIV_i. Вследствие того что в нижней части дерева возрастает избыточность хранения значений типов элементов, при указанном подходе минимизируется размер верхней части дерева и этим обеспечивается управление размером нижней части. На рис. 15.8 приведен пример файла или подфайла, включающего три активных типа элемента данных: значения $NIV_1 = 1$, $NIV_2 = 3$ и $NIV_3 = 7$ — и восемь уникальных комбинаций ключевых значений. На рис. 15.8, *б* показана минимальная конфигурация дерева для этого файла, когда типы элементов присвоены уровням в порядке возрастания значений NIV_i. На рис. 15.8, *г* представлена максимальная конфигурация, имеющая место при присвоении типов элементов уровням в порядке уменьшения значений NIV_i. Рис. 15.8 позволяет определить теоретически допустимые верхнюю и нижнюю оценки количества вершин в NIL-уровневом индексе двусвязанного дерева:

$$\sum_{i=1}^{NIL} NIV_i \leqslant INODES \leqslant (NIL - 1) \times UKV + NIV_{макс}, \quad (15\text{-}38)$$

где INODES обозначает количество вершин индекса, UKV — количество уникальных комбинаций ключевых значений в записях данных, а $NIV_{макс}$ — максимальное значение NIV_i для всех элементов данных типа i.

Основная стратегия минимизации времени обслуживания ввода-вывода для структуры двусвязанного дерева состоит в размещении типов элементов данных на уровнях 1 — NIL в порядке уменьшения частот их использования в запланированных запросах со стороны прикладных программ и в порядке возрастания частот обновления. Тогда на уровне 1 располагаются значения наиболее часто используемого типа элемента данных. При таком подходе минимизируется общее количество вершин индекса, просматриваемых во многих, но не во всех случаях. Кроме того, на практике рекомендуется спроектировать несколько вариантов структуры и оценить каждый с точки зрения запланированных операций выборки и обновления.

Как установлено, двусвязанное дерево имеет ряд явных преимуществ в сравнении с инвертированным файлом. Во-первых, вследствие того, что вторичные ключи, используемые в индексе, вынесены из записей данных, уменьшается требуемый объем памяти. Во-вторых, в силу того что обновление вторичного ключа ограничено обновлением индекса, выполнение операции обновления требует меньшего времени ввода-вывода. В случае инвертированного файла подобные обновления затрагивают и индекс, и записи данных. В-третьих, можно ускорить обработку запланированных запросов путем конструирования структуры

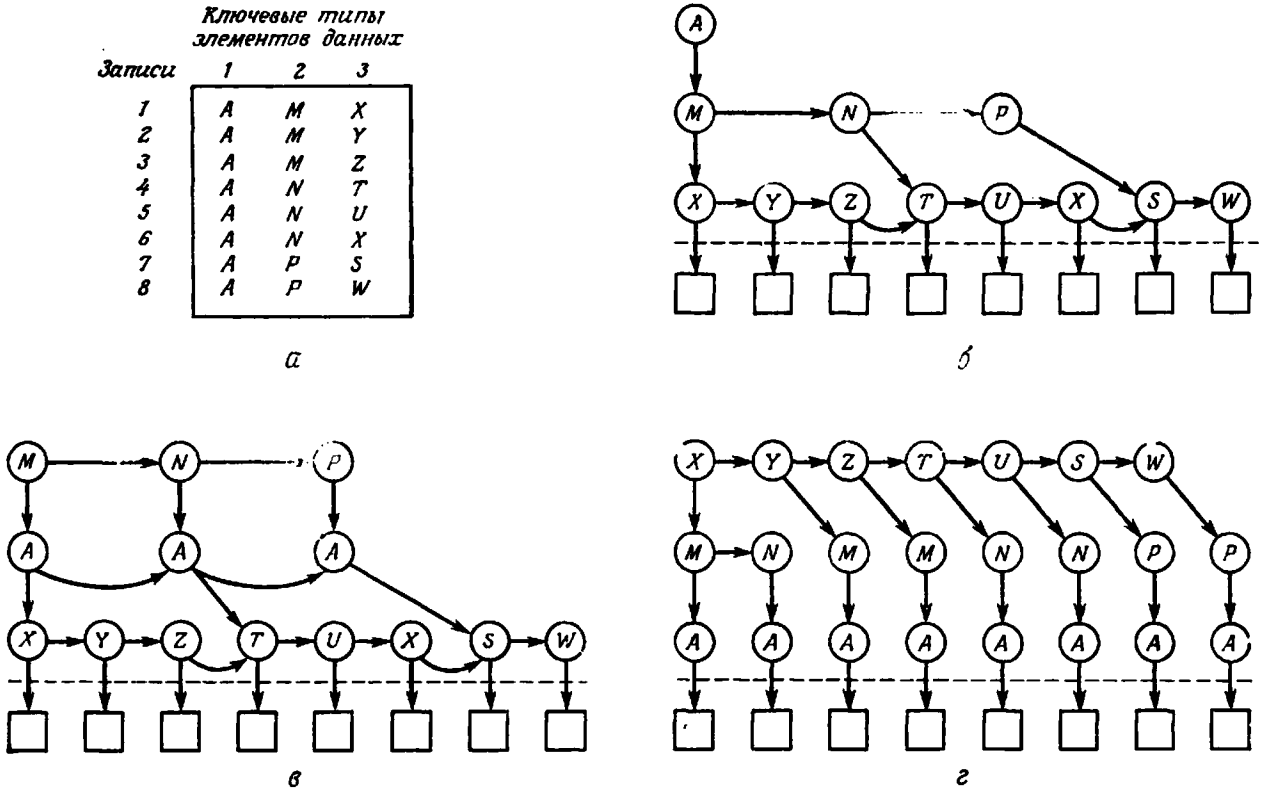


Рис. 15.8. Организация хранения индекса двусвязанного дерева.

а — примеры значений ключевого элемента; *б* — упорядоченность типов элементов данных: 1, 2, 3 (12 вершин); *в* — упорядоченность типов элементов данных: 2, 1, 3 (14 вершин); *г* — упорядоченность типов элементов данных: 3, 2, 1 (23 вершины).

дерева, минимизирующей количество обращений к вершинам индекса в процессе доступа к блоку записей-целей. К сожалению, существуют конфигурации записей данных и требования к обработке, препятствующие проявлению перечисленных преимуществ. Фиксированная структура дерева индекса может оказаться крайне неэффективной для обработки незапланированных запросов. В следующих разделах выводятся оценки операций выборки и обновления двусвязанных деревьев; метод оценки согласуется с вышеизложенным методом оценки мультисписковых и инвертированных файлов, что обеспечивает возможность сравнительного анализа их производительности.

Ниже перечислены основные вопросы проектирования двусвязанного дерева; их подробный анализ содержится в следующих разделах:

- *Тип обработки.* Наиболее подходят обобщенные запросы и обновления; незапланированные запросы в отличие от запланированных; сложность запроса.
- *Метод присвоения типов ключевых элементов уровням индекса.* Влияет на объем памяти и время доступа.
- *Частота использования отдельных ключевых элементов в запросах.* Влияет на метод присвоения ключевых элементов уровням индекса.
- *Упорядоченность индекса.*
- *Размер блока индекса и записей данных.*
- *Ограничения на объем памяти.*

15.4.1. Производительность выборки при обработке запросов

В основе оценки производительности структуры двусвязанного дерева и его механизма поиска используются представление дерева (рис. 15.7) и определение обобщенного запроса из разд. 15.1. Вначале мы рассмотрим случай запроса, составленного из нескольких условий элементов, причем каждое условие элемента включает одно атомное условие: $УЭ_1 \wedge УЭ_2 \wedge \ldots \wedge УЭ_q$, где для любого i $УЭ_i$ имеет вид «тип элемента $\left\{\begin{matrix}<\\=\\>\end{matrix}\right\}$ значение элемента». В этом случае механизм поиска выполняет просмотр дерева, начиная с уровня 1 и продвигаясь уровень за уровнем вниз по дереву до уровня NIL. На каждом уровне возможны два случая. В первом случае ключевой тип элемента уровня i входит в условие запроса, тогда поиск заданного в запросе значения элемента продолжается на уровне i. При условии успешного поиска средняя длина пути составляет половину длины списка индекса на уровне i. Случай неуспешного поиска рассмотрен ниже. Во втором случае ключевой тип элемента на

уровне i не входит в запрос; это означает, что допустимы все значения ключевого элемента и на этом уровне необходим доступ ко всем элементам списка.

По мере продвижения по дереву количество просматриваемых порожденных подмножеств (фрагментов индекса) сильно возрастает. По отношению к предыдущему уровню, исходной вершине, снова возможны два случая. В первом случае ключевой тип элемента входит в запрос, тогда во фрагменте индекса существует в точности одна вершина, удовлетворяющая условию запроса на уровне $i-1$ и имеющая порожденное подмножество на уровне i. Во втором случае ключевой тип элемента не входит в запрос; это означает, что необходимо просмотреть все вершины во фрагменте индекса на уровне $i-1$ и порожденные ими подмножества на уровне i. Таким образом теперь можно оценить возможную длину пути поиска на каждом уровне индекса. Для $1 \leqslant q \leqslant \mathrm{NIL}$

$$\mathrm{PBA}(\text{ПОЛУЧИТЬ НЕКОТОРЫЕ, } q \text{ УЭ/УЗ, 1 АУ/УЭ}) = \\ = \text{поиск в индексе} + \text{поиск в блоках данных} = \\ = \sum_{i=1}^{\mathrm{NIL}} \mathrm{LEN}_i\,\mathrm{rba} + \mathrm{NTB}\,\mathrm{rba}, \qquad (15\text{-}39)$$

где $\mathrm{LEN}_i = \left(\prod_{j=1}^{i-1} \mathrm{CARD}_j\right) \times \mathrm{WID}_i$ равно количеству просмотренных вершин на уровне i.

Аналогично

$$\mathrm{WID}_i = \begin{cases} \left\lceil \dfrac{1+\mathrm{NIV}_i}{2} \right\rceil, & \text{если тип элемента } i \text{ входит в запрос,} \\ \mathrm{NIV}_i & \text{в противном случае} \end{cases}$$

$$\text{и } \mathrm{CARD}_j = \begin{cases} 1, & \text{если тип элемента } j \text{ входит в запрос,} \\ \mathrm{NIV}_j & \text{в противном случае.} \end{cases}$$

Переменная NTB обозначает количество блоков, содержащих NTR записей-целей запроса. Надо заметить, что NTB не меньше, чем $\mathrm{LEN}_{\mathrm{NIL}+1}$ или $\left\lceil \dfrac{\mathrm{NTR}}{\mathrm{EBF}_{\mathrm{NIL}+1}} \right\rceil$.

В случае $q = \mathrm{NIL}$, то есть когда каждый тип элемента индекса входит в условие элемента запроса, уравнение (15-39) можно упростить. Для этого случая имеем

$$\mathrm{PBA}(\text{ПОЛУЧИТЬ НЕКОТОРЫЕ, } q = \mathrm{NIL} \text{ УЭ/УЗ, 1 АУ/УЭ}) = \\ = \sum_{i=1}^{\mathrm{NIL}} \frac{1+\mathrm{NIV}_i}{2}\,\mathrm{rba} + \mathrm{NTB}\,\mathrm{rba}. \qquad (15\text{-}40)$$

При условии, что выборка значений вторичных ключей происходит независимо от выборки значений хранимых ключей, уравнения (15-39) и (15-40) остаются справедливыми для любого способа упорядочения списков уровней индекса.

В случае неуспешного поиска средиюю длину поиска определить затруднительно. Нижняя оценка равна NR_i — максимальной длине первого уровня индекса. При этом, если искомое значение ключа на первом уровне не найдено, поиск немедленно завершается. Верхняя оценка определена уравнением (15-39), за тем исключением, что $WID_i = NIV_i$ независимо от того, входит i-й тип элемента в запрос или нет.

Если помимо нескольких условий элемента в запросе каждое условие состоит из нескольких атомных условий, то также возможны два случая. В первом случае для проверки атомных условий необходимо выполнить доступ к порожденным фрагментам индекса на всех уровнях ниже корня (на пути доступа). Для неупорядоченных списков индекса всегда имеет место первый случай. В этом случае для оценки производительности используется уравнение (15-39) при условиях $WID_i = NIV_i$ и $CARD_j = NIV_j$. Во втором случае списки упрядочены; благодаря этому, согласно теореме 12.1, длину поиска можно уменьшить до (максимум) $p_i/(p_i + 1)$, где p_i обозначает среднее количество атомных условий в условии элемента для i-го типа элемента. Однако такое незначительное уменьшение длины поиска, вероятно, не стоит дополнительных расходов на упорядочение условий элемента в запросе в соответствии с упорядоченностью списка индекса. Вследствие этого верхняя и нижняя оценки для WID_i и $CARD_j$ в случае неупорядоченного списка совпадают с оценками упорядоченного списка.

В случае нескольких условий записей в запросе их оценка выполняется отдельно, и для получения общей оценки запроса берется сумма оценок ввода-вывода для всех условий записи. С целью выявления и удаления избыточных записей-целей можно использовать операцию сортировки.

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, } r \text{ УЗ/запрос)} =$$

$$= \sum_{j=1}^{r} \text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, } q \text{ УЭ/УЗ, } p \text{ АУ/УЭ)}_j + \\ + \text{SORT}\left(\sum_{j=1}^{r} \text{NTR}_j\right). \qquad (15\text{-}41)$$

15.4.2. Производительность обобщенного обновления

Метод анализа операций обновления для двусвязанного дерева подобен методу анализа мультисписковых и инвертированных файлов. Вновь обратимся к рис. 15.7 как к общей мо-

дели структуры двусвязанного дерева и предположим, что предварительный поиск изменяемой или обновляемой записи был успешным. Поскольку записи данных сблокированы, оценка операции обновления записей-целей некоторого запроса больше не является линейной функцией числа NTR. В любой заданный момент времени блокн могут оказаться заполненными лишь частично; следовательно, без анализа реальных статистических данных из опыта прошлого использования блоков трудно предсказать, сколько блоков данных сейчас активны. Подобные статистические данные принципиально важны для анализа средних значений производительности; в противном случае можно вычислить только ее верхнюю и нижнюю оценки.

Прежде всего заметим, что изменение неключевых значений в записи данных влечет за собой только перезапись измененного блока.

$$\text{РВА (ИЗМЕНИТЬ неключевое значение)} = = \begin{cases} 1\,\text{sba} & \text{для изменения 1 записи,} \\ \text{NTB}\,\text{sba} & \text{для изменения NTR блокированных записей.} \end{cases} \tag{15-42}$$

Операция удаления записи данных выполняется посредством ее физического удаления из блока и перезаписи измененного блока. Кроме того, если это последняя запись в списке блоков, то с вероятностью $\text{PRDEL}_{\text{NIL}}$ необходимо удалить элемент индекса на уровне NIL, соответствующий удаляемому блоку данных. Удаление элементов индекса может затронуть все уровни до уровня 1, но по мере приближения к уровню 1 и уменьшения повторяющихся ключевых значений наблюдается снижение вероятности удаления. Поскольку вершина индекса реализуется в виде связанной последовательной структуры, для ее удаления необходимо изменить значение указателя СЛЕДУЮЩАЯ в предыдущей вершине. В предположении, что предыдущая вершина (или элемент) все еще находится в буфере, куда она была занесена при поиске удаляемой записи-цели, достаточно выполнить лишь ее физическую перезапись.

$$\text{РВА (УДАЛИТЬ)} = \text{РВА (перезаписать блок данных)} + + \sum_{i=1}^{\text{NIL}} \text{PRDEL}_i \times \text{РВА (перезаписать исходный элемент индекса)} = = \begin{cases} 1\,\text{sba} + \sum_{i=1}^{\text{NIL}} \text{PRDEL}_i \times (1\,\text{sba}) & \text{при удалении 1 записи,} \\ 0\,\text{sba} + \sum_{i=1}^{\text{NIL}} \text{PRDEL}_i \times (1\,\text{sba}) & \text{при удалении всех NTR записей.} \end{cases} \tag{15-43}$$

Простейший способ включения записи данных состоит в определении местоположения соответствующего блока данных, физическом включении записи данных в блок и последующей перезаписи блока. С вероятностью $PRINS_{NIL}$ блока данных, предназначенного для записей, которые удовлетворяют заданным условиям элемента, может еще не быть в файле. В этом случае необходимо создать соответствующие элементы индекса, а также создать и записать блок данных, содержащий включаемую запись. Для включения нового элемента в индекс требуется установить соответствующее значение указателя СЛЕДУЮЩАЯ в предшествующей записи, перезаписать ее и записать новый элемент индекса после установки в нем значения указателя СЛЕДУЮЩАЯ.

$$\text{РВА (ВКЛЮЧИТЬ)} + \text{РВА (записать или перезаписать блок данных)} + \sum_{i=1}^{NIL} PRINS_i \times \text{РВА (записать новый элемент индекса} + \text{перезаписать предшествующий элемент)} =$$

$$= [PRINS_{NIL} \times (1\ rba) + (1 - PRINS_{NIL}) \times (1\ sba)] + \sum_{i=1}^{NIL} PRINS_i \times (1\ rba + 1\ sba). \qquad (15\text{-}44)$$

По мере приближения к уровню 1 вероятность PRINS уменьшается подобно вероятности PRDEL.

Изменение значения вторичного ключа в группе записей не затрагивает физические (хранимые) записи, а касается только индекса. После завершения предварительного поиска записей данных выполняется изменение значений соответствующих указателей данных в индексе:

$$\text{РВА (ИЗМЕНИТЬ вторичный ключ)} = \text{РВА (УДАЛИТЬ NTR записей)} + \text{РВА (найти путь для включения нового вторичного ключа)} + \text{РВА (ВКЛЮЧИТЬ — перезаписать блок данных).} \qquad (15\text{-}45)$$

15.4.3. Объем памяти

Требования к памяти со стороны двусвязанного дерева состоят из требований к хранению записей индекса и записей дан-

ных. В случае несблокированных записей индекса

$$\text{BLKSTOR (двусвязанное дерево)} = \text{объем памяти индекса} + \text{объем памяти данных} = \sum_{i=1}^{\text{NIL}} \left(\prod_{j=1}^{i} \text{NIV}_j \text{(активные)} \right) \times \text{SRS} + \left\lceil \frac{\text{NR}}{\text{EBF}_2} \right\rceil \times \text{BKS}_2 \text{ байт}, \quad (15\text{-}46)$$

где подстрочный индекс 1 обозначает область индекса, а подстрочный индекс 2 — область данных. В случае сблокированных записей индекса

$$\text{BLKSTOR (двусвязанное дерево)} = \left\lceil \sum_{i=1}^{\text{NIL}} \left(\frac{\prod_{j=1}^{i} \text{NIV}_j}{\text{EBF}_1} \right) \right\rceil \times \text{BKS}_1 + \left\lceil \frac{\text{NR}}{\text{EBF}_2} \right\rceil \times \text{BKS}_2. \quad (15\text{-}47)$$

15.4.4. Сравнение двусвязанного дерева с инвертированным и мультисписковым файлами

Из практических соображений полезно уметь определять условия, при которых двусвязанное дерево обеспечивает лучшую производительность в сравнении с инвертированным или мультисписковым файлом. Запрос из примера 15-2 (см. разд. 15.3.4 и 15.3.5) позволяет сравнить производительность операций выборки, а приведенный ниже пример запроса 15-3 позволит получить сравнительную оценку производительности операций обновления.

Пример запроса 15-2 (продолжение).

$$\text{NTR} = 500 \text{ записей-целей}$$

Предположим, что блоки данных в усредненном случае заполнены наполовину, а в минимальном (лучшем) случае — полностью. Рассмотрим указанные случаи более подробно.

$$\text{EBF}_2 = \left\lfloor \frac{4000}{200 - 3 \times 6} \right\rfloor = 21, \quad \text{NBLK} = \left\lceil \frac{\text{NR}}{\text{EBF}_2} \right\rceil \times 2 = \left\lceil \frac{10^6}{21} \right\rceil \times 2 = 95\,240.$$

В данном примере имеется только один прикладной запрос, поэтому распределение уровней индекса не оказывает существенного влияния на эффективность выборки. Однако оно

влияет на требуемый объем памяти. Поэтому упорядочим уровни индекса по возрастанию значений NIV_i.

Уровень 1 = ДОЛЖНОСТЬ	$NIV_1 = 5$,
Уровень 2 = КАТЕГОРИЯ	$NIV_2 = 13$,
Уровень 3 = ГОРОД	$NIV_3 = 100$.

В минимальной конфигурации размер вершины индекса в среднем равен 6 байт для значения элемента и 2×4 байт для двух указателей.

$$EBF_i = \left\lfloor \frac{4000}{14} \right\rfloor = 285,$$

$$BLKSTOR_{ср} = \left\lceil \frac{5 + 5 \times 13 + 5 \times 13 \times 100}{285} \right\rceil \times 4000 + 95\,240 \times$$

$$\times 4000 = 24 \times 4000 + 95\,240 \times 4000 = 381{,}056 \text{ Мбайт},$$

$$BLKSTOR_{мин} = 24 \times 4000 + \frac{95\,240}{2} \times 4000 = 190{,}076 \text{ Мбайт}.$$

Минимальный требуемый объем памяти на 5 % меньше в сравнении с инвертированной, секционной инвертированной и мультисписковой структурами, в усредненном случае требуемый объем памяти на 90 % больше, чем во всех других структурах.

В случае несблокированного индекса

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, 3 УЭ/УЗ)} =$$

$$= \left(\left\lceil \frac{1+5}{2} \right\rceil + \left\lceil \frac{1+13}{2} \right\rceil + \left\lceil \frac{1+100}{2} \right\rceil \right) rba_1 + NTB\ rba_2 =$$

$$= (3 + 7 + 51)\, rba_1 + 48\, rba_2 = 61\, rba_1 + 48\, rba_2,$$

$$NTB_{ср} = \left\lceil \frac{NTR}{EBF_2} \right\rceil \times 2 = \left\lceil \frac{500}{21} \right\rceil \times 2 = 48,$$

$$TIO = 61 \times 33{,}4 \text{ мс} + 48 \times 36{,}7 \text{ мс} = \underline{3{,}8 \text{ с.}}$$

В случае сблокированного индекса

$$\text{РВА(ПОЛУЧИТЬ НЕКОТОРЫЕ, 3 УЭ/УЗ)} =$$

$$= 3\, rba_2 + 48\, rba_2 = 51\, rba_2,$$

$$TIO = 51 \times 36{,}7 \text{ мс} = \underline{1{,}9 \text{ с.}}$$

В табл. 15.1 для запроса из примера 15-2 приведены сравнительные характеристики производительности операции выборки для пяти вариантов методов доступа.

Из табл. 15.1 видно, что в случае двусвязанного дерева наблюдается значительное уменьшение времени ввода-вывода при организации записей данных в блоки. В худшем случае, если бы в каждом блоке находилась только одна запись-цель, было бы NTR = NTB = 500 блоков данных и разница во временах ввода-

вывода была бы еще более ощутима. Использование сблокированных индексов также оказывает значительное влияние на производительность, но не в большей степени, чем блокирование записей данных. Таким образом, в случае если невозможно обеспечивать высокую степень заполненности блоков данных,

Таблица 15.1. **Оценки времени ввода-вывода операции выборки и объема памяти для примера запроса 15-2**

Метод доступа	Время ввода-вывода для обработки запроса, с	Объем памяти, Мбайт
(Физически) последовательный	585	200,0
Мультисписковый	367	212,0
Инвертированный	29	212,0
Секционный инвертированный	19	200,4
Двусвязанное дерево		
Минимальная конфигурация	1,9—3,8 [1])	190,6
Усредненная конфигурация	3,6—5,6 [1])	381,1

[1]) Нижняя оценка соответствует случаю сблокированного индекса; верхняя — случаю несблокированного индекса.

двусвязанному дереву свойственна вероятность больших служебных издержек памяти. □

Нам осталось сравнить вторичные методы доступа на основе оценки операций обновления. Давайте рассмотрим пример процедуры обновления и для оценки производительности вариантов вторичных методов доступа воспользуемся прежней аналитической моделью.

Пример запроса 15-3. Для запроса из примера 15-2 вычислить время ввода-вывода следующих операций обновления: ИЗМЕНИТЬ неключевой элемент, УДАЛИТЬ, ВКЛЮЧИТЬ И ИЗМЕНИТЬ вторичный ключ ГОРОД. Предположим, что для всех уровней PRDEL = PRINS = 0,1, и рассмотрим два случая NTR = 1 и 100.

Основные параметры

PRDEL = PRINS = 0,1,
NTR = 1100,
$NIV_{ГОРОД} = 100$,
$NRIV_{ГОРОД} = 10\,000$,
$NBIV_{ГОРОД} = \frac{10\,000}{EBF_3} \times 2 = 20$ наполовину заполненных блоков.

$EBF_1 = EBF_2 = 400$ для индекса$_1$ и индекса$_2$,
$EBF_3 = 1000$ для списков указателей доступа,
$EBF_4 = 20$ для блоков данных в (случае инвертированной и мультисписковой организации)
$= 21$ (в случае двусвязанного дерева),
$NTB_{ср} = 48$ для двусвязанного дерева.

Таблица 15.2. **Оценка ввода-вывода (количества обращений к физическим блокам) операций обновления на примере запроса 15-3 с NTR = 1**

	Мультисписковая организация	Инвертированная организация	Секционная инвертированная организация	Двусвязанное дерево
ИЗМЕНИТЬ неключевое значение	1 sba	1 sba	1 sba	1 sba
УДАЛИТЬ запись данных	1,2 sba	2,2 sba	2,2 sba	1,3 sba
ВКЛЮЧИТЬ запись данных	(0,2 sba + 1 rba)	(2,2 sba + 1 rba)	(2,2 sba + 1 rba)	(1,2 sba + 0,4 rba)
ИЗМЕНИТЬ вторичный ключ	(2,4 sba + 2 rba)	(3,2 sba + 4,2 rba)	(3,2 sba + 4,2 rba)	(1,5 sba + 4,4 rba)

Таблица 15.3. **Оценка ввода-вывода (количества обращений к физическим блокам) операций обновления на примере запроса 15-3 с NTR = 100**

	Мультисписковая организация	Инвертированная организация	Секционная инвертированная организация	Двусвязанное дерево
ИЗМЕНИТЬ неключевое значение	100 sba	100 sba	100 sba	10 sba
УДАЛИТЬ запись данных	1,2 sba	20,2 sba	2,2 sba	0,3 sba
ВКЛЮЧИТЬ запись данных	(0,2 sba + 100 rba)	(20,2 sba + 100 rba)	(2,2 sba + 20 rba)	(90,3 sba + 10,3 rba)
ИЗМЕНИТЬ вторичный ключ	(101,4 sba + 2 rba)	(138,2 sba + 4,2 rba)	(102,2 sba + 2,2 rba)	(85,6 sba + 61,3 rba)

Ниже выводится оценка количества обращений к физическим блокам для операций ИЗМЕНИТЬ вторичный ключ:

Мультисписковая организация

РВА(ИЗМЕНИТЬ втор. ключ) = РВА(УДАЛИТЬ) + РВА (найти элемент индекса$_2$ для нового значения ключа) + РВА(повторно ВКЛЮЧИТЬ) = 1,2 sba + 2 rba + 1,2 sba = 2,4 sba + + 2 rba.

Инвертированная организация

РВА(ИЗМЕНИТЬ втор. ключ) = РВА(перезаписать запись данных) + РВА (найти список указ. дост. для нового значения ключа) + РВА(прочитать и перезаписать список указателей доступа) + РВА(прочитать и перезаписать индекс$_2$) × (PRDEL + + PRINS) = 1 sba + 2 rba + 2 × (1 rba + 1 sba) + (1 rba + 1 sba) × × 0,2 = 3,2 sba + 4,2 rba.

Секционная инвертированная организация

РВА(ИЗМЕНИТЬ втор. ключ) = 1 sba + 2 rba + 2 × (1 rba + + 1 sba) + (1 rba + 1 sba) × 0,2 = 3,2 sba + 4,2 rba.

Двусвязанное дерево

РВА(ИЗМЕНИТЬ втор. ключ) = РВА(УДАЛИТЬ) + РВА (найти место для включения измененного блока данных) + РВА (ВКЛЮЧИТЬ — перезаписать блок данных) = 1,3 sba + 4 rba + + (1,2 sba + 0,4 rba — 1 sba) = 1,5 sba + 4,4 rba. □

Из табл. 15.2 и 15.3 видно, что ни один из рассмотренных методов доступа не превосходит остальные по производительности всех типов операций обновления. Однако в случае преобладания определенных типов обновления можно подобрать наиболее эффективный метод доступа. Например, если ожидается интенсивное включение новых записей данных, секционная инвертированная структура и ее механизм поиска оказываются значительно более производительными в сравнении со всеми другими методами доступа. Надо заметить, что разница в производительности становится более ощутимой для NTR равного 100 записям, чем 1 записи. Это является следствием организации индекса и данных в блоки, что неэффективно для небольших совокупностей записей-целей.

15.5. Способы организации инвертированного индекса

Для каждого из рассмотренных выше вторичных методов доступа характерна уникальная структура организации хранения данных. Однако, по существу, в мультисписковом и инвертированном файлах используется одна и та же структура индексирования. В дальнейшем при сравнении различных конфигураций инвертированного индекса будем использовать эту структуру в качестве прототипа. До сих пор предполагалось, что с целью ускорения выборки данных фрагмент индекса, со-

ответствующий значениям элемента данных определенного типа (индекс$_2$), плотный и упорядоченный. Предполагалось также, что в целях выявления пересечений множеств записей-целей при обработке конъюнктивных запросов список указателей доступа, соответствующий каждому элементу индекса$_2$, также плотный и упорядоченный. Условимся называть варианты индекса, получающиеся в результате отступлений от этих правил, и связанные с ними изменения в процедурах обновления способами организации индекса. В работе [7] в связи с проблемой выбора индекса приведен и проанализирован целый список различных способов организации индекса; здесь мы расширим использованные понятия, адаптируя их к введенной нами модели метода доступа.

При изложении материала ниже подразумевается, что обновленне индекса выполняется сразу же, так что для любого запроса состояние базы данных корректно. Кроме того, из рассмотрения исключена возможность взаимных блокировок процессов, работающих с базой данных; предполагается однопрограммиая операционная среда; однако при оценке времеии ответа в мультипрограммной операционной среде это обстоятельство необходимо учитывать [см. параметр QDELAY в гл. 9]. Для этого требуется использовать динамическую модель базы данных с очередями [186, 189].

Давайте классифицируем возможные способы организации индекса и определим для них основные характеристики производительности. Варианты 1—9 относятся только к индексу; спнски указателей доступа по-прежнему считаются плотными н упорядоченными.

1. Упорядоченная последовательная структура индекса

Указанная структура представляет стандартный прототип организации инвертированного файла. В ней и фрагменты индекса, соответствующие значениям элементов данных (ннднекс$_2$), и списки указателей доступа организованы как плотные упорядоченные физически последовательные файлы, что обеспечивает быстрое выполнение операций выборки. Включению нового элемента в индекс предшествует последовательный поиск соответствующей позиции в упорядоченном списке и сдвиг остальных элементов списка с целью сохранения его плотности и упорядоченности. Для удаления элемента из индекса также требуется выполнить последовательный поиск и сдвиг элементов индекса. Производительность выборки элементов из индекса можно улучшить посредством применения метода бинарного поиска или механизма поиска для многоуровневого индекса. Основные характеристики производительностн обновления пред-

ставляют:

$$\text{РВА(ВКЛЮЧИТЬ запись данных)} = \text{РВА}_1 + \text{РВА}_2 + \\ + \text{PRINS} \times \text{РВА}_3, \quad (15\text{-}48)$$

где $\text{РВА}_1 = \text{РВА}$(записать блок данных), $\text{РВА}_2 = \text{РВА}$(прочитать и переписать список указателей доступа) и $\text{РВА}_3 = \text{РВА}$ (прочитать и перезаписать индекс$_2$).

$$\text{РВА(УДАЛИТЬ запись данных)} = \text{РВА}_2 + \text{PRDEL} \times \text{РВА}_3, \quad (15\text{-}49)$$

$$\text{РВА(ИЗМЕНИТЬ ключ)} = \text{РВА(УДАЛИТЬ запись данных)} + \\ + \text{РВА(найти новый список указателей доступа)} + \\ + \text{РВА(ВКЛЮЧИТЬ запись данных, при этом вместо «записать» необходимо «перезаписать блок данных»).} \quad (15\text{-}50)$$

При оценке предполагается, что операция чтения или записи произвольного (или связанного) блока требует 1 rba, операция перезаписи блока — 1 sba, а совокупность операций чтения или записи физически последовательных блоков составляет 1 sba для каждого блока.

2. Упорядоченная неплотная структура индекса

Данная структура отличается от упорядоченной последовательной структуры тем, что в ней блоки индекса могут быть неплотными. Это достигается путем задания коэффициента загрузки < 1, благодаря чему в блоках предусматривается некоторый запас свободного пространства с целью последующего расширения индекса без перемещения остальных его элементов. Вследствие того что при удалении записи выполняется специальная ее пометка или освобождается занимаемое ею пространство памяти, выполнение этой операции упрощается. Все варианты операции выборки применительно к блокам индекса можно реализовать с использованием первичных методов доступа; операции обновления могут быть выполнены также более эффективно. Ниже показаны различия в производительности рассмотренной структуры индекса:

ВКЛЮЧИТЬ запись данных: применимо уравнение (15-48), за тем исключением, что

$$\text{РВА}_3 = P_1 \times \text{РВА(перезаписать один блок индекса}_2) + \\ + (1 - P_1) \times \text{РВА(сдвиг элементов индекса}_2 \text{ с целью освобождения места для нового элемента),} \quad (15\text{-}51)$$

где P_1 — вероятность наличия в блоке свободного места, достаточного для нового элемента.

УДАЛИТЬ запись данных: применимо уравнение (15-49), за тем исключением, что

$$\text{РВА}_3 = \text{РВА (перезаписать один блок индекса}_2\text{)}. \quad (15\text{-}52)$$

В силу того что операция ИЗМЕНИТЬ ключ определяется на более высоком уровне абстракции (т. е. она определяется через операции ВКЛЮЧИТЬ И УДАЛИТЬ), выражение оценки ее производительности остается неизменным для всех способов организации индекса. Следовательно, достаточно сослаться на уравнение (15-50).

3. Упорядоченная структура индекса с делением блоков

Данный способ представляет модификацию предыдущего способа в том аспекте, что в случае отсутствия в блоке индекса свободного места для нового элемента выделяется новый блок (как в методе ISAM) или выполняется деление блока (как в методе VSAM). В случае деления блока перемещать элементы индекса не требуется. Операция удаления выполняется путем специальной пометки элемента или освобождения занимаемого им места и также не требует перемещения данных. Если при выполнении операции УДАЛИТЬ запись данных блок оказывается свободным, то в случае метода деления блоков его, возможно, потребуется удалить.

ВКЛЮЧИТЬ запись данных: применимо выражение (15-48), за тем исключением, что

$$\text{РВА}_3 = P_1 \times \text{РВА (перезаписать один блок индекса}_2\text{)} + \\ + (1 - P_1) \times \text{РВА (разделить блок)}, \quad (15\text{-}53)$$

где

$$\text{РВА}_4\text{ (разделить блок)} = \text{РВА (перезаписать старый блок} + \\ + \text{записать новый блок)}. \quad (15\text{-}54)$$

УДАЛИТЬ запись данных: применимо выражение (15-49), за тем исключением, что

$$\text{РВА}_3 = P_2 \times \text{РВА (перезаписать один блок индекса}_2\text{)} + \\ + (1 - P_2) \times \text{РВА (перезаписать один блок индекса}_2 + \\ + \text{перезаписать предыдущий блок после того, как в нем} \\ \text{изменено значение указателя)}, \quad (15\text{-}55)$$

где P_2 обозначает вероятность полного освобождения блока индекса в результате операции УДАЛИТЬ.

4. Упорядоченная связанная структура индекса

В случае когда индекс организован в виде упорядоченного связанного последовательного файла, его обновление выполняется путем последовательного поиска соответствующей позиции

и модификации значения указателя СЛЕДУЮЩАЯ (и, возможно, ПРЕДЫДУЩАЯ). Все операции выборки выполняются с помощью механизма последовательного поиска. Для случая однонаправленной связанной последовательной организации:

ВКЛЮЧИТЬ запись данных: применимо выражение (15-48), за тем исключением, что

$$\text{РВА}_3 = \text{РВА}(\text{модифицировать значение указателя СЛЕДУЮЩАЯ в новом элементе и в предшествующем ему элементе и перезаписать оба элемента}). \quad (15\text{-}56)$$

УДАЛИТЬ запись данных: применимо выражение (15-49), за тем исключением, что

$$\text{РВА}_3 = \text{РВА}(\text{модифицировать значение указателя СЛЕДУЮЩАЯ в предыдущем элементе и перезаписать предыдущий блок}). \quad (15\text{-}57)$$

5. Неупорядоченная последовательная структура индекса

В случае неупорядоченной плотной структуры новые элементы включаются в конец индекса за последним блоком. Обработка переполнения осуществляется путем назначения нового последнего блока. В целях сохранения плотности при удалении элемента требуется сдвиг оставшихся элементов индекса. Операции выборки выполняются всегда последовательно.

ВКЛЮЧИТЬ запись данных: применимо выражение (15-48), за тем исключением, что

$$\text{РВА}_3 = P_1 \times \text{РВА}(\text{перезаписать последний блок индекса}_2) + (1 - P_1) \times \text{РВА}(\text{записать новый блок индекса}_2). \quad (15\text{-}58)$$

УДАЛИТЬ запись данных: применимо выражение (15-49), за тем исключением, что

$$\text{РВА}_3 = \text{РВА}(\text{прочитать и перезаписать индекс}_2). \quad (15\text{-}59)$$

6. Неупорядоченная неплотная структура индекса

Данная структура подобна упорядоченной неплотной структуре, за тем исключением, что предшествующая включению элемента операция выборки осуществляет поиск первого блока с достаточным свободным пространством; все прочие операции выборки выполняются последовательно.

7. Неупорядоченная структура индекса с делением блоков

Данная структура подобна упорядоченной структуре индекса с делением блоков, за исключением того, что предшествующая включению элемента операция выборки осуществляет поиск пер-

вого подходящего блока; все прочие операции выборки выполняются последовательно.

8. Неупорядоченная связанная структура индекса

Данная структура подобна упорядоченной связанной структуре индекса, за исключением того, что предшествующая включению элемента операция выборки относится к заглавному указателю индекса.

9. Неупорядоченная прямая структура индекса

Данная структура подобна последовательной неупорядоченной структуре индекса, за исключением того, что предшествующая включению элемента операция выборки реализуется в виде одного обращения к физическому блоку относительно последнего блока фрагмента индекса$_2$. С этой целью в структуре предусмотрен специальный указатель.

Для списков указателей доступа также возможны варианты, подобные вариантам организации индекса 1—9; для всех этих случаев необходимо вывести новые выражения оценки производительности, аналогичные вышеприведенным, но содержащие $РВА_2$ вместо $РВА_3$.

Глава 16. Выбор вторичного индекса

В гл. 15 рассмотрены традиционные вторичные методы доступа, обычно реализуемые в СУБД и других программных системах. Мультисписковый и инвертированный файлы служат примерами одноключевого метода индексирования (метода отдельных индексов); в нем фрагмент индекса строится на основе значений одного ключевого типа элемента данных. Двусвязанное дерево представляет пример гибридной схемы индексирования; в ней каждый уровень соответствует одному типу ключевого элемента, но проход через несколько уровней можно рассматривать как поиск в многоключевом (составном) индексе. В двусвязанном дереве все обращения к индексу (к элементам внутри уровня индекса и к уровням индекса) концептуально выглядят как связанные последовательные обращения, хотя на практике они часто реализуются в виде обращений к одному и тому же физическому блоку.

Теперь мы приступим к исследованию нескольких других возможных методов многоключевого индексирования и рассмотрим проблему выбора ключей, использование которых в качестве индексов обеспечивает наилучшую производительность системы. Выбор индексов представляет важный этап процесса проектирования базы данных и включает выбор первичных ключей, вторичных ключей для использования в качестве отдельных индексов и вторичных ключей для использования в качестве составных индексов. Это в значительной степени неформальный процесс, требующий от проектировщика интуитивной догадки и оценки многочисленных вариантов. Для автоматизации этого процесса были разработаны некоторые вспомогательные средства проектирования, но в настоящее время эта работа находится в стадии эксперимента; на практике имеет место автоматизированный анализ при непосредственном участии проектировщика или же просто анализ проектировщика без применения средств автоматизации. В данной главе для сравнительной оценки многочисленных вариантов вторичных методов доступа и возможных выборов ключей по-прежнему используется построенная ранее аналитическая модель базы данных.

16.1. Многоключевое (составное) индексирование

Инвертированный файл, являющийся примером метода одноключевого индексирования, обеспечивает высокую производительность операций выборки при обработке простых запросов,

включающих значения одного ключа, и приемлемую производительность при обработке конъюнктивных запросов, включающих несколько условий элементов. В последнем случае для получения записей-целей, удовлетворяющих всему условию запроса, приходится выполнять дополнительную работу по объединению списков указателей доступа; тем не менее, как было показано на примере запроса 15-2, эта процедура оказывается намного быстрее в сравнении с поиском записей данных в последовательном или мультисписковом файле. В то же время для указанного примера еще более производительной оказалась гибридная схема составного индексирования (двусвязанное дерево). В случае коротких списков указателей доступа инвертированный файл обеспечил бы более высокую производительность в сравнении с двусвязанным деревом, и вообще эта структура обладает большей степенью гибкости при обработке незапланированных запросов любой сложности. Тем не менее для многих случаев прикладных запросов целесообразнее использовать некоторую форму составного индексирования.

Для более наглядной демонстрации достоинств составного индексирования расширим пример запроса 15-2. Предположим, что указанный запрос является представителем целого класса запросов одного и того же формата, но с разными значениями ключевых элементов в условиях элементов, что, как известно, на практике встречается довольно часто. В этом случае благодаря статичности запросов можно было бы повысить эффективность операции выборки за счет создания специального индекса, упорядоченного по трем заданным типам элементов данных (ГОРОД, ДОЛЖНОСТЬ, КАТЕГОРИЯ), интерпретации комбинации ключей как одного составного ключа и применения для поиска требуемого элемента индекса соответствующего первичного метода доступа. Благодаря такому подходу поиск в индексе для всего запроса уменьшился бы до одного или самое большее до нескольких обращений к блокам. При обработке запросов с несколькими условиями записей для каждого условия потребовалась бы выборка одного элемента индекса; тем не менее производительность такой обработки оказалась бы значительно выше в сравнении с инвертированным файлом.

Если бы вместо списка указателей доступа для условия записи использовался участок удовлетворяющих ему записей-целей, как в двусвязанном дереве и в алгоритмах хеширования, можно было бы еще уменьшить общее время операции выборки. На рис. 16.1 приведен пример файла с составным индексом.

Запрос из примера 15-2 можно было бы обработать ценой трех обращений к блокам (3 rba) при индексно-последовательном поиске составного индекса и последующих $500/20 = 25$ обращений к блокам для 500 записей-целей (в лучшем случае

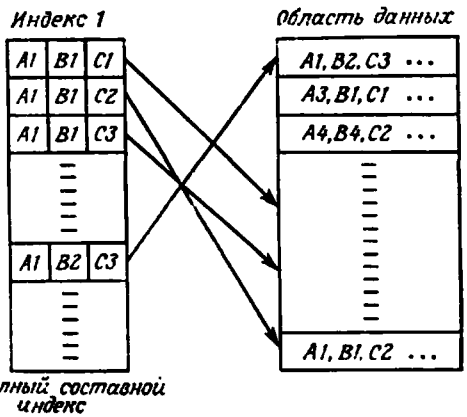


Рис. 16.1. Пример файла с составным индексом.

1 rba + 24 sba). Общее время операции выборки составило бы (4 rba) × 36,7 + (24 sba) × 11,7 = 427,6 мс. Эта оценка в 44 раза лучше в сравнении с секционным инвертированным файлом и сопоставима с производительностью двусвязанного дерева при использовании блокированного индекса.

Как и в случае двусвязанного дерева, значительное повышение производительности операции выборки имеет свою цену. Это двойная цена: во-первых, составному индексу присуща большая степень избыточности; во-вторых, структура оказывается крайне чувствительна к любому изменению содержимого базы данных или используемого типа запроса; такие изменения вызывают реорганизацию индекса. Указанная избыточность подобна избыточности первой нормальной формы реляционной базы данных (плоского файла, представляющего составной индекс) в сравнении со второй и третьей нормальными формами, которые подобны совокупности одноключевых индексов. В составном индексе для запроса из примера 15-2 необходимо хранить $\prod_{i=1}^{3} \mathrm{NIV}_i = 100 \times 5 \times 13 = 6500$ элементов. Если на каждое значение отводится 6 байт и 4 байт на указатель (10 байт на каждый элемент), общий объем памяти составит 65 К. Для отдельных одноключевых индексов потребовалось бы $\sum_{i=1}^{3} \mathrm{NIV}_i = 100 + 5 + 13 = 118$ элементов, или 1,18 К. (Надо заметить, что аналогия с реляционными нормальными формами не совсем точна. Вторая и третья нормальные формы все же обладают некоторой избыточностью.) Хотя индекс размером 65К, как в данном примере, допустим в средних и больших вычислительных системах, но очень велика разница в требуемых объемах

памяти, а для некоторых классов запросов составной индекс на практике просто невозможно было бы реализовать.

Для иллюстрации проблемы чувствительности к изменениям расширим требования к обработке запросами с третьим ключевым компонентом КАТЕГОРИЯ. Программные средства обслуживания инвертированного файла в этом случае обычно выполняют обращение к индексу значений элементов типа КАТЕГОРИЯ (индекс$_2$) и находят соответствующий список указателей доступа. В силу того что файл составного индекса не упорядочен по значениям элементов типа КАТЕГОРИЯ и записи данных хранятся в произвольном порядке, необходимо выполнить его последовательный просмотр. Например, если условию элемента КАТЕГОРИЯ-10 соответствует 10^5 записей-целей, произвольно распределенных в 50 000 блоках, то оценки количества обращений к блокам для одноключевого секционного инвертированного метода доступа и соответственно для метода доступа с составным индексом таковы:

Секционный инвертированный файл

РВА (ПОЛУЧИТЬ индекс$_1$) = 1 rba, РВА (ПОЛУЧИТЬ индекс$_2$) = 1 rba,

РВА (ПОЛУЧИТЬ список указателей доступа) = 1rba + $+\left(\frac{5000}{10^5}-1\right)$ sba = 1 rba + 4 sba,

РВА (ПОЛУЧИТЬ записи данных) = 5000 rba,

РВА (ПОЛУЧИТЬ НЕКОТОРЫЕ, всего) = 5003 rba + + 4 sba (TIO = 184 с).

Файл с составным индексом

РВА (ПОЛУЧИТЬ ВСЕ, последовательный просмотр) = = 50 000 sba (TIO = 585 с).

По причине большого размера списка записей-целей инвертированный файл оказался бы крайне неэффективен (TIO = = 3670 с). Применение составного индекса также неэффективно, но эта неэффективность объясняется необходимостью просмотра всего файла с произвольно расположенными в нем записями-целями. Благодаря преимуществу организации блоков на уровне записей данных секционный инвертированный файл оказывается наиболее приспособленным к большим спискам записей-целей. Очевидно, что преимущества составного индекса наиболее ярко проявляются в случае заранее определенных часто используемых статичных запросов. В противном случае в добавление к постоянным значительным служебным издержкам памяти наблюдается значительный спад производительности операции выборки.

В целом все условия записей можно разбить на три категории по степени соответствия сложности запроса используемому индексированию [7]. Условие записи называется *последо-*

вательным конъюнктом, если в него входят исключительно неиндексируемые элементы данных (т. е. неключевые элементы данных). В этом случае необходимо выполнить последовательный просмотр записей, и каждую из них проверить на ошибочный результат поиска. Условие записи называется *чистым конъюнктом*, если в него входят только индексируемые элементы данных, поэтому для определения местоположения записей-целей запроса достаточно выполнить поиск в индексе (а также при необходимости объединение списков указателей доступа). Условие записи называется *смешанным конъюнктом*, если оно содержит и индексируемые, и неиндексируемые элементы данных. В этом случае обычная стратегия поиска записей-целей состоит в определении списка записей, удовлетворяющих условиям индексируемых элементов и последующей их проверке на удовлетворение условиям неиндексируемых элементов. Очевидно, что наиболее эффективно обслуживается чистый конъюнкт, поэтому при проектировании схем индексирования наибольшее внимание уделяется заранее известным прикладным запросам.

16.2. Классификация вторичных методов доступа

Выше были наглядно продемонстрированы различия между вторичными методами доступа. В целом указанные различия можно представить в виде следующих четырех групп:

Группа 1. Структура индекса. Основными параметрами являются:
- NIF, количество фрагментов ключевых значений (в индексе$_2$),
- NKEY, количество ключей, представленных в одном элементе фрагмента (ключевых значений) в индексе$_2$.

Группа 2. Способы поиска в индексе.

Группа 3. Способы поддержания индекса.

Группа 4. Методы доступа к записям данных:
- Список указателей доступа для отдельных экземпляров записей данных.
- Список указателей доступа только для блоков данных.
- Кластеризация (и блокирование) записей данных при отсутствии списков указателей доступа.
- Некластеризованная связанная последовательная организация записей данных.

Каждая группа параметров считается независимой, то есть для определения полной конфигурации вторичного метода доступа можно рассматривать любую комбинацию параметров из разных групп. Условимся называть параметры групп 1—3 определением *конфигурации индекса*.

Вначале проанализируем параметры структуры индекса (группа 1). На рис. 16.2 представлена классификация вторичных методов доступа в двумерном пространстве, определеином параметрами NIF и NKEY. Рассмотрим подробно каждую представленную категорию.

Группа 1. Структура индекса

Полностью инвертированный файл (FIV): NKEY = 1, NIF = = NIT. Во многих современных СУБД предусмотрены средства управления полностью инвертированными файлами; первоиа-

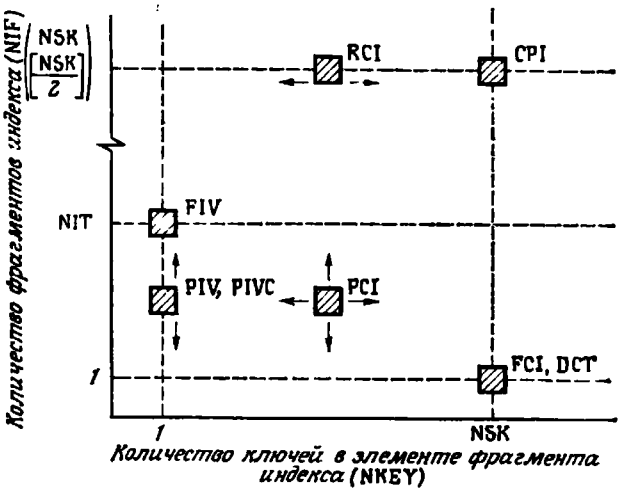


Рис. 16.2. Классификация вторичных методов доступа на основе структуры индекса.

чально они были введены как обязательный компонент системы управления базами данных TDMS [35]. Наиболее распространена реализация полной инверсии в виде списка указателей доступа, значения которых адресуют конкретные записи-цели; в секционном инвертированном файле элементы списка указателей доступа адресуют блоки данных, содержащих записи-цели. По аналогии с индексно-последовательным методом доступа для организации поиска в индексе часто используются индексы блока или дорожки. Конкретные правила управления индексом зависят от используемой системы.

Частично инвертированный файл (PIV): NKEY = 1, NIF < < NIT. Частично инвертированный файл представляет распространенный метод организации данных в коммерческих СУБД [58, 299]; в отличие от полностью инвертированного файла здесь нет необходимости инвертировать (индексировать) данные по каждому типу элемента данных. В секционном частично инвер-

тированном файле предусматриваются списки указателей доступа для блоков данных, содержащих записи-цели. Отличительную черту мультисписковой структуры представляет метод доступа к записям.

Полный составной индекс (FCI) *и двусвязанное дерево* (DCT): NKEY = NSK, NIF = 1. Концептуально обе указанные структуры эквивалентны; в обеих предусмотрен один большой индекс. Однако реализация их различна. Хранение полного составного индекса организовано в виде физически последовательного файла, в то время как при хранении и поиске в многоуровневом индексе двусязанного дерева используется структура последовательного связанного файла. В обоих случаях наилучшая производительность наблюдается, когда представленные в индексе ключевые типы элементов эквивалентны ключам поиска в запросах. В то же время служебные издержки памяти в случае двусвязанного дерева меньше за счет использования иерархической структуры индекса. В одной из экспериментальных реализаций схемы полного составного индекса, известной под названием сбалансированной схемы организации файлов [134], определяются бакеты, содержащие списки указателей доступа, а в качестве списков указателей доступа используются указатели отдельных физически последовательных записей.

Частичный составной индекс (PCI): $1 \leqslant \text{NKEY} \leqslant \text{NSK}$, $1 \leqslant \text{NIF} \leqslant \text{NIT}$. Эта структура индекса представляет общую основу построения полностью и частично инвертированных файлов, а также полного составного индекса [201]. Вторичный метод доступа с несколькими многоключевыми индексами занимает промежуточное место между полным составным индексом, которому свойственны быстрая операция выборки, но большой объем памяти и отсутствие должной гибкости, и полностью и частично инвертированными файлами, которым присуща более медленная операция выборки, но меньший объем памяти и бо́льшая гибкость. Примером реализации схемы частичного составного индекса служит организация вторичного индекса в системе IMS.

Составное индексирование (CPI): NKEY = NSK, $\text{NIF} = \binom{\text{NSK}}{\lceil \text{NSK}/2 \rceil}$. Предложенная Ламом [201] схема составного индексирования расширяет схему полного составного индекса, делая ее еще более избыточной с целью ускорения операции выборки. Каждый из используемых в ней нескольких индексов представляет собой полный составной индекс, но отличается от других порядком ключей. Хотя теоретически возможно всего NSK! способов упорядочения ключей, в работе показано, что достаточно рассмотреть

$$\text{NIF} = \binom{\text{NSK}}{\lceil \text{NSK}/2 \rceil} \qquad (16\text{-}1)$$

способов; это обеспечит ответ на все возможные коньюнктивные запросы из NSK или менее ключей, так что отпадает необходимость в последующей сортировке для выявления пересечения списков указателей доступа записей-целей. Например, если в запросе примера 15-2 заменить названия типов элементов на A, B и C, то все возможные конъюнктивные запросы из 1, 2 и 3-го условий элемента можно представить в виде следующих 15 случаев [241, 286]: A, B, C, $A \wedge B$, $B \wedge A$, $A \wedge C$, $C \wedge A$, $B \wedge C$, $C \wedge B$, $A \wedge B \wedge C$, $A \wedge C \wedge B$, $B \wedge A \wedge C$, $B \wedge C \wedge A$, $C \wedge A \wedge B$, $C \wedge B \wedge A$.

Однако Лам предложил структуру из трех фрагментов:

Ключи индекса	Возможные запросы
1. ABC	A, $A \wedge B$, $B \wedge A$, $A \wedge B \wedge C$, $A \wedge C \wedge B$
2. BCA	B, $B \wedge C$, $C \wedge B$, $B \wedge C \wedge A$, $B \wedge A \wedge C$
3. CAB	C, $C \wedge A$, $A \wedge C$, $C \wedge A \wedge B$, $C \wedge B \wedge A$

Эта структура обеспечивает ответ на любой из 15 типов запросов без необходимости выявления пересечения списков указателей доступа или просмотра всего индекса.

В соответствии с уравнением (16-1) получаем следующие значения количества фрагментов индекса как функции количества вторичных ключей:

NSK	2	3	4	5	10
NSK!	2	6	24	120	3 628 800
NIF	2	3	6	10	252

Редуцированный составной индекс (RCI): $\mathrm{NKEY} \leqslant \mathrm{NSK}$, $\mathrm{NIF} = \binom{\mathrm{NSK}}{\lfloor \mathrm{NSK}/2 \rfloor}$. Данная схема представляет вариант схемы составного индексирования, в котором избыточность отдельных индексов уменьшена [286], хотя количество индексов осталось прежним. Например, размер предложенных Ламом трех индексов для запроса из примера 15-2 с $\mathrm{NSK} = 3$ можно уменьшить так, как показано в табл. 16.1, где редуцированные составные индексы представляют ABC, BC и CA.

Таблица 16.1. **Количество элементов в составном и редуцированном составном индексах для запроса из примера 15-2 с NSK = 3**

Индекс	Составной индекс	Редуцированный составной индекс
1. A B C	$100 \times 5 \times 13 = 6\,500$	$100 \times 5 \times 13 = 6500$
2. B C | A	6 500	$5 \times 13 = 65$
3. C A | B	6 500	$13 \times 100 = 1300$
Общее количество элементов	19 500	7865

На рис. 16.3 приведены примеры составного и редуцированного составного индексов. Полные индексы описывают схему составного индексирования, а получаемые из них в результате исключения элементов справа от вертикальной линии частичные индексы представляют схему редуцированного составного индексирования. Укороченные индексы означают, что схема редуцированного составного индексирования способна уменьшить не

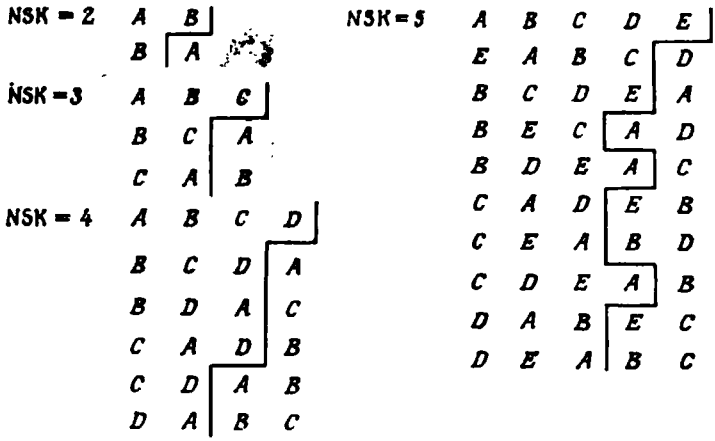


Рис. 16.3. Составной индекс и редуцированный составной индекс для ключей *A, B, C, D* [286].

только объем памяти, но и время поиска записей-целей в индексе. В работах [201, 286] рассмотрены также варианты указанной схемы, позволяющие при специальных условиях еще больше уменьшить время выборки.

Хотя в действительности алгоритмы хеширования не являются схемами индексирования, но концептуально они подобны способу организации частичного составного индекса, при котором значение составного ключа преобразуется в физический адрес бакета записей данных, каждая из которых содержит полное ключевое значение [4, 49]. Для сравнительной оценки применения методов индексирования и хеширования к организации составного индекса необходимо дополнительное исследование.

Группа 2. Способы поиска в индексе

В качестве механизма поиска в индексе можно использовать любой из первичных методов доступа, рассмотренных в гл. 12—14; это последовательный поиск, бинарный поиск, (многоуровневая) структура индексов блоков, хеширование и поиск в дереве. Выбор способа зависит от представления индекса в виде физически последовательного или связанного последовательного файлов.

Группа 3. Способы поддержания индекса

Способы поддержания индекса рассмотрены в разд. 15.5. В зависимости от представления индекса в виде физически последовательного или связанного последовательного файла возможны варианты.

Группа 4. Методы доступа к записям данных.

Для любой из рассмотренных в группах 1—3 возможных конфигураций индекса допускается применение своего метода доступа к записям данных. Основную структуру инвертированного файла, рассмотренного в гл. 15, представляет список указателей доступа, адресующий записи данных. Основную структуру секционного инвертированного файла представляет список указателей доступа, адресующий блоки данных. Основной формой организации двусвязанного дерева служат кластеры записей данных, связаных несколькими ключевыми значениями. Данная форма наиболее эффективна для статичных файлов и статичных запросов, но становится неэффективной в случае динамического изменения данных и (или) запросов. Для мультисписконого подхода характерны некластеризованные записи данных. В каждом из указанных методов применяется простой механизм последовательного поиска.

Перечисленные выше группы и варианты вторичных методов доступа представляют важный аспект в решении проблемы выбора индекса. Рассмотрим теперь эту проблему подробнее.

16.3. Проблема выбора индекса

Проблема выбора индекса представляет одну из наиболее сложных проблем проектирования базы данных. Мы определяем ее как проблему выбора первичных ключей, вторичных ключей и конфигурации индекса, позволяющих обеспечить достаточно высокую производительность системы баз данных для всех типов требуемых прикладных функций. *Проблема выбора вторичного индекса* более локальна; она состоит в выборе множества вторичных ключей и конфигурации индекса, обеспечивающих высокую производительность системы баз данных. Мы отличаем указанные проблемы от *проблемы выбора оптимального вторичного индекса*, при котором выбирается конфигурация вторичных ключей и индекса, оптимизирующая производительность системы баз данных. Для определения конфигурации индекса служат характеристики групп 1—3 разд. 16.2. Для оценки производительности используются различные рассмотренные выше критерии или их подмножество: время ввода-вывода для операций выборки и обновления, время служебного использования CPU, объем памяти.

На практике отличие проблемы выбора индекса от проблемы выбора оптимального индекса состоит в том, что хотя существуют отдельные эвристические методы выбора индекса, но отсутствуют средства проектирования, обеспечивающие оптимальное решение проблемы в целом. Известно несколько подходов к получению оптимального решения в отдельных случаях одноключевых (отдельных) индексов и многоключевых (составных) индексов, но не для обоих типов индексирования одновременно. В разд. 16.4 рассмотрен один из подходов. Для выбора оптимального индекса было разработано автоматизированное средство [269], но оно ограничено только конфигурацией одноключевого индекса. В работе [87] показано, что даже для проблемы выбора одноключевого индекса сложность любого алгоритма для худшего случая не меньше 2^{NIT}; это потенциально сложная (то есть NP-полная) проблема. Хотя во многих случаях проблема может быть решена за несколько шагов, в динамической среде, когда компоненты индекса часто меняются, может оказаться нецелесообразным использовать подобные алгоритмы. Автор работы [87] предложил исследовать возможность использования для выбора оптимального индекса аппроксимирующих алгоритмов как средства выявления решений, близких к оптимальным, ценой небольших вычислительных затрат.

Независимо от того, ставится проблема выбора оптимального или просто «хорошего» индекса, в процессе ее решения можно выделить следующие основные этапы:

1. Выявить типы требуемых ключей и перечислить подходящие ключи-кандидаты.
2. Исключить неудовлетворительные варианты ключей (ограничить пространство решений).
3. Выбрать подмножество ключей-кандидатов, обеспечивающих требуемую производительность.

Рассмотрим каждый из этих этапов подробнее.

Этап 1: Перечисление ключей — кандидатов

Источником информации о возможных ключах служит логическая структура базы данных, предназначенная для удовлетворения требований пользователей к информации и ее обработке. В ней отражены связи данных, влияющие на выбор ключей. Для более точного определения типов элементов данных весьма полезны также словари и справочники данных, соответствующие логической структуре базы данных.

Выбор первичных ключей вызывается необходимостью определенной упорядоченности данных в целях эффективного выполнения последовательных (ПОЛУЧИТЬ ВСЕ) или произвольных (ПОЛУЧИТЬ УНИКАЛЬНУЮ) операций. Кроме того, упорядоченность по первичному ключу облегчает пакетную об-

работку запросов и обновлений. В простейшем случае первичный ключ представляет собой одноатрибутный ключ (тип элемента), имеющий уникальное значение для каждой записи данных. Примерами одноатрибутных идентификаторов служат учетный номер клиента, номер страхового полиса или номер накладной. Как правило, большая часть первичных ключей бывает определена еще на этапе концептуального проектирования во время выбора объектов и атрибутов и на данном этапе только проверяется.

Иногда атрибуты оказываются слишком длинными; поэтому для удобства обработки используются только частичные ключи. К сожалению, значения частичных ключей, как правило, повторяются; для обеспечения их уникальности проектировщику приходится использовать некоторые суффиксы. Следовательно, в качестве варианта первичного ключа возможно сцепление нескольких ключей. В качестве примера предположим, что при определении частичных ключевых значений были использованы пять первых букв фамилии. Наличие одинаковых или похожих фамилий (например, JOHNS, JOHNSON) вызвало бы появление повторяющихся значений частичного ключа. В качестве суффиксов полезно использовать значения других ключей или частичных ключей, оставшихся в записи, порядковые номера или значения относительных адресов. Интересный способ сцепления, предложенный в работе [154], состоит в том, чтобы добавить к ключу один или несколько битов, значения которых сигнализируют об изменении записи, частоте ее изменения и типе произведенного изменения.

Реализованное в системе IMS понятие сцепленного ключа полезно как средство описания иерархической связи данных, а также обеспечения уникальности ключевого значения. В некоторых случаях иерархически определенный сцеплеиный ключ может оказаться слишком длинным либо иметь переменную длину. В обеих ситуациях наблюдается необходимость дополнительного служебного использования CPU.

В качестве других способов определения первичных ключей можно указать следующие: способ создания сцепленных ключей с целью обеспечения требуемой упорядоченности данных для конкретного отчета; способ создания индексов, адресующих записи данных с нулевыми значениями определеиных ключей; способ размещения данных пересечения (в связях типа m:n) в ключах, а не в отдельных записях. Последний способ, как правило, требует дополнительного объема памяти вследствие возможного дублирования данных пересечения в ключах.

Этап 2: Исключение явно неудовлетворительных решений

При анализе кандидатов в ключи следует прежде всего

определить необходимые первичные ключи и исключить их из дальнейшего рассмотрения. Выбирая вторичные ключи, мы стремимся избежать первичных ключей или ключей со значениями, близкими к уникальным, из-за присущих им чрезмерно длинных списков значений элементов и коротких списков указателей доступа. Для первичного ключа список значений элементов данных имеет длину NR, а список указателей доступа — длину 1. По этой причине для таких ключей рекомендуется использовать первичные методы доступа. Еще один класс ключей, которые следует исключить из рассмотрения, представляют ключи с небольшим количеством значений, но большим количеством повторов, как, например, тип элемента ПОЛ. Для запросов, содержащих подобные ключевые значения, как правило, более эффективной оказывается последовательная обработка. Оставшиеся кандидаты во вторичные ключи необходимо исследовать на предмет их возможного использования при обработке множества прикладных запросов.

Этап 3: Окончательный выбор множества ключей и структура индекса

Цель выбора вторичных ключей состоит в обеспечении обработки запросов, о которых известно или можно предположить, что они будут использоваться довольно часто. С одной стороны, индексирование по каждому кандидату в ключи обеспечило бы обработку всех текущих и будущих простых запросов (т. е. запросов из одного условия элемента) без просмотра всей базы данных. С другой стороны, такая гарантия быстрого выполнения операции выборки требует дополнительных служебных издержек памяти и процедур поддержания индекса. Если анализ затрат показывает, что инверсия по всем ключам неразумна, необходим более продуманный подход. Конечно, можно сократить список кандидатов в ключи, оставив в нем только необходимые для обслуживания основных известных и ожидаемых запросов. Вследствие того что использование частичных вторичных ключей может вызвать дополнительные служебные издержки CPU, способ построения составных многоключевых индексов путем сцепления ключей, конечно, является более реальным, особенно в случае известных сложных запросов, включающих более одного условия элемента. Основой выбора кандидатов во вторичные ключи служат главным образом анализ требований к обработке и возможность удовлетворить эти требования с точки зрения содержимого данных. Основой окончательного выбора множества вторичных ключей и способов их организации в конфигурацию индекса служат определенные пользователем требования к производительности системы.

16.4. Выбор оптимального вторичного индекса

Как отмечалось ранее, выбор оптимального множества вторичных ключей с вычислительной точки зрения представляет сложную проблему. Однако следует заметнть, что для простых случаев можно вычислить оптимальное решение, а для многих случаев средней и повышенной сложности можно получить решение, близкое к оптимальному. Большинство исследовательских работ, проведенных в 70-е годы, были посвящены поиску приближенных методов решения проблемы выбора одноключевого (отдельного) и многоключевого (составного) индексов [7, 57, 143, 144, 153, 201, 203, 204, 269, 301].

Сейчас мы определим общий метод выбора вторичного индекса, в основе которого лежит использование модели последовательного и произвольного доступов, а также соответствующей ей аналитической формулировки понятия производительности вторичного метода доступа. Мы используем указанную основу и критерий выбора индекса, согласующийся с критериями теоретических моделей и алгоритмов, известных из литературы. Модель доступа позволяет нам определить любую структуру отдельного или составного индекса, показанных на рис. 16.2, включая метод поиска и поддержания индекса, учесть сложные условия запросов, рассмотренные в разд. 15.1, и построить оценочную функцию операций выборки, обновления и хранения. Оптимизация выполняется методом ручного перебора возможных решений. Для усовершенствования процедуры оптимизации можно создать вычислительную программу оценки стоимости варианта, производящую расчет по приведенным ниже уравнениям, и предусмотреть алгоритм исключения неудовлетворительных вариантов. Глобальная *проблема выбора оптимального метода индексирования* вторичных ключей и конфигурации индекса, минимизирующих общую стоимость варианта, по-прежнему остается темой для научно-исследовательской работы.

Для определения компонентов общей оценки варианта индексирования за единицу времени, выбранную в качестве стандартной меры, воспользуемся нисходящей методологией (сверху вниз) поэтапного приближения:

$$\text{TCOST (выбор индекса } x) = \sum_{\substack{\text{для всех } i \\ \text{операций ЯМД}}} (\text{CR}_i + \text{CU}_i) + \text{CS}, \quad (16\text{-}2)$$

где CR_i обозначает стоимость i-й операции выборки, CU_i — стоимость i-й операции обновления, а CS — стоимость памяти для хранения данных. В качестве еще одной условной функции, которую можно было бы использовать как меру эффективности, является функция, равная разности между TCOST (нет индексов) и TCOST (выбор индекса x); тогда задача сводится

к задаче поиска ее максимума [7]. Используя выражение (16-2), оценим физически последовательный метод доступа как эталон для последующего сравнения:

$$CR_i = CR_i(IO) + CR_i(CPU), \qquad (16\text{-}3)$$

$$CR_i(IO) = TIO_i(\text{ПОЛУЧИТЬ НЕКОТОРЫЕ}) \times C_{IO} \times FGS_i, \qquad (16\text{-}4)$$

$$CR_i(CPU) = TCPU_i(\text{ПОЛУЧИТЬ НЕКОТОРЫЕ}) \times \times C_{CPU} \times FGS_i, \qquad (16\text{-}5)$$

где TIO_i (ПОЛУЧИТЬ НЕКОТОРЫЕ) обозначает время ввода-вывода при обслуживании запроса i и представляет линейную функцию от PBA_i (ПОЛУЧИТЬ НЕКОТОРЫЕ), определенную уравнениями (9-5)—(9-7), C_{IO} обозначает стоимость единицы времени ввода-вывода, C_{CPU} — стоимость единицы процессорного времени, а FGS_i определяет частоту операций (запроса) i ПОЛУЧИТЬ НЕКОТОРЫЕ; $TCPU_i$ (ПОЛУЧИТЬ НЕКОТОРЫЕ) представляет общее время CPU, связанное с обработкой запроса i, и подробно определяется ниже:

$$TCPU_i(\text{ПОЛУЧИТЬ НЕКОТОРЫЕ}) = CPU.MERGE(c, a) + + CPU.TEST(q, a) + CPU.EXCH \times \times PBA_i(\text{ПОЛУЧИТЬ НЕКОТОРЫЕ}), \qquad (16\text{-}6)$$

$CPU.MERGE(c, a) = (c \times a) \times CPU.1$ — время CPU на объединение c упорядоченных списков из a элементов каждый, (16-7)

где CPU.1 обозначает время CPU на выполнение одной операции сравнения значений двух элементов данных,

$CPU.TEST(q, a) = a \times CPU.q$ — время CPU на выполнение проверки q условий элемента для каждого из a элементов (записей) списка, (16-8)

где $CPU.q = q \times CPU.1$. (16-9)

CPU.EXCH обозначает время CPU на выполнение программы канала ЭВМ для операции ввода-вывода одного физического блока, а PBA_i (ПОЛУЧИТЬ НЕКОТОРЫЕ) представляет количество обращений к физическим блокам, требуемое для обслуживания i-й операции ПОЛУЧИТЬ НЕКОТОРЫЕ. Этот же параметр PBA_i (ПОЛУЧИТЬ НЕКОТОРЫЕ) использован в расчете TIO_i (ПОЛУЧИТЬ НЕКОТОРЫЕ).

Аналогично вычисляется стоимость операций обновления за определенный период времени с учетом того факта, что воз-

можны четыре типа обновления: ВКЛЮЧИТЬ (*I*), УДАЛИТЬ (*D*), ИЗМЕНИТЬ ключ (CK) и ИЗМЕНИТЬ неключевой элемент (CNK).

$$CU_i = CU_i(IO) + CU_i(CPU), \quad (16\text{-}10)$$

$$CU_i(IO) = CI_i(IO) + CD_i(IO) + CCK_i(IO) + CCNK_i(IO), \quad (16\text{-}11)$$

$$CU_i(CPU) = CI_i(CPU) + CD_i(CPU) + CCK_i(CPU) + CCNK_i(CPU). \quad (16\text{-}12)$$

Стоимость ввода-вывода для отдельных операций:

$$CI_i(IO) = TIO_i(\text{ВКЛЮЧИТЬ, NTR.}I) \times C_{IO} \times FI_i, \quad (16\text{-}13)$$

$$CD_i(IO) = TIO_i(\text{УДАЛИТЬ, NTR.}D) \times C_{IO} \times FD_i, \quad (16\text{-}14)$$

$$CCK_i(IO) = TIO_i(\text{ИЗМЕНИТЬ ключ, NTR.CK}) \times C_{IO} \times FCK_i, \quad (16\text{-}15)$$

$$CCNK_i(IO) = TIO_i(\text{ИЗМЕНИТЬ неключевое значенне, NTR.CNK}) \times C_{IO} \times FCNK_i. \quad (16\text{-}16)$$

Стоимость работы CPU определяется аналогичнo:

$$CI_i(CPU) = TCPU_i(\text{ВКЛЮЧИТЬ, NTR.}I) \times C_{CPU} \times FI_i, \quad (16\text{-}17)$$

$$CD_i(CPU) = TCPU_i(\text{УДАЛИТЬ, NTR.}D) \times C_{CPU} \times FD_i, \quad (16\text{-}18)$$

$$CCK_i(CPU) = TCPU_i(\text{ИЗМЕНИТЬ ключ, NTR.CK}) \times \times C_{CPU} \times FCK_i, \quad (16\text{-}19)$$

$$CCNK_i(CPU) = TCPU_i(\text{ИЗМЕНИТЬ неключевое значение, NTR.CNK}) \times C_{CPU} \times FCNK_i. \quad (16\text{-}20)$$

В выражениях (16-13)—(16-20) частота выполнения операций обновления каждого типа обозначена буквой F, за которой указана аббревиатура типа используемой операции обновления. $TCPU_i$ (тип обновления) представляет общее время CPU, требуемое операции обновления дополнительно к времени CPU на выполнение предшествующей операции выборки обновляемой записи. Префикс NTR специфицирует среднее количество записей-целей для указанной операции. Упрощая обозначения констант, получаем:

$$TCPU_i(\text{ВКЛЮЧИТЬ, NTR.}I) = \text{CPU.}I \times \text{NTR.}I_i + \text{CPU.EXCH} \times \text{PBA(ПЕРЕЗАПИСАТЬ)}_i, \quad (16\text{-}21)$$

$$TCPU_i(\text{УДАЛИТЬ, NTR.}D) = \text{CPU.}D \times \text{NTR.}D_i + \text{CPU.EXCH} \times \text{PBA(ПЕРЕЗАПИСАТЬ)}_i. \quad (16\text{-}22)$$

$$TCPU_i(\text{ИЗМЕНИТЬ ключ, NTR.CK}) = \text{CPU.CK} \times \text{NTR.CK}_i + \text{CPU.EXCH} \times \text{PBA(ПЕРЕЗАПИСАТЬ)}_i, \quad (16\text{-}23)$$

$$TCPU_i(\text{ИЗМЕНИТЬ неключевое значение, NTR.CNK}) = \\ = \text{CPU.CNK} \times \text{NTR.CNK}_i + \text{CPU.EXCH} \times \\ \times \text{PBA(ПЕРЕЗАПИСАТЬ)}_i. \quad (16\text{-}24)$$

Префиксом CPU. обозначены оценки времени, зависящие от быстродействия используемого CPU и эффективности программного обеспечения. Эпизодически возникает необходимость дополнительного использования CPU:

$\text{CPU.SORT}(a) = a \log_2 a \times \text{CPU.1}$ — время CPU на сортировку списка из a элементов (для способов сортировки класса $\log_2 a$); (16-25)

$\text{CPU.BSEARCH}(b, a) = b \log_2 a \times \text{CPU.1}$ — время CPU на выполнение бинарного поиска в списке из a элементов; поиск выполняется для каждого из b элементов данных в списке, сравниваемом со списком из a элементов. (16-26)

В случае неупорядоченного списка (например, списка указателей доступа) выполнение сортировки и сравнения оценивается как сумма уравнений (16-25) и (16-26). При отсутствии предварительной сортировки время сравнения увеличивается:

$$\text{CPU.USEARCH}(b, a) = b \times a \times \text{CPU.1}. \quad (16\text{-}27)$$

Стоимость хранения данных за определенный промежуток времени получается в результате расширения выражений для вычисления BLKSTOR в гл. 12—15. Надо заметить, что стоимость хранения не зависит от операций ЯМД.

$$CS = (\text{BLKSTOR(индекс)} + \text{BLKSTOR(данные)}) \times C_{ST}, \quad (16\text{-}28)$$

где C_{ST} обозначает единицу стоимости одного байта памяти. На этом расчет общей оценки завершен.

• **Пример 16-1.** Пусть база данных состоит из 10^5 экземпляров записей данных одного типа. В табл. 16.2 приведены сведения

Таблица 16.2. **Объем обработки для примера 16-1 (частота в единицу времени)**

Ключи в условиях элемента в запросе	ПОЛУЧИТЬ НЕКОТОРЫЕ	*I*	*D*	CHK	CHNK	NRIV (NTR)	NIV	NR
A	10	—	—	—	—	10^4	10	10^5
B	10	—	—	—	—	10^2	10^3	10^5
AB	80	10	10	10	10	10	10^4	10^6
Итого	100	10	10	10	10			

о прикладных запросах и операциях обновления. Требуется для указанного объема обработки и значений параметров базы данных выполнить выбор индекса, минимизирующего общую стоимость, заданную уравнением (16-2). Возможные варианты выбора индекса:

1. Нет индекса. 2. Только индекс A. 3. Только индекс B. 4. A, B — отдельные индексы ($A — B$). 5. A, B — составной индекс (AB).

Параметры базы данных:

NR = 10^5 экземпляров записей	CPU. EXCH = 10^{-3} с
SRS = 50 байт	CPU. 1 = 10^{-5} с
KS = 6 байт (в среднем)	CPU. $n = n \times 10^{-5}$ с
PS = 4 байт	$C_{IO} = 1$, $C_{CPU} = 10$, $C_{ST} = 10^{-4}$ [1]
BKS = 1000 байт	sba = 10 мс
LF = 1	rba = 40 мс
BOVHD = 0	CPU. $I = 10 \times 10^{-5}$ с
PRDEL = PRINS = 0	CPU. $D = 5 \times 10^{-5}$ с
(нет перезаписи индекса$_2$)	CPU. CK = 2×10^{-5} с
Фиксированная организация индекса: упорядоченный, плотный, последовательный	CPU. CNK = 2×10^{-3} с
Фиксированный выбор упорядоченности списка указателей доступа: всегда упорядоченный	
Оценка генерации отчета не включается (предполагается, что это постоянная величина)	

[1]) Параметры Мичиганской терминальной системы.

Оптимальный способ индексирования для заданных прикладных функций представляет составной индекс из ключевых элементов A и B (см. табл. 16.3). Еще один хороший способ индексирования, обеспечивающий производительность, близкую к оптимальной, представляет один индекс из ключа B. Самые большие оценки времени имеют место для операций выборки; оценки времени CPU и ввода-вывода для операций выборки являются величинами одного порядка. Хотя стоимость хранения данных велика, но различие в ее значениях для разных способов индексирования базы данных несущественно. Для случаев индексирования все оценки операций обновления невелики; причина этого в том, что они применяются только к 10 записям-целям (0,01 % базы данных), в то время как операции выборки затрагивают 10^4 записей (10 % базы данных). Создание индекса из ключа A неоправданно в силу того, что время ввода-вывода операций выборки 10^4 записей для прикладных запросов, условия элементов которых содержат ключ A, оказывается значи-

Таблица 16.3. **Оценка вариантов индексирования (в секундах)**

	Вариант индексирования				
	1 (нет индекса)	2 (только *A*)	3 (только *B*)	4(*A* — *B*)	5(*AB*)
CR (IO)	5000	36,5K	870	4370	551K
CI (IO)	500	84	6	6	6
CD (IO)	500	80	2	2	2
CCK (IO)	500	165	9	9	9
CCNK (IO)	500	1	1	1	1
CR (CPU)	6800	2616	622	1022	670
CI (CPU)	500	1	1	1	9
CD (CPU)	500	1	1	1	9
CCK (CPU)	500	3	3	3	7
CCNK (CPU)	500	1	1	1	9
CS	500	540	541	581	550
TCOST	16,300 K	39,992K	2,057K	5,997K	1,823K

тельно больше, чем время ввода-вывода последовательного поиска в базе данных. В то же время производительность двух выбранных вариантов индексирования гораздо выше в сравнении с конфигурацией без индекса.

Часть V

Специальные вопросы проектирования

Глава 17. Реорганизация

17.1. Введение

Реорганизация представляет собой изменение либо концептуальной, либо логической, либо физической структуры базы данных. Изменение концептуальной или логической структуры называется реструктурированием, а изменение физической структуры — реформатированием. В качестве примеров реорганизации можно привести добавление нового типа элемента в структуру записи, изменение связей между двумя или более типами записей, перекодирование данных из кода ASC II в код EBCDIC, изменение метода хеширования или же перемещение записей из области переполнения в первичную область данных.

Необходимость реорганизации может возникнуть по целому ряду причин. Она может улучшить использование памяти, уменьшить время выборки и обновления данных, а следовательно, увеличить производительность. Как будет показано в примере, приведенном ниже, реорганизация может стать логически необходимой при изменениях связей между объектами, сведения о которых содержатся в базе данных. Далее приводится ряд примеров, показывающих, при каких обстоятельствах может потребоваться реорганизация базы данных [295].

- *Изменение связей между объектами.* Например, некоторая компания первоначально разрешала каждому из своих сотрудников одновременно работать лишь над одним проектом, позже компания изменила свой принцип и разрешила сотрудникам работать над несколькими проектами одновременно. При этом $1:N$ связь между проектами и сотрудниками должна быть заменена связью типа $M:N$.
- *Добавление новых типов данных.* В этом случае может потребоваться увеличение размеров логических и физических записей для включения в них новых типов элементов данных.
- *Изменение законодательных актов.* Например, ограничения на распространение информации среди правительственных организаций могут потребовать расчленения записей для хранения в защищенных и незащищенных сегментах.

• *Создание новой базы данных на основе уже имеющихся баз данных или файлов.* Например, некоторая компания, присоединившая к себе другую компанию, приняла решение провести слияние баз данных обеих компаний, содержащих сведения о клиентах. Эти базы могли поддерживаться различными СУБД, иметь различные форматы записей, следовательно, их слияние потребует проведения соответствующих преобразований.
• *Изменение характеристик использования.* Например, при проведении нового социологического исследования, использующего базу данных о населении, потребовалось осуществлять доступ к данным по некоторому специфическому ключу. Это может повлечь создание нового вторичного индекса.
• *Увеличение объема данных.* База данных может быть перенесена на более быстрое и емкое запоминающее устройство, что может потребовать изменения размещения записей на физическом носителе.
• *Анализ эксплуатационных характеристик может выявить необходимость настройки или повторного выбора физических параметров базы данных, таких, как способ индексирования или размер блока.*

Некоторые структуры памяти могут выбираться и поддерживаться автоматически с использованием СУБД. Даже для такой системы может потребоваться время от времени ручная или автоматическая реорганизация структуры памяти, а также структуры реализации.

Существует много вариантов реорганизации, начиная от физического форматирования и кончая концептуальным реструктурированием.

С целью классификации типов реорганизации выделим три основных уровня абстракции при проектировании базы данных: концептуальный уровень, уровень реализации и физический уровень.

Реорганизация на концептуальном уровне

Реорганизация на концептуальном уровне может оказаться необходимой в случае изменения информационных требований или же значительной эволюции условий функционирования. Результирующая информационная структура, возможно, выразится в изменениях сущностей, их атрибутов и/или связей. Сущности и атрибуты могут быть добавлены, удалены, объединены, разделены или переименованы. Могут быть созданы, удалены или переименованы связи. Может быть изменен также тип связи: 1 : 1, 1 : *N*, *M* : *N*.

Реорганизация на уровне реализации

Любое изменение на концептуальном уровне, связанное с сущностями, атрибутами и связями, должно также касаться записей, типов элементов и связей между записями на уровне

реализации. Подобные изменения могут быть выполнены в ответ на проведенные на концептуальном уровне изменения, которые должны быть распространены на схему реализации, либо могут возникнуть непосредственно на уровне реализации из-за изменений требований обработки данных, ограничений целостности, безопасности базы данных и т. д. Например, некоторые изменения на уровне элементов могут состоять из изменений масштаба, уровня агрегации (учет рабочих часов за месяц, а не за день) и диапазона значений, которые могут быть связаны с типом элементов. Изменения в схеме и подсхеме СУБД также оказывают непосредственное влияние на разработку прикладных программ.

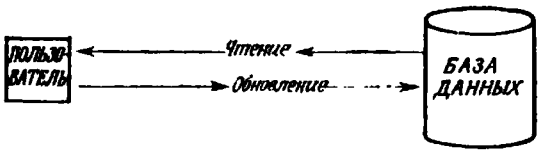
Реорганизация на физическом уровне

Любые изменения в форматах или содержании хранимых записей, в методах доступа, блокирования, кластеризации хранимых записей подпадают под категорию реорганизации физических записей, или реформатирования. Приведем несколько характерных примеров этих и других изменений:

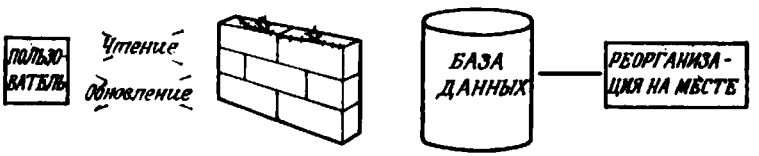
- Возможности выбора указателя (например, последовательный иерархический вместо прямого иерархического в СУБД IMS или же добавление указателей на владельцев в CODASYL).
- Конфигурация индексирования.
- Балансирование иерархии индексов.
- Управление безопасностью (например, замок секретности в CODASYL).
- Явное хранение данных вместо динамического их получения (например, в CODASYL ACTUAL RESULT (действительный результат), а не VIRTUAL RESULT (виртуальный результат)).
- Замена базисного представления данных на кодированное, криптографирование и сжатие.
- Размер элемента.
- Фиксированная, а не переменная длина записи.
- Изменение метода доступа.
- Расчленение либо кластеризация хранимых данных.
- Контроль сохранности цепочек указателей (обнаружение пропущенных или ошибочных указателей).
- Устранение переполнения первичной области данных; сжатие первичной области.
- Настройка, вызванная заменой типов устройств хранения данных.

17.2. Стратегии реорганизации

Мы рассмотрим четыре стратегии, которые могут быть использованы при реорганизации базы данных [295]. Три первые из них обычно применяются для существующих в настоящее

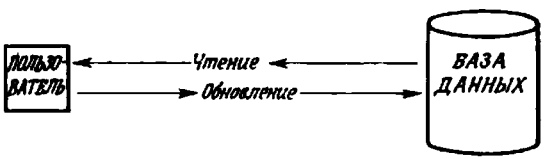


Шаг 1. Обычный доступ

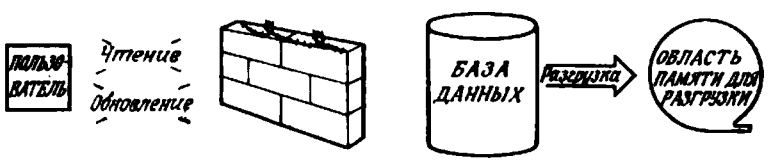


Шаг 2. Реорганизация на месте

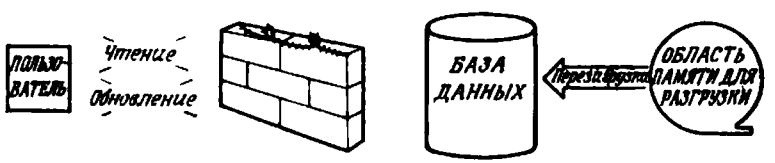
Рис. 17.1. Реорганизация на месте [295].



Шаг 1. Обычный доступ



Шаг 2. Разгрузка



Шаг 3. Перезагрузка и реорганизация

Рис. 17.2. Реорганизация путем разгрузки и перезагрузки [295].

время СУБД, а четвертая — для баз данных, поддерживаемых некоторыми разрабатываемыми СУБД. При использовании первых двух стратегий база данных или же ее часть, подлежащая реорганизации, обычно выводится в автономный режим и становится недоступной для нормального использования на несколько часов.

1. *Реорганизация на месте.* Иллюстрация метода приведена на рис. 17.1. На шаге 1 возможен обычный доступ пользователей к базе данных. Далее, на шаге 2 все запросы пользователей блокируются на время проведения реорганизации, после завершения которой пользователям вновь разрешается доступ к базе данных. Один из вариантов этой стратегии, использующийся

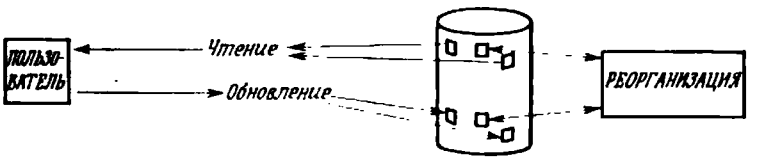


Рис. 17.3. Реорганизация параллельно с эксплуатацией [295].

в ряде случаев, состоит лишь в изменении определения базы данных без проведения физической реорганизации.

2. *Реорганизация путем разгрузки и перезагрузки.* Первоначально, на шаге 1, разрешается обычный доступ пользователей к базе данных (см. рис. 17.2). На шаге 2 блокируются все запросы пользователей и проводится разгрузка базы данных в предназначенную для этого область памяти, и затем на шаге 3 проводится перезагрузка базы данных уже в реорганизованном формате. После завершения реорганизации разрешается обычный доступ к базе данных. Один из вариантов такой стратегии заключается в реорганизации базы данных путем копирования из одной рабочей области в другую без использования промежуточной области.

3. *Реорганизация приращениями.* Стратегией реорганизации, не предусматривающей перевод базы данных в автономное состояние, является реорганизация приращениями, выполняемая по мере ссылок на элементы данных. При этом необходимая реорганизация протекает приращениями, когда пользователь ссылается на какую-либо единицу данных в базе данных (например, перемещение синонима хеширования в его собственный бакет в случае удаления записи, ранее располагавшейся в этом бакете).

4. *Реорганизация параллельно с эксплуатацией.* Другой стратегией, не требующей перевода базы данных в автономное состояние, является реорганизация, проводимая параллельно с

эксплуатацией. Согласно этой стратегии, пользователи имеют доступ к реорганизованной части базы данных, в то время как один или более процессов реорганизации осуществляют модификацию базы данных либо на месте, либо путем загрузки и перезагрузки. Иллюстрация этой стратегии приведена на рис. 17.3. Примером этой стратегии является способ динамического расчленения блоков, применяемый при реализации *B*-деревьев, включая VSAM.

Многие находящиеся в эксплуатации коммерческие СУБД содержат обширные программы реорганизации, использующие различные комбинации приведенных выше стратегий. Полный обзор этих вопросов дан в [295].

17.3. Роль администратора базы данных

Реорганизация базы данных обычно проводится под руководством администратора базы данных (АБД) [52, 102, 103, 208, 339].

Именно АБД определяет необходимость и наилучший момент проведения реорганизации и непосредственно осуществляет ее средствами СУБД или специально разработанного программного обеспечения. При выполнении этих функций АБД решает следующие вопросы [295]:

1. Определяет *необходимость* реорганизации.

2. Решает, *какие* новые структуры должны быть конечным результатом реорганизации.

3. Решает, *когда* осуществлять реорганизацию. Для эксплуатации может существовать оптимальный период времени между реорганизациями. Этот вопрос обсуждается далее в разд. 17.4.

4. Выбирает *способы* проведения реорганизации, состоящие:

а) в выборе стратегий реорганизации, которые могут представлять из себя разгрузку и перезагрузку, реорганизацию на месте с прерыванием режима эксплуатации, постепенную реорганизацию по мере ссылок на элементы данных или же реорганизацию в процессе эксплуатации;

б) в определении подходящих средств проведения реорганизации.

5. Определяет *выгоду*, которую дает реорганизация базы данных в целом. Это может быть увеличение производительности при работе с базой данных, увеличение ее функциональных возможностей или же улучшение использования памяти ЭВМ.

6. Оценивает *затраты*, связанные с проведением реорганизации, а именно:

а) людские и вычислительные ресурсы, необходимые при планировании, собственно реорганизации, изменениях в программном обеспечении и обучении персонала;

б) отказы в обслуживании пользователей при реорганизации с прерыванием режима эксплуатации либо снижение производительности при реорганизации в режиме эксплуатации.

7. Определяет, *кто* будет *затронут* и *что* будет *затронуто* реорганизацией. Инструментом, с помощью которого часто можно определить влияние реорганизации, является словарь-справочник данных, который содержит информацию об элементах данных, об их взаимосвязях и о том, в каких прикладных задачах эти данные используются. Некоторые прикладные задачи могут выиграть от реорганизации, в то время как другие — проиграть, если база данных после реорганизации не будет для них оптимальной. В этой ситуации АБД должен выступать как арбитр, следить, чтобы затронутое программное обеспечение дорабатывалось, и обеспечивать, чтобы обучение персонала осуществлялось для всех пользователей, затронутых реорганизацией.

8. *Документирует* любые изменения, являющиеся результатом реорганизации. Часть этой документации может обеспечиваться словарем-справочником данных.

9. *Контролирует* получение желаемого результата. Например, это может потребовать проверок с целью установления, что новые указатели правильно поддерживают новые связи.

17.4. Когда проводить реорганизацию: эвристический подход

В этом разделе рассматривается влияние различных системных и пользовательских параметров на выбор оптимальных точек реорганизации. Вводится эвристическое правило, основанное на суммарной стоимости и определяющее множество квазиоптимальных точек реорганизации. Стоимость за период времени t определяется в данном контексте как некоторая функция поиска и обновления базы данных, которая также учитывает стоимость памяти. Этот анализ касается главным образом реорганизации, проводимой с целью сопровождения базы данных, например путем устранения переполнения, а не с целью однократного реформатирования или реструктурирования, основанных на учете изменившихся требований.

В работе [285] в отношении определения оптимальных точек реорганизации предполагалась линейная функция стоимости, линейное увеличение объема базы данных и известное время жизни базы данных T. Для фиксированной длины интервала реорганизации t_s оптимальное решение было получено в замкнутой форме. Для линейной скорости роста и линейно возрастающей стоимости реорганизации было найдено, что t_s должно быть приблизительно пропорционально $T^{1/2}$. При $T \to \infty$ t_s становится неопределенным. Оптимальное решение при переменной

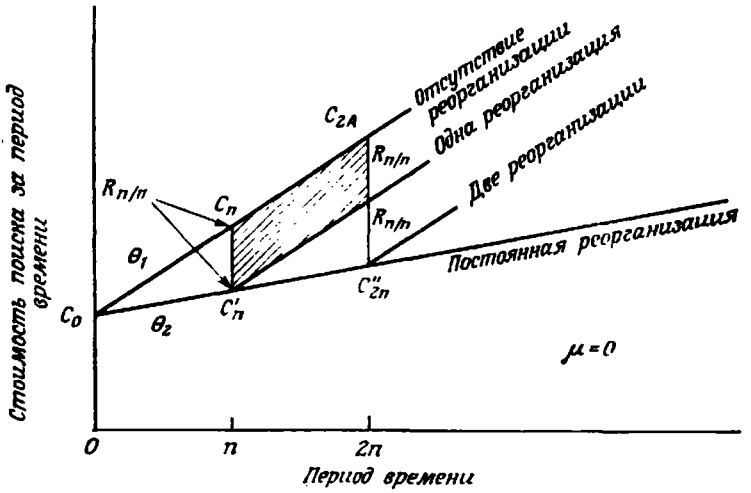


Рис. 17.4. Стоимость поиска за период времени как функция времени и количества реорганизаций.

длине интервала реорганизации и линейно возрастающем объеме базы данных было получено позднее [327].

Определение роста объема базы данных в единицах количества экземпляров элементов и экземпляров записей часто приводит к нелинейной функции стоимости, возрастающей быстрее линейной. Можно показать, что при нелинейной стоимости реорганизации, неизвестном времени жизни базы данных и фиксированном уровне рабочей нагрузки время между точками реорганизации увеличивается пропорционально объему базы данных [[351]. С другой стороны, уровень рабочей нагрузки, пропорциональный размеру базы данных, приводит к последовательности быстро сокращающихся интервалов между точками реорганизации. Квазиоптимальность эвристического алгоритма демонстрируется путем сравнения суммарной стоимости с точным решением, полученным для линейных моделей стоимости при различных значениях времени жизни базы данных.

Допустим, что поиск в базе данных имеет начальную стоимость C_0. Время поиска может ухудшиться из-за вставок и других операций обновления данных. С целью снижения стоимости поиска необходимо периодически проводить реорганизацию базы данных. Пусть θ_1 — скорость увеличения стоимости поиска при отсутствии реорганизации базы данных, а θ_2 — скорость увеличения стоимости поиска при периодической реорганизации базы данных. Предположим далее, что реорганизация осуществляется через определенные периоды времени. На рис. 17.4 показана зависимость стоимости поиска в базе данных от вре-

мени, где C_n — стонмость поиска после n-го периода времеии при отсутствии реорганизации, C'_n — стоимость поиска после n-го периода времени, если последняя реорганизация проводилась в конце n-го периода времени, R_n — стоимость реорганизации в конце n-го периода временн, R_0 — начальная стоимость реорганизации, а μ — скорость увеличения стоимости реорганизации. Другими словами, имеем

$$C_n = C_0 + \theta_1 n, \tag{17-1}$$

$$C'_n = C_0 + \theta_2 n, \tag{17-2}$$

$$R_n = R_0 + \mu n. \tag{17-3}$$

Если предположить скорость изменения линейной, то как C_n, так и C'_n увеличиваются с ростом n, поскольку время поиска запнси увеличивается по мере роста объема базы данных во времени. R_n также возрастает из-за того, что период проведения реорганизации обычно является возрастающей функцией от количества экземпляров записей в базе даиных. Например, предположим, что реорганизация проводится с целью перемещения записей переполиеиия в первичную область данных и физического удаления тех записей, которые помечены флажками удаленнн. В этом случае R_n есть стоимость разгрузки всех записей, включая удаляемые записи, прииадлежащих одиому или нескольким определенным типам, плюс стоимость перезагрузки неудаляемых записей в предположении, что при реоргаиизации используется стратегия разгрузки и перезагрузки. Если NR есть общее количество записей в подфайле в иачале первого периода времени, а x и y обозначают количество вставляемых и удаляемых записей соответственно, то количество записей, к которым производится обращение при реорганизации, равио $NR + x$, а количество переписаиных записей $NR + x - y$. Количество записей переполиения непосредственно перед реорганизацией находится между 0 и x в зависимости от степени коэффициента загрузки первичной области памяти.

Критерий для определения точек реорганизации, который учитывает скорость роста базы данных, уровень активности пользователей н время жизни базы данных, должен удовлетворять требованиям оптимальиости н вычислнтельиой эффективности. Критерий оптимальиости предполагает миинмизацию суммарной стоимости реорганизации, но если время жизни базы данных неизвестно, то суммарная стоимость реоргаиизации также неизвестиа. Следовательио, так как время жизни базы данных становится известным либо в момент прекращения существования базы данных, либо незадолго до этого, оптимальное решение может быть получено лишь приближенно. Во внимание должна приниматься также эффективность вычислений

точек реорганизации. Следующее эффективное правило является эвристическим. Оно учитывает в лучшем случае предшествующий и n последующих периодов времени.

Критерий динамической реорганизации. Пусть база данных должна быть реорганизована по прошествии n-го периода времени после самой последней реорганизации, где n является наименьшим целым числом, для которого

$$C_n \geqslant C'_n + R_n/n, \tag{17-4}$$

где C_n — стоимость поиска без реорганизации, C'_n — стоимость поиска после реорганизации, а R_n — стоимость текущей реорганизации.

В первую очередь рассмотрим задачу для непрерывной линейной функции стоимости (линейный рост и постоянная активность пользователей). На рис. 17.4 первая точка реорганизации появилась в конце n-го периода времени. Полная стоимость поиска к моменту времени t равна $\int_0^t C_x\,dx$, что соответствует площади под кривой. Заштрихованная площадь между параллельными линиями соответствует экономии затрат, возникшей в результате проведения реорганизации. При $C_n = C'_n + R_n$ площадь, как видно, будет равна R_n, т. е. точно соответствовать стоимости реорганизации в конце n-го периода времени. Поэтому реорганизация после n-го периода времени даст общее снижение суммарной стоимости лишь только после дополнительных n периодов времени. В пределе при $\mu = 0$ (постоянной стоимости реорганизации) вторая реорганизация наступит в момент $2n$.

Суммарная стоимость эксплуатации базы данных есть сумма полной стоимости поиска и стоимости реорганизации:

$$TC_t = \int_0^t C_x\,dx + \sum_m R_m, \tag{17-5}$$

где m обозначены периоды времени, после которых проводилась реорганизация. На рис. 17.5 показана зависимость суммарной стоимости от времени при $\mu = 0$. Кривая, соответствующая проведению одной реорганизации, пересекает кривую, соответствующую отсутствию реорганизаций, в момент времени $2n$. Вторая реорганизация в момент времени $2n$ снижает стоимость поиска до $C'_{2n} = C_{2n} - 2 \times R_n/n$ и удваивает экономию стоимости поиска (см. рис. 17.4). Поэтому требуется только $n/2$ дополнительных периодов времени, чтобы вновь снизить суммарную стоимость по сравнению с отсутствием реорганизации, и n периодов до пересечения кривой, соответствующей одной реорга-

низации. В итоге достигается стабильное состояние, при котором отсутствие реорганизации всегда приводит к более высокой полной стоимости. Для $\mu > 0$ (при линейном росте стоимости реорганизации) R_n увеличивается со временем, однако разность между C'_n и C_n не меняется, вторая точка реорганизации совпадает с точкой $2n$ либо будет дальше ее (см. рис. 17.6).

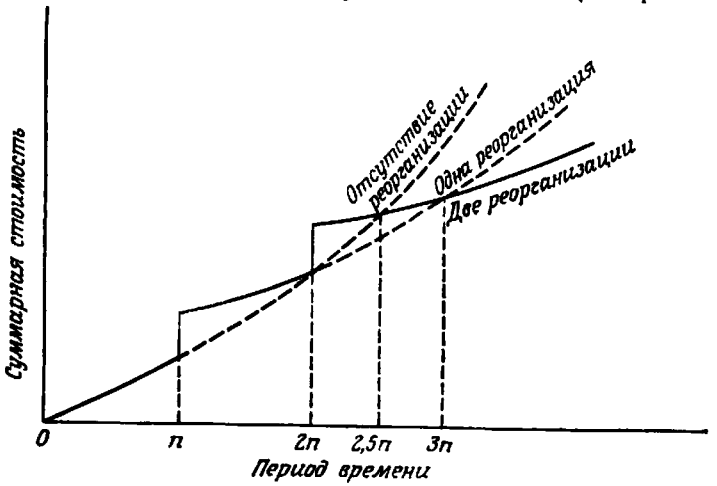


Рис. 17.5. Суммарная стоимость эксплуатации базы данных при реорганизации и при ее отсутствии.

Последующие реорганизации будут проводиться через все бóльшие и бóльшие периоды времени. Начальная экономия в стоимости с увеличением времени будет уменьшаться и приводить к большей стоимости при превышении определенного момента времени.

Для заданного времени жизни базы данных и линейного роста объема базы данных можно определить оптимальное значение фиксированного интервала реорганизации, т. е. получить значение интервала фиксированной длины между точками реорганизации. Оптимальный интервал реорганизации был получен [285] в следующем виде:

$$t_s = \left(\frac{2 \times R_0 + \mu \times T}{\theta_1 - \theta_2}\right)^{1/2}, \qquad (17\text{-}6)$$

где R_0 — начальная стоимость реорганизации, θ_1 — скорость роста без реорганизации, θ_2 — скорость роста с непрерывной реорганизацией, μ — скорость роста R_n в единицах стоимости за период времени.

Если время жизни базы данных неизвестно, то получить оптимальный интервал реорганизации невозможно. Критерий дина-

мической реорганизации может применяться и при неизвестном T. Следующая теорема задает точки, определяющие динамику реорганизации, начиная с наиболее ранней.

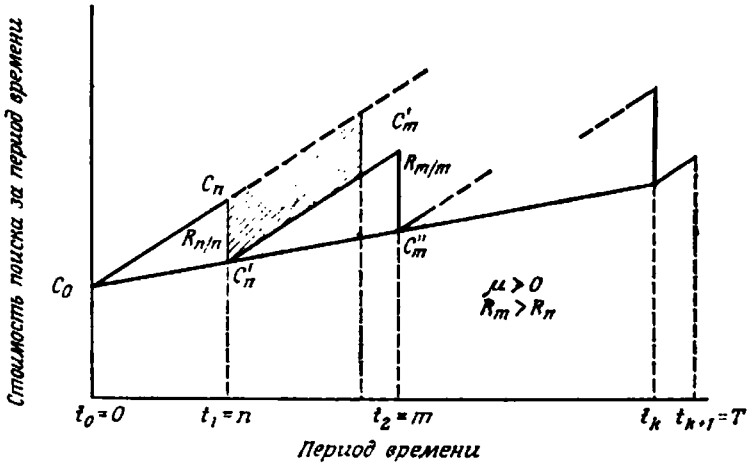


Рис. 17.6. Стоимость поиска за период времени при возрастающей стоимости реорганизации.

• **Теорема 17.1.** Для критерия динамической реорганизации точка реорганизации определяется из следующего выражения:

$$t_n = \frac{\mu + [\mu^2 + 4 \times R_0 (\theta_1 - \theta_2)]^{1/2}}{2 (\theta_1 - \theta_2)}. \qquad (17\text{-}7)$$

Доказательство. Пусть критерий реорганизации определяется следующим образом:

$$C_n \geqslant C_n' + R_n/n \text{ или } C_0 + \theta_1 t_n \geqslant C_0 + \theta_2 t_n + (R_0 + \mu t_n)/t_n;$$

умножая на t_n обе части неравенства и проводя упрощение, получим

$$C_0 t_n + \theta_1 t_n^2 \geqslant C_0 t_n + \theta_2 t_n^2 + R_0 + \mu t_n,$$

или $\theta_1 t_n^2 \geqslant \theta_2 t_n^2 + R_0 + \mu t$.
В случае равенства $t_n^2(\theta_1 - \theta_2) + t_n(\mu) - R_0 = 0$.
Решая квадратное уравнение, получим

$$t_n = \frac{\mu + [\mu^2 + 4 \times R_0 (\theta_1 - \theta_2)]^{1/2}}{2 (\theta_1 - \theta_2)}. \quad \square$$

Отметим, что выражение (17-7) должно использоваться после каждой проведенной реорганизации с целью определения следующей точки реорганизации и что R_0 заменяется на текущее

значение R_n после реорганизации. Точка реорганизации существует для любых заданных величин $\theta_1 - \theta_2$, R_0 и μ.

В реорганизации нет необходимости лишь при $t_n > T$.

В работе [351] показано, что для критерия динамической реорганизации суммарная стоимость самое большее на 6 % выше, чем оптимальное значение суммарной стоимостн для фиксированных точек реорганизации, определяемых согласно работе [285]. При $\mu > 0$ критерий динамнческой реорганизации дает лучшие результаты, чем метод с фиксированными точками реорганизации, хотя ни один из них не является оптимальным. Оптимальная стратегия и близкое к оптимальному приближение при снятии ограничения в виде фиксированной длины интервала реорганизации были получены автором работы [327]. Он также показал, что критерий динамической реорганизации достаточно близок к оптимальному, хотя и не настолько, насколько близко предложенное им самим приближенное решение. Полезность эвристического подхода состоит в применимости его к реальиым ситуациям, таким, как неизвестное время жизни T, нелинейная скорость роста базы данных илн нелинейное увеличение рабочей нагрузки базы данных.

Эксперименты, проведенные при использовании эвристического подхода, показывают, что полученное при этом значение интервала времени между реорганизациями крайне чувствительно к способу организации файла. По мере возрастания стоимости поиска (доступа) интервалы реорганизации имеют тенденцию к уменьшению, если только стоимость реорганизации не увеличивается с большей скоростью. Возрастание активности пользователей базы данных, илн рабочей нагрузки, приведет к возрастанию стоимости обработки приложений без увеличения стоимости реорганизации. Если объем базы данных не увеличивается, то это приводит к быстрому уменьшеиию интервалов времени между реорганизациями. Если же объем базы данных увеличивается, то возрастают как стоимость обработки приложений, так и стоимость реорганизации. Следовательно, интервал между реорганизациями может возрастать или уменьшаться в зависимостн от нзменения параметров баз данных.

Отметим, что на практнке функции стоимости ие всегда столь легко определять. Реорганизации часто проводятся, либо когда стоимость реорганнзации крайне мала, например в выходные или праздничные днн, либо непосредственно перед моментом времени, когда ожидаемая стоимость эксплуатацин базы данных становится очень большой. Однако эвристическое правнло применимо и в этих случаях, так как оно согласуется с интуитнвиым пониманием того, когда проводить реорганизацию.

17.5. Реструктурирование

Целью реструктурирования является преобразование логической структуры базы данных, обеспечивающее возможность удовлетворения новых информационных требований или требований обработки данных.

В работе [236] предложены категорин операций реструктурирования для класса иерархических логических структур и определены три основных типа модификации логических структур: поименование, отношение и соединение, которые описываются с помощью нескольких операций реструктурирования более низкого уровня.

На рис. 17.7 показаны некоторые операции иерархического реструктурирования [120, 122]. Часть *а* этого рисунка описывает иерархические связи между ПРЕЗИДЕНТЫ, СУПРУГИ и ДЕТИ. Записи типа ПРЕЗИДЕНТЫ содержат имена президентов США. Тип записи второго уровня, СУПРУГИ, связан с типом записи ПРЕЗИДЕНТЫ посредством типа набора «женат на», с помощью которого указываются имена всех супруг каждого президента. Таким образом, набор представляет собой отображение $1:M$ между двумя типами записей. Тип записей третьего уровня, ДЕТИ, связан с типом записи СУПРУГИ аналогичным образом и содержит имена детей каждой СУПРУГИ.

В части *б* рис. 17.7 исходный тнп записи ДЕТИ расчленен на два типа записей: СЫНОВЬЯ и ДОЧЕРИ на основании элемента данных «пол». Расчленение является операцией реструктурирования, при которой один тип записей разделяется на два или более различиых типов на основании значения одного или более элементов данных. Отметим, что элемент данных «пол» должен быть исключен из логической структуры, представленной в части *а* рисунка, так как эта информация теперь представлена логической структурой, показанной в части *в* рисунка.

Операция реструктурирования, обратная расчленению, называется соеднненнем. Она состоит в объединении экземпляров двух или более типов записей в единственный тип записей. Чтобы сохранить информацию, представляемую ранее логнческой структурой, часто в тип записи добавляется элемент данных. На рис. 17.7 экземпляры двух типов записей СЫНОВЬЯ и ДОЧЕРИ должны быть соединены в экземпляры единственного типа записей — ДЕТИ. Термины «исходный» и «результирующий» являются условнымн и зависят от выбора направления преобразования при реструктурированнн. В части *а* рис. 17.7 представлена исходная логическая структура для прнмера расчленения, которая является результирующей для примера соединения. Для сохранения информацни, представленной ранее семантически в виде исходной логической структуры в

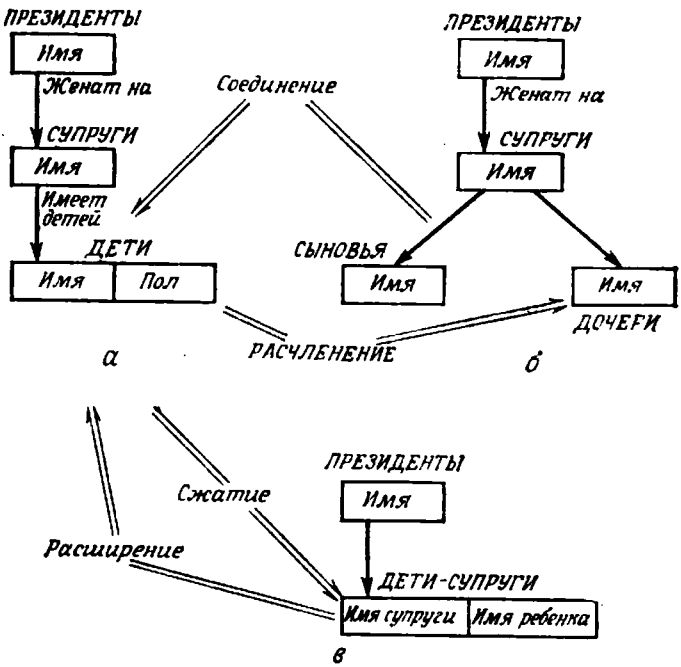


Рис. 17.7. Операции реструктурирования.

части *б* на рис. 17.7, тип записи ДЕТИ дополняется типом элемента «пол».

Еще одна операция реструктурирования заключается в сжатии двух или более иерархических уровней в один. На рис. 17.7 два исходных типа записей СУПРУГИ и ДЕТИ должны быть сжаты в единственный тип записи СУПРУГИ-ДЕТИ. На уровне экземпляров записей эта операция реализуется путем повторения в каждом экземпляре записи типа ДЕТИ, связанного с ними экземпляра записи типа СУПРУГИ.

Операция, обратная сжатию,— расширение. Она расширяет один уровень иерархии на два или более уровней путем расщепления выбранных записей данных. Это иллюстрируется на рис. 17.7 (часть *в*), где один тип записи СУПРУГИ-ДЕТИ должен быть расщеплен на два уровня.

17.5.1. Сетевое реструктурирование

В классе сетевых логических структур операции реструктурирования разделить на категории гораздо труднее. В первую очередь это происходит из-за многообразия и сложности име-

ющихся структур. Например, рис. 17.8 иллюстрирует преобразование сетевой структуры, состоящее в изменении типа связи $1:M$ на связь типа $M:M$. В исходной логической структуре

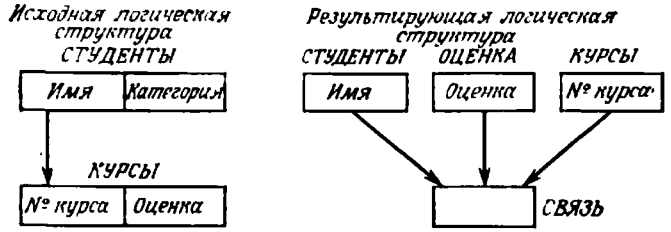


Рис. 17.8. Пример реструктурирования сетевой структуры.

представлены следующие сведения: о студентах, о принадлежности к категории (выпускники, дипломники, специализация, иностранцы и т. д.), о курсах, изучаемых студентом, и об итоговых оценках, полученных студентом за каждый курс. В результирующей логической структуре представлены те же самые сведения, за исключением того, что связь между студентами, курсами и оценками осуществляется с использованием типа записи

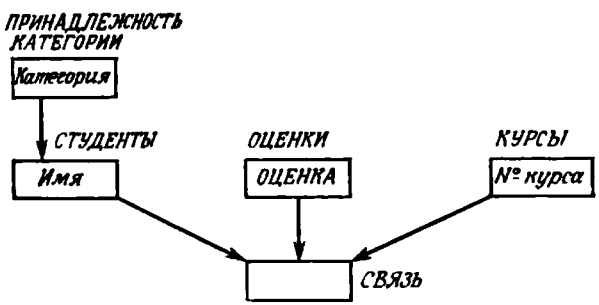


Рис. 17.9. Введение набора индексирования.

СВЯЗЬ. Это является типичным примером операций реструктурирования, характерных для сетевых структур, в которых используется несколько типов записей и типов наборов.

Другой операцией реструктурирования является введение индексирования типов наборов и типов записей. Это можно достичь путем переноса элемента данных «категория» в тип записи ПРИНАДЛЕЖНОСТЬ-КАТЕГОРИИ. Такой перенос служит целям указания сведений о распределении студентов по категориям (рис. 17.9).

Имеется еще одна важная операция реструктурирования сетевой структуры, позволяющая получать из исходной базы дан-

ных не только фактически существующие в ней данные, но также и неявные данные. (Под неявными данными понимаются сведения, которые могут быть выведены из фактически существующих исходных данных.) На рис. 17.10 приводится пример, показывающий различие между неявными и явными сведениями.

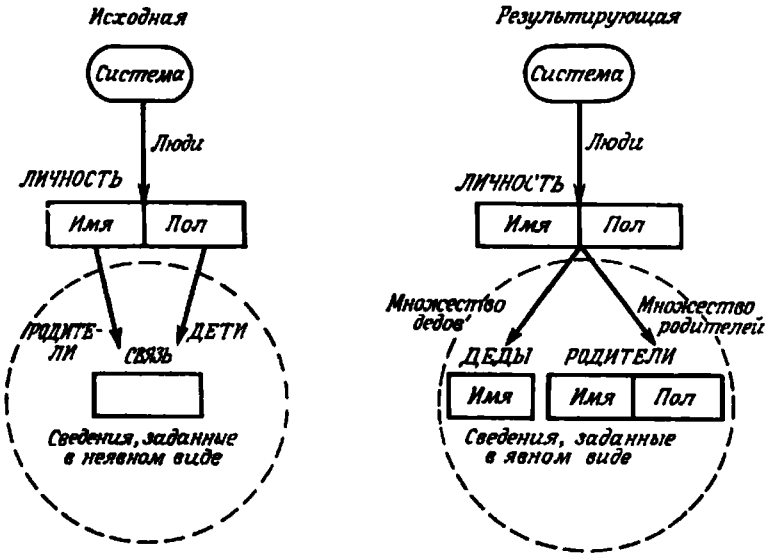


Рис. 17.10. Выборка данных, заданных неявно.

Он описывает исходную базу данных, содержащую два типа записей: ЛИЧНОСТЬ и СВЯЗЬ. Тип записи ЛИЧНОСТЬ содержит типы элементов «имя» и «пол». Тип записи СВЯЗЬ не содержит данных, но обеспечивает информационную связь наборов РОДИТЕЛИ и ДЕТИ. Результирующая структура также содержит тип записи ЛИЧНОСТЬ. Записи этого типа предназначены для хранения данных, явно содержащихся в исходном файле. Напротив, типы записей РОДИТЕЛИ и ДЕДЫ результирующей структуры непосредственно не соответствуют никакому типу записей исходного файла, а содержат сведения, неявно содержащиеся в исходном файле. Нет необходимости говорить о том, что такое преобразование не может быть описано при помощи операций над иерархическими подструктурами.

17.5.2. Инженерные подходы к реструктурированию

Использование элементарных операций

Существует интересная аналогия между системами реструктурирования и справочными системами высокого уровня. Дей-

ствительно, мы можем рассматривать запрос как ограниченное реструктурирующее преобразование, целью которого является получение некоторой иной формы представления информации. Использование элементарных операций позволило создать эффективные простые языки запросов для класса иерархических логических структур [287, 288, 289]. Эти системы дают возможность пользователю сформировать последовательность элементарных операций над иерархической структурой, позволяющую получить ответ на запрос. Подобные языки легко изучать и применять, поскольку для реализации большинства запросов они используют небольшое число весьма простых операций.

Для решения задачи реструктурирования и создания языков реструктурирования с использованием элементарных операций было предпринято несколько попыток. Так был создан язык высокого уровня CONVERT, который реализует все возможности реструктурирования иерархических структур [289]. Он базируется на концепции «формы данных» в связи с набором реструктурирующих функций, называемых «операциями над формами».

Авторы работ [287, 288] описали подобный подход к реструктурированию, применяя набор «преобразующих функций» для определения операций реструктурирования.

Под исходной базой данных в методе элементарных операций подразумевают набор данных с определенной логической структурой и форматами. Под результирующей базой данных подразумевают те же данные, но с отличающейся логической структурой и форматом. Следовательно, реструктурирование есть процесс обработки исходных данных с целью их преобразования к виду, определяемому результирующими логической структурой и форматом.

Исследования метода реструктурирования с использованием элементарных операций привели к использованию операций низкого уровня (примитивов), осуществляющих преобразование элементов из одной логической структуры в другую. Одним из преимуществ такого подхода является то, что при этом значительно упрощается структура программного обеспечения, реализующего реструктурирование. Оно сводится к набору подпрограмм низкого уровня, в точности соответствующих элементарным операциям. Таким образом, спецификация процесса реструктурирования состоит в задании последовательности примитивов, которые могут быть непосредственно преобразованы в последовательность вызовов подпрограмм; подпрограммы осуществляют фактическое реструктурирование. К сожалению, пользователи должны детально разбираться во всех вопросах реструктурирования и в совершенстве представлять себе функции каждой операции для того, чтобы пользоваться ею. Следо-

вательно, даже язык высокого уровня требует, чтобы пользователь представлял себе реструктурирование как последовательность этапов низкого уровня.

Ряд других проблем возникает при применении подхода с использованием операций низкого уровня к реструктурированию сетевых баз данных. Вообще говоря, элементарные операции задуманы для работы с небольшими логическими подструктурами (состоящими из одного или двух типов записей и одного набора), при этом происходит создание новой логической подструктуры. Поскольку может быть определено лишь ограниченное количество операций, то только лишь конечное число исходных подструктур могут быть кандидатами для реструктурирования. И чем сложнее структура, тем больше вероятность того, что она содержит подструктуры, которые не являются допустимыми для ряда имеющихся операций. В этом смысле элементарные операции не очень хорошо подходят к описанию реструктурирующих преобразований сложных сетевых логических структур. В дополнение к этому сложное реструктурирующее преобразование (имеется в виду то, что оно образовано из элементарных операций) требует выполнения сложной последовательности элементарных операций, которую трудно анализировать.

Другим классом реструктурирующих преобразований, который трудно выполнить посредством использования элементарных операций, является извлечение неявной информации. Мы показали, что такие преобразования (см. пример на рис. 17.10) обычно не могут быть описаны последовательностью операций над иерархическими подструктурами. Даже если эти преобразования и могут быть описаны последовательностью элементарных операций над сетевыми подструктурами, то такие описания бывают обычно громоздкими и сложными.

Следуя нашей аналогии, отметим, что очень трудно применить элементарные операции справочных систем высокого уровня к сетевым базам данных. Использование элементарных операций при разработке справочных систем для сетевых баз данных ведет по сравнению с иерархическими к уменьшению мощности системы и делает ее громоздкой. Вообще говоря, все недостатки вытекают из следующих отмеченных выше проблем: ограниченность допустимых входных структур, чрезмерно сложные описания, трудности, связанные с наличием связующих записей, использование других методов запоминания неявной информации. Метод, основанный на использовании путей доступа, возник из попыток разработать стратегию реструктурирования, более подходящую к сетевым базам данных.

Использование путей доступа

Следуя принятой в настоящее время тенденции в области систем баз данных с включающими языками, использующимися

для обработки сетевых баз данных, выберем для реструктурирования метод, основанный на использовании путей доступа. Исходная база данных рассматривается как совокупность сведений, часть которых представлена данными в явном виде, а часть представлена неявно. Подобным же образом предполагается, что результирующая база данных содержит подмножество сведений из исходной, и все данные в результирующей базе данных взяты из исходной. Тогда выполнение реструктурирующего преобразования сводится попросту к обходу исходной базы данных для получения данных, необходимых для создания результирующей базы данных, и их запоминания в соответствии с результирующей логической структурой.

Существуют многочисленные разновидности такого подхода. Например, проводимые исследования в области спецификации реструктурнрования пытаются развить языки спецификации реструктурирования, основанные на концепциях стратегий доступа и критериях выбора. Так как эта проблема тесно связана с разработкой языков запросов, то в ней могут быть использованы результаты предыдущих исследований. Метод, осиованный на использовании путей доступа, влияет также на разработку фактических алгоритмов реструктурирования. Разработаны алгоритмы, осуществляющие эффективный и исчерпывающий доступ к исходной базе данных. Наряду с этим они осуществляют тестирование данных согласно внешне определенным стратегиям доступа и критериям проверки. В этом состоит коренное отличие рассматриваемого метода от метода элементарных операций, который ведет к развитию подпрограмм нижнего уровня.

Наиболее важное следствие такого подхода состоит в создании мощных обобщенных средств реструктурирования сетевых структур. Поскольку реструктурирование рассматривается как операция, осуществляющая доступ к данным, метод не подвержен влиянию изменений логической структуры базы данных. Такая независимость гарантирует, что любая база данных будет являться допустимой для реструктурирования независимо от сложности логической структуры (иерархической, сетевой и т. п.). Более того, нет необходимости в сходстве исходной и результирующей структур, так как результирующая база данных получается на основе данных, содержащихся в исходной базе. Итак, в отличие от метода элементарных операций неявные сведения могут извлекаться и реструктурироваться столь же легко, сколь и явные (см. рис. 17.10). Таким образом, явные сведения могут превратиться в неявные и наоборот.

И наконец, система по самой своей сути является простой. Пользователи, знакомые с базами данных и их использованием, легко поймут язык описания операций реструктурирования, основанный на таких ориентированных на приложения концепциях,

как стратегия доступа и критерий выбора. Не вызовут затруднений и более сложные реструктурирующие преобразования. Алгоритм, разработанный для реализации этого процесса, также является простым, открытым и надежным (достоверным).

Все операции реструктурирования осуществляются при помощи одной и той же последовательности шагов независимо от особенностей преобразования:

- Осуществление доступа ко всем исходным данным в соответствии со спецификацией доступа.
- Проверка данных на основе критерия выбора.
- Создание экземпляров результирующих записей результирующей базы данных, содержащих релевантные данные.

Более полное теоретическое обоснование этого метода содержится в [106].

Архитектура языка спецификации путей доступа

Построение языка спецификации путей доступа (Access Path Specification Language — APSL) основывается на высокоуровневой спецификации путей доступа. Главные компоненты языка описывают стратегии доступа и критерии выбора. Стратегии доступа описываются с помощью оператора пути доступа. Этот оператор определяет схему обхода, требуемую для получения данных для каждого результирующего типа записи. Критерий выбора устанавливает исходные (и, косвенно, результирующие) требования к данным. Детальное описание APSL выходит за рамки этой книги (см. [312]), однако основные черты языка здесь будут рассмотрены.

APSL является языком с явно выраженной блочной структурой, в котором каждый блок содержит единственный оператор результирующей записи. Таким образом, существует один оператор результирующей записи для каждого типа записи в результирующей базе данных. Эта структура отражает подход, основанный на использовании путей доступа, поскольку каждый результирующий тип записи представляет определенную порцию сведений, которые могут быть получены из исходной базы данных. Следовательно, каждое описание результирующей записи содержит спецификации для схемы доступа в исходной базе данных и для критерия выбора.

Второй уровень структуры в APSL представляет собой оператор результирующего набора (см. рис. 17.11). Каждый оператор результирующей записи включает в себя один или несколько операторов результирующих наборов, идентифицирующих наборы, в которых результирующие типы записей являются членами.

Третий уровень структуры APSL — оператор пути доступа. Этот оператор точно специфицирует то, как должен быть совершен обход исходной логической структуры, с целью получения

данных, необходимых для создания экземпляра записи результирующего логического типа. Это могут быть отдельные операторы пути доступа, один или несколько для каждого оператора результирующего набора, поскольку данные, составляющие тип результирующей записи, могут принадлежать нескольким различным типам исходных типов записей.

На четвертом уровне структуры существуют два типа операторов APSL: оператор нового результирующего элемента и оператор исходной записи. Для каждого оператора пути доступа может быть либо нуль, либо несколько операторов новых результирующих элементов. Такой оператор определяет результирующий элемент, которому присваивается постоянное значение каждый раз, когда создается экземпляр результирующей записи с использованием определенного пути доступа. Так как путь доступа указывает структуру в исходной базе данных, оператор нового элемента является полезным в тех случаях, когда семантически представленная в исходной структуре информация преобразуется в результирующей структуре в фактические значения данных.

Вторым оператором APSL на четвертом уровне является оператор исходной записи. Он определяет исходную запись (записи) в пути доступа, которая будет использоваться при создании экземпляра результирующей записи для получения значения элемента и/или для проверки исходных данных. Для каждого оператора пути доступа может существовать один или несколько операторов исходной записи, так как данные могут поступать (и/или проверяться) из нескольких различных типов исходных записей. Более того, каждому типу исходной записи может быть поставлен в соответствие индексный номер, однозначно определяющий исходную запись внутри пути доступа. Допускается возможность организации циклов или петель, что обеспечивает тем самым многократное использование одной и той же исходной записи.

На пятом, и конечном уровне структуры APSL существуют два типа операторов: оператор квалификации элемента и оператор присваивания. Оператор квалификации элемента используется для установления критерия выбора, используемого при проверке исходных данных. Таким образом, имеется возможность специфицировать постоянное значение и затем сравнивать его со значением исходного элемента, для того чтобы определить, должен ли быть создан экземпляр результирующей записи, использующий текущий набор исходных данных. Оператор присваивания используется для получения значения исходного элемента и присвоения его в дальнейшем соответствующему результирующему элементу. Структура APSL показана на рис. 17.11.

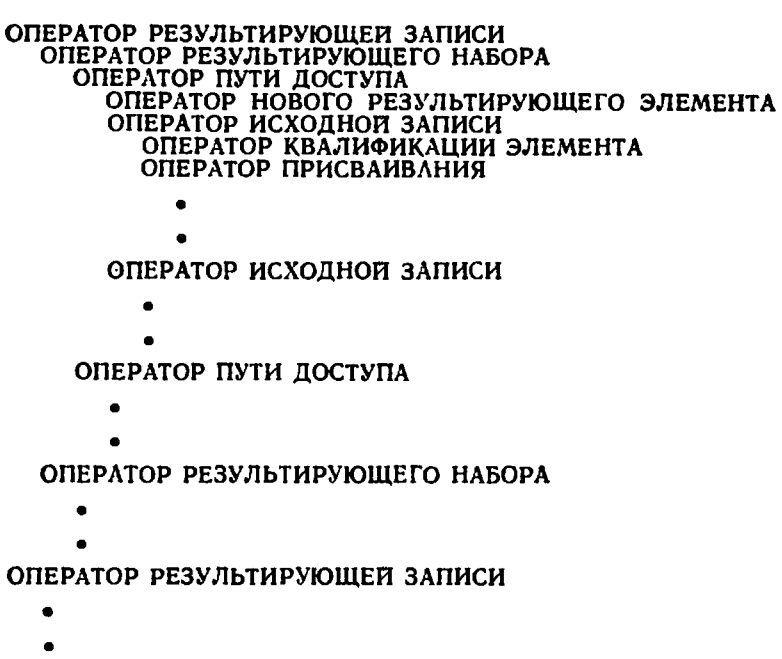


Рис. 17.11. Структура языка APSL.

Необходимо отметить, что APSL не дает явного описания логической структуры ни исходной, ни результирующей базы данных. Мичиганский транслятор данных получает соответствующие описания об исходной и результирующей базах данных без ссылок на описание реструктурирования.

Таким образом, APSL описывает только информацию, относящуюся к фактическому преобразованию исходных данных в результирующие. При этом считается, что логические структуры баз данных должны быть предварительно определены и доступны для реструктурирующих действий.

Пример реструктурирования с использованием APSL

Для иллюстрации был выбран пример, демонстрирующий реструктурирующие возможности языка APSL, которые не могут быть классифицированы в терминах иерархических структур или элементарных операций (рис. 17.12). Пример предполагает выделение неявной информации, содержащейся в сетевой структуре, приведенной на рис. 17.10. Программа на APSL, необходимая для проведения нужных преобразований, приведена на рис. 17.13. Для ясности текст программы несколько упрощен. Ясно видна также блочная структура языка.

Так как имеются три типа результирующих записей, то имеются также три оператора результирующих записей. Опера-

тор 1 начинает описание для результирующего типа записи ЛИЧНОСТЬ. Тип записи ЛИЧНОСТЬ является членом лишь одного набора (из системного уровня доступа); как следствие этого имеется лишь один оператор пути доступа (2). Оператор 3 определяет исходный путь доступа, который должен быть использован для определения данных, необходимых для создания

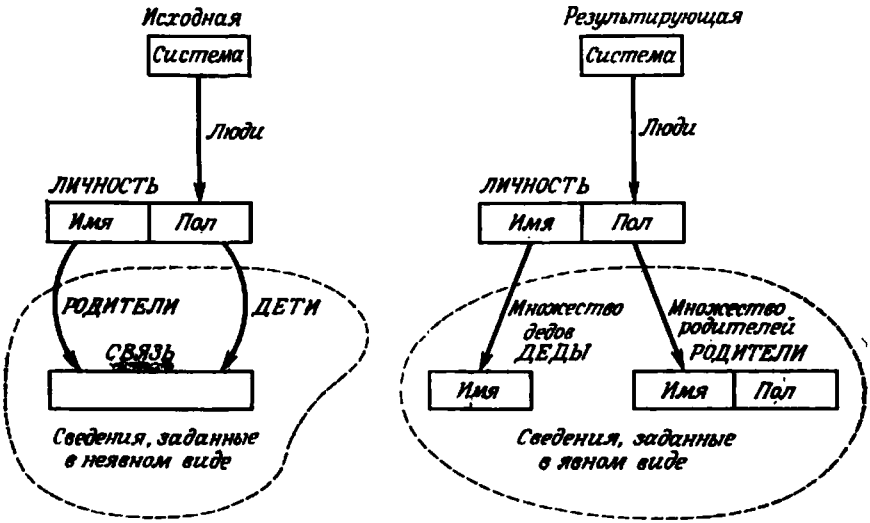


Рис. 17.12. Пример реструктурирования с использованием языка APSL.

результирующего типа записей. ДОСТУП-ЧЕРЕЗ-ЛЮДИ описывает путь доступа от исходной вершины СИСТЕМА через набор ЛЮДИ к типу записей ЛИЧНОСТЬ. Оператор 4 переносит данные из исходной записи в результирующую запись. ФАКТИЧЕСКИЕ-ДАННЫЕ-УПОРЯДОЧЕНЫ указывает, что значения в типах исходных элементов записей «имя» и «пол» должны быть присвоены результирующим типам элементов «имя» и «пол». Описание типа записей ДЕДЫ начинается оператором 12. Как и ранее, имеется лишь один оператор пути доступа (13), так как тип записи является членом лишь одного набора (НАБОР-ДЕДЫ). Операторы 14, 17, 18 и 19 определяют стратегию исходного пути доступа, которая должна быть использована для получения нужной информации. (Заметим, что это лишь одна из трех возможных альтернатив.)

Приведенная стратегия доступа может быть прокомментирована следующим образом. Тип записи СВЯЗЬ в исходной базе данных моделирует отношение между родителями и детьми. Для заданного конкретного экземпляра записи ЛИЧНОСТЬ путем

```
 1. РЕЗУЛЬТИРУЮЩАЯ ЗАПИСЬ ЛИЧНОСТЬ
 2.   УОПД ЛИЧНОСТЬ
 3.     ИСХОДНАЯ ЗАПИСЬ ЛИЧНОСТЬ ДОСТУП-ЧЕРЕЗ-ЛЮДИ
 4.       ФАКТИЧЕСКИЕ-ДАННЫЕ-УПОРЯДОЧЕНЫ
 5. РЕЗУЛЬТИРУЮЩАЯ ЗАПИСЬ РОДИТЕЛИ
 6.   УОПД РОДИТЕЛИ
 7.     ИСХОДНАЯ ЗАПИСЬ ЛИЧНОСТЬ ИД=РЕБЕНОК ДОСТУП-
        ЧЕРЕЗ-ЛЮДИ
 8.       ИМЯ ПРИСВОИТЬ ИМЯ <НАБОР-РОДИТЕЛИ>
 9.     ИСХОДНАЯ ЗАПИСЬ СВЯЗЬ ДОСТУП-ЧЕРЕЗ-РОДИТЕЛИ ИЗ
        ИД=РЕБЕНОК
10.     ИСХОДНАЯ ЗАПИСЬ ЛИЧНОСТЬ ИД=РОДИТЕЛИ ДОСТУП-
        ЧЕРЕЗ-ДЕТИ
11.       ФАКТИЧЕСКИЕ-ДАННЫЕ-УПОРЯДОЧЕНЫ
12. РЕЗУЛЬТИРУЮЩАЯ ЗАПИСЬ ДЕДЫ
13.   УОПД ДЕДЫ
14.     ИСХОДНАЯ ЗАПИСЬ ЛИЧНОСТЬ ИД = ДЕДЫ ДОСТУП-ЧЕРЕЗ-
        ЛЮДИ
15.       ПОЛ ВЫБОР ЕСЛИ РАВНО 'М'
16.       ИМЯ ПРИСВОИТЬ ИМЯ
17.     ИСХОДНАЯ ЗАПИСЬ СВЯЗЬ ИД=ШАГ 1 ДОСТУП-ЧЕРЕЗ-ДЕ-
        ТИ ИЗ ИД=ДЕДЫ
18.     ИСХОДНАЯ ЗАПИСЬ ЛИЧНОСТЬ ИД=РОДИТЕЛИ ДОСТУП-
        ЧЕРЕЗ-РОДИТЕЛИ ИЗ ИД=ШАГ 1
19.     ИСХОДНАЯ ЗАПИСЬ СВЯЗЬ ИД=ШАГ 2 ДОСТУП-ЧЕРЕЗ-ДЕ-
        ТИ ИЗ ИД=РОДИТЕЛИ
20.       ИМЯ РОДИТЕЛИ ПРИСВОИТЬ ИМЯ <НАБОР-ДЕДЫ>
```

Рис. 17.13. Запрос на языке APSL.

использования типов наборов РОДИТЕЛИ и ДЕТИ, связанных посредством типа записи СВЯЗЬ, можно получить доступ к родителям. Аналогично все деды конкретных личностей получаются путем выделения всех родителей родителей. Операторы путей доступа, расположенные в строках 14, 17, 18 и 19, отражают самую сущность этой стратегии. Используем набор ЛЮДИ для достижения типа записи ЛИЧНОСТЬ (идентификатор ДЕДЫ). Переходим в наборе ДЕТИ к типу записи СВЯЗЬ (идентификатор ШАГ 1), затем через набор РОДИТЕЛИ вновь к типу записи ЛИЧНОСТЬ (идентификатор РОДИТЕЛИ). Далее, переходим в наборе данных ДЕТИ снова к типу записи СВЯЗЬ и, наконец, через набор РОДИТЕЛИ вновь к типу записей ЛИЧНОСТЬ (идентификатор РЕБЕНОК).

Заметим, что исходный тип записи ЛИЧНОСТЬ используется дважды при описании пути доступа. Следовательно, оператор исходной записи каждый раз, когда он используется, должен указывать, какой тип исходной записи ЛИЧНОСТЬ или СВЯЗЬ должен быть использован. При первом использовании записи типа ЛИЧНОСТЬ ей приписывается идентификатор ДЕДЫ, при втором — РОДИТЕЛИ. В этом же смысле должен различаться и тип записи СВЯЗЬ. Оператор 15 является оператором квалификации элемента, используемым для выбора толь-

ко мужчин из родителей родителей, чтобы получить требуемую информацию в результирующей записи, т. е. дедов. Оператор 16 присванвает значение исходного типа элемента «имя» результирующему типу элемента «имя».

Должно быть ясно, что результирующий тип записи РОДИТЕЛИ создается аналогичным образом, отличаясь лишь использованием более простой стратегии доступа и отказом от выбора критерия.

В заключение отметим, что были получены существенные результаты в реализации обобщенного реструктурировщика. И хотя обобщенный реструктурировщик технически реализуем, пока еще слишком рано предсказывать снижение людских и машинных затрат за счет его использования. Существует определенный интерес при рассмотрении больших баз данных, так как производительность реструктурировщика пропорциональна объему базы данных. Однако были предприняты попытки оптимизации его работы, что должно привести к существенному повышению эффективности. Помимо этого, должна быть достигнута еще и полная применимость к процессу физического преобразования. А это представляется значительно более трудной задачей, ибо чем глубже мы вникаем в фактическое представление информации, тем более сложным становится процесс описания и реализации модулей физического преобразования.

Глава 18. Распределенные базы данных: обзор

18.1. Введение

Современные тенденции развития информационных систем состоят в переходе от централизованных вычислительных систем 70-х годов к распределенным системам 80-х годов. Такой переход был подготовлен развитием межмашинных связей и поистине взрывным характером развития мини-ЭВМ в последние годы.

Для того чтобы рассматривать общие характеристики распределенных систем, необходимо провести ясное различие между системами распределенных баз данных и системами распределенной обработки данных. В *системе распределенных баз данных* базы данных распределены между несколькими (возможно, территориально разобщенными) ЭВМ и обеспечены соответствующие возможности для управления этими разделенными частями. По-иному построены *системы распределенной обработки данных*, которые имеют распределенные между взаимосвязанными ЭВМ вычислительные мощности и программное обеспечение, но централизованную базу данных.

В этой главе основное внимание будет сосредоточено исключительно на системах распределенных баз данных. Будут рассмотрены некоторые проблемы проектирования и реализации систем распределенных баз данных, а также основы проектирования распределенных баз данных.

Основной целью системы распределенных баз данных является обеспечение управляемого доступа и независимого обращения к данным, распределенным в сети ЭВМ. Под управляемым доступом понимается степень безопасности, необходимая для защиты данных от неавторизованного доступа. Независимость обращения, или разделимость, позволяет пользователям получать доступ к данным через различные, подчас значительно удаленные вычислительные средства. Сеть ЭВМ представляет совокупность неоднородных вычислительных средств, связанных между собой высокоскоростными каналами связи.

Для обеспечения разделимости данных необходимо согласованное функционирование различных средств доступа и систем управления базами данных. Технология работы с распределенными базами данных развивается очень быстро.

Технологические проблемы в распределенных базах данных в зависимости от их происхождения могут быть разделены на две большие категории. Проблемы, возникающие на стадии

проектирования, будь то проектированне сети, системы управления базой данных или самой базы данных, определяются как *проблемы проектирования.* Аналогнчно проблемы, затрагивающие фуикционированне распределенной системы, именуются как *проблемы реализации.* Несмотря на то что этн проблемы могут быть классифицированы, исходя нз целей опнсания, следует отметить, что между ними существует иепосредственная прнчиниая связь. В связи с тем, что проектирование предшествует эксплуатации, принятые на стадии проектирования решения оказывают непосредственное влияине на реализацию и последующее функционированне системы. Принятые проектные решення, касающнеся таких вопросов, как архитектура системы, размещение данных, распределение сетевого справочнпка и однородность программных и аппаратных средств, непосредственно влияют на результаты реализацин контроля, снихронизации, блокирования и трансляцни.

Хотя и был предложен ряд оригинальных решеиий н некоторые даже реализованы, их влияние на функционирование системы еще до конца не понято. В заключение отметнм, что возиикновенне новой, сетевой технолгии будет оказывать решающее воздействие на современные технические решения и эффективность систем.

18.2. Архитектура распределенных СУБД

Программное обеспечение систем управлення распределеннымн базами данных (СУРБД) обычно имеет многоуровневую архитектуру [13, 262, 303]. Как показано на рнс. 18.1, в такой архитектуре существует пять уровией, которые могут быть подразделены на две основные части. Верхние четыре уровня процессоров — пользовательский, глобальный логический, фрагментный и распределенный уровни представленнй могут быть сгруппированы вместе и названы сетевой СУБД. Нижний уровень — процессор узлового уровня представления может быть назван локальной СУБД. Межузловая связь связывает узлы сетевой СУБД с узлами локальной СУБД.

Каждый из этнх уровней поддерживает различные представления базы данных. Каждый уровень взаимодействует только с непосредственно смежными уровнями представлення. Самым верхним уровнем структуры является интерфейс прнкладиой программы, или нитерфейс процессора запроса.

Каждый уровень представления базы данных необходим для того, чтобы в явном внде представлять определенный аспект логической или физической структуры базы данных. На рнс. 18.2—18.6 приведен пример, нллюстрирующий задание уровней представлення данных. В этом примере база данных

представлена в виде нескольких таблиц, с помощью которых задаются указанные выше уровни представления. Первый уровень, называемый глобальным логическим уровнем представления, соответствует логической структуре всей сетевой базы данных, как она представляется с точки зрения администратора базы данных. Этот уровень подобен концептуальному уровню представления в концепции ANSI [8]. Глобальный логический уровень представления, изображенный на рис. 18.2, задан в виде трех таблиц (отношений), в которых содержатся сведения по

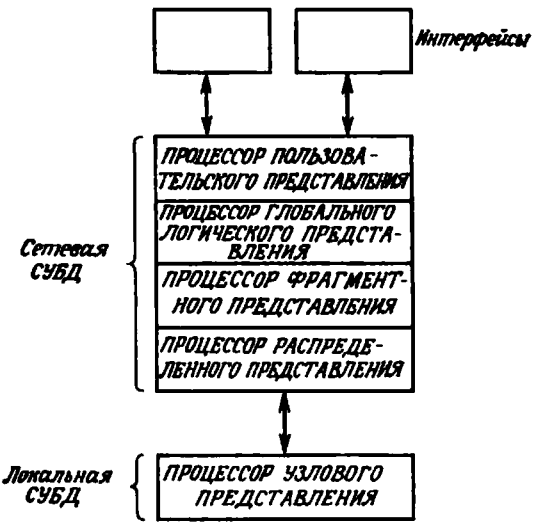


Рис. 18.1. Архитектура СУРБД.

нескольким заводам. Следующий уровень представления называется пользовательским уровнем представления, поскольку он описывает часть базы данных, доступную конкретному пользователю для использования. Эта часть является подмножеством глобального логического представления и подобна внешнему представлению в концепции ANSI [8]. Каждый пользователь может иметь отличное от других пользователей представление, соответствующее его требованиям и требованиям защиты. На рис. 18.3 приведен пример одного пользовательского представления, относительно которого можно сделать несколько замечаний. Первое из них состоит в том, что не все таблицы глобального логического представления доступны конкретному пользователю. В данном примере таблица СЫРЬЕ не содержится в приведенном представлении пользователя, означая тем самым, что пользовательский уровень представления может содержать не все таблицы глобального логического уровня представления. Второе замечание состоит в том, что дальнейшее

совершенствование пользовательского представления может быть достигнуто путем использования лишь некоторого подмножества столбцов таблиц. Столбец ТАРИФ из таблицы СЛУЖАЩИЕ не включен в приведенное пользовательское представление. И наконец, пользовательское представление может включать только подмножество строк таблицы. В приведенном примере в таблице СЛУЖАЩИЕ пользовательского представления содержатся лишь строки, соответствующие заводу, номер которого равен 3.

СЛУЖАЩИЕ

№	ИМЯ	ЗАВОД №	ТАРИФ
100	БИЛЛ	1	6,00
101	ДЖИМ	1	6,00
102	МАЙК	2	10,00
103	ХАУТАН	2	12,00
104	ДОН	3	2,90
105	СТИВ	3	3,00

ЗАВОД

ЗАВ. №	НАЗВАНИЕ
1	АНН-АРБОР
2	ДЕТРОЙТ
3	НЬЮ-ЙОРК

СЫРЬЕ

№	ЭЛЕМЕНТ	КОЛИЧЕСТВО
1	ГЛИНА	500
1	ГИПС	100
2	УГОЛЬ	940
3	ГЛИНА	75

Рис. 18.2. Глобальный логический уровень представления.

Существование третьего и четвертого уровней представления объясняется распределенной природой базы данных и решением использовать управляемую избыточность. Третий уровень представления — фрагментное представление. Используя это представление, АБД определяет несвязанные подмножества базы данных, называемые *логическими фрагментами*, каждый из которых является подмножеством строк в таблице. На рис. 18.4 показаны логические фрагменты базы данных. В рассмотренном примере таблица ЗАВОД разделена на три логических фрагмента согласно конкретным значениям номеров заводов. Географическое расположение экземпляра каждого фрагмента определяется на четвертом уровне представления — представления распределения. В этом представлении разрешается существование нескольких физических копий одного фрагмента. Рис. 18.5 иллюстрирует распределение и дублирование хранимых фрагментов в трехузловой системе. *Хранимые фрагменты* являются физической реализацией логических фрагментов. В примере предполагается, что завод № 1 является головным для всей организации, поэтому сотрудникам соответствующих служб за-

СЛУЖАЩИЕ

№	ИМЯ	ЗАВ. №
104	ДОН	3
105	СТИВ	3

ЗАВОД

ЗАВ. №	НАЗВАНИЕ
1	АНН-АРБОР
2	ДЕТРОЙТ
3	НЬЮ-ЙОРК

Рис. 18.3. Пользовательский уровень представления

СЛУЖАЩИЕ

	№	ИМЯ	ЗАВ. №	ТАРИФ
ФРАГМЕНТ 1	100	БИЛЛ	1	6,00
	101	ДЖИМ	1	6,00
ФРАГМЕНТ 2	102	МАЙК	2	10,00
	103	ХАУТАН	2	12,00
ФРАГМЕНТ 3	104	ДОН	3	2,90
	105	СТИВ	3	3,00

ЗАВОД

	ЗАВ. №	НАЗВАНИЕ
ФРАГМЕНТ 4	1	АНН-АРБОР
	2	ДЕТРОЙТ
	3	НЬЮ-ЙОРК

СЫРЬЕ

	ЗАВ. №	ЭЛЕМЕНТ	КОЛИЧЕСТВО
ФРАГМЕНТ A	1	ГЛИНА	500
	1	ГИПС	100
ФРАГМЕНТ B	2	УГОЛЬ	940
ФРАГМЕНТ C	3	ГЛИНА	75

Рис. 18.4. Фрагментный уровень представления.

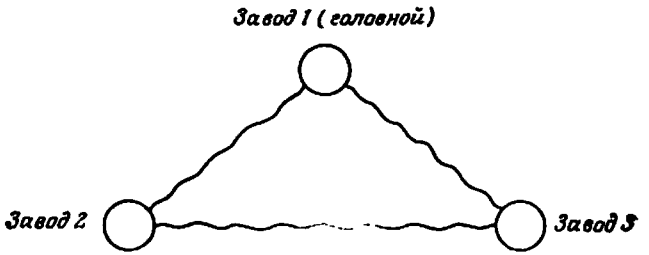


ТАБЛИЦА	ФРАГМЕНТ	РАСПОЛОЖЕНИЕ
СЛУЖАЩИЕ	1 2 3	1 1,2 1,3
ЗАВОД	α	1,2,3
СЫРЬЕ	A B C	1 2 3

Рис. 18.5. Распределенный уровень представления.

СЛУЖАЩИЕ

ФРАГМЕНТ 3

№	ИМЯ	ЗАВ. №	ТАРИФ
104	ДОН	3	2,90
105	СТИВ	3	3,00

ЗАВОД

ФРАГМЕНТ α

ЗАВ №	НАЗВАНИЕ
1	АНН-АРБОР
2	ДЕТРОЙТ
3	НЬЮ-ЙОРК

СЫРЬЕ

ФРАГМЕНТ C

ЗАВ. №	ЭЛЕМЕНТ	КОЛИЧЕСТВО
3	ГЛИНА	25

Рис. 18.6. Локальный уровень представления на заводе 3.

вода № 1 было бы желательно иметь эффективный доступ к тем частям базы даниых, которые описывают другие заводы. Заводы № 2 и 3 взаимодействуют главным образом между собой. По этой причине копии всех трех фрагментов таблицы СЛУЖАЩИЕ имеются на заводе № 1, в то время как на заводах № 2 и 3 имеются копии только соответствующих им фрагментов. Необходимо отметить, что выбор управляемой избыточиости позволяет иметь большую гибкость. Таблица ЗАВОД дублируется полностью, таблица СЫРЬЕ расчленяется, но не дублируется, а таблица СЛУЖАЩИЕ распределена с использованием как расчлеиения, так и дублирования.

Конечное, или локальное, представление есть представление части базы данных, существующей в конкретном узле (отсюда «локальное»). Другими словами, оно описывает базу данных, доступную локальной СУБД. На рис. 18.6 показано локальное представление базы данных узла 3 рис. 18.5. Безусловно, база данных, расположенная в узле, может рассматриваться как с точки зрения логической, так и физической структуры. Локальное представление, как было определено, является логической структурой, а физнческая структура при этом является скрытой. Локальная СУБД в свою очередь имеет иесколько уровней представления данных, однако в данном рассмотренин такне детали не учитываются.

Логическая структура базы данных

Модель данных, соответствующая логической структуре базы данных, является вариантом реляционной модели. Она достаточно мощная для представления как реляционных структур, так и логических структур других моделей данных. В расширенной реляционной модели база данных состоит из нескольких таблиц. Для заданиых миожеств значений $D_1, D_2, \ldots, D_n$ таблица определяется как множество строк, в каждой из которых первый элемент берется из D_1, второй из D_2 ... и n-й элемент из D_n. Множества D_i иазываются доменами. Допустимыми типами данных для этих доменов являются символьные строки, целые и действительиые числа, логические переменные. Каждая строка таблицы должна иметь первичиый ключ, состоящий из значений одного или нескольких столбцов, которые однозначио определяют строку. Все строки таблнцы являются различными (первичиые ключи не дублируются).

Базы данных с древовидной структурой явио определяются в момент определения данных посредством промежуточной таблицы, называемой ассоциацией. Ассоциации есть явно определениые связи между двумя таблицами. Они используются для представления связи типа $1:M$ в древовидных базах данных. В этой связи одна таблица может быть названа «владелец», а другая «члеи». Для строк таблицы-члена, которые связаны со

строками таблицы-владельца, может быть определен один или более критериев упорядочивания.

Как было описано в примере, существуют два типа представлений логической структуры базы данных: глобальное логическое представление и пользовательское представление. Глобальное логическое представление содержит логическую структуру всей распределенной базы данных. Пользовательское представление является подмножеством глобального логического представления, которое администратором базы данных объявляется доступным конкретному пользователю. Это упрощенное представление содержит только те таблицы, столбцы и строки, которые требуются конкретному пользователю. С одним и тем же глобальным логическим представлением может быть связано несколько пользовательских представлений. Для администратора базы данных пользовательское представление является средством управления доступом и безопасности. Для пользователя его представление есть средство упрощения структуры базы данных.

Физическая структура базы данных

Часть базы данных, распределенная в одном узле, называется хранимым фрагментом. Хранимый фрагмент, как реализация логического фрагмента, содержит подмножество строк таблицы. Таблица должна быть расчленена на не пересекающиеся между собой фрагменты. Наименьшим фрагментом таблицы является одиночная строка. Каждая строка фрагмента является полной в том случае, когда она содержит все столбцы, определенные в таблице. Фрагмент может повторяться с целью повышения надежности и производительности. Может существовать любое количество копий отдельного фрагмента, каждая из которых размещается в отдельном узле. При запросе таблицы в первую очередь необходимо определить, какие фрагменты таблицы могут содержать запрашиваемые строки, а затем расположение «лучшей» копии фрагмента или «лучших» копий фрагментов.

Гибкость проектирования обеспечивается наличием возможности сочетать копирование и расчленение базы данных. Определение логических фрагментов и размещения копий хранимых фрагментов позволяет проектировщику удовлетворять предъявляемые к времени отклика и надежности системы требования, управляя при этом дублированием базы данных.

18.3. Проблемы проектирования распределенных баз данных

Множество возможных вариантов построения систем распределенных баз данных отображает разнообразие целей системы и широту возникающих при этом проблем проектирования. Несмотря на то что в этой области существует значительное ко-

личество литературы, большинство проводимых исследований ориентировано на моделирование стратегий распределения данных при строгих допущениях, причем очень мало сообщается о решении существующих проблем реализации.

18.3.1. Стратегии распределения данных

Стратегии распределения данных по узлам сети ЭВМ могут классифицироваться в зависимости от количества узлов, содержащих данные, и наличия дублирования информации. Допустимые стратегии определяются архитектурой системы и программным обеспечением системы управления базой данных. Особенности реализации стратегий распределения данных определяются обычно в процессе проектирования базы данных. Рассмотрим четыре альтернативные стратегии распределения данных:

1. Централизация (единственная копия базы данных, расположенная в одном узле).

2. Расчленение (единственная копия базы данных, непересекающиеся подмножества распределены по различным узлам).

3. Дублирование (несколько копий базы данных, в каждом узле располагается полная копия всех данных).

4. Смешанная (несколько копий подмножеств базы данных, в каждом узле может содержаться произвольный фрагмент базы данных).

Система управления распределенными базами данных, допускающая лишь централизованное распределение, является простейшей, а система, допускающая смешанное распределение данных, — наиболее сложной. Стратегии расчленения и дублирования являются в различной степени более сложными, чем централизованная. Стратегия расчленения предполагает наличие лишь одной копии базы данных, но при этом необходимо знать, какая часть базы данных расположена в каждом узле. Стратегия дублирования предполагает наличие в каждом узле полной копии базы данных, причем все копии должны обслуживаться согласованно для обеспечения их полноты и целостности.

Смешанная стратегия сочетает сложности двух других распределенных стратегий, приобретая при этом гибкость и достоинства обеих стратегий. Для системы управления распределенными базами данных может потребоваться следить за изменением состояний копий каждого подмножества базы данных, а также за размещением каждой копии.

Можно отметить различия между стратегией распределения данных, реализуемой сетевой СУБД, и стратегией, применяемой при конкретной реализации базы данных. Конкретная реализация базы данных может использовать не все возможности сетевой СУБД. Например, централизованная база данных может

быть реализована на основе сетевой СУБД, допускающей три варианта стратегий реализации распределения данных. Другим, более интересным примером является случай, когда сетевая СУБД поддерживает смешанную стратегию, но реализация базы данных использует лишь один ограниченный вариант: расчлененное или дублированное распределение данных. Если реализация базы данных является лишь частным случаем того, что допускает сетевая СУБД, то результатом этого обычно бывает лишь напрасная трата ресурсов как с позиции ненужной сложности программного обеспечения, так и с позиции эффективности обработки. Объем потеряиных ресурсов будет значительно изменяться от одной реализации к другой. Однако наличие более гибкой сетевой СУБД позволяет иметь преимущество в смысле дальнейших модификаций, даже если полные возможности СУБД и не будут использованы немедленно. В остальной части настоящего раздела будем полагать, что четыре стратегии распределения данных описывают возможности классов сетевых СУБД, а не конкретных реализаций баз данных. Читатель легко сможет распространить результаты применительно к частным случаям.

Существуют как преимущества, так и недостатки каждой из четырех стратегий. Далее будут рассмотрены вопросы, касающиеся надежности, хранения данных, времени отклика при выборке и обновления данных, а также различные механизмы управления и сопутствующие им стоимости программного обеспечения и связи. Сейчас же мы рассмотрим преимущества и недостатки стратегий и типичные ситуации, когда каждая из этих стратегий является наиболее подходящей.

Стратегия централизации

Основным преимуществом централизованной базы данных, безусловно, является простота. Все операции осуществляются под контролем единственного узла, все проблемы и действия полностью ясны, по крайней мере по сравнению с распределенной базой данных. Распределенные стратегии должны доказать свое преимущество путем преодоления некоторых недостатков, присущих централизованным системам. Так как в централизованных базах данных все данные располагаются в единственном узле, то наличие вторичной памяти в этом узле ограничивает возможный размер базы данных. Все запросы на выборку и обновление данных должны направляться в центральный узел со всеми сопутствующими затратами на стоимость связи и временну́ю задержку. Если предположить, что в узле находится лишь одна ЭВМ, то это приведет к ограничениям на параллельную обработку, следовательно, скорость обработки будет ограничена быстродействием процессора. Центральный узел может стать узким местом всей системы, хотя вся остальная сеть мо-

жет функционировать нормально. Одна из главных проблем рассматриваемой стратегии — ограниченные доступность и надежность. База данных становится недоступной из удаленного узла при появлении ошибки в системе связи и полностью выходит из строя при выходе из строя центрального узла. Любая из трех других стратегий распределения данных преодолевает некоторые из этих недостатков, но ценой определенных затрат.

Стратегия расчленения

При распределении данных на основе стратегии расчленения база данных распределяется по многим узлам сети, однако существование копий отдельных частей базы данных не допускается. Как было описано ранее, база данных разделяется на непересекающиеся подмножества, называемые логическими фрагментами, и каждый логический фрагмент размещается в отдельном узле. Такой метод имеет ряд преимуществ перед стратегией централизации. Объем базы данных теперь ограничивается уже объемом вторичной памяти, имеющейся во всей сети, а не в единственном узле. Так как запросы на выборку и обновление направляются в узлы, где расположены запрашиваемые данные, стоимость связи может быть снижена за счет того, что большая часть запросов к базе данных будет осуществляться к своим локальным частям. С другой стороны, запрос может потребовать доступа ко всем узлам сети, и это приведет к большей стоимости связи и к большему времени задержки, чем в случае централизованной базы данных. Время отклика может быть меньше по сравнению с централизованной базой данных, если сетевая СУБД использует возможный параллелизм. Система становится менее чувствительной к узким местам при передаче данных в отдельном узле, поскольку нагрузка по передаче данных будет более равномерно распределена по сети в целом. Доступность и надежность базы данных могут быть повышены по сравнению с централизованным подходом. Если часть или даже все средства связи выйдут из строя или если выйдет из строя один или несколько узлов, то система все же может оказаться частично работоспособной. Доступными могут оставаться части базы данных в отдельных узлах или в узлах, еще остающихся связанными в сети. Ключевым фактором, влияющим на надежность и доступность базы данных, является так называемая локализация ссылок (т. е. расположение запрашиваемых данных, исходя из удовлетворения запросов пользователей). Если база данных распределена по сети таким образом, что данные, расположенные в узле, запрашиваются почти исключительно пользователями этого узла, то говорят, что существует высокая степень локализации ссылок. Если же подобное расчленение базы данных невозможно, то говорят, что степень локализации ссылок мала. Следствием степени локализации

ссы... ...является и бóльшая доступность базы данных. Например, если запрос пользователя может быть удовлетворен с помощью локально хранимых данных, то ошибки в других узлах или ошибки в устройствах связи не окажут влияния на этот запрос. Если степень локализации ссылок мала или запрос пользователя является сложным, то могут потребоваться данные, хранимые в различных узлах. И если при этом будет недоступен хотя бы один узел, то запрос не будет удовлетворен. В этой ситуации доступность базы данных может быть хуже, чем при стратегии централизации. Вероятность того, что по меньшей мере один узел сети будет недоступен, очевидно, выше вероятности недоступности единственного узла. Следовательно, база данных может быть доступна меньшую часть времени, чем при использовании стратегии централизации.

В заключение отметим, что стратегия расчленения наиболее подходит для случая, когда либо локальная вторичная память ограничена по сравнению с объемом базы данных (например, при использовании мини-ЭВМ), либо недостаточна надежность централизованной базы данных, либо когда должна быть повышена эффективность функционирования. Эффективность функционирования может быть обычно повышена, если запросы к базе данных будут обладать высокой степенью локализации ссылок. При отсутствии локализации ссылок эффективность может упасть довольно быстро из-за большого объема передачи данных.

Стратегия дублирования

При распределении данных с использованием стратегии дублирования в каждом узле сети размещается полная копия базы данных (т. е. в каждом узле, в котором имеются данные, имеется вся база данных). Сетевая система управления базой данных должна согласовывать состояние многих копий данных, однако здесь отсутствует проблема определения, какую конкретно часть базы данных содержит каждый узел, как это было в стратегии расчленения. Сравнивать две указанные стратегии, исходя из их сложности, нельзя, поскольку они решают различные задачи. Основное преимущество стратегии дублирования относится к областям надежности, доступности и эффективности выборки. Уровень надежности, обеспечиваемый этой стратегией, самый наивысший из возможных, но при явных затратах используемой вторичной памяти. Объем базы данных также ограничен объемом вторичной памяти в каждом узле. Значительная часть обработки может быть проведена локально, но с целью согласования множественных копий базы каким-то образом должна осуществляться их синхронизация. Конкретные способы проведения согласования меняются от системы к системе в широких пределах, а уровень определенных накладных расходов будет

зависеть от имеющегося уровня постоянства данных. Эта стратегия не столь легко реализует параллельную обработку одного запроса, как стратегия расчленения, что связано с необходимостью согласования копий и сложностью управления, однако каждый узел может работать асинхроино. Возможно получение очень быстрых ответов на запросы пользователей, особенно в снтуациях, когда нет необходимости в межузловой связи для согласования копий баз данных, например при поиске в базе данных. Надежность базы данных является высокой не только из-за доступности данных при нарушении работоспособности узла или части сети, но также и из-за простоты замены разрушениой копии базы данных или возможности продолжения обработки, несмотря иа вышедший из строя узел. Еще одним преимуществом стратегии дублирования является простота операций восстановления базы данных. Согласованная копия базы данных может быть получена из любого рабочего узла. Если часть сети недоступна по какой-либо причине, то, возможно, придется ограничить выполнение некоторых операций (например, обновлеиия) для того, чтобы поддержать согласованность базы данных. Иными словами, если разрешены две операции обновления в двух разных узлах и не может быть осуществлена синхронизация, то при возобиовлении нормальиого функционирования сети возможно нарушение согласованности базы данных. В заключение отметим, что стратегия распределения данных с полиой избыточностью наиболее подходит для тех ситуаций, когда фактор надежности является критическим, база данных — иебольшой, а иитенсивность обновления может быть невысокой (например, базы данных с интеисивными запросамн справочного типа). Это очевидный пример того, как проблема реализации, такая, как выбор способа синхронизации, оказывает большое влияние на эффективность распределения базы данных. Определение количественных характеристик этой зависимости является в настоящее время важной темой исследований.

Смешанная стратегия

Смешанная стратегия распределения данных объединяет подходы, связаниые с расчленением и дублированием данных с целью приобретения преимуществ, которыми они обладают. Но к сожалению, эта стратегия приобретает сложиости каждого из объединяемых подходов. Эта стратегия подразделяет базу данных на логические фрагменты, как это сделано в стратегии расчленеиия, но в дополнение к этому дает возможность иметь произвольное количество физических копий каждого фрагмента, иазываемых хранимыми фрагмеитами. Эта стратегия является общей в том, что любая часть базы данных может быть дублирована произвольное количество раз и в каждом узле может

содержаться желаемая часть базы данных. Недостатком сетевой СУБД является необходимость хранить информацию о том, где иаходятся данные в сети, и согласовывать произвольное количество хранимых фрагментов, связанных с каждым логическим фрагментом. Обработка и оптимизация запросов являются при использовании смешанной стратегии нетривиальными задачами.

Ключевым преимуществом смешанной стратегии является гибкость. Например, можно установить компромисс между объемом памяти, используемой в целом и в каждом отдельном узле, обеспечиваемым уровнем надежности и различными мерами эффективности. К примеру, архивные данные необходнмо запоминать только в одном месте, иапротив, более критические данные могут быть дублированы, еслн требуется достичь требуемого уровня надежности. При дублировании логического фрагмента (запоминаиии более одного фрагмента) стоимость согласования, включающая стоимость связи, возрастает, однако большее количество данных становится локальио доступным, что ведет к снижению количества пересылок и стоимости связи при выполнении запросов. Происходит это из-за того, что степень локализацни ссылок может возрастать за счет дублирования. Система допускает относительно простую реализацию параллельной обработки данных, что делает возможным получение малого времени отклика. Узкие места, возникающие при связи, во многих случаях могут быть устраиены. В связи с тем, что в каждом узле сети может находиться произвольное подмножество базы данных, можно получить практически любую степень надежиости при возникновении ошибок в узлах и устройствах связи. Функционирование может продолжаться, хотя фуикциональные возможности будут ограниченны. Как и для случая стратегии дублирования, выполнение операций обиовления может быть ограничено с тем, чтобы автоматически поддерживалась согласованность базы данных при возобновлении функционирования сети в полном объеме.

Хотя распределенная СУБД, реализующая смешанную стратегию, и является предельно гибкой, остается проблема взаимозависимости различных факторов, влияющих на производительность системы, ее надежность и требования к памяти. Изолировать один фактор от другого весьма трудно. Механизмы, используемые распределенными СУБД, оказывают большое влияиие на производительность н другие параметры. Предположеиия, касающиеся использования базы данных, оказывают большое влиянне на конечное распределение даиных. Смешанная стратегия является приемлемой тогда, когда ии одиа из более простых стратегий не является удовлетворительной. Это случается достаточно часто. В качестве примера рассмотрим большую-

базу данных, в которой требования высокой надежности предъявляются лишь к определеиным ее частям. Каждый узел может обращаться к некоторым частям базы данных часто, а к некоторым — редко (т. е. имеется разнообразне локализации ссылок). В этом случае стратегия расчленения не может обеспечить достаточной надежности, а стратегия дублирования может быть иевыгодиой или иеприемлемой из-за требований на вторичную память.

18.3.2. Распределение сетевого справочника данных

Справочник сетевой базы данных может быть распределен по узлам сети согласно любой из четырех стратегий, применяющихся для самой базы данных: централизованной, расчлененной, дублированной или смешанной. Какая стратегия является иаиболее подходящей, определяется обычно сочетанием стратегий распределения данных, используемым сетевой СУБД, и допущениями относительно требований (пользователей и приложений) к системам баз данных. Это означает, что обычно основная стратегия будет предоставляться сетевой СУБД и, следовательно, ее воздействия должны быть оценены до выбора именно сетевой СУБД. Многое из сказаиного о распределении данных может быть перенесено и на распределеиие справочника. Например, наличие нескольких копий справочника приводит к необходимости согласоваиия при его модификации.

Хотя теоретически можно пользоваться любой стратегией распределения справочника независимо от стратегии распределения данных, на практике используется лишь несколько сочетаний. Использование стратегии централизации при распределении справочника совместно со смешанной стратегией распределения данных обычно приводит к потерям многих преимуществ, присущих смешанной стратегии распределения даниых. Необходимо отметить, что характеристики запросов к сетевому справочиику, такие, как отношение количества обновлений к количеству выборок (изменчивость), и ряд других важных при выборе стратегии распределения характеристик могут сильно различаться для сетевого справочника и собственно для базы даниых, управляемой этим справочииком. Запросы иа выборку из справочника возникают при обработке любого запроса пользователя, что служит доводом в пользу расчлененного и дублированного справочника. При использовании смешаниой стратегии требуется иметь справочник справочника, чтобы обеспечить возможность сетевой СУБД найти необходимую часть справочиика. В системе SDD-1 [262] такой справочник называется указателем справочника и дублируется в каждом узле. В общем сетевой справочиик должен рассматриваться как часть сетевой

СУБД, а не как отдельный объект. Применяемая стратегия распределения справочника должна рассматриваться в свете стратегии распределения данных и целей функционирования базы данных.

Проектирование сетевого справочника может потребоваться уже при реализации базы данных, хотя при этом возможно возникновение большего количества ограничений по сравнению с количеством ограничений на этапе проектирования базы данных. Некоторые СУБД могут потребовать размещения сетевого справочника аналогично размещению данных. Другими словами, сетевой справочник может быть разделен на логические фрагменты, а затем логические фрагменты могут быть размещены по узлам сети, возможно с использованием избыточности. Если справочник обрабатывается аналогично базе данных, то такие механизмы поддержания базы данных, как управление параллелизмом и безопасность, также применяются и для поддержания справочника. При этом необходимо еще раз оценить такие факторы, как надежность и характеристики производительности.

Размещение справочников в распределенной системе исследовалось в Калифорнийском университете Лос-Анджелеса [75]. Это исследование было направлено на анализ схем размещения справочника в различных ситуациях.

Авторы работы [304] рассматривали функцию справочника данных как связь между именами объектов и их размещением в системе распределенных данных. К решению этой проблемы существуют два подхода: использование полного имени доступа или функции каталогизации. При использовании полного имени доступа связь осуществляется путем включения поля, указывающего размещение распределенных данных непосредственно в имени каждого файла, что дает явное указание размещения. Функция каталогизации обеспечивает для распределенных разделяемых систем данных поддержание внутренних связей имя-расположение таким образом, что пользователю нет надобности знать расположение файлов.

18.3.3. Однородные и неоднородные системы баз данных

Распределенные системы часто строятся путем «интеграции» разнородных аппаратных и программных средств. Следовательно, должен быть сделан выбор между однородной и неоднородной вычислительными системами.

Во-первых, необходимо определить связь сетевой и локальных СУБД. В случае однородных СУБД нет проблем ни с моделью данных, ни с языком запросов, ни с другими средствами, которые должны быть предоставлены пользователям. Все это

совпадает с тем, что поддерживается локальной СУБД. Однако все же существует ряд вопросов, которые необходимо решить, например, должны ли абсолютно все пользователи взаимодействовать с сетевой СУБД или же те пользователи, которым нужны данные, хранящиеся в локальном узле, должны взаимодействовать непосредственно с локальной СУБД. Так как желательно, чтобы возможности, предоставляемые пользователям сетевой СУБД, были эквивалентны или по меньшей мере весьма сходны, вопрос касается главным образом архитектуры программного обеспечения и преимуществ общего программиого интерфейса по сравнеиию с затратами на его поддержание. Если же распределенная база данных поддерживается неоднороднымн СУБД, то вопросы усложняются. Использование неоднородных СУБД обычио является следствием формирования распределенной базы данных нз ряда существовавших ранее автономных баз даиных. Стоящая перед разработчиками цель — достнчь прозрачностн доступа, что представляет собой нечто большее, чем простое обеспечение доступа к удаленным СУБД и нх базам данных. Прозрачность означает, что пользователю либо неизвестно расположение данных, либо такне сведения для работы с базой данных ему не требуются. В системах с неоднородными СУБД такое возможно лишь в том случае, если локальная СУБД, управляющая данными, также «прозрачна», т. е. пользователь ие обязан знать, какая локальная СУБД обслуживает его запрос. Это можно реализовать двумя принциаиально отличными путями. Один путь состоит в том, чтобы дать пользователю возможность использовать в каждом узле пользовательский интерфейс, предоставляемый локальиой СУБД. При этом имеющаяся схема должна быть расширена с целью включения даиых, имеющихся в других узлах, ио пользователи должны работать так, как если бы удалеиные данные были добавлены в данную локальную базу данных. Проблема заключается в том, чтобы программное обеспечение сетевой СУБД в каждом узле позволяло обращаться к данным любого другого узла независимо от модели данных и других факторов. При увеличении количества разнородных локальных СУБД количество типов преобразования схем быстро растет. Если имеется n локальных СУБД, то необходимо $n(n-1)$ типов преобразований схем, что никак нельзя назвать эффективным решеннем.

Использование единого для всей сети стандартного пользовательского интерфейса и стандартных внутренних форм представления запроса облегчает решение проблемы преобразования схем. При таком подходе все пользователи или по меньшей мере те из них, кому нужны данные из удаленных узлов, используют общий интерфейс, который может быть отличен от

использующегося в любой локальной СУБД. Должна существовать одна схема сетевой базы данных, а не различные схемы в каждом узле, которые зависят от конкретной локальной СУБД. Каждому типу локальиой СУБД должен соответствовать свой тип преобразования схемы в общую форму, при этом используется лишь n типов преобразованнй схем для n различных СУБД по сравнению с n $(n-1)$, как было ранее. Недостатком является то, что пользователи должны изучать новую систему, сетевую СУБД. Итак, все еще остается вопрос: разрешать ли пользователям, нуждающимся только в локальных данных, пользоваться непосредственно локальиой СУБД или заставлять их использовать сетевой стандарт? Возможно, последняя альтернатива не очень хороша с точки зрения управлення или эффективности, но она может дать много технологических преимуществ, такнх, как управление параллелизмом.

Еще одна сложность состоит в том, что современное состояние исследований касается лишь половины проблемы, а именно выборки данных. Реализацня запросов на обновление данных в неоднородных СУБД является нерешенной задачей, по крайней мере в общем внде.

В большинстве исследований в этой области предполагалось, что преобразование данных будет проводиться в статическом состоянии, т. е. база данных является недоступной пользователям в периоды, когда она преобразуется из одной формы представления в другую. Требования производительиости, возникающие при дииамическом преобразовании данных, затрудняют применение хорошо работающнх в статике схем. В заключение отметим, что технические проблемы, возникающие прн преобразовании данных, тем труднее, чем неодиороднее система баз данных.

18.4. Основы проектированиия распределенной базы даиных

Поэтапная методология проектирования цептрализованных баз данных была предложена и развита в гл. 1. В настоящем разделе обсуждается возможное расшнрение методологии с целью охватить также и распределенные базы даиных [66, 230]. В распределенных системах баз данных логически целостная база данных может быть фрагментирована и широко распределена по сети с целью улучшения производительиости системы. Фрагментацня и распределение базы данных без вниматсльиого централизованного планирования часто приводят к беспорядку и иесогласованности при использованин базы данных. Предлагаемая процедура поэтапного проектирования распределенной базы данных учитывает этот важный факт.

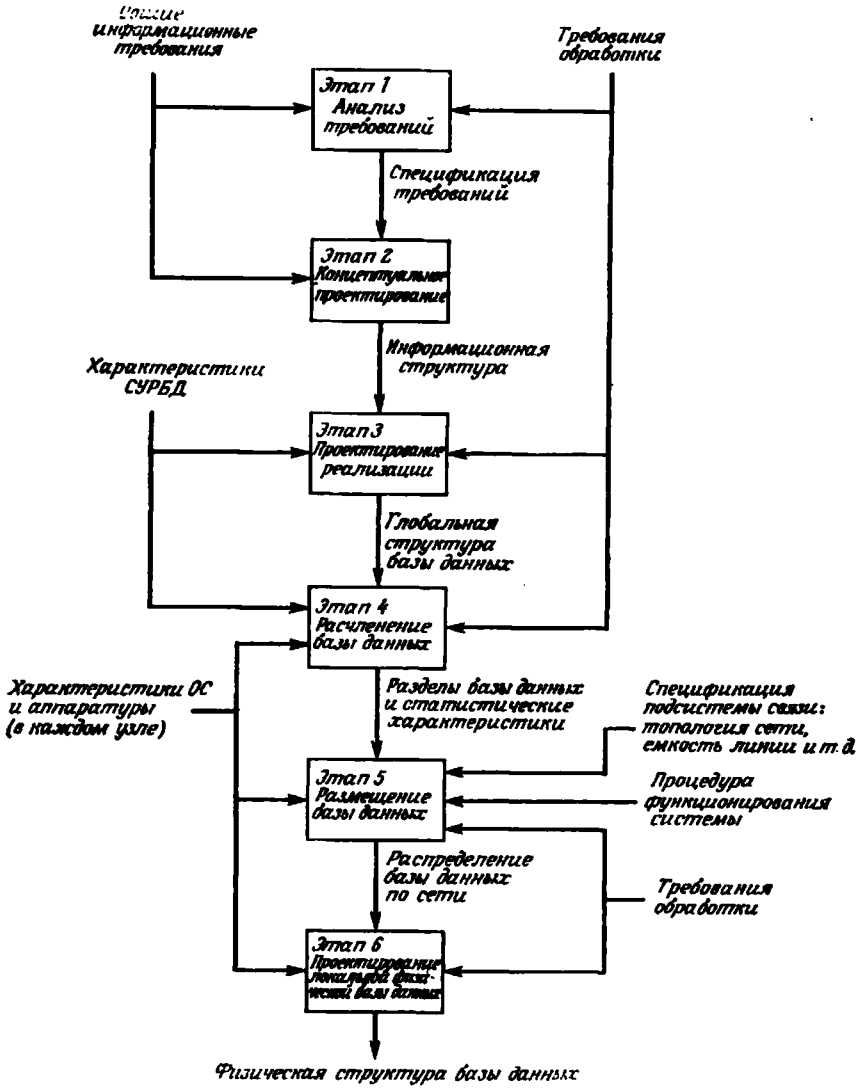


Рис. 18.7. Этапы проектирования распределенной базы данных.

Основные этапы последовательности проектирования распределенной базы данных показаны на рис. 18.7. Полагаем, что рассматриваемая распределенная СУБД (СУРБД) позволяет определить структуру всей базы данных, расчленение и размещение базы данных, а также автоматически обрабатывать при-

кладные запросы независимо от того, как распределены хранимые фрагменты. Пример такого типа СУРБД предложен в [108]. СУРБД с такими возможностями обычно используют одинаковые модели данных, описывающие каждую локальную базу данных, входящую в систему, и поддерживают некоторым образом справочник о расчленении и размещении базы данных. С учетом этого факта, после выполнения проектировщиком этапов 1—5, приложения получают возможность использовать преимущества однородности, повышается согласованность баз данных, что эквивалентно использованию общесистемного стандарта, принятого в организации, являющейся владельцем системы. Этап 6 (физическое проектирование) может быть выполнен в каждом узле системы автономно с тем, чтобы учесть особенности локальных ЭВМ и таким образом улучшить результаты проектирования.

Отметим, что этапы 1, 2, 3 и 6 подобны этапам 1—4 при проектировании централизованной базы данных и описаны в гл. 2, поэтому в следующих разделах мы будем рассматривать только этапы 4 и 5.

18.4.1. Расчленение базы данных

Этап расчленения базы данных связан с расчленением глобальной базы данных и синтезом различных приложений на основе модели. Как показано на рис. 18.8, существуют три класса выходных данных этапа расчленения: 1) совокупность расчлененных частей базы данных (разделов), 2) размер каждого раздела, 3) модели и частоты использования приложений. В следующих параграфах мы опишем эти классы выходных данных и коротко обсудим вопросы, связанные с ними.

Совокупность расчлененных частей базы данных (разделов) $\{F_1, \ldots, F_n\}$. На этом этапе проектирования исходная глобальная база данных расчленяется на множество подфайлов $\{F_1, \ldots, F_n\}$. Требуется, чтобы расчлененные подфайлы содержали в точности все сведения, имевшиеся в глобальной базе данных. Помимо требования о сохранении информации часто требуется совместимость ограничений на разделы базы данных. Вообще говоря, совместимость структуры базы данных дает возможность проектировать для всех структурно допустимых разделов некие составляющие блоки (или агрегаты). Составляющие блоки могут быть и столь малыми, как кортежи (экземпляры записей) в реляционной модели данных, и столь большими, как связанные компоненты (посредством связи владелец-член) в сетевой модели данных. Существуют два момента, которые накладывают ограничения на процесс формирования разделов из составляющих блоков. Это допустимый размер и про-

изводительность. При оценке производительности должны учитываться также два основных фактора: время отклика и надежность системы. Необходимо объединять в подфайл такие часто используемые совместно записи, чтобы он (будучи размещен с учетом частоты использования) улучшал характеристики отклика системы. Необходимо пытаться получить требуемый уровень надежности, используя по возможности меньшую кратность дублирования.

Размер раздела F_i. Каждый подфайл в расчлененной базе данных должен выбираться как неделимая единица размещения данных. Более того, если каждая ЭВМ в системе имеет

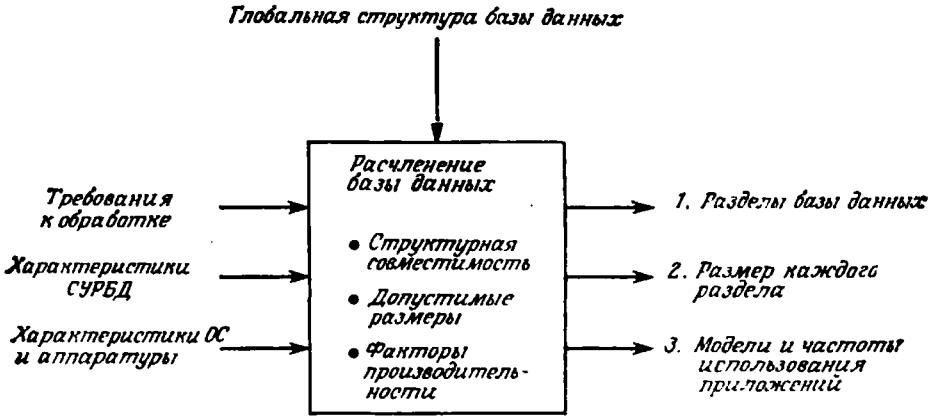


Рис. 18.8. Этап расчленения распределенной базы данных (этап 4).

фиксированный объем памяти, это будет являться определенным ограничением на класс допустимых расчленений. Простой пример подобного ограничения — размер подфайла не может превышать максимальный объем памяти, имеющейся в узле.

Модели и частоты использования приложений. На этом этапе проводится анализ того, как приложения базы данных используют возможные разделы базы данных. Связь между разделом базы данных и приложениями характеризуется идентификатором типа приложения, идентификатором узла сети, создающего приложение (т. е. транзакцию), частотой использования приложения и моделью приложения.

Модели приложений могут быть классифицированы следующим образом:

1. Приложения, использующие единственный файл.
2. Приложения, использующие несколько файлов:

а) приложения, допускающие независимую параллельную обработку;

б) приложения, допускающие синхронизированную обработку.

Стоимость обработки и побочные эффекты синхронизации вместе с расчленением и размещением разделов оказывают важное влияние на производительность базы данных. Понимание этой взаимосвязи является определяющим в процессе проектирования распределенных баз данных.

18.4.2. Размещение базы данных

Поэтапная методика размещения отличается от классического подхода по меньшей мере в двух аспектах.

1. Расчленение базы данных является неотъемлемой частью решения задачи размещения. При классическом методе решения считается, что расчленение базы данных задано. На этапе расчленения базы данных мы используем столько ограничений, сколько нужно, чтобы сузить класс допустимых расчленений. Тем не менее в общем случае на этапе 4 выбор единственного расчленения невозможен. Для того чтобы получить лучшую структуру, этап 5 повторяется для каждого возможного варианта расчленения, полученного на шаге 4. Этот процесс длится до тех пор, пока не будет получен удовлетворительный результат или не будут исчерпаны все допустимые расчленения.

2. При классическом подходе предполагается, что приложения обращаются лишь к единственному файлу. В отличие от этого предположение о транзакциях, являющихся результатом этапа 4, должно учитывать приложения, использующие несколько файлов. Для такого обобщения требуется новая модель.

Размещение распределенной базы данных является многовариантной задачей. Количество возможных вариантов реализации расчлененной или смешанной базы данных огромно. Для того чтобы выбрать наиболее подходящую стратегию распределения данных, необходимо еще до выбора СУБД провести оценку пользовательских и системных требований, если, конечно, это возможно. Могут быть также определены допущения, упрощающие сетевую систему управления базой данных, например, что запросы на обновление исходят только из одного узла или допустимость временного рассогласования базы данных. В рамках существующих четырех стратегий распределения данных допущения такого рода будут иметь большое влияние на выбор конкретной системы управления сетевой базы данных, которая в свою очередь оказывает влияние на фазу проектирования распределенной базы данных. При этом необходимо иметь информацию о данных, узлах сети, приложениях, их связях, а также требования, предъявляемые ко времени отклика и надежности, и ограничения, накладываемые аппаратными и программными средствами сети. Может оказаться необходимым выполнять при проектировании итерации до достижения прием-

лемого компромисса. После того как выбраны стратегия расчленения и конкретная система управления базой данных, встает задача квазиоптимального или по крайней мере рационального размещения данных по узлам сети. На этой стадии предполагается, что логическая структура базы данных, или схема, фиксирована.

Размещение данных по узлам сети является сравнительно простой задачей для двух из четырех стратегий размещения данных. При использовании централизованной стратегии встает только один вопрос: в каком узле должна быть размещена центральная база данных? В большинстве случаев даже это является очевидным, если учитывать сетевую архитектуру и наличие вторичной памяти. При использовании дублированной стратегии для каждого узла существуют лишь два варианта: размещать или не размещать полную копию базы данных. Некая комбинация из требований пользователей, перечня альтернатив и здравого смысла может решить много проблем или по меньшей мере уменьшить количество альтернатив настолько, чтобы каждую из них в дальнейшем можно было проверить.

Более трудной является задача размещения данных по узлам сети при стратегии расчленения и гораздо более трудной для смешанной стратегии. В случае стратегии расчленения следует решить следующие вопросы: 1) как расчленить базу данных на логические фрагменты (этап 4 на рис. 18.7) и 2) как разместить каждый логический фрагмент в конкретном узле (т. е. один хранимый фрагмент (этап 5 на рис. 18.7)). Если принимается смешанная стратегия, то решение становится более сложным, так как каждый фрагмент может быть размещен в любом количестве узлов (несколько хранимых фрагментов). Это эквивалентно принятию решения, какая часть базы данных должна быть расположена в каждом узле. Количество перестановок логических и хранимых фрагментов растет весьма быстро. Это является одной из причин того, что целью проектирования часто является рациональная, а не оптимальная схема размещения. Ручная оценка вариантов структур распределенных баз данных весьма сложна, много проще автоматическое решение. Дополнительные сложности, такие, как проектирование сетевого справочника данных и его влияние на производительность, совершенно не учитываются в этом разделе, чтобы не затенять вопросов проектирования собственно базы данных, однако они должны учитываться в процессе реализации базы данных.

По вопросу размещения базы данных существует ряд исследований. Большинство работ рассматривают совокупность файлов, которая не обязательно является структурированной в логическую схему. Термин «файл», который часто используется

в литературе, может также обозначать логический фрагмент. Термин «копия» является эквивалентным хранимому фрагменту.

Одной из первых значительных работ в области физического распределения данных является работа [74], в которой рассматривается задача оптимального размещения файлов по узлам сети, решенная методом линейного программирования. Решение было получено в булевых переменных при следующих допущениях: 1) количество копий файлов известно, 2) запросы существуют по всем файлам, 3) модели запросов известны, 4) поток запросов пуассоновский. Хорошо известно, что такое решение возможно лишь для задач небольшой размерности, так как количество переменных и ограничений быстро возрастает при возрастании количества узлов сети. В работе [336] более широко рассмотрена задача проектирования системы связи ЭВМ, но использован подход, предложенный в [74] для определения оптимального размещения файлов. В работах [60, 61,62] часть ограничений, принятых в модели [74], снята и показано, что отношение количества запросов на обновленне к количеству запросов на выборку определяет верхнюю границу числа копий файлов в сети. В [210] разработан эвристический алгоритм для совместного решения задачи размещения файлов и распределения емкости каналов в сети с фиксированной топологией. В работе [183] исследована другая сетевая топология, обеспечивающая более высокую надежность при меньшей стоимости. При этом использована эвристическая процедура, минимизирующая полную стоимость хранения базы данных и использования каналов связи при заданных ограничениях на надежность сети, доступность файлов и время задержки в каналах связи.

Распределение данных в сети не является независимым от программ, работающих с этими данными. Авторы работы [232] распространили стратегию распределения данных на случай, учитывающий влияние программ. Была сформулирована модель, учитывающая эффект взаимодействия программ и данных и их оптимальное размещение в сети. Задача размещения была разделена на три уровня. Предполагалось, что на первом уровне типовые запросы известны и носят статический характер. Задача оптимального размещения файлов была решена методом линейного программирования с использованием булевых переменных. На втором уровне к типовым запросам предъявлены менее жесткие требования и был сформулирован подход к решению задачи методом динамического программирования. Третий и последний уровень относится к ситуации, когда типовые запросы первоначально неизвестны. Был разработан статистический метод для определения этих запросов, который включен в модель размещения файлов. Такой подход полезен при адаптивном перераспределении файлов в сети.

18.5. Дифференциальные файлы

Дифференциальный файл в базе данных аналогичен списку опечаток в книге. Вместо того чтобы печатать новое издание книги, всякий раз, когда требуется внести изменения в текст, издатель описывает исправление и указывает номер страницы и строки, куда следует поместить исправление, а затем вносит это описание в список исправлений, который помещает в каждую книгу. Издание, имеющее список исправлений, продается по более низкой цене. При использовании откорректированного варианта книги читатель, прежде чем начать чтение основного текста, должен просмотреть список опечаток. Увеличение времени доступа компенсируется таким образом снижением стоимости поддержания. Если количество изменений в тексте растет, то также растет список опечаток. Он может достичь такого размера, при котором становятся оправданными затраты на реорганизацию. При этом все изменения должны быть непосредственно включены в книгу, тем самым образуя новое издание.

Обновление больших баз данных ставит аналогичную задачу. Как и в случае с книгой, обычно наиболее простой и наименее дорогой путь состоит в накоплении изменений за период времени и в последующем перенесении их всех вместе во вновь созданную редакцию (поколение) базы данных. Гораздо более дорогой, основываясь на стоимости хранения, времени поддержания и суммарной сложности системы, является модификация базы данных всякий раз, когда выполняется транзакция обновления. В качестве компромиссного решения может быть использован дифференциальный файл, аналогичный списку опечаток, который осуществляет сбор и идентификацию будущих изменений записей. Предварительное обращение к дифференциальному файлу, являющееся первым шагом при выполнении операции выборки, — эффективное средство доступа к самому последнему состоянию базы данных. Таким образом, путем увеличения времени доступа затраты на обновление базы данных могут быть уменьшены. Когда дифференциальный файл достигает достаточно больших размеров, проводится реорганизация, в процессе которой все изменения, хранящиеся в дифференциальном файле, вносятся в базу данных, образуя тем самым ее новое поколение. А опустевший дифференциальный файл может снова начать накапливать изменения.

Концепция дифференциального файла для различных приложений и различных вариантов появлялась много раз. Применительно к распределенным системам баз данных она рассмотрена в работе [279]. Здесь будут описаны три способа организации дифференциального файла, описанные в литературе. Впервые в работе [328] обрисована дифференциальная структура для

ленточных систем, которая была разработана для того, чтобы исключить запись неизменяемых данных при последовательной пакетной обработке обновлений. Файл данных разбивается на одинаково упорядоченные подфайлы: большая совокупность записей, для которых разрешено только чтение, хранится на одной ленте, в то время как небольшая совокупность модифицируемых записей содержится на отдельной «ленте изменений». Для обновления файла данных обе ленты сливаются, при этом получается новая измененная лента. Неизменяемые записи с ленты, доступной только для чтения, никогда не записываются. В данной работе впервые рекомендовано проводить реорганизацию файла данных после модификации половины всех записей.

В работе [264] также предложено использовать принцип дифференциального файла для обработки изменений в файловой системе с прямым доступом. Система обращается к записям через уникальный идентификатор, и любая ссылка на данные проходит через индекс базы данных, который адресует все записи. Созданный однажды, главный файл данных никогда не модифицируется. Новые записи базы данных обращаются к индексу, но запоминаются в отдельной области переполнения. Все модификации записей данных трактуются как записи добавления. При этом создается новая копия записи и обновляется индекс для указания на область переполнения. Старая запись не уничтожается, а, наоборот, поддерживается как предшествующий образ, на который указывает новая запись, и свидетельствует о том, что новая запись в результате обновления увеличивается в размерах без нарушения размещения соседних записей.

Система с подобной структурой описана в работе [254]. Она была разработана с целью обеспечения восстановления базы данных при внезапном отключении электрического питания. И в этом случае все обращения осуществляются через системный индекс и все модификации выделяются в файл изменений, называемый MODFILE. Каждая измененная запись указывает на свой предыдущий образ. При отключении электрического питания информация из журнала транзакций совместно с информацией из файла MODFILE используется для удаления незавершенных обновлений.

Всякий раз, когда запись обновляется одним из описанных в [264, 254] способов, механизм поиска записи (связанный ранее лишь с главным файлом) модифицируется таким образом, чтобы указывать на новую копию записи, которая, что особенно важно, запоминается в дифференциальном файле. Как показано на рис. 18.9, *а*, доступ к текущему значению идентифицированной записи независимо от того, находится ли она в главном или дифференциальном файле, осуществляется при помощи общего механизма поиска — системного индекса.

Обобщение такой стратегии доступа показано на рис. 18.9, *б*. При наличии у каждой записи базы данных своего идентификатора поиск вначале всегда проводится в дифференциальном файле. Если запись там не найдена, выборка осуществляется из главного файла. Подразумевается, что каждый файл может иметь свой собственный механизм поиска. При этом индекс главного файла является неизменяемым и может быть быстро восстановлен в случае сбоя по копии. Изменяемая часть индекса перенесена в меньший и более легко восстанавливаемый индекс дифференциального файла.

Чтобы защитить главный файл и его механизм поиска от изменений, приходится идти на накладные расходы в виде поиска в дифференциальном файле при каждом обращении к записи. В том случае, когда оба файла могут быть размещены на отдельных устройствах и доступ к ним осуществляется через независимые каналы, поиск проводится параллельно и пользователи системы не чувствуют дополнительной задержки. Если же параллельный поиск невозможен, приходится ожидать увеличения среднего времени выборки на время произвольного доступа к вторичной памяти (используется стратегия доступа к дифференциальному файлу, предложенная в работах [279, 280]). При этом дополнительное время доступа может оказаться сравнительно большим, что значительно снижает производительность системы.

В тех случаях, когда значительное увеличение времени доступа недопустимо, предлагается применять модифицированную стратегию поиска, использующую предпоисковый фильтрующий алгоритм для уменьшения количества ненужных обращений к дифференциальному файлу (рис. 18.9, *в*). Фильтрующая схема для определения появления редких событий, описанная в [36], практически полностью исключает безуспешные попытки поиска. В предложенном методе дифференциальному файлу ставится в соответствие двоичный вектор B длины M, размещаемый в основной памяти, и некоторое количество X функций хеширования, которые отображают идентификаторы записей в адреса разрядов вектора. Когда в начальный момент времени дифференциальный файл пуст, все разряды находятся в состоянии 0. Когда запись запоминается в дифференциальном файле, ее идентификатор преобразуется каждой из X функций хеширования и разряд, адрес которого получен в результате каждого преобразования, устанавливается в 1.

Запросы к базе данных обрабатываются теперь следующим образом. Идентификатор записи, к которой производится обращение, преобразуется указанным выше способом, и ко всем X значениям адресованных разрядов применяется логическая операция И. В результате получается двоичное значение 0 или 1.

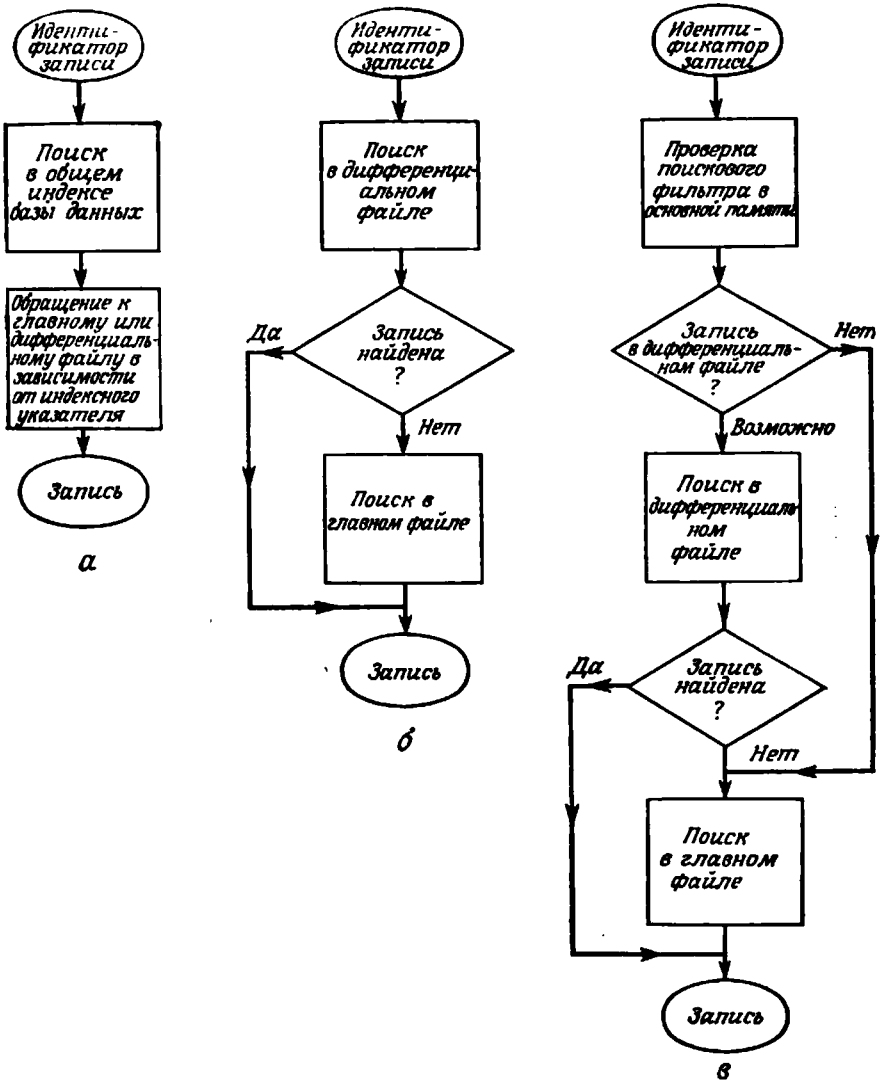


Рис. 18.9. Альтернативные стратегии доступа.
а — общий индекс; *б* — отдельные индексы; *в* — отдельные индексы и поисковый фильтр.

Значение 0 однозначно указывает на то, что последняя версия записи находится в главном файле. Поиск в дифференциальном файле не нужен, и обращение производится к главному файлу. Значение 1 указывает на то, что обновленная копия записи, вероятно, находится в дифференциальном файле, в котором и производится поиск. Тем не менее существует вероятность того, что поиск не приведет к успеху, так как разряды, связанные с идентификатором, могли быть установлены в 1 при совпадении отображения другой обновляемой записи. Поиск в обоих файлах будет осуществляться только в случае такой ошибки фильтрации.

Преимущества дифференциального файла

Как показывает история попыток создания баз данных, простота проекта базы данных является главным показателем того, что его реализация будет успешной. И хотя понятие дифференциального файла концептуально весьма просто, на практике любое усложнение системы должно быть обосновано реальной выгодой. Потенциальные преимущества дифференциальной организации базы данных полностью не оценены. Даже существующие и эксплуатируемые в настоящее время системы мотивированы довольно слабо. Это послужило поводом назвать и обсудить в данном разделе восемь основных преимуществ, которые дает использование дифференциальной организации баз данных. Пять из них связаны с целостностью базы данных. Они устанавливают тот факт, что при использовании дифференциального файла можно не только снизить стоимость копирования и скорость восстановления, но даже минимизировать вероятность серьезных потерь данных. Оставшиеся три преимущества связаны с эксплуатацией; дифференциальный файл может привести к повышению доступности данных и одновременно снизить затраты памяти и стоимость выборки. В сумме эти преимущества дают очень весомый аргумент для более широкого использования дифференциальных файлов, в особенности при поддержании очень больших баз данных.

1. *Снижение стоимости создания дампа базы данных.*

Для восстановления физически разрушенной базы данных обычно используется один из вариантов процедуры развертывания (roll-forward) [355]. Вначале путем перезагрузки восстанавливается состояние базы данных, существовавшее в некоторый предыдущий момент времени. Затем с помощью специальной процедуры обработки в базу данных добавляется накопленный результат обработки всех транзакций обновления, выполненных с момента последнего копирования базы данных. Частота, с которой осуществляется копирование базы данных, является основным параметром при разработке рассмотренного варианта процедуры восстановления [65, 110]. Частое копирование обес-

печивает быстрое восстановление базы данных, но связано с большими системными издержками.

Поскольку время, требующееся на копирование, пропорционально объему копируемых данных [171], применение дифференциального файла может значительно снизить стоимость восстановления большой базы данных, в особенности если доля записей, измененных с момента последнего копирования, невелика. Рассмотрим для примера базу данных, имеющую 10^7 записей, длиной 500 символов каждая и размещенную на диске IBM 3330 в виде блоков размером в одну дорожку. Предположим, что база данных эксплуатируется 5 дней в неделю по 10 часов в день, интенсивность изменений — 100 изменений в час. При использовании быстрой утилиты копирования/восстановления [171] полный дамп базы данных потребует более 6 часов. В то же время дамп дифференциального файла, его двоичного вектора и требуемых рациональным механизмом поиска данных даже после недельной эксплуатации может быть получен менее чем за две минуты. В сумме он будет занимать менее 300 дорожек по сравнению с 51 пакетом дисков, занимаемых базой данных.

2. *Возможность создания дампа приращениями*

В ряде случаев неразумно создавать дамп всей базы данных за один раз. Разработана стратегия дампирования приращениями [268], называемая также «дифференциальным дампированием диска» [355]. Согласно этой стратегии, физические сектора базы данных периодически просматриваются и, если в них произошли изменения, дампируются. Реализация дифференциального файла, при которой новые записи последовательно располагаются во вторичной памяти (например, в системе [254]), совершенно естественно приводит к той же самой стратегии. Для того чтобы в любое время обеспечить полное восстановление базы данных, необходимо лишь добавить записи дифференциального файла, созданные после последнего копирования базы данных, к текущему состоянию текущей копии файла. При каждом дампировании приращениями возможно сохранить текущее состояние двоичного вектора дифференциального файла и индекс поиска. То и другое могут быть восстановлены путем простого просмотра дифференциального файла.

3. *Возможность дампирования и реорганизации параллельно с обновлением*

В связи с тем, что дамп представляет собой мгновенное состояние базы данных в фиксированный момент времени, обычные процедуры восстановления в процессе своего выполнения запрещают все изменения базы данных. Путем создания дампа дифференциального файла время, в течение которого запрещены запросы на обновление, может быть существенно снижено. Од-

нако можно полностью устранить запрет на внесение изменений путем построения «дифференциально-дифференциального» файла, в котором будут запоминаться изменения, возникающие во время создания дампа дифференциального файла. Для большинства практических случаев такой файл чрезвычайно мал и его можно разместить в оперативной памяти. Действуя как кэш-память во время создания дампа, этот файл должен просматриваться перед каждой выборкой. После завершения создания дампа содержащиеся в нем записи должны быть включены в основной дифференциальный файл. Очевидно, что эта же идея дает возможность организовать реорганизацию параллельно с эксплуатацией. Так как генерация нового основного файла может потребовать большого количества времени, дифференциально-дифференциальный файл должеи поддерживаться во вторичной памяти. После завершения реорганизации происходит замена старого дифференциального файла.

Подобные процедуры дампирования и реоргаиизации базы данных особенно подходят для таких приложений, как резервирование авиабилетов, которые требуют 24-часовой эксплуатации, но в то же время у них имеются периоды снижения интенсивности запросов. Любая процедура дампирования или реорганизации может быть активизирована в период снижения рабочей нагрузки и может выполняться без введения ограничений на внесение изменений, пока уровень загрузки системы вновь не возрастет.

4. *Ускорение восстановления после потери данных*

Выход из строя запоминающих устройств не является единствеиной причиной потери данных. Причинами могут быть также иеправильная модификация данных программой пользователя, программный сбой, возиикновение тупика в системе, машинный сбой. Все они могут прекратить выполнение программы обработки запроса, изменяющей данные, до ее завершения. При этом любая ошибка может привести к нарушению содержательной и/или структурной целостности базы данных. В работе [254] предложен рабочий пример дифференциального файла (используемого для поддержания эксплуатациоиного журнала предшествующих образов), который позволил проводить быстрое восстановление системы посредством отката неправильно обработанной или частично не завершенной транзакции.

5. *Снижение риска безвозвратной потери данных*

При восстановлении базы данных настроенная соответствующим образом утилита копирования-восстановления обеспечивает перезагрузку практически с предельной скоростью передачи существующей аппаратуры (порядка 10^5 символ/с). Основная часть времени восстановления тратится затем на индивидуальное восстановление небольших частей восстанавливаемых запи-

сей. Это небольшое подмножество измененных записей является ахиллесовой пятой системы. Традиционная организация файла с обновлением на месте размещает измененные записи по всей вторичной памяти. Это практически гарантирует, что даже локальная неисправность в аппаратуре (например, пропуск дорожки или неисправность головки в одном из устройств очень большой базы данных) потребует проведения длительного процесса восстановления. В этом случае путем концентрации изменений на малом физическом пространстве дифференциальный файл дает следующие три потенциальных преимущества:

а) Минимизируется критическая незащищенная область базы данных. Большинство физических нарушений может быть быстро восстановлено с использованием локальной процедуры копирования файла.

б) Критическая область может быть размещена на более надежном устройстве по сравнению с размещением главного файла.

в) Небольшая критическая область может быть дублирована для достижения максимальной надежности при ограниченных затратах на стоимость обработки.

6. *Эффективная поддержка «файла памяти»*

Правильное обновление базы данных в режиме on-line требует сложных программных средств, обеспечивающих управление многопользовательским доступом и гарантирующих восстановление данных [184]. Для того чтобы исключить существенные издержки, связанные с применением таких программных средств, многие системы, работающие в режиме on-line, в действительности лишь накапливают изменения для их пакетной обработки после окончания рабочего дня. Например, системы материально-технического снабжения обычно могут допустить некоторую потерю точности в период между циклами обновления, конечно, если целостность базы данных полностью восстанавливается после обработки каждого пакета изменений. В системах, где можно допустить контролируемое запаздывание информации (например, в банковских системах), может быть использована концепция «файла памяти» [100] для поддержания «вероятностно-точных» данных без потребности в сложном программном обеспечении. Идея состоит в использовании программного обеспечения, не защищенного от всевозможных непредвиденных событий (таких, как параллельное обновление, системные сбои, неисправность головок), для обновления копии базы данных. В конце дня изменения вносятся в действующую базу данных. Использование при этом дифференциального файла является очевидным.

7. *Упрощение разработки программного обеспечения*

В системе с дифференциальным файлом основной файл данных и связанный с ним индекс не подвергаются действию обновлений — это дает возможность использовать имеющиеся средства для разработки и проверки нового программного обеспечения обработки данных. При использовании двух дифференциальных файлов можно представить ситуацию, при которой разрабатываемая и эксплуатируемая системы работают параллельно, имея доступ к одному и тому же главному файлу, но при этом они модифицируют каждая свой дифференциальный файл. При отладке нового программного обеспечения можно проводить непосредственное сравнение данных, поддерживаемых обеими системами. Такое использование дифференциального файла особенно важно для больших баз данных, когда либо невозможно создать вторую копию базы данных специально для экспериментов, либо невозможно поддерживать в режиме on-line одновременно обе копии.

В связи с тем, что главный файл является неизменяемым в период между реорганизациями, структуры, необходимые для его хранения, являются по-настоящему простыми и эффективными. При первоначальной загрузке базы данных может быть достигнута большая плотность хранения, поскольку нет необходимости ни в выделении свободного пространства памяти, ни в хранении связующих записей [264]. Если пользовательская программа обращается к данным, о которых известно, что они являются либо постоянными, либо относительно неизменными, то такой запрос может, минуя дифференциальный файл, обратиться непосредственно к главному файлу без постановки в очередь для обеспечения монопольного доступа.

8. *Снижение стоимости хранения базы данных в будущем*

В обозримом будущем с помощью одной из конкурирующих технологий мы получим устройства массовой памяти с прямым доступом емкостью в триллионы битов. При этом ожидается, что стоимость записи-считывания будет на порядок больше стоимости только считывания. Использование концепции дифференциального файла при работе с таким оборудованием очевидно: большой главный файл предназначен только для чтения, и снижение стоимости, обеспечиваемое использованием дифференциального файла, значительно увеличит возможную сферу применения автоматизированных информационных систем.

Приложение Б

Упражнения по теме «Физическое проектирование баз данных»

Упражнение Б1

Для задач концептуального проектирования и проектирования реализации, сформулированных в приложении А, определить значения физических параметров, обеспечивающих эффективную реализацию выбранной СУБД-ориентированной логической структуры базы данных, предполагая, что СУБД-независимая логическая структура фиксирована. Следует использовать такие критерии эффективности, как время ввода-вывода и объем памяти, а также время обслуживания CPU, если выбранные СУБД и ОС позволяют получать разумные оценки этой величины.

Проанализируйте следующие классы характеристик:

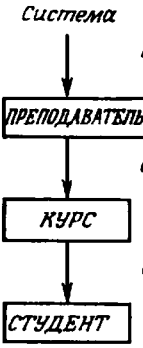
1) варианты указателей;
2) методы доступа (последовательный, произвольный и т. п.);
3) вторичные индексы;
4) кластеризацию (области, подфайлы, группы вторичных наборов данных);
5) размер блока;
6) любые другие параметры, которые специфичны для выбранной СУБД (например, распределение буферов, распределение вторичной памяти, соотношение между объемами внешней и оперативной памяти, методы сжатия данных и т. п.).

Упражнение Б2

В следующем примере структура базы данных, описывающая модель «преподаватель-курс-студент», представлена «простой» сетевой схемой. Предполагается, что СУБД не поддерживает индексы и пользователь должен сам создавать специальные типы записей для обеспечения индексации. Требуется расширить предложенную схему так, чтобы реализовать каждый из следующих ниже методов доступа:

1) индексный последовательный доступ к записям о преподавателе;
2) индексный произвольный доступ к записям о курсе по номеру курса;

3) инвертированный доступ к записям о студентах по наименованию кафедры, где учится студент;
4) индексный доступ к записям о преподавателе с использованием *B*-дерева;
5) индексный доступ к записям о курсе с использованием *TRIE*-структуры.



Исходная структура

ЗАПИСЬ	ПРЕПОДАВАТЕЛЬ	
ЭЛЕМЕНТ	SSN	9 символов
ЭЛЕМЕНТ	ФАМИЛИЯ	20 символов
ЭЛЕМЕНТ	ИМЯ	20 символов
ЭЛЕМЕНТ	ОТЧЕСТВО	1 символ
ЭЛЕМЕНТ	КАФЕДРА	30 символов
ЗАПИСЬ	КУРС	
ЭЛЕМЕНТ	КОЛИЧЕСТВО	8 символов

ЭЛЕМЕНТ	ИМЯ	25 символов
ЗАПИСЬ	СТУДЕНТ	
ЭЛЕМЕНТ	ИДЕНТИФ. №	10 символов
ЭЛЕМЕНТ	ФАМИЛИЯ	20 символов
ЭЛЕМЕНТ	ИМЯ	20 символов
ЭЛЕМЕНТ	КАФЕДРА	30 символов
СВЯЗЬ	ФАКУЛЬТЕТ	(Отсортировано по SSN)
ИСХОДНЫЙ	СИСТЕМА	
ПОРОЖДЕННЫЙ	ПРЕПОДАВАТЕЛЬ	
СВЯЗЬ	ОБУЧЕНИЕ	
ИСХОДНЫЙ	ПРЕПОДАВАТЕЛЬ	
ПОРОЖДЕННЫЙ	КУРС	
СВЯЗЬ	СПИСОК	(Отсортировано по фамилии)
ИСХОДНЫЙ	КУРС	
ПОРОЖДЕННЫЙ	СТУДЕНТ	

Упражнение Б3

а. Для простейшей файловой организации массива данных о служащих университета, приведенной ниже, спроектировать организацию данных для каждого из перечисленных методов доступа: 1) последовательный; 2) произвольный; 3) индексно-произвольный; 4) индексно-последовательный; 5) инвертированный; 6) двусвязанное дерево; 7) *B*-дерево.

Решение представить в графической форме. Каждая из этих организаций данных должна обеспечивать обработку следующих транзакций:

Приложение А: получить упорядоченный в алфавитном порядке список всех служащих (частота использования — 10 %).

Приложение Б: вывести на печать все данные о служащем по заданному идентификационному номеру (частота использования — 50 %).

Приложение В: вывести на печать имена служащих с указанием названия кафедры, которые относятся к возрастной группе XXX и имеют степень YYY (частота использования — 40 %).

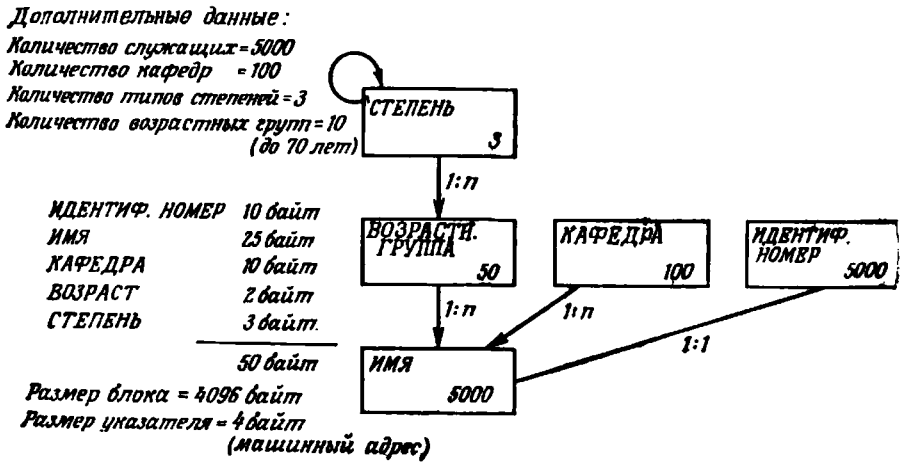
Желательно получить по возможности эффективные решения.

б. Сравнить полученные решения по требуемому объему памяти и времени доступа для каждого приложения. Определить полное время доступа через время произвольного доступа к блоку (TRBA) и время последовательного доступа к блоку (TSBA). Предполагается, что используется дисковая память, но для расчета следует использовать только параметры, определенные выше. Результаты представить в виде следующей таблицы.

Таблица Б1

	Приложение А	Приложение Б	Приложение В	Объем памяти
Последовательный				
Произвольный				
и т. п.				

в. Определить, какой метод доступа обеспечивает наименьшее время доступа для всех трех типов приложений. Если для доступа к записям в приложении Б использовались также переменные ВОЗРАСТ, КАФЕДРА или СТЕПЕНЬ, то остался бы этот метод доступа по-прежнему наиболее эффективным? Дайте обоснование своего ответа.



Упражнение Б4

Сведения о местонахождении морских судов, участвующих в различных исследовательских проектах, объединнены в простейший последовательный файл. Файл расположен последовательно на накопителе на магнитном диске, который обеспечивает требуемые временные характеристики и содержит 25 000 записей по 80 байт в каждой. При работе с файлом примерно 20 % запросов приходится на операции обновления, которые осуществляются в пакетном режиме, и 80 % — на индивидуальные запросы пользователей. Обновление требует в среднем 600 транзакций на один пакет. В результате обработки запроса выбирается в среднем 2000 логических записей.

1. Если пользователю требуются только ответы на запросы, то какие методы доступа обеспечивают минимальное время ввода-вывода: последовательный, индексно-последовательный или инвертированный? (Спецификации инвертированного файла включают 8 типов элементов, каждый из которых может принимать 25 возможных значений.) Изобразить графически время ввода-вывода в зависимости от количества полученных на запрос «справок».

2. Рассчитать время обновления и время ответа на запрос; определить метод доступа и структуру, обеспечивающие наименьшее время ввода-вывода. Изобразить графически время ввода-вывода в зависимости от объема обновления пакета (от 0 до 100 %).

3. Определить степень зависимости результатов, полученных в пп. 1 и 2, от размера блока, размера индекса блока, количества записей, выдаваемых на один запрос, режима использования НМД — совместного и монопольного.

Упражнение Б5. Сокращение времени отклика

Для обслуживания терминального комплекса, работающего на линии с ЭВМ в оперативном режиме, вычислительному центру потребуется удвоить объем основной оперативной памяти (до 1 Мбайт) и увеличить быстродействие и емкость дисковой памяти. Необходимость модернизации аппаратного обеспечения обосновывается требованием сокращения времени отклика до 1 мин для специального набора приложений, связанных с поиском данных в файле. В настоящее время обработка производится в пакетном режиме и составляет от 1 до 4 ч на одно приложение. Предполагается, что время «прогона» одного задания ограничено полным временем ввода-вывода и равно 4 мин. Задача упражнения состоит в том, чтобы предложить такие альтернативные методы реформатирования базы данных,

которые обеспечат требуемую оперативность, и сравнить относительные преимущества и недостатки выбранных вариантов.

Последовательность действий при обработке приложений изображена на рис. Б1. Прежде всего формируется критерий поиска записей, а затем производится считывание последовательного файла фиксированной длины, состоящего из форматированных записей, при этом определяются статистические данные по отдельным группам записей. Так как заданы строгие

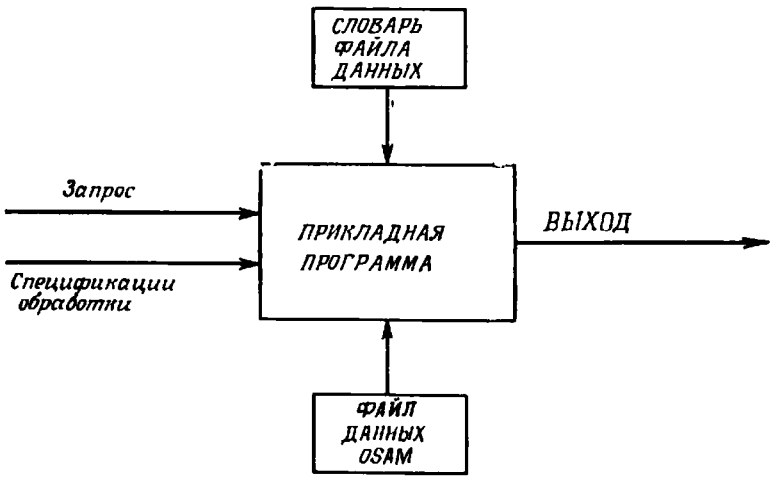


Рис. Б1.

ограничения на объем оперативной памяти, то приложения спроектированы с использованием коэффициента блокирования, равного 5, и одной области памяти для буферов.

Таблица Б5. а.

Спецификации набора данных	
Количество логических записей	30 000
Количество элементов данных	200
Средняя длина элемента данных, байт	3
Длина логической записи, байт	690
Длина идентификатора (числового), байт	6

Использование набора данных

От 5 до 50 пользователей в день запрашивают суммарные статистические данные, например итоговую сумму по опреде-

Таблица Б5.б

Изменчивость набора данных	
Удаление	2500 в мес
Модификация	Нет
Добавление	2500 в мес
Оперативность внесения изменений не критична, добавления вводятся в пакетном режиме один раз в месяц, старые транзакции переписываются на магнитную ленту (архив)	

ленному подмножеству записей, количество которых обычно составляет 0,1 %—10 % от их общего числа. В общем случае критерий поиска записи может быть произвольным, однако выбор 80 % записей осуществляется по 5—10 атрибутам, специфичным для каждого месяца. Обычно активно используется от 10 до 100 ключей (т. е. булевых комбинаций пар вида «атрибут-значение»). В течение года интерес могут представлять разные атрибуты.

Новые технические средства предполагают наличие достаточного количества каналов, блоков управления и накопителей, чтобы можно было пренебречь вероятностью блокировки при передаче по каналу сообщений от разных источников. Кроме того, имеется возможность увеличить объем оперативной памяти для прикладной программы до 50К.

Таблица Б5.в

Аппаратное обеспечение	НМД (старая система)	НМД (новая система)	НМД (старая система)
Время полного оборота	25 мс	17 мс	
Время доступа			
минимальное	25 мс	10 мс	4 мс (стартстопный режим)
среднее	75 мс	16,7 мс	3 мс (без остановки)
Плотность записи (симв. на дорожку)	7294	13 030	1600 бит/дюйм
Количество дорожек на цилиндре	20	19	
Скорость передачи данных (кбайт/с)	312	806	320

Таблица Б5.г

Количество физических блоков на одной дорожке	Максимальный размер физического блока в байтах	
	НМД (старая система)	НМД (новая система)
1	7294	13 030
2	3521	6 447
3	2298	4 253
4	1693	3 156
5	1332	2 498
6	1092	2 059
7	921	1 745
8	793	1 510
9	694	1 327
10	615	1 181
11	550	1 061
12	496	962
13	450	877
14	411	805
15	377	742
16	347	687

Вопросы для анализа.

1. Сколько времени требуется для анализа файла с использованием «старых» технических средств?

2. Как будет влиять на время анализа увеличение быстродействия новых устройств?

3. Сравнить время анализа файла с использованием «старых» технических средств с временем, которое требуется для анализа файла, содержащего 30 000 записей и размещенного на магнитной ленте.

4. Как на каждое из этих времен будет влиять увеличение коэффициента блокирования и количества буферов?

5. Как для сокращения времени отклика использовать мультисписковую или инвертированную организацию? Какие списки следует поддерживать, как они должны быть организованы, что следует использовать в качестве указателя на конкретную запись и как эти указатели должны преобразовываться в адрес записи? Обсудите варианты.

Упражнение Б6. Проектирование алгоритма поиска записей

Рассмотрим оперативную базу данных объемом 100 000 150-символьных записей, которую необходимо разместить на дисковом запоминающем устройстве IBM 3330, затраты на экс-

плуатацию которого составляют 0,5 долл. за дорожку в месяц. Характеристики ЗУ и размещения базы данных приведены ниже. Поисковые запросы на требуемую запись используют шестизначный цифровой идентификатор и поступают от 600 терминалов, работающих в режиме «on-line». Частота поступления запросов должна составлять 1 запрос в минуту с одного терминала, что соответствует 10 запросам в секунду или 100 запросам на одну запись в месяц. Время ввода-вывода оценивается 0,02 долл. за секунду, а задержка во времени отклика термина — 0,03 долл. за секунду.

Нужно спроектировать для этой задачи эффективный алгоритм поиска, а также рассмотреть возможность использования метода поиска типа ISAM и преобразования хеширования. Как следует выполнить хеширование, если идентификаторы представляют собой шестисимвольные алфавитные последовательности?

Таблица Б6.а

Аппаратное обеспечение	
Время полного оборота, мс	16,7
Время доступа, мс	
минимальное	10
среднее	30
Плотность записи (символов на дорожку)	13 030
Количество дорожек на цилиндре	19
Скорость передачи данных, кбайт/с	806
Длина адреса записи на устройстве, байты	4

Таблица Б6.б

Количество физических блоков на одной дорожке	Максимальный размер физического блока, байты	Количество физических блоков на одной дорожке	Максимальный размер физического блока, байты
1	13 030	9	1 327
2	6 447	10	1 181
3	4 253	11	1 061
4	3 156	12	962
5	2 498	13	877
6	2 059	14	805
7	1 745	15	742
8	1 510	16	687

Упражнение Б7

Эта задача позволяет использовать для проектирования файла таксономию, предложенную в работе [278]. Ниже приведены исходные данные для проектирования структуры базы данных: содержание файла и записи, требования к обновлению и поиску записей, а также характеристики внешней памяти.

а. *Содержание файла и записи*

Страховая компания поддерживает файл страховых полисов на автомобили, который содержит около 200 000 записей, по одной на каждый выданный полис. Файл хранится на диске, что позволяет обрабатывать его в оперативном режиме.

Атрибуты записей можно разбить на 7 групп: номера полиса, данные о страховых премиях, владелец полиса, описание автомобиля, страховой агент, претензии заявителя, страховые полисы дочерних компаний. В качестве идентификатора записи используется номер полиса. Сегмент «претензии заявителя» в полисе представляет собой повторяющуюся группу длиной 150 байт. В среднем на один страховой полис приходятся 2 претензии. Средняя длина записи — 468 байт.

б. *Обновление записи*

Каждый месяц оформляется 1000 новых страховых полисов и аннулируется 500 старых, что составляет увеличение количества страховых полисов на 6000 в год. Ежедневно обрабатывается 300 претензий; каждая обработка требует обновления записи. В связи с изменением характеристик клиентов (изменение адреса, условий страхования, типа автомобиля и т. п.) в среднем за день производится коррекция 80 записей. Ежеквартальные взносы клиентов поступают регулярно (16 000 в неделю).

в. *Поиск записи*

От агентов фирмы и клиентов поступает около 600 запросов в день. Цифровой 15-значный номер полиса в трети случаев бывает неизвестен, и для его поиска используется фонетическое имя владельца полиса. (Хеширование, как способ генерации номера страхового полиса, будет рассмотрено ниже.) Фирма работает ежедневно, и для управления требуются ежедневные отчеты об активности заявителей и еженедельные отчеты о выплате страховых премий. Другие отчеты требуются редко и их здесь можно не учитывать.

г. *Аппаратное обеспечение* (запоминающее устройство IBM 3330)

Время полного оборота, мс — 16,7
Время доступа, мс:
  минимальное — 10
  среднее — 30
Количество дорожек на цилиндре — 19

Скорость передачи данных, кбайт/с — 806
Длина адреса записи, байты — 4
Емкость дорожки — (см. табл. в упражнении Б6)

Алгоритм получения фонетического кода

1. Сохранить первую букву заданного имени.
2. Вычеркнуть следующие буквы: A, E, I, O, U, Y, W, H.
3. Закодировать оставшиеся буквы следующим образом:

Код	Буквы
1	B F P V
2	C G J K Q S X Z
3	D T
4	L
5	M N
6	R

4. Отбросьте оставшиеся буквы (или дополните нулями) так, чтобы длина кода равнялась 4.

Примеры

S	E	V	E	R	A	N	C	E		K	N	U	T	H
↓		↓		↓		↓				↓	↓		↓	
S		1		6		5				K	5		3	0

Ниже перечислены некоторые шаги, связанные с выбором проекта базы данных.

1. Определение наиболее часто используемых поисковых запросов и выбор соответствующего путн доступа.
2. Выбор стратегии поддержания базы данных.
3. По мере необходимости выбор вторичных структур для остальных запросов.
4. Выбор типа указателей.
5. Расчет требуемого объема памяти для всей системы.

Упражнение Б8

Часто подписчики газеты запрашивают необходимые им данные в информационной базе без указания своего идентификационного номера. Когда это происходит, служащий, обрабатывающий запрос, должен иметь возможность найти номер, идентифицирующий подписчика. Эту операцию выполняет транзакция в базе данных ALPHA, где первичным ключом является фамилия подписчика.

Существующая реализация базы данных удовлетворяет ограничениям по времени отклика, но не обеспечивает возможность ежедневного обновления. База данных перезагружается на основе ежемесячных сведений. Нужно разработать метод доступа к файлу (т. е. структуру памяти и механизм поиска), который

не снижал бы времени ответа на запрос, но допускал ежедневное обновление базы данных в пакетном режиме.

Спецификации обработки

1. Поисковый запрос (4200 в день).

Необходимо обеспечить поиск по следующим данным:

а) только по фамилии (в 10 % случаев);
б) по имени и фамилии (59 %);
в) по фамилии и адресу (30 %);
г) по имени, фамилии и адресу (1 %).

Таблица Б8. а. **Элементы данных**

Наименование	Длина (в символах)
Фамилия	15
Имя	8
Отчество	1
Адрес	32
Идентификационный номер	9
Суффикс фамилии	3
	68

Таблица Б8. б. **Распределение экземпляров данных**

Колнчество имен	Диапазон колнчества экземпляров имени	Общее колнчество экземпляров в данном диапазоне
85 420	1—1	85 420
112 402	2—100	1 011 839
1 711	100—200	236 271
633	200—300	154 435
320	300—400	109 690
214	400—500	95 960
146	500—600	79 629
101	600—700	64 894
63	700—800	46 313
61	800—900	51 789
47	900—1000	44 949
292	1001—16 410	619 105
		2 600 294

2. Обновление в пакетном режиме.
   а. Обновление данных ежедневно.

1) 26К обновлений в месяц;
2) 3700 изменений фамилий в месяц;
3) 19К изменений адреса в месяц;
4) 2300 изменений имени в месяц;
5) 20К удалений в месяц.

б. Реорганизация по мере необходимости.

в. Удаление данных ежемесячно.

Таблица Б8. в. **Некоторые специфические экстремумы распределения**

Число экземпляров	Инициалы и фамилия	Общее количество фамилий
1725	Р. Смит	Всего 16 410 фамилий
1565	М. Смит	
1482	У. Смит	
1333	Д. Смит	
1217	Е. Смит	
1300	Р. Джонсон	Всего 11 350 фамилий
1773	Дж. Салливан	Всего 10 341 фамилия
1660	М. Салливан	
739	Е. Салливан	

Упражнение Б9

Исследовать использование различных способов обработки переполнения при хешировании и при индексно-последовательной организации; изобразить графически изменение времени поиска в зависимости от увеличения объема базы данных за счет вставок.

Упражнение Б10

Сравнить эффективность использования *SPB**-дерева и *B*-дерева для поиска и обновления данных при длине разделителя, равной 1/2 длины ключа (см. гл. 14).

Упражнение Б11

Вывести и сравнить между собой выражение для расчета объема памяти физического блока при организации базы данных в виде *B**-дерева и *B*-дерева.

Упражнение Б12

Определить зависимость между размером блока и оптимальным кластером, рассчитанным по методу, предложенному в [270] (см. гл. 11).

Приложение В

Список обозначений

ANC	Усредненное количество экземпляров записей, порожденных одной исходной записью
BA	Количество обращений к бакету
BF	Коэффициент блокирования
BKS	Размер блока (в байтах)
BLKPC	Количество физических блоков на одном цилиндре
BLKPT	Количество физических блоков на одной дорожке
BLKSTOR	Общий объем памяти, требуемой для блоков базы данных (в байтах)
BNODES	Количество ветвящихся вершин в B-дереве
BOND	Сила связи «ближайший-соседний»
BOVHD	Размер служебных полей блока (в байтах)
BPT	Количество байтов на одной дорожке (максимум — размер дорожки)
C	Стоимость поиска
CA	Стоимость одного обращения к блоку
CCK	Стоимость операции ИЗМЕНИТЬ ключ
CCNK	Стоимость операции ИЗМЕНИТЬ неключевой элемент
CD	Стоимость операции УДАЛИТЬ
CI	Стоимость операции ВКЛЮЧИТЬ
CIO	Затраты времени ввода-вывода
COMMDELAY	Оценка усредненной задержки связи
COMPR	Коэффициент сжатия
CPD	Количество полезных цилиндров на одном диске
CPU.BSEARCH	Оценка времени процессора на выполнение бинарного поиска
CPU.EXCH	Оценка времени процессора на выполнение программы канала ввода-вывода для операций чтения или записи
CPU.MERGE(c, a)	Оценка времени процессора на выполнение

	объединения *c* упорядоченных списков, в каждом из которых *a* элементов
CPU.1	Оценка времени процессора на выполнение одной операции сравнения значений двух элементов данных
CPU.SORT (*a*)	Оценка времени процессора на сортировку списка из *a* элементов
CPU.TEST	Оценка времени процессора на проверку соответствия записи условию запроса
CPU.USEARCH	Оценка времени процессора на последовательный поиск в неупорядоченном списке
CR	Стоимость операции выборки
CS	Стоимость хранения данных
CSORT	Затраты времени на сортировку
CT	Оценка передачи (одного байта) данных
CU	Стоимость операции обновления
D	Среднее расстояние между экземплярами записи (в алгоритмах кластеризации)
DSTOR	Объем памяти для данных (в байтах)
EBF	Коэффициент полезного блокирования
EBKS	Полезный размер блока
ESRS	Средний размер хранимой записи для всех типов записей в базе данных
F	Частота выполнения прикладной программы базы данных
FCK	Частота операции ИЗМЕНИТЬ ключ
FCNK	Частота операции ИЗМЕНИТЬ неключевой элемент
FD	Частота операции УДАЛИТЬ
FGS	Частота операции ПОЛУЧИТЬ НЕКОТОРЫЕ
FI	Частота операции ВКЛЮЧИТЬ
IBNODES	Количество ветвящихся вершин в индексе *TRIE*-структуры
INODES	Количество вершин в индексе двусвязанного дерева
IS	Размер элемента данных (в байтах)
KS	Размер ключа (ключевого элемента данных) (в байтах)
LEN	Количество просмотренных вершин на одном уровне индекса (двусвязаное дерево)
LF	Коэффициент загрузки
LNODES	Количество вершин-листьев в *B*-дереве
LRA	Количество обращений к логическим записям
LRAB	Количество обращений к логическим записям при поиске бакета

MAXSK	Максимальное расстояние установки на CPD — 1 цилиндрах
MINSK	Минимальное время установки на один цилиндр
NB	Количество бакетов в файле произвольного доступа
NBIV	Среднее число блоков, содержащих записи данных с заданным значением элемента
NBLK	Количество блоков, требуемых для базы данных; количество блоков заданного размера в базе данных
NBLKC	Количество блоков в индексе цилиндра
NBLKOV	Количество физических блоков, отведенных под область переполнения
NBLKT	Количество блоков в индексе дорожки
NBUF	Количество буферов, отведенных прикладной программе
NCYL	Количество непрерывных цилиндров на диске, необходимых для хранения базы данных
NIL	Количество уровней индекса в методе доступа
NIT	Количество типов элементов данных в базе данных
NIV	Количество экземпляров одного типа элемента в базе данных; усредненное количество значений одного типа элемента
NPTR	Среднее количество указателей в записи
NR	Количество записей в файле или базе данных
NREC	Количество экземпляров записи
NRIV	Среднее колнчество записей, содержащих заданное значение элемента
NTB	Количество блоков, содержащих NTR записей-целей запроса
NTR	Количество записей-целей запроса
NTRK	Количество дорожек, необходимых для базы данных
NVAL	Количество значений типа элемента данных, требуемых прикладной программе
OVFLCH	Средняя длина цепочки переполнения
OVFLPR	Вероятность переполнения бакета
OVPBA	Количество обращений к физическим блокам переполнения
P	Вероятность события
PBA	Обращение к физическому блоку
PMS	Вероятность наличия произвольно запрошенного блока в буфере оперативной памяти

PRDEL	Вероятность удаления старого значения ключа из индекса
PRI	Вероятность наличия в бакете не меньше *i* записей
PRINS	Вероятность включения нового значения ключа в индекс
PRMV	Вероятность пересылки записи данных в новый блок
PS	Размер указателя (в байтах)
PTRSTOR	Объем памяти указателей (в байтах)
R	Оценка реорганизации
RECSIZE	Усредненный размер логической записи (в байтах)
REDUC	Усредненное уменьшение размера данных вследствие сжатия
ROT	Время полного оборота диска
ROVHD	Размер служебных полей записи (в байтах)
RPB	Усредненное количество хранимых записей в одном бакете
RPBD	Количество записей в пакете для операции удаления пакета записей
RPBI	Количество записей в пакете для операции включения пакета записей
RPBR	Количество записей в пакете для операции выборки пакета записей
RSPB	Количество фрагментов записей в одном бакете
S	Мера подобия
SDIST	Расстояние поиска (на диске)
SEEK (CPD)	Среднее время установки для CPD цилиндров на диске
SEEK (NCYL)	Среднее время установки для NCYL непрерывных цилиндров базы данных
SRS	Размер хранимой записи
SRSTOR	Общий объем памяти для хранимых записей
TC	Общая стоимость обработки базы данных, включая стоимость поиска и реорганизации
TCOST	Общие затраты на поиск и хранение данных
TCPU	Общее время процессора
TCPUB	Среднее время процессора на обработку данных одного физического блока
TFBA	Среднее время передачи последовательного физического блока с диска в оперативную память

TIO	Время обслуживания ввода-вывода (в секундах)
TPC	Количество полезных дорожек на одном цилиндре
TR	Скорость передачи данных на диск (с диска) (в байтах на секунду)
TRBA	Среднее время доступа и передачи произвольного физического блока с диска в оперативную память
TRESP	Время отклика
TRKSTOR	Объем памяти для дорожек, содержащих базу данных (в байтах)
TRVOL	Объем передачи (в байтах)
TSBA	Среднее время доступа и передачи последовательного физического блока с диска в оперативную память
UKV	Количество уникальных комбинаций ключевых значений в записях двусвязаиного дерева
USED	Фактически использованное пространство в блоке
V	Количество знаков в алфавите
WAIT	Общее время ожидания в очередях ресурсов, включая задержки взаимных блокировок процессов

Толковый словарь

Агрегация (aggregation). Способ абстракции, при котором взаимосвязь объектов рассматривается как объект более высокого уровня.

Администратор базы данных (database administrator). Лицо которому, возможно, помогает служебный персонал, ответственное за базу данных в организации на протяжении ее жизненного цикла.

Аномалии (операций добавления, удаления, обновления) (anomalies (add, delete, update)). Три возможных нарушения правил манипулирования данными вследствие нарушения условия нормальной формы отношения. При удалении последнего экземпляра, описывающего связь элементов данных, может произойти потеря информации о возможности такой связи.

Атомное условие (atomic condition). Базисное условие квалификации запроса, имеющее вид: «имя{<=>} значение», где «имя» представляет имя типа элемента данных, а «значение» — его значение.

Атрибут (attribute). Информация, описывающая объект и, возможно, служащая его идентификатором. В записи данных атрибут представлен типом элемента данных и может использоваться в качестве первичного ключа, вторичного ключа или неключевого элемента.

AVL-дерево (AVL-tree). Сбалансированное по высоте дерево, такое, что его левые и правые поддеревья различаются по высоте не больше чем на 1.

База данных (database). Автоматизированное хранилище совокупности операционных данных, предназначенных для обслуживания многих пользователей в организации или в определенной части организации.

Бакет (bucket). Область памяти, которая может содержать несколько записей и адресуется как единое целое с помощью некоторого способа адресации. Основная адресуемая единица в функциях хеширования, рандомизации или вычисления.

Безопасность (security). Защита данных от преднамеренного или непреднамеренного нарушения секретности, искажения или разрушения данных.

Бинарное дерево поиска (binary search tree). Бинарное дерево, каждой вершине которого приписано отдельное имя, и все

вершины расположены в соответствии с лексикографическим порядком имен, то есть для любой вершины *i* имена вершин ее левого поддерева лексикографически предшествуют имени вершины *i* и именам вершин ее правого поддерева.

Бинарный поиск (binary search). Метод поиска в физически последовательном файле. Процедура поиска заключается в выборе верхней или нижней половины просматриваемого участка; выбор основан на анализе значения ключа в середине просматриваемого участка. Выбранная часть затем снова делится пополам и т. д., пока не будет найден искомый элемент.

B-дерево (*B*-tree). Расширение понятия бинарного дерева: из каждой его вершины могут выходить две и более ветвей.

B*-дерево (*B**-tree). Вариант *B*-дерева: в листьях (конечных вершинах) размещены ключи и соответствующие данные, а доступ к ним осуществляется через *B*-дерево индекса.

Виртуальное поле (или запись) (virtual field (or record)). Поле или запись, физически не существующие, но кажущиеся существующими; значение такого поля физически формируется в момент его запроса со стороны прикладной программы. Оно строится или извлекается из существующих физических данных.

Внешняя схема (по определению ANSI-SPARC) (external schema (ANSI-SPARC)). Представление данных с позиции прикладного программиста, обычно нашедшее отражение в схеме реализации; логическая структура базы данных.

Внутренняя схема (по определению ANSI-SPARC) (internal schema (ANSI-SPARC)). Физическая структура данных; представление данных с позиций вычислительной системы, как они на самом деле выглядят на запоминающем устройстве; обычно отражается в структуре хранения данных.

Возможный ключ (candidate key). Элемент данных, такой, что каждый его экземпляр в группе идентифицирует экземпляр объекта; идентификатор.

Восстановление (recovery). Предусмотренная в СУБД способность восстанавливать целостность (правильное состояние) базы данных вслед за любым видом сбоя системы.

Время ввода-вывода (I/O time). Время, затрачиваемое на выборку данных с внешней памяти в оперативную, включая время передачи данных.

Время доступа (access time). Промежуток времени между выдачей команды, содержащей обращение к некоторым данным, и фактическим получением данных для обработки.

Время отклика (response time). Промежуток времени между вводом запроса к базе данных в ЭВМ и завершением обработки запроса с представлением результатов.

Вторичная группа наборов данных (термин IMS) (secondary data set group (IMS)). Физическая кластеризация сегментов, исключая корневой сегмент (IMS).

Вторичный ключ (secondary key). Тип атрибута или элемента данных, используемый для индексирования записей, но необязательно однозначно их идентифицирующий (то есть одно и то же значение ключа может входить в несколько экземпляров записи).

Вторичный метод доступа (secondary access method). Совокупность средств, предусмотренных для обеспечения эффективного доступа к записям-целям в соответствии с набором значений вторичного ключа в запросе.

Выделенное устройство (dedicated device). Внешнее запоминающее устройство (диск), выделенное для использования в одной прикладной программе пользователя; благодаря этому повышается эффективность ее выполнения за счет более эффективного использования физических характеристик устройства.

Глобальная информационная структура (global information structure). Эффективное соединение двух и более локальных информационных структур с минимизацией избыточности и длины пути доступа к записи.

Данные (data). Нечто, известное из опыта или введенное с определенной целью; устаиовленный факт или принцип; то, на чем основаны логический вывод или обоснование; основа построения любой интеллектуальной системы.

Двусвязанное дерево (doubly chained tree). Инвертированная организация файла, предусматривающая в иерархической структуре индекса один уровень для каждого ключевого типа элемента данных; в элементы индекса включены значения ключевых типов элементов данных, а сами записи данных не содержат значений ключевых типов элементов, используемых для их индексирования.

Длина пути доступа (access path length). Количество экземпляров или логических записей базы данных, просматриваемых при выполнении заданной прикладной функции.

Домен (domain). Множество элементов (полей) данных одного и того же типа в отношении.

Жизненный цикл базы данных (database system life cycle). Основные этапы процесса проектирования, реализации и реорганизации базы данных и ее прикладного программного обеспечения.

Запись (record). Поименованная совокупность элементов данных, рассматриваемая прикладной программой как одно целое.

Запись-цель (target record). Запись, удовлетворяющая условиям квалификации запроса.

Запрос (query). Запрос на выборку определенных данных из базы данных; часто формулируется в виде булевой функции, построенной с помощью логических конъюнкций «И» и дизъюнкций «ИЛИ».

Значение элемента данных (item value). Значение элемента данных в экземпляре записи.

Идентификатор (identifier). Атрибут, значения которого однозначно определяют экземпляры моделируемого объекта предметной области; тип элемента данных, однозначно идентифицирующий экземпляры конкретного типа записи (то есть первичный ключ).

Иерархический порядок (или иерархическая последовательность) (hierarchical order (or sequence)). В системе IMS порядок размещения экземпляров сегмента по возрастанию значений иерархического ключа, который состоит из (справа налево) значения ключевого поля сегмента, кода типа сегмента и, возможно, значения иерархического ключа исходного сегмента, если таковой имеется.

Избыточность (redundancy). Дублирование элементов и (или) записей данных.

Инвертированный файл (inverted file). Организация файла, обеспечивающая возможность быстрого поиска данных для обычных запросов, основанного на значениях вторичного ключа. Она предусматривает многоуровневую структуру индекса и совокупность списков указателей доступа, адресующих данные с определенным значением ключа. Частичный инвертированный файл инвертируется только относительно некоторых типов элементов данных (предусматривает для них элементы индекса), а полный инвертированный файл инвертируется относительно всех типов элементов данных.

Индекс (index). Таблица, содержащая информацию о записях или элементах данных с указанием их местоположения.

Индекс блока (block index). Упорядоченный индекс значений первичного ключа, организованный так, что каждое значение в нем соответствует наибольшему значению ключа в записях блока. С каждым значением ключа в индексе связан указатель соответствующего блока. Понятие индекса блока служит основой индексно-последовательного метода доступа.

Индексно-последовательный метод доступа (indexed sequential access method). Метод доступа к упорядоченному последовательному файлу, использующий иерархическую структуру индексов блока; каждый индекс упорядочен в соответствии со значением первичного ключа (также упорядочены записи в файле); этот метод позволяет уменьшить время произвольного доступа к записям, сохраняя при этом эффективную последовательную обработку данных. При наличии переполнения вслед-

ствне добавлений записей в базу даиных эффективность обработки резко падает.

Индексно-произвольная организация (indexed random). Организация файла с полным индексом.

Интеграция представлений (view integration). Объединеиие и представление требований отдельных пользователей в единую форму. По желанию можно выполнять отдельно интеграцию требований, зависимых от обработки, и требоваиий, независимых от обработки.

Информационная структура (information structure). Схема представления примитивных объектов данных и связей между ними, абсолютно независимая от характеристик СУБД. Для графического представления объектов данных обычио используются прямоугольники, а связи изображаются в виде дуг или стрелок.

Информация (information). Знание, сообщенное кем-то или полученное в результате исследования, анализа или обучения; знание, почерпнутое из данных [334].

Информация, соответствующая концептуальному прикладному представлению (usage perspective (UP)). Информация, описывающая требования к обработке; информация, зависимая от обработки.

Информация, соответствующая концептуальному структурному представлению (information structure perspective (ISP)). Информация, описывающая естественные и хоицептуальные связи между всеми данными в базе данных и ие ограниченная необходимостью обеспечить только какие-то прикладные функции; независимая от обработки информация.

Кластеризация (clustering). Размещение связанных данных поблизости друг от друга в целях повышения эффективности доступа. При логической кластеризации определенные типы элементов данных объединяются в записи; при физической кластеризации экземпляры записей одного или разных типов помещаются в один блок, одну область или иа одно запоминающее устройство.

Кластеризация записей (record clustering). Кластеризация хранимых записей в набор непрерывных экстентов.

Ключ, ключевой, ключевая (key). Элемент данных, используемый для идентификации или определения местоположения записи. Организация хранения данных, предусматривающая доступ к экземплярам записи через цепь (связаиный список). В цепи может быть предусмотрено (но необязательно) логическое упорядочивание.

Ключ базы даиных (database key). Уникальный идентификатор, ассоциируемый с каждым экземпляром записей в базе даиных.

Ключ поиска (search key). Ключ или идентификатор в запросе, в соответствии с которым осуществляется поиск данных. Полное описание всех типов записей, наборов, элементов данных в том виде, как они представлены в базе данных. Во многих промышленных версиях СУБД описание физической структуры включено в описание схемы, однако условимся отделять при анализе понятие логической схемы от физической.

Количество обращений к физическим блокам (physical block accesses (PBA)). Общее количество физических блоков, доступ к которым обеспечивает выполнение прикладной функции базы данных. Последовательный доступ к блоку на диске включает задержку вращения и время передачи (время ввода-вывода), произвольный доступ к блоку (доступ по указателю) дополнительно включает усредненную задержку поиска.

Коллизия (collision). Случай преобразования значения ключа в физический адрес, который уже занят.

Конвертирование данных (data translation). Модификация физического (а иногда и логического) представления данных, реализованного с использованием одних программно-технических средств, в целях совместимости с другими программно-техническими средствами.

Конвертирование программы (program translation). Модификация текста исходной программы, первоначально написанной для манипулирования данными в одной программно-технической среде, с целью обеспечения ею эквивалентных функций применительно к конвертированным данным в другой программно-технической среде.

Контроль правильности данных (certification, validation). Контроль данных до и после загрузки в базы данных на наличие ошибок. Проверка формата, области значений, уникальности значения ключевого поля и допустимого количества экземпляров.

Концептуальная схема (по определению ANSI-SPARC) (conceptual schema (ANSI-SPARC)). Высокоуровневое представление пользователя о данных, обычно в виде объектов и связей (т. е. в виде информационной структуры).

Концептуальное проектирование (conceptual design). Анализ формально специфицированных и независимых от обработки требований к информации и проектирование информационной структуры (концептуальной схемы), представляющей точную модель предметной области в виде элементов данных и их связей, независимую от конкретной СУБД.

Конъюнкт (conjunct). Условие записи в запросе, конъюнкция (логическая функция «И») условий элементов данных. Конъюнктивный запрос представляет собой запрос в виде бу-

левой функции, построенной с помощью конъюнкций и дизъюнкций элементов данных.

Коэффициент блокирования (blocking factor). Количество логических записей в одиом физическом блоке.

Коэффициент загрузки (loading factor). Отиошение, характеризующее максимальную долю полезного пространства в физическом блоке после вычисления издержек на служебные поля блока.

Коэффициеит начальной загрузки (percent fill). Параметр (со значениями от 0 до 100 %), определяющий в процентах долю памяти в каждом физическом блоке, отводимую для заполнеиия даиными в момеит начальной загрузки файла или базы данных. Благодаря этому обеспечивается возможность последующего расширения базы данных путем включеиия новых записей в соответствующий блок, не прибегая к переполнеиию в случае необходимости расположить записи в определеином порядке или поблизости друг от друга.

Коэффициент успеха (hit ratio). Отиошение количества записей в файле или базе даиных, удовлетворяющих запросу, к общему количеству запнсей.

Логическая структура базы данных (logical database structure). Схема для коикретной СУБД или определеиие даиных, полученное в результате проектироваиия реализации.

Локальная информационная структура (local information structure). Информационная структура, удовлетворяющая основным требованиям к обработке со стороиы конкретного приложения.

Массив указателей (pointer array). Список указателей записей, удовлетворяющих некоторому специфическому критерию; список указателей доступа.

Метод доступа (access method). Метод и средства организации хранения и выборки данных на физическом устройстве, обычно во внешней памяти.

Метод срастающихся цепочек (coalesced chaining). Метод органиизации цепочек для функций хешироваиия, предусматривающий глобальный список свободного пространства для всех адресуемых бакетов. Для каждого бакета строится своя цепочка, но следующий ее элемент может располагаться в любом свободном бакете, на который указывает первый элемент в списке свободного пространства.

Методология проектироваиия базы даиных (database design methodology). Совокупность методов и средств, используемых в рамках некоторой организационной схемы, для разработки последовательности проектов развития структуры базы данных.

Механизм поиска (search mechanism). Алгоритм определения специфического пути доступа к данным в структуре базы

данных и просмотра данных по этому пути в случае вызова его из транзакции.

Модель ANSI-SPARC (ANSI-SPARC model). Архитектура системы баз данных, основанная на понятии трех уровней схем спецификации данных (внешняя схема, концептуальная схема, внутренняя схема); предложена в 1975 г. в отчете рабочей группы ANSI/X3/SPARC в качестве проекта национального стандарта для системы баз данных. Внешняя схема основана на представлении данных с позиции прикладного программиста; концептуальная схема соответствует обобщенному представлению о всех записях базы данных; внутренняя схема отражает представление данных с точки зрения организации их хранения [97].

Модель данных (data model). Представление о предметной области в виде данных и связей между ними. Модель данных может отображать либо концептуальное представление данных, либо представление реализации.

Модель DIAM (DIAM model). Понятие независимой от данных модели доступа представляет формальную концепцию данных в СУБД, основанных на четырехуровневой архитектуре: от логической модели объекта до физической модели.

Мощность отношения (cardinality of a relation). Количество кортежей в отношении.

Мультисписковый файл (multilist file). Совокупность связанных последовательных структур (связанных списков), каждая из которых состоит из записей с одинаковыми значениями определенного типа элемента данных. В целях обеспечения быстрого доступа к записям по вторичному ключу указанные списки, как правило, индексируются.

Набор (термин CODASYL) (set (CODASYL)). Связь записей данных; поименованная совокупность исходного и подчиненных типов записей.

Начальная загрузка (базы данных) (populating (a data base)). Начальная загрузка данных в базу данных.

Независимость данных (data independence). Возможность изменения структуры базы данных без изменения пользующихся ею прикладных программ. Логическая независимость данных означает возможность изменять логическую структуру базы данных (схему реализации), не затрагивая прикладных программ; физическая независимость данных подразумевает такую же возможность изменения физической структуры базы данных.

Незапланированный запрос (ad hoc query). Запрос, точная спецификация которого (формат, значение ключа, размер искомой записи) заранее не известна.

Ненормализованное отношение (unnormalized relation). Отношение, не являющееся отношением в первой нормальной форме.

Непротиворечивость (consistency). Свойство базы данных, особенно в случае дублирования данных и наличия многих пользователей, заключающееся в том, что в любой момент времени все пользователи базы данных получают одинаковые ответы на одинаковые запросы. Для обеспечения непротиворечивости во время обновления доступ операций чтения к изменяющимся данным блокируется до завершения обновления всех копий данных.

Нормализация (normalization). Декомпозиция сложных структур данных в структуру из одного или нескольких плоских файлов (отношений); для определения разных уровней нормализации (т. е. нормальных форм) необходим анализ функциональных зависимостей.

Область (area). Подраздел базы данных; поименованная часть адресуемого физического пространства памяти базы данных; область (по определению CODASYL) может содержать экземпляры разного типа записей, наборов и частей наборов в соответствии с определением администратора базы данных.

Область переполнения (overflow area). Физический экстент памяти, предназначенный для размещения данных, не поместившихся в основную область данных. Области переполнения могут быть организованы внутри записей, физических блоков, дорожек и цилиндров диска.

Обновление (update). Любая модификация базы данных посредством операций добавления, удаления или изменения данных.

Обобщение (generalization). Вид абстракции, при которой множество подобных объектов рассматривается как обобщенный объект.

Объем данных (data volume). Количество экземпляров каждого типа записей, хранящихся в базе данных в текущее время.

Объем обработки (processing volume). Произведение частоты обработки и объема данных.

Объем передачи (transport volume). Общее количество байтов, переданных по линиям связи ЭВМ для удовлетворения (выполнения) прикладной функции базы данных.

Определение данных (data definition). Как правило, заключается в объявлении имени типа элемента данных, его свойств (например, знаковый или цифровой) и связей с другими типами элементов данных (включая организацию сложных групп).

Организация файла (file organization). Представление записей данных, составляющих файл, с отражением их взаимосвя-

зей и определением физических параметров, таких, как указатели и индексы.

Отношение (relation). Плоский файл или таблица; файл, записи которого (называемые «кортежи») не могут содержать повторяющихся групп.

Ошибочный результат поиска (false drop). Запись, которая по значениям указателя блока в индексе удовлетворяет запросу, но не является записью-целью этого запроса.

Параллельная реорганизация (concurrent reorganization). Стратегия реорганизации базы данных, обеспечивающая пользователям доступ к реорганизуемой части одновременно с выполнением одного или нескольких процессов реорганизации.

Первичный ключ (primary key). Ключ, однозначно идентифицирующий запись.

Плоский файл (flat file). Физически последовательная структура; файл.

Повторяющаяся группа (repeating group). Поименованная совокупность элементов данных, имеющая переменное количество экземпляров.

Подсхема (subschema). Определение представления данных с позиции отдельного пользователя-программиста; возможное подмножество схемы.

Позиционирование (currency). Метод запоминания адреса последнего выбранного экземпляра для каждого типа записи или каждого типа набора (в CODASYL) (т. е. записи владельца и т. д.) с целью обеспечения быстрого доступа к ним в случае последующего продолжения просмотра базы данных.

Поле (field). Тип элемента данных (термин IMS).

Полная функциональная зависимость (full functional dependence). Атрибут Y полностью функционально зависит от атрибута X, если он функционально зависит от X и не зависит ни от какого подмножества X (X должен быть составным).

Полный индекс (full index). Организация файла, предусматривающая для каждого экземпляра его записей соответствующий элемент индекса. Элемент индекса состоит из значения первичного ключа и указателя записи, содержащей это значение. Как правило, данный метод неэффективен, так как характеризуется большими служебными издержками памяти, за исключением случая файлов небольшого размера. Указанная организация известна также под названием «индексно-произвольной организации».

Полный сцепленный ключ (по определению IMS) (fully concatenated key (IMS)). Сцепление значений ключевых полей всех сегментов на иерархическом пути от корневого сегмента до выбираемого сегмента.

Порожденное множество (filial set). Множество всех имеющихся в наличии вершин, порожденных одной исходной вершиной.

Последовательный метод доступа (sequential access method). Последовательная обработка последовательного файла.

Последовательный файл (sequential file). Файл, занимающий непрерывное пространство физических адресов.

Проектирование базы данных (database design). Процесс разработки базы данных от требований пользователей до структуры реализации.

Проектирование реализации (implementation design). Этап проектирования базы данных, на котором выполняется уточнение и преобразование концептуальной схемы в схему конкретной СУБД.

Произвольная обработка (random processing). Сочетание организации базы данных и метода выборки записей-целей в прикладных программах пользователей, приводящее к произвольному физическому расположению записи-цели по отношению к предыдущей выбранной записи.

Произвольный метод доступа (random access method). Метод доступа, использующий функцию хеширования для получения адреса записи по значению ее ключевого элемента.

Протокол (protocol). Правила взаимодействия сетевого программного обеспечения узлов в вычислительной сети.

Прямой метод доступа (direct access method). Метод определения уникального адреса в памяти по каждому значению первичного ключа (идентификатора). Характерен большими служебными издержками памяти, но чрезвычайно полезен для организации прямого доступа к данным в случае небольшого набора возможных значений первичного ключа.

Разделяемое устройство (shared device). Внешнее запоминающее устройство, одновременно предоставленное многим пользователям; вследствие взаимных наложений наблюдается тенденция к рандомизации адресов при выборке данных. Например, выполнение прикладной функции с последовательной обработкой на разделяемом диске может вызвать произвольную установку его считывающей головки при каждом обращении к физическому блоку.

Размер блока (block size). Количество байтов в одном блоке.

Распределенная база данных (distributed database). Единая база данных, представленная в виде нескольких отдельных (возможно, избыточных и перекрывающихся) разделов на разных вычислительных установках.

Реляционная алгебра (relational algebra). Набор операций манипулирования данными, операндами которых служит одно или несколько отношений, а результатом — новое отношение.

Реорганизация (reorganization). Процесс изменения концептуальной, логической или физической структуры базы данных. Процесс изменения логической структуры называется реструктурированием, процесс изменения физической структуры называется реформатированием.

Реструктурирование (restructuring). Логическая реорганизация данных (например, изменением проекта схемы). Автоматизированная система реструктурирования представляет собой пакет прикладных программ, выполняющий реорганизацию экземпляров записей в целях совместимости с измененной схемой.

Реформатирование (reformatting). Физическая реоргаинзация данных (например, изменение представления знаков, длины слова или переиоса записей переполнения в основную область данных).

Сбалансированное дерево (balanced tree). Дерево называется сбалансированным, если разница по высоте между его корнем и любыми двумя листьями (конечными вершинами) не превышает 1.

Связь (relationship). Ассоциация между экземплярами примитивных или агрегированиых объектов (записей) даиных (например, 1 : 1, 1 : m, m : n).

Связь атрибутов (attribute relationship). Неквалифицированное отношение принадлежиости на миожестве атрибутов, описывающих один и тот же объект или связь объектов.

Связь сущностей (entity relationship). Квалифицированное или неквалифицированиое отиошение прииадлежности на множестве объектов разных типов.

Связь «сущность-атрибут» (entity-attribute relationship). Функциональная зависимость между сущностью и одиим из ее атрибутов.

Связь $M:N$ ($M:N$ relationship). Говорят, что между двумя объектами A и B существует связь $M:N$, если каждому элемеиту из A соответствует несколько элементов из B, и каждому элементу из B соответствует несколько элементов из A.

Сегмент (термин IMS) (segment (IMS)). Поименоваиная совокупность данных, включающая одно или несколько полей; запись.

Сегментация записи (record segmentation). Разбиение записи на части; размещение отдельных элементов даиных хранимой записи в отдельных экстентах, возможно, на разных физических устройствах.

Секционная инвертированная организация (cellular inverted). Вариант инвертированной оргаиизации, когда каждый элемент списка указателей доступа представляет собой указатель физического блока, содержащего хотя бы одну запись с соответствующим значением ключа.

Секционная мультисписковая организация (cellular multilist). Вариант мультисписковой организации, когда указатели связывают не записи, а только физические блоки, каждый из которых содержит хотя бы одну запись с соответствующим значением ключа.

Сетевая структура (network structure). Связь между записями (или другими группами данных), в которой подчиненная запись может иметь несколько исходных записей.

Сжатие (compression). Способ кодирования элементов данных, обеспечивающий уменьшение их размера в сравнении с обычным представлением и сохранность их информационного содержимого.

Система баз данных (database system). Совокупность программного обеспечения СУБД, прикладного программного обеспечения, базы данных, операционной системы и технических средств, задействованных с целью обеспечения информационного обслуживания пользователей.

Система обработки информации (information processing system). Вычислительная система вместе с набором программ и данных пользователя.

Система управления базами данных (СУБД) (database management system (DBMS)). Обобщенное понятие механизма управления базами данных большой размерности, включающего специализированное программное обеспечение для организации хранения, обновления, справочного обслуживания и анализа данных. В помощь всем пользователям (от чиновника до администратора базы данных) предусмотрен широкий набор языковых средств.

Системный журнал (audit trail). Журнал регистрации всех изменений базы данных.

Словарь данных (data dictionary). Каталог всех типов элементов в базе данных, включающий для каждого типа его определение, формат, источник и применение; широко распространены автоматизированные словари данных.

Смещение (displacement). Количество бакетов, просмотренных в линейном поиске, прежде чем найдена возможность включить новую запись.

CODASYL DBTG. Рабочая группа по базам данных (РГБД), специальный комитет ассоциации CODASYL, созданный в конце 60-х годов с целью разработки стандарта на современные системы управления базами данных.

Составное индексирование (compound indexing). Организация нескольких составных индексов, отличающихся порядком расположения ключевых типов элементов данных и представляющих все возможные перестановки этих ключей.

Составной индекс (combined index). Индекс, каждый элемент которого содержит сцепление значений разных ключевых типов элементов данных. Полный составной индекс состоит из значений всех ключевых типов элементов данных в записи, а частичный составной индекс содержит значения только некоторого подмножества ключевых типов элементов в записи.

Список указателей доступа (accession list). Физически последовательный список указателей записей, содержащих одно и то же значение определенного типа ключевого элемента.

Способ проектирования (design tool). Любой аналитический, эвристический или процедурный метод, применимый к проектированию базы данных и реализованный в программном обеспечении.

Средство анализа и (или) проектирования (analysis and/or design tool). Автоматизированное или ручное средство проектирования базы данных.

Средство автоматизированной оценки (evaluator). Пакет прикладных программ, обеспечивающий средства оценки заданной (логической или физической) структуры базы данных по одному или нескольким определенным критериям производительности.

Средство автоматизированного проектирования (designer). Пакет прикладных программ, обеспечивающий средства построения возможной (логической или физической) структуры базы данных на основе заданных требований или же средства оптимизации структуры базы данных по заданному критерию (критериям).

Структура TRIE (TRIE-structure). Произвольный метод доступа, основанный на использовании отдельных знаков, составляющих значение первичного ключа; в нем на каждом уровне используется структура многоходового дерева в соответствии с возможными значениями знака $i+1$ в ключе для заданного конкретного значения знака i.

Структура хранения (storage structure). Описание способа организации физического хранения данных в системе: указатели, представление знаков, плавающая запятая, блокирование, метод доступа и т. д.

Сущность (entity). Примитивный объект данных, отображающий элемент предметной области (человека, место, вещь и т. д.).

Схема (schema). Графическое или формальное определение логической структуры базы данных. В системах CODASYL схема состоит из предложений на языке определения данных.

Сцепление (concatenation). Объединение в одну вершину B-дерева двух его вершин, заполненных меньше чем наполовину, в особенности после удаления ключа.

Таблица хеширования (scatter table). Массив указателей, соответствующий функции хеширования и обеспечивающий промежуточный уровень в организации произвольного доступа к данным; благодаря этому достигается большая гибкость при обеспечении физического упорядочивания областей данных.

Тип элемента данных (item type). Тип элемента данных или поле. Основная единица данных, описывающая объект; основной компонент записи.

Транзакция (transaction). Прикладная программа базы данных; как правило, программа обновления.

Требования к информации (information requirements). Независимые от обработки требования к информации представляют собой требования к представлению в базе данных определенных типов данных в целях точного отображения организации, а не в силу необходимости обеспечить выполнение текущих прикладных функций, целостность и безопасность данных. Зависимые от обработки требования включают требования к обеспечению эффективного и правильного выполнения прикладных программ, использующих конечную структуру базы данных.

Указатель (pointer). Индикатор, который ведет к заданной записи из какой-то другой записи в базе данных; указателем может служить абсолютный или относительный адрес записи или ее символический идентификатор.

Файл (file). Совокупность аналогично построенных хранимых записей фиксированной или переменной длины одного типа.

Физическая база данных (physical database). Организация хранения взаимосвязанных данных в виде одного или нескольких типов хранимых записей.

Физическая последовательная организация (physical sequential organization). Организация файла или физической базы данных, предусматривающая хранение записей в последовательных областях памяти, при которой логический и физический порядки следования записей совпадают.

Физическая структура базы данных (physical database structure). Формат хранимых записей, их логическое или физическое упорядочивание, пути доступа и распределение на устройстве базы данных со многими типами записей.

Функциональная зависимость (functional dependency (FD)). Для любого заданного отношения R множество его атрибутов B называется функционально зависимым от множества атрибутов A, если в любой момент времени каждое значение A в R связано только с одним значением B.

Хеширование (hashing). Метод доступа, обеспечивающий прямую адресацию данных путем преобразования значения ключа, выполняемого функцией рандомизации, в относительный или абсолютный физический адрес.

Хранимая запись (stored record). Совокупность связанных элементов данных, соответствующая одной или нескольким логическим записям и содержащая все необходимые указатели, длину записи и другие служебные данные, а также схемы кодирования для представления знаков.

Целостность (integrity). База данных обладает свойством целостности, если все данные в ней удовлетворяют условиям допустимых диапазонов их значений, и это свойство сохраняется при всех манипуляциях с базой данных.

Цепь (chain). Связанный список с нулевым значением поля связи (указателя) в последнем элементе; возможны одионаправленные и двунаправленные цепи.

Цепь вычисления (по определению CODASYL) (Calc chain (CODASYL)). Связанный список логических записей, которые в результате хеширования (вычисления) указывают на один и тот же физический блок. При этом учитываются также записи переполнения.

Частота обработки (processing frequency). Частота выполнения отдельной транзакции обновления или запроса (приложения базы данных).

Экземпляр (occurrence). Отдельный экземпляр объекта, записи, элемента данных, набора CODASYL и так далее, представленный совокупностью значений его составных частей.

Экспертная оценка проекта (design review). Рассмотрение спецификаций проекта структуры базы данных, вызванное необходимостью изменений в прикладном программном обеспечении системы баз данных.

Экстент (extent). Непрерывная область в пространстве хранения данных.

Элемент данных (data item). Наименьшая единица данных, имеющая смысл при описании информации; наименьшая единица поименованных данных.

Элементарные данные (data element). Примитивный объект данных в предметной области.

Язык запросов (query language). Высокоуровневый язык манипулирования данными, обеспечивающий средства взаимодействия пользователя с файлом или базой данных.

Язык манипулирования данными (ЯМД) (data manipulation language (DML)). Язык, используемый программистом для загрузки, доступа и обновления базы данных.

Язык определения данных (ЯОД) (data definition language (DDL)). Язык определения модели данных вместе с ее (частичным) отображением в структуру хранения; подсхема ЯОД представляет собой язык определения подмодели данных.

Литература

1. Aanstad P. S., Skylstad G., Solvberg A., CASCADE — A Computer-Based Documentation System, *Computer-Aided Information Systems, Analysis and Design*, J. A. Bubenko, B. Langefors, A. Solvberg, eds., Lund, Sweden, 1972, pp. 93—112.
2. ACM/NBS, Data Base Directions: The Next Steps, NBS/ACM Workshop, J. Berg, ed., NBS Spec. Pub 451, U. S. Dept. of Commerce, Washington, DC, 1976.
3. Адельсон-Вельский Г. М., Ландис Ю. М. Алгоритм организации информации. ДАН СССР, т. 146, с. 236—266, 1962.
4. Aho A. V., Ullman J. D., Optimal Partial-Match Retrieval When Fields Are Independently Specified, *ACM TODS*, **4**, 2, 168—179 (June 1979).
5. Aigner M., Combinatorial Theory, Springer-Verlag, New York, 1979, p. 92.
6. Alford M. W., A Requirements Engineering Methodology for Real-Time Processing Requirements, *IEEE Trans. Softw. Eng.*, **SE-3**, 1, 60—69 (1977).
7. Anderson H. D., Berra P. B., Minimum Cost Selection of Secondary Indexes for Formatted Files, *ACM Trans. Database Syst.*, **2**, 68—90 (1977).
8. ANSI/X3/SPARC/Study Group-Database Management Systems, The ANSI/X3/SPARC DBMS Framework, *Inf. Syst.*, **3**, 3, 173—191 (1978).
9. Aronson J., Data Compression: A Comparison of Methods, in U. S. Dept. of Commerce, Washington, DC, Computer Science and Technology, NBS Spec. Publ. 500—12, June 1977, 31 pp.
10. Aschim F., Some Design and Analysis Tools for Design of Databases for Database Oriented Information Systems, Central Institute for Industrial Research, Oslo, Norway, Feb. 1975.
11. Ashany R., Application of Sparse Matrix Techniques to Search, Retrieval, Classification and Relationship Analysis in Large Database Systems, Proc. 4th Int. Conf. Very Large Data Bases (ACM), 1978, pp. 499—516.
12. Astrahan M. M., Chamberlin D. D., Implementation of a Structured English Query Language, *Commun. ACM*, **18**, 10, 580—588 (Oct. 1975).
13. Astrahan M. M. et al., System R: Relational Approach to Database Management, *ACM TODS*, **1**, 2, 97—137 (1976).
14. Babad J. M., A Record and File Partitioning Model, *Commun. ACM*, **20**, 1, 22—30, (Jan. 1977).
15. Bach T. J., Goguen N. H., Kaplan M. M., The ADAPT System: A Generalized Approach towards Data Conversion, Proc. 5th Int. Conf. Very Large Data Bases (ACM), 1979, pp. 183—193.
16. Bachman C. W., Data Structure Diagrams, *Database*, **1**, 2, 4—10, (1969).
17. Bachman C. W., The Evolution of Storage Structures, *Commun. ACM* 15, 7, 628—624, (1972).
18. Bachman C. W., Implementation Techniques for Data Structure Sets, in Data Base Management Systems, D. A. Jardine, ed., North Holland, Amsterdam, 1974.
19. Bachman C. W., Daya M., The Role Concept in Data Models, Proc. 3rd Int. Conf. Very Large Data Bases (ACM), 1977, pp. 464—476.
20. Bayer R., Symmetric Binary B-trees: Data Structure and Maintenance Algorithms, *Acta Inf.*, **1**, 4, 290—306, (1972).
21. Bayer R., McCreight E., Organization and Maintenance of Large Ordered Indexes, *Acta Inf.*, **1**, 3, 173—189, (1972).

22. Bayer R., Schkolnick M., Concurrency of Operations on B-Trees, *Acta Inf.*, **9**, 1, 1—21 (1977).
23. Bayer R., Unterauer K., Prefix B-trees, *ACM TODS*, 2, 11—26 (Mar. 1977).
24. Belford G., Dynamic Data Clustering, Research in Data Management and Resource Sharing, University of Illinois at Urbana-Champaign, May 1975.
25. Bell T. E., Bixler D. C., Dyer M. E., An Extendable Approach to Computer-Aided Software Requirements Engineering, *IEEE Trans. Softw. Eng.*, **SE-3**, 1, 49—60 (1977).
26. Bentley J. L., Friedman J. H., Data Structures for Range Searching, *ACM Comp. Surv.*, **11**, 4, 397—409 (Dec. 1979).
27. Berelian E., Irani K. B., Evaluation and Optimization, Proc. 3rd Int. Conf. Very Large Data Bases (ACM), 1977, pp. 545—555.
28. Bernstein P. A., Synthesizing Third Normal Form Relations from Functional Dependencies, *ACM Trans. Database Syst.*, 1, 4, 277—298 (1976).
29. Bernstein P. A., Goodman N., Concurrency Control in Distributed Database Systems, *ACM Comput. Surv.*, 13, 2, 185—221 (1981).
30. Bernstein P. A., Swenson J. R., Tsichritzis D. C., A Unified Approach to Functional Dependencies and Relations, Proc. ACM-SIGMOD Int. Conf. Manage. Data (ACM), 1975, pp. 237—245.
31. Bernstein P. A., Shipman D. W., Wong W. S., Formal Aspects of Serializability in Database Concurrency Control, *IEEE Trans. Softw. Eng.*, **SE-5**, 3, 203—216 (1979).
32. Bernstein P. A., Shipman D. W., Rothnie J. B., Concurrency Control in a System for Distributed Databases, (SDD-1), *ACM TODS*, 5, 1, 18—51, (1980).
33. Birss E. W., Fry J. P., Generalized Software for Translating Data, Proc. 1976 Nat. Comput. Conf. AFIPS Press, Arlington, VA, 1976, pp. 889—899.
34. Blasgen M. W., Eswaran K. P., Storage Access in Relational Data Bases, *IBM Syst. J.*, 4, 363—377 (1977).
35. Blier R., Vorkaus A., File Organization in the SDC Time Shared Data Management (TDMS) System, Proc. IFIP Congr., 1968, pp. F92—F97.
36. Bloom B. H., Space/Time Trade-offs in Hash Coding with Allowable Errors, *Commun. ACM*, 13, 7, 422—426 (July 1970).
37. Boehm B. W., Software and Its Impact: A Quantitative Assessment, Datamation, 48—59, May 1973.
38. Boehm B. W., Software Engineering, TRW Tech. Rep. TRW-SS-76-98, Oct. 1976, 40 pp.
39. Bolour A., Optimality Properties of Multiple Key Hashing Functions, *J. ACM*, **26**, 2, 196—210 (Apr. 1979).
40. Brinch Hansen P., Operating System Principles, Prentice-Hall, Englewood Cliffs, NJ, 1973, pp. 55—141.
41. British Computer Society, Data Dictionary Systems Working Party, *Data Base*, **9**, 2 (1977), *SIGMOD Record*, **9**, 4, 24 (1977).
42. Brown L., Data Base Review Methodology and IMS On-Line Performance Guide-lines, Data Systems Center, University of Michigan, Ann Arbor, MI, Mar. 1977.
43. Bubenko J. A., IAM: An Inferential Abstract Modeling Approach to Design of Conceptual Schema, Proc. ACM/SIGMOD Int. Conf. Manage. Data (ACM), 1977a, pp. 62—74.
44. Bubenko J. A., IAM: Inferential Abstract Modeling — An Approach to Design of Information Models for Large Shared Data Bases, IBM Res. Rep. No. RC6343 (27297), Jan. 4, 1977b, 79 pp.
45. Bubenko J. A., Validity and Verification Aspects of Information Modeling, Proc. 3rd Int. Conf. Very Large Data Bases, Tokyo (ACM), 1977c, pp. 556—565.
46. Bubenko J., Berild S., Lindencrona-Ohlin E., Nachmens S. A. From Information Requirements to DBTG Data Structures, Proc. ACM/SIGMOD/SIG-

PLAN Conf. Data: Abstraction, Definition and Structure (ACM), 1976, pp. 73—85.
47. Buchholz W., File Organization and Addressing, *IBM Syst. J.*, 2, 86—111 (June 1963).
48. Burkhard W. A., Hashing and Trie Algorithms for Partial-Match Retrieval, *ACM TODS*, 1, 2, 175—187 (June 1976).
49. Burkhard W. A., Partial-Match Hash Coding: Benefits of Redundancy, *ACM TODS*, 4, 2, 228—239 (June 1979).
50. Burroughs Corporation, B6700/B7700 DMS II Data and Structure Definition Language (DASDL) Reference Manual, Burroughs, Corp., Detroit, MI, 1974.
51. Caine S. H., Gordon E. K., PDL: A Tool for Software Design, Proc. 1975 Natl. Comput. Conf. (AFIPS), Vol. 44, AFIPS Press, Montvale, NJ, 1975, pp. 271—276.
52. Canning R. G., The Data Administrator Function, *EDP Anal.*, 10, 11 (Nov. 1972).
53. Canning R. G., The Data Dictionary/Directory Function, *EDP Anal.*, 12, 10 (1974).
54. Canning R. G., Getting the Requirements Right, *EDP Anal.*, 15, 7 (1977).
55. Canning R. G., Installing a Data Dictionary, *EDP Anal.*, 16, 1 (1978).
56. Cardenas A. F., Evaluation and Selection of File Organization — A Model and System, Commun. ACM 16, 9, 540—548 (1972).
57. Gardenas A. F., Analysis and Performance of Inverted Data Base Structures, *Commun. ACM*, 13, 5, 253—264 (1975).
58. Cardenas A. F., Data Base Management Systems, Allyn and Bacon, Boston, 1979.
59. Gardenas A. F., Sagamang J. P., Doubly-Chained Tree Data Base Organization — Analysis and Design Strategies, *Comput. J.*, 20, 1, 15—26 (1977).
60. Casey R. B., Allocation of Copies of a File in an Information Network, Proc. AFIPS Spring Joint Comput. Conf., Vol. 40, AFIPS Press, Arlington, VA, 1972, pp. 617—625.
61. Casey R. G., Design of Tree Networks for Distributed Data, AFIPS NCC Proc. 42 (1973a), pp. 251—257.
62. Casey R. G., Design of Tree Structures for Efficient Querying, *Commun. ACM*, 16, 549—556 (1973b).
63. Chamberlin D., Relational Data-Base Management Systems, *Comput. Surv.*, 8, 1, 43—66 (1976).
64. Chamberlin D. D., Boyce R. F., Sequel: A Structured English Query Language, Proc. 1974 ACM SIGFIDET Workshop, Apr. 1974, pp. 249—264.
65. Chandy K. M., Brown J. C., Dissly C. W., Uhrig M. R., Analytic Models for Rollback and Recovery Strategies in Database Systems, *IEEE Trans. Softw. Eng.*, SE-1, 1, 100—110 (Mar. 1975).
66. Chang D., A Stepwise Distributed Database Design, Database Systems Research Group Tech. Memo 79DS 1.1(R), University of Michigan, Ann Arbor, MI, May 6, 1980.
67. Chang S.-K., Cheng W.-H., Database Skeleton and Its Application to Logical Database Synthesis, *IEEE Trans. Softw. Eng.*, SE-4, 1, 18—30 (1978).
68. Chen P., The Entity-Relationship Model — Towards a Unified View of Data, *ACM TODS*, 1, 1, 9—36 (Mar. 1976).
69. Chen P., The Entity-Relationship Model — A Basis for the Enterprise View of Data, Proc. AFIPS Conf., Vol. 46, AFIPS Press, Arlington, VA, 1977a, pp. 77—84.
70. Chen P., The Entity-Relationship Approach to Logical Data Base Design, Q. E. D. Monograph Series, Wellesley, MA, 1977b.
71. Chen P., Applications of the Entity-Relationship Model, NYU Symp. Database Design, May 1978, pp. 25—33.

72. Chen P., Yao S. B., Design and Performance Tools for Database Systems, Proc. 3rd Int. Conf. Very Large Data Bases (ACM), 1977, pp. 3—15.
73. Chen D. D., Fry J. P., Teorey T. J., The Hierarchical Evaluator, Rep. No. DSRG 797DE6.2, Database Syst. Res. Group, University of Michigan, Ann Arbor, MI, Aug. 1979.
74. Chu W. H. W., Optimal File Placement in a Computer Network, Computer Communication Networks, N. Abramson, F. Kuo, eds., Prentice-Hall, Englewood Cliffs, NJ, 1973, pp. 82—94.
75. Chu W. H. W., Performance of File Directory Systems for Databases in Distributed Networks, Proc. 1976 Natl. Comput. Conf. (ACM), Vol. 45 (1976), pp. 577—587.
76. Clark J. D., An Attribute Access Probability Determination Procedure, Ph. D. thesis, Case Western Reserve University, School of Management, June 1977.
77. Clark J. D., Hoffer J. A., A Procedure for the Determination of Attribute Access Probabilities, Proc. ACM/SIGMOD Int. Conf. Manage. Data (ACM), 1978, pp. 110—117.
78. Clark J. D., Hoffer J. A., Physical Database Record Design, Q. E. D. Monograph Series, Q. E. D. Information Systems, Inc., Wellesley, MA, 1979, 110 pp.
79. CODASYL — Storage Structure Definition Language Task Group (SSDLTG) of CODASYL Systems Committee, Introduction to Storage Structure Definition, (by J. P. Fry); Informal Definitions for the Development of a Storage Structure Definition Language, (by W. C. McGee); A Procedural Approach to File Translation (by J. W. Young, Jr.); Preliminary Discussion of a General Data to Storage Structure Mapping Language (by E. H. Sibley, R. W. Taylor), Proc. ACM-SIGFIDET Workshop Data Description, Access, Control, E. F. Codd, ed., ACM, New York. Nov. 1970, pp. 368—380.
80. CODASYL Data Base Task Group, April 1971 Report, ACM, New York, 1971a.
81. CODASYL Systems Committee, Feature Analysis of Generalized Data Base Management Systems, ACM, New York, 1971b.
82. CODASYL Systems Committee, Selection and Acquisition of Data Base Management Systems, ACM, New York, Mar. 1976.
83. CODASYL — The Stored-Data Definition and Translation Task Group, Stored-Data Description and Data Translation: A Model and Language, *Inf. Syst.*, 2, 3, 95—148 (1977).
84. CODASYL Data Description Language Committee, *J. Dev.*, 1978.
85. Codd E. F., A Relational Model of Data for Large Shared Data Banks, *Commun. ACM*, 13, 6, 377—387 (1970).
86. Coffman E. G., Eve J., File Structures Using Hashing Functions, *Commun. ACM*, 13, 7, 427—432 (July 1970).
87. Comer D., The Difficulty of Optimum Index Selection, *ACM TODS*, 3, 4, 440—445 (Dec. 1978).
88. Comer D., Heuristics for Trie Index Minimization, *ACM TODS*, 4, 3, 383—395, (Sept. 1979a).
89. Comer D., The Ubiquitous B-Tree, *ACM Comput. Surv.*, 11, 2, 121—137 (June 1979b).
90. Courtice R. M., Data Base Design Using IMS/370, Proc. AFIPS 1972 Fall Joint Comput. Conf., Vol. 41, Thompson Book Co., Washington, DC, pp. 1105—10.
91. Curtice R. M., Data Base Design Using a CODASYL System, Proc. 1974 Natl. Comput. Conf. (ACM), Vol. 43, AFIPS Press, Arlington, VA, pp. 473—480.
92. Curtice R. M., Access Mechanisms and Data Structure Support in Data Base Management Systems, Q. E. D. Monograph Series No. 1, Q. E. D. Information Systems, Inc., Wellesley, MA, 1975.

93. Curtice R. M., Jones P. E., Key Steps in the Logical Design of Data Bases, NYU Symp. Database Design, May 1978, pp. 51—66.
94. Das K. Sundar, A Scheduling Methodology for Computer Operations, Ph. D. thesis, Dept. of Industrial and Operations Engineering, University of Michigan, Ann Arbor, 1977.
95. Database Administration Working Group, B. C. S./CODASYL DDIC, Rep. British Computer Society, London, June 1975.
96. Data Dictionary System, Technical Overview, International Computers Ltd., London, 1977.
97. Date C. J., An Introduction to Database Systems, Addison-Wesley, Reading, MA, 1975. [Имеется перевод: Дейт К. Введение в системы баз данных. — М.: Наука, 1980, 464 с.]
98. Davenport R. A., Data Analysis for Database Design, *Aust. Comput. J.*, **10**, 4, 122—137 (Nov. 1978).
99. Davenport R. A., Design of Distributed Database Systems, Infotech State of the Art Report: Distributed Databases, Vol. 2, Infotech International, Maidenhead, Berkshire, England, 1979, pp 87—114.
100. Davis G. B., Management Information Systems: Conceptual Foundations, Structure and Development, McGraw-Hill, New York, 1974, p. 278.
101. Davis C. G., Vick C. R., The Software Development System, *IEEE Trans. Softw. Eng.*, **SE-3**, 1, 70—84 (1977).
102. DeBlasis J. P., Johnson T. H., Database Administration — Classical Pattern, Some Experiences and Trends, Proc. AFIPS NCC, Vol. 46, AFIPS Press, Arlington, VA, 1977, pp. 1—7.
103. DeBlasis J. P., Johnson T. H., Review of Database Administrators Functions from a Survey, Proc. ACM/SIGMOD Int. Conf. Manage. Data (ACM), 1978, pp. 101—109.
104. De la Briandais R., File Searching Using Variable Length Keys, Proc. 1959 West. Joint Comput. Conf., pp. 295—298.
105. Denning D. E., Denning P. J., Data Security, *ACM Comput. Surv.*, **11**, 3, 227—249 (Sept. 1978).
106. Deppe M. E., A Relational Interface Model for Database Restructuring, Tech. Rep. 76 DT 3, Data Translation Project, Graduate School of Business Administration, University of Michigan, Ann Arbor, MI, 1976.
107. DeSmith D., Fry J. P., CODASYL Report on Distributed Database Systems, 1980.
108. DeSmith D. A., Aghili H., Grocock M., Matthews W A Distributed Database Management System for the IBM Series/1: Functional Capabilities, 79DS8.1(R), Database Systems Research Group, University of Michigan, Ann Arbor, MI, 1979.
109. Dijkstra E. W., A. Discipline of Programming, Prentice-Hall, Englewood Cliffs, NJ, 1976. [Имеется перевод: Дейкстра Э. Дисциплина программирования. — М.: Мир, 1978, 280 с.]
110. Drake R. W., Smith J. L., Some Techniques for File Resovery, *Aust. Comput. J.*, 3, 4, 162—170 (Nov. 1971).
111. Eisner M. J., Severance D. G., Mathematical Techniques for Efficient Record Segmentation in Large Shared Databases, *J. ACM 23*, 4, 619—635 (1976).
112. Everest G. C., Characteristics of Inter-entity Relationships and a Graphical, Notation, MISRC-WP-77-04, Graduate School of Business Administration, University of Minnesota, Minneapolis, MN, June 1977
113. Fagin R., Multi-valued Dependencies and a New Normal Form for Relational Data-bases, *ACM Trans. Database Syst.*, **2**, 3, 262—278 (1977).
114. Fagin R., Normal Forms and Relational Database Operations, Proc. ACM/SIGMOD Int. Conf. Manage. Data (ACM), 1979, pp. 153—160.
115. Fagin R., Nieverrgelt J., Pippenger N., Strong H. R., Extendible Hashing — A Fast Access Method for Dynamic Files, *ACM TODS*, **4**, 3, 315—344 (Sept. 1979).

116. Falkenberg E., Concepts for Modelling Information, in Modelling in Data Base Management Systems, G. M. Nijssen, ed., IFIP'76, North-Holand, Amsterdam, 1976, pp. 95—110.
117. Foster C. C., Information Storage and Retrieval Using AVL Trees, Proc. ACM 20th Natl. Conf., 1965, pp. 192—205
118. Foster C C., A Generalization of AVL Trees, *Commun. ACM*, **16**, 8, 513—517 (Aug. 1973).
119. Fredkin E., Trie Memory, *Commun. ACM*, **3**, 9, 490—500 (Sept. 1960).
120. Fry J. P., prep. The Technology of Data Base Translation — Data Base Restructuring, Auerbach Information Management Series: Current Directions in DBM Development (24-01-11), Auerbach Publishers, Pennsauken, NJ, 1978.
121. Fry J. P., Deppe M. E., Distributed Data Bases: A Summary of Research, *Comput. Networks*, **1**, 2, 1—13 (1976).
122. Fry J. P., Jeris D., Towards a Formulation of Data Reorganization, Proc. 1974 ACM/SIGMOD WorkshopData Description, Access, Control, R. Rustin, ed., ACM, New York, pp. 83—100.
123. Fry J. P., Kahn B. K., A Stepwise Approach to Database Design, Proc. ACM Southeast Reg. Conf., 1976, pp. 34—43.
124. Fry J. P., Maurer J., Operational and Technological Issues in Distributed Databases, Auerbach Database Management Series, Portfolio No. 24-01-02, 1977.
125. Fry J. P., Sibley E. A., Evolution of Database Management Systems, *Comput. Surv.*, **8**, 1, 7—42 (Mar. 1976).
126. Fry J. P., Teorey T. J., DeSmith D. A., Oberlander L. B., Survey of State-of-the-Art Database Administration Tools: Survey Results and Evaluation, with Appendixes A, B, and C, DSRG Tech. Rep. 78DE14.2, Database Systems Research Group, Graduate School of Business Administration, University of Michigan, Ann Arbor, MI, 1978a.
127. Fry J. P., DeSmith D. A., Oberlander L. B., Database Research and Systems Bibliography, attachment to DSRG Tech. Rep. 78DE14.2, Database Systems Research Group, Graduate School of Business Administration, University of Michigan, Ann Arbor, MI, 1978b.
128. Gambino T. J., Gerritsen R., A Data Base Design Decision Support System, Proc. 3rd Int. Conf. Very Large Data Bases (ACM), 1977, pp. 534—544.
129. Gane C., Sarson T., Structured System Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1979.
130. Gerritsen R., Understanding Data Structures, NTIS AD-A008-937, Springfield, VA, 1975a.
131. Gerritsen R., A Preliminary System for the Design of DBTG Data Structures, *Commun. ACM*, **18**, 10, 557—567 (Oct. 1975b).
132. Gerritsen R., Steps toward the Automation of Database Design, NYU Symp. Database Design, May 1978, pp. 91—99.
133. Ghosh S., Data Base Organization for Data Management, Academic Press, New York, 1976.
134. Ghosh S. P., Abraham C. T., Application of Finite Geometry in File Organization for Records with Multiple-Valued Attributes, *IBM. J. Res. Dev.*, **12**, 2, 180—187 (1968).
135. Gray J. N., Notes on Database Operating Systems, Operating Systems — An Advanced Course, Vol. 60, Lecture Notes in Computer Science, Springer-Verlag, New York, 1978, pp. 393—481.
136. Gray J. N., Lorie R. A., Putzolu G. R., Granularity of Locks and Degrees of Consistency in a Shared Database, Proc. 1st Int. Conf. Very Large Data Bases (ACM), Sept. 1975, pp. 428—451.
137. Guide International, Comparison of Data Base Management Systems, Oct. 1971.

138. Guide International, The Data Base Administrator, Nov. 1972.
139. Guide International, The Data Base Design Guide, Aug. 1974.
140. Haerder T., Implementing a Generalized Access Path Structure for a Relational Database System, *ACM Trans. Database Syst.*, 3, 3, 285—298 (1978).
141. Hall P., Owlett J., Todd S., Relations and Entities, in Modelling in Data Base Management Systems, G. M. Nijssen, ed., North-Holland, New York, 1976.
142. Hammad P., Raviart T., Formulation of Choice Criterions for File Organizations, *Inf. Syst.*, 3, 2, 123—130 (1978).
143. Hammer M., Chan A., Index Selection in a Self-Adaptive Data Base Management System, Proc. ACM/SIGMOD Int. Conf. Manage. Data (ACM), 1976a, pp. 1—8.
144. Hammer M., Chan A., Acquisition and Utilization of Access Patterns in a Relational Data Base Implementation, in Pattern Recognition and Artificial Intelligence, Academic Press, New York, 1976b, pp. 292—313.
145. Hammer M., Niamir B., A Heuristic Approach to Attribute Partitioning, Proc. ACM/SIGMOD Int. Conf. Manage. Data (ACM), 1979, pp. 93—100.
146. Hardgrave W. T., A Technique for Implementing a Set Processor, Proc. ACM/SIGMOD/SIGPLAN Conf. Data: Abstraction, Definition and Structure, Mar. 1976, pp. 86—94.
147. Hebalkar P. G., Zilles S. M., Graphical Representation and Analysis Information Systems Design, IBM Res. Rep. RJ2465, Jan. 1979.
148. Held G., Stonebraker M., B-trees Reexamined, Commun. *ACM*, 12, 2, 139—143 (Feb. 1978).
149. Hellerman H., Smith H. J., Jr., Throughput Analysis of Some Idealized Input, Output, and Compute Overlap Configurations, *ACM Comput. Surv.*, 2, 2, 111—118 (June 1970).
150. Hersney E. A. et al., Problem Statement Language Version 3.0 Language Reference Manual, Working Paper 68, ISDOS Research Project, Universtiy of Mich.gan, Ann Arbor, MI, May 1975.
151. Hibbard T., Some Combinatorial Properties of Certain Trees with Applications to Searching and Sorting, *J. ACM*, 9, 1, 13—28 (Jan. 1962).
152. Hoffer J. A., A Clustering Approach to the Generation of Subfiles for the Design of a Computer Database, Ph. D. thesis, Dept of Operations Research, Cornell University, Jan. 1975.
153. Hoffer J. A., Selection of Secondary Indexes in a Minicomputer Environment, Tech. Rep., Case Western Reserve University, 1978.
154. Hoffer J. A., A Survey of Primary and Secondary Keys through a Case Study, *Inf. Manage*, 2, 99—106 (1979).
155. Hoffer J. A., Severance D. G., The Use of Cluster Analysis in Physical Data Base Design, Proc. First Intl. Conf. VLDB, Framigham, MA, Sept. 1975, ACM, New York, pp. 69—86.
156. Housel B. C., Waddle V., Yao S. B., The Functional Dependency Model for Logical Database Design, Proc. 5th Int. Conf. Very Large Data Bases (ACM), Oct. 3—5, 1979, pp. 194—203.
157. Hsiao D., Harary F., A Formal System for Information Retrieval from Files, *Commun. ACM*, 14, 2, 67—73 (1970).
158. Hubbard G. U., Technique for Automated Logical Database Desing, NYU Symp. Database Design, May 1978, pp. 85—90.
159. Hubbard G., Raver N., Automating Logical File Design, in Proc. 1st Int. Conf. Very Large Data Bases (ACM), 1975, pp. 227—253.
160. Huffmann D. A., A Method for the Construction of Minimum Redundancy Codes, Proc. IRE 40 (Sept. 1952), pp. 1098—1104.
161. Hulten C., Soderlund L., A Simulation Model for Performance Analysis of Large Shared Data Bases, Proc. 3rd Int. Conf. Very Large Data Bases (ACM), 1977, pp. 524—532.

162. IBM, DBPROTOTYPE General Information Manual GH20-1272-0, IBM Mechanicsburg, PA, 1973.
163. IBM, IBM Data Base Design Aid-A Designer's Guide, Program No. 5748-XX4, GH20-1627-0, 1975.
164. IBM, Structured Walk-Throughs: A Project Management Tool, IBM, Aug. 1973.
165. IBM, IMS/VS System/Application Design Giude, NH20-0919, IBM, 1974.
166. IBM, OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide, IBM GC26-3838-2, 1976.
167. IBM, Planning for Enhanced VSAM under OS/VS, IBM GC26-3842-2, 1977.
168. IBM, IBM 3350 Direct Access Storage, GA26-1638, rev. 1979.
169. IBM, Introduction to IBM System/360 Direct Access Storage Devices and Organjzation Methods, IBM C20-1649-4, 1966.
170. INFOTECH, Database Technology, INFOTECH State of the Art Report, INFOTECH International, Maidenhead, Berkshire, England, 1978.
171. Innovation Data Processing, Inc. Fast Dump Restore and Data Set Functions, User Documentation, Innovation Data Processing, Inc., Clifton, NJ, July 1973.
172. Irani K., Purkayastha S., Teorey T. J., A Designer for DBMS — Processable Logical Database Structures, Proc. 5th Int. Conf. Very Large Data Bases (ACM), Oct. 3—5, 1979, pp 219—231.
173. Jackson M. A., Principles of Program Design, Academic Press, New York, 1975.
174. Janning M., Machmens S., Berild S, CS4: An Introduction to Associative Data Bases and the CS4 — System, Studentlitteratur, Lund, Sweden, ISBN 91-44-17111-0, 1981, 251 pp.
175. Jones P. E., Data Base Design Methodology — Logical Framework, Q. E. D. Monograph Series, Q. E. D. Information Systems, Inc., Wellesley, MA, 1976.
176. Kahn B. K., A Method for Describing the Information Required by the Data Base Design Process, Proc. Int. ACM/SIGMOD Conf. Manage. Data, 1976, pp. 53—64.
177. Kahn B., A Structured Logical Data-Base Design Methodology, in NYU Symp. Database Design, May 1978, pp. 15—24.
178. Karlton P., Fuller S., Scroggs R., Kachler E., Performance of Height Balanced Trees, *Commun. ACM*, **19**, 1, 23—28 (Jan. 1976).
179. Keehn D., Lacy J., VSAM Data Set Design Parameters, *IBM Syst. J.*, **13**, 3, 186—212 (1974).
180. Kent W., Entities and Relationships in Information, in Architecture and Models in Data Base Management Systems, G. Nijssen, ed., North-Holland, Amsterdam, 1977.
181. Kent W., Data and Reality: Basic Assumptions in Data Processing Reconsidered, North-Holland, Amsterdam, 1978, p. 211.
182. Kerschberg L., et al. A Taxonomy of Data Models, Tr. Csrb-70, Computer System Research Group, University of Toronto, Toronto, May 1976.
183. Khabbaz N. G., A Combined Communication Network Design and File Allocation for Distributed Databases, Ph. D. dissertation, University of Michigan, Ann Arbor, 1979.
184. King P. F., Collmeyer A. J., Database Sharing — An Efficient Mechanism for Supporting Concurrent Processes, Proc. AFIPS 1973 NCC, Vol. 42, AFIPS Press, Montvale NJ, 1973, pp. 271—275.
185. King W. F., On the Selection of Indices for a File, IBM Tech. Rep. BJ 1341, San Jose, CA, 1974.
186. Kleinrock L. Queuing Systems, Wiley, New York, 1975. [Имеется перевод: Клейнрок Л. Теория массового обслуживания. Пер. с англ. — М.: Машиностроение, 1979, 432 с.; Клейнрок Л. Вычислительные системы с очередями. Пер. с англ. — М.: Мир, 1979, 600 с.]

187. Knuth D. E., The Art of Computer Programming, Vol. 1: Fundamental Algorithms, Addison-Wesley, Reading, MA, 1968. [Имеется перевод: Кнут Д. Искусство программирования для ЭВМ, т. 1, Основные алгоритмы. — М.: Мир, 1972, 735 с.]
188. Knuth D. E., The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, MA, 1973. [Имеется первод: Кнут Д. Искусство программирования для ЭВМ. Т. 3, Сортировка и поиск — М.: Мир, 1978, 844 с.]
189. Kobayashi H., Modeling and Analysis: An Introduction to System Performance Evaluation Methodology, Addison-Wesley, Reading, MA, 1978, 446 pp.
190. Lampson B., Sturgis H., Crash Recovery in a Distributed Data Storage System, Tech. Rep., Comp. Sci. Lab., Xerox Palo Alto Research Center, Palo Alto, CA,, 1976.
191. Landauer W. I., The Balanced Tree and Its Utilization in Information Retrieval, *IEEE Trans. Electron. Comput.*, EC-12, 5, 863—871 (Dec. 1963).
192. Langefors B., Information Systems, Proc. IFIP Congr., North-Holland, Amsterdam, 1974, pp. 937—945.
193. Lefkovitz D., File Structures for Online Systems, Spartan Books, Rochelle Park, NJ, 1969.
194. Lefkowitz D., Data Management for Online System, Hayden, Rochelle Park, NJ, 1974.
195. Leong-Hong B., Marron B., Technical Profile of Seven Data Element Dictionary Directory Systems, NBS Spec. Publ. 500-3, U. S. Dept. of Commerce, Washington, DC, Feb. 1977.
196. Leong-Hong B., Marron B., Database Administration: Concepts, Tools, Experiences, and Problems, NBS Spec. Publ. 500—528, U. S. Dept. of Commerce, Washington, DC, Mar. 1978.
197. Liou J. H., Yao S. B., Multidimensional Clustering for Data Base Organizations, *Inf. Syst.*, 2, 4, 187—198 (1977).
198. Liu J. W. S., Algorithms for Parsing Search Queries in Systems with Inverted File Organizations, *ACM TODS*, 1, 4, 299—316 (Dec. 1976).
199. Lohman G. M., Muckstadt J. A., Optimal Policy for Batch Operations: Backup, Checkpointing, Reorganization, and Updating, *ACM Trans. Database Syst.*, 2, 3, 209—222 (1977).
200. Lowe T. C. The Influence of Data Base Characteristics and Usage on Direct Access File Organization, *J. ACM*, 15, 4, 535—548 (1968).
201. Lum V. Y., Multi-attribute Retrieval with Combined Indexes, *Commun. ACM*, 13, 1, 660—665 (Nov. 1970).
202. Lum V. Y., Yuen P. S. T., Additional Results on Key-To-Address Transform Techniques, *Commun. ACM*, 15, 11, 996—927 (1972).
203. Lum V. Y., Ling H., Senko M. E., Analysis of a Complex Data Management Access Method by Simulation Modeling, Proc. AFIPS 1970 Fall Joint Comput. Conf. AFIPS Press, Arlington, VA, 1970, pp. 211—222.
204. Lum V. Y., Yuen P. S. T., Dodd M. Key-to-Address Transformation Techniques: A Fundamental Performance Study on Large Existing Formatted Files, *Commun. ACM*, 14, 4, 228—239 (1971).
205. Lum V. Y., Senko M. E., Wang C. P., Ling H., A Cost Oriented Algorithm for Data Set Allocation in Storage Hierarchies, *Commun. ACM*, 18, 6, 318—332 (June 1975).
206. Lum V., et al. 1978 New Orleans Data Base Design Workshop Report, IBM Tech. Rep. No. RJ2554 (33154), July 1979a.
207. Lum V. Y., et al. 1978 New Orleans Data Base Design Workshop Report, Proc. 5th Int. Conf. Very Large Data Bases (ACM), Oct. 3—5, 1979b. pp. 328—339.
208. Lyon J. K., Data Base Administrator, Wiley-Interscience, New York, 1976a.

209. Lyon J. K., Introduction to Data Base Design, Wiley Interscience Communigraph Series on Business Data Processing, Wiley-Interscience, New York, 1976b.
210. Mahmoud S. A., Riordon J. S., Toth K. S., Distributed Database Partitioning and Query Processing, Proc. IFIP TC-2 Working Conf. Database Architecture, Venice, Italy, June 1979.
211. Manola F., An Evaluation of the New CODASYL and ANSI/SPARC Database Proposals, Infotech State of the Art Report: Data Base Technology, Vol. 2, Infotech International, Maidenhead, Berkshire, England, 1978, pp. 131—150.
212. March S. T., Models of Storage Structures and the Design of Database Records Based upon a User Charapcterization, Ph. D. dissertation, Cornell University, 1978.
213. March S. T., Severance D. G., The Determination of Efficient Record Segmentations and Blocking Factors for Shared Files, *ACM Trans. Database Syst.*, 2, 3 (1977), pp. 279—296.
214. March S. T, Severance D. G., A Mathematical Modeling Approach to the Automatic Selection of Database Designs, Proc. ACM/SIGMOD Int. Conf. Manage. Data, 1978, pp. 52—65.
215. March S. T., Severance D. G., Wilens M. E., Frame-Memory: A Storage Architecture to Support Rapid Design and Implementation of Efficient Databases, *ACM TODS*, 6, 3, 441—463 (1981).
216. Martin J., Design of Real-Time Computer Systems, Prentise-Hall, Englewood Cliffs, NJ, 1967.
217. Martin J., Teleprocessing Network Organization, Prentice-Hall, Englewood Cliffs, NJ, 1970. [Имеется перевод: Мартин Дж. Сети связи и ЭВМ. Пер. с англ. Ч. 1. — М.: Связь, 1974, 230 с.; Ч. 2. — М.: Связь, 1975, 208 с.]
218. Martin J., Computer Data-Base Organization, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1977. [Имеется перевод: Мартин Дж. Организация баз данных в вычислительных системах. — М.: Мир, 1980, 664 с.]
219. Maryanski F. J., Fisher P. S., Rollback and Recovery in Distributed Database Management Systems, Proc. 1977 ACM Annu. Conf., 1977, pp. 33—38.
220. Maurer W. D., An Improved Hash Code for Scatter Storage, *Commun. ACM*, 11, 1, 35—38 (Jan. 1968).
221. Maxwell W. L., Severance D. G., Comparison of Alternatives for the Representation of Data Items Values in an Information System, Proc. Wharton Conf. Res. Comput. Organ. Oper. Res 20—5 (1973), 16 pp.
222. McCormick W. T., Jr., Sweitzer P. J., White T. W. Problem Decomposition and Data Reorganization by a Clustering Technique, *Oper. Res.*, 20, 5, 993—1009 (Sept. — Oct. 1972).
223. McIlroy M. D., A Variant Method for File Searching, *Commun. ACM*, 6, 1, p. 101 (Jan. 1963).
224. Menasce D. A., Ropek G. J., Muntz R., A Locking Protocol for Resource Coordination in Distributed Database Systems, Supplement to ACM SIGMOD 1978, pp. 1—4.
225. Methlie L. B., Schema Design Using a Data Structure Matrix, *Inf. Syst.*, 3, 2, 81—91 (1978).
226. Miller G. A., The Magical Number Seven, Plus-or-minus Two: Some Limits on Our Capacity for Processing Information, *Psychol. Rev.*, 63, 2, 81—97 (Mar. 1956).
227. Mills H. D., Mathematical Foundations for Structured Programming, IBM Tech. Rep FSC72-6012, Feb. 1972.
228. Mitoma M. F., Optimal Data Base Schema Design, Ph. D. dissertation, University of Michigan, Ann Arbor, 1975.
229. Mitoma M. F., Irani K. B., Automatic Data Base Schema Design, Proc. 1st Int. Conf. Very Large Data Bases (ACM), 1975, pp. 286—321.

230. Mohan C., Yeh R. T., Distributed Database Systems — A Framework for Database Design, INFOTECH State of the Art Report: Distributed Database, Vol. 2, INFOTECH International, Maidenhead, Berkshire, England, 1979, pp. 237—256.
231. Mommens J., Smith S., Automatic Generation of Physical Data Base Structures, Proc. Int. ACM/SIGMOD Conf. on Manage. Data, 1975, pp. 157—165.
232. Morgan H. L., Levin H. D., Optimal Program and Data Locations in Computer Networks, *Commun. ACM*, **20**, 5, 315—322(1977).
233. Morris R., Scatter Storage Techniques, *Commun. ACM*, **11**, 1, 38—43 (1968).
234. Muntz R., Uzgalis R., Dynamic Storage Allocation for Binary Search Trees in a Two-Level Memory, Proc. 4th Annu. Princeton Conf., Princeton, NJ, 1970, pp. 539—549.
235. Nakamura F., Yoshida I., Kando H., A Simulation Model for Data Base System Performance Evaluation, Proc 1975 Natl. Comput. Conf., Vol. 44, AFIPS Press, Montvale, NJ, pp. 459—465.
236. Navathe S. B., Fry J. P., Restructuring for Large Data Bases: Three Levels of Abstraction, *ACM Trans. Database Syst.*, **1**, 2, 138—158 (1976).
237. Navathe S. B., Gadgil S. G., A Methodology for View Integration in Logical Database Design, Database Systems Research and Development Center Tech. Rep. University of Florida, 1980.
238. Navathe S. B., Schkolnick M., View Representation in Logical Database Design, Proc. ACM/SIGMOD Int. Conf. Manage. Data, 1978, pp. 144—156.
239. Niamir B., Attribute Partitioning in a Self-Adaptive Relational Database System, MIT Lab. Comput. Sci. Tech. Rep. 192, Cambridge, MA, Jan. 1978.
240. Nievergelt J., Binary Search Treess and File Organization, *ACM Comput. Surv.*, **6**, 3, 195—207 (Sept. 1974).
241. Nievergelt J., Reingold E. M., Binary Search Trees of Bounded Balance, *SIAM J. Comput.*, **2**, 1, 33—43 (Mar. 1973).
242. Nijssen G. M., Current Issues in Conceptual Schema Concepts, Proc. IFIP TC2 Conf., Nice, France, North-Holland, Amsterdam, Jan. 1977.
243. Novak D., Fry J., The State of the Art of Logical Database Design, Proc. 5th Texas Conf. Comput. Syst. (IEEE), Long Beach CA, 1976, pp. 30—39.
244. Oberlander L. B., Physical Design of Database Structures, Ph. D. thesis, Dept. of Computer and Communications Science, University of Michigan, Ann Arbor, 1979.
245. Olle W. T., UL-1 — A non-procedural Language for Retrieving Information from Data Bases, Information Processing — 68, North-Holland, Amsterdam (1968), pp. 572—578.
246. Olson C. A., Random Access File Organization for Indirectly Addressed Records, Proc. 1969 ACM Natl. Conf., pp. 539—549.
247. Oren O. A., Aschim F., Statistics for the Usage of a Conceptual Data Model as a Basis for Logical Data Base Design, Proc. 5th Int. Conf. Very Large Data Bases (ACM), Oct. 3—5, 1979, pp. 140—145.
248. Orr K. T. Structured Systems Design, Langston Kitch and Associates, Topeka, Kans., 1976.
249. Palmer I., Practicalities in Applying a Formal Methodology to Data Analysis, NYU Symp. Database Design, May 1978, pp. 67—84.
250. Parnas D. L., On the Criteria to be Used in Decomposing Systems into Models, *Commun. ACM*, **15**, 12, 1053—1058 (Dec. 1972).
251. Reterson W. W., Addressing for Random Access Storage, *IBM J. Res. Dev.*, **1**, 130—146 (Apr. 1957).
252. Proceedings of Wharton Seminar on Database Administration, University of Pennsylvania Press, Philadelphia, 1977.
253. Rameriz J. A., Rin N. A., Prywes N. S., Automatic Conversion of Data Conversion Programs Using a Data Description Language, Access and

Control, Proc. ACM/SIGFIDET Workshop Data Description (ACM), 1974, pp. 207—225.
254. Rappaport R. L., File Structure Design to Facilitate On-Line Instantaneous Updating, Proc. 1975 ACM/SIGMOD Conf., pp. 1—14.
255. Ries D. R., Stonebraker M. R., Locking Granularity Revisited, *ACM Trans. Database Syst.*, **4**, 2, 210—227 (June 1979).
256. Rivest R. L., Partial Match Retrieval Algorithms, *SIAM J. Comput.*, **5**, 1, 19—50 (Mar. 1976).
257. Roche D. J. M., Practical Aspects of Randomizing, IFIP, ADP Group, IAG, File Organization and Search Techniques Seminar Notes, Nov. 1969, pp. C-1 — C-24.
258. Rosenkrantz D., Sterns R., Lewis R., A System Level Councurrency Control for Distributed Database Systems, Proc. 2nd Berkeley Workshop, May 22—27, 1977.
259. Ross D. T., Structured Analysis (SA): A Language for Communicating Ideas, *IEEE Trans. Softw. Eng.*, **SE-3**, 1, 16—34 (1977).
260. Ross D. T., Schoman K. B., jr., Structured Analysis for Requirements Definition, *IEEE Trans. Softw. Eng.*, **SE-3**, 1, 6—15 (Jan. 1977).
261. Rothnie J B., Goodman N., An Overview of the Preliminary Design of SDD-1: A System for Distributed Database, Proc. 1977 Berkeley Workshop Data Manage. Comput. Networks, Lawrence Berkeley Lab., University of California, Berkeley, CA, May 1977, pp. 39—57.
262. Rothnie J. B. et al., Introduction to a System for Distributed Databases (SDD-1), *ACM Trans. Database Syst.*, **5**, 1, 1—17 (1980).
263. Rotwitt T., DeMaine P. A. D., Storage Optimization of Tree Structured Files Representing Descriptor Sets, Proc. 1971 SIGFIDET (SIGMOD) Workshop, San Diego, CA.
264. Roycroft A. J., Techniques for Handling Variable Length Logical Records on IBM Direct Access Storage Devices, Proc. FILE68 Int. Seminar File Org., Copenhagen, 1968, pp. 701—720.
265. Rubin F., Experiments in Text File Compression, *Commun. ACM*, **19**, 11, 617—623 (Nov. 1976).
266. Rund D. S., Data Base Design Methodology — Part I & II, Auerbach Data Base Management Series, Portfolios Nos. 23-01-01 and -02, 1977.
267. Salton G., Wang A., Generation and Search of Clustered Files, *ACM Trans. Database Syst.*, **3**, 4, 321—346 (1978).
268. Sayani H. H., Restart and Recovery in Transaction-Oriented Information Processing System, Proc. 1974 ACM/SIGMOD Workshop Data Description, Access, Control, May 1974, pp. 351—366.
269. Schkolnick M., The Optimal Selection of Secondary Indices for Files, *Inf. Syst.*, 1, 141—146 (1975).
270. Schkolnick M., A Clustering Algorithm for Hierarchical Structures, *ACM Trans. Database Syst.*, 2, 1, 27—244 (1977).
271. Schkolnick M., Physical Database Design Techniques, NYU Symp. Database Design, May 1978, pp. 99—110.
272. Scidmore A. K., Weinberg B. L. Storage and Search Properties of a Tree-organized Memory System, *Commun. ACM*, **6**, 1, 28—31 (Jan. 1963).
273. Selinger M. M., Astrahan M. M., Chamberlin D. D., Lorie R. A., Price T. G., Access Path Selection in a Relational Database Management System, Proc. ACM/SIGMOD Int. Conf. Manage. Data, 1979, pp. 23—34.
274. Senko M. E., Lum V. Y., Owens P. J., A File Organization Evaluation Model (FOREM), Proc. 1968 IFIP Cong., Spartan Books, Washington, DC, pp. C19 — C23.
275. Senko M. et al., Data Structures and Accessing in Data-base Systems, *IBM Syst. J.*, **12**, 1, 30—93 (1973).
276. Severance D. G., Some Generalized Modeling Structures for Use in Design of File Organization, Ph. D. dissertation, University of Michigan, Ann Arbor, 1972.

277. Severance D. G., Identifier Search Mechanisms: A Survey and Generalized Model, *Comput. Surv.*, **6**, 3, 175—194 (1974).
278. Severance D. G., Carlis J. V., A Practical Approach to Selecting Record Access Paths, *ACE Comput. Surv.*, **9**, 4, 259—272 (1977).
279. Severance D. G., Duhne R. A., A Practitioner's Guide to Addressing Algorithms, *Commun. ACM*, **19**, 6, 314—326 (1976).
280 Severance D. G., Lohman G. M. Differential Files: Their Application to the Maintenance of Large Databases, *ACM TODS*, **1**, 3, 256—267 (Sept. 1976).
281. Sharmen G., Winterbottom N., The Data Dictionary Facilities of NDB, in Proc. 4th Int. Conf. Very Large Data Bases (ACM), 1978, pp. 186—197.
282. Sheppard D. L., Data Base: a Business Approach to System Design, CONCOM Systems, Inc., Cincinnati, OH, Aug. 1974.
283. Sherman S. W., Brice R. S., Performance of a Database Manager in a Virtual Memory System, *ACM Trans. Database Syst.*, **1**, 4, 317—343 (1976a).
284. Sherman S. W., Brice R. S., An Extension of the Performance of a Database Manager in a Virtual Memory System Using Partially Locked Virtual Buffers, *ACM Trans. Database Syst.*, **2**, 2, 196—207 (1977b).
285. Shneiderman B., Optimum Data Base Reorganization Points, *Commun. ACM*, **16**, 6, 363—365 (1973).
286. Shneiderman B., Reduced Combined Indexes for Efficient Multiple Attribute Retrieval, *Inf. Syst.*, **2**, 4, 149—154 (1977), Pergamon Press, Oxford.
287. Shoshani A., A Logical-Level Approach to Data Base Conversion, Proc. 1975 ACM/SIGMOD Int. Conf. Manage Data (ACM), pp. 112—122.
288. Shoshani A., Brandon K., On the Implementation of a Logical Data Base Converter, Proc. First Int. Conf. Very Large Data Bases (ACM), 1975, pp. 529—531.
289. Shu N. C., Housel B.C., Lum V. Y., CONVERT: A High-Level Translation Language for Data Conversion, *Commun. ACM*, **18**, 10, 557—567 (1975).
290. Shu H. C. et al., EXPRESS: A Data Extraction, Processing, and Restructuring System, ACM TODS, June 1977, pp. 134—174.
291. Siler K. F. A Stochastic Evaluation Model for Database Organizations in Data Retrieval Systems, *Commun. ACM*, **19**, 2, 84—95 (Feb. 1976).
292. Smith J. M., Smith D. C. P., Database Abstractions: Aggregation and Generalization, *ACM Trans. Database Syst.*, **2**, 2, 105—133 (1977a).
293. Smith J. M., Smith D. C. P., Database Abstractions: Aggregation, *Commun. ACM*, **20**, 6, 405—413 (June 1977b).
294. Smith J. M., Smith D. C. P., Principles of Database Design, NYU Symp. Database Design, May 1978, pp. 35—50.
295. Sockut G. H., Goldberg R. P., Database Reorganization — Principles and Practice, *ACM Comput. Surv.*, **11**, 4, 371—395 (Dec. 1979).
296. Soderlund Lars., A Study on Concurrent Database Reorganization, Ph. D. thesis. of Information Processing and Computer Science, The Royal Institute of Technology and The University of Stockholm, Sweden, ISBN 91-85212-57-1, 1980
297. Sowa J., Conceptual Graphs for a Data Base Interface, *IBM J. Res. Dev.*, 20, 4, 336—357 (1976).
298. Sperry Univac, Data Management System (DMS1100) Level 8R1 System Support Functions Data Administrator Reference, Univac Publ. UP 7909.1, 1978.
299. Sprowls R. C., Management Data Bases, Wiley/Hamilton, Santa Barbara, CA 1976.
300. Stone H. S., Multiprocessor Scheduling with the Aid of Network Flow Algorithms, *IEEE Trans. Softw. Eng.*, **SE-3**, 1, 85—93 (Jan. 1977).
301. Stonebraker M., The Choice of Partial Inversions and Combined Indices, *J. Comput. Inf. Sci.*, 3, 2, 167—188 (1974).

302. Stonebraker M., Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES, *IEEE Trans. Softw. Eng.*, SE-5, 3, 1979, pp. 188—194.
303. Stonebraker M., Neuhold E., A Distributed Data Base Version of INGRES, Proc. Berkeley Workshop Distributed Data Manage. Comput. Networks, 1977, pp. 19—36.
304. Stonebraker M. et al., The Design and Implementation of INGRES, *ACM TODS*, Sept. 1976, pp. 189—222.
305. Strong H. R., Markowsky G., Chandra A. K. Search within a Page, *J. ACM*, **26**, 2, 457—482 (July 1979).
306. Sundgren B., An Infological Approach to Databases, Ph. D. dissertation, University of Stockholm, Sweden, 1973.
307. Sundgren B., Data Base Design in Theory and Practice — Towards an Integrated Methodology, Proc. 4th Conf. Very Large Data Bases (ACM), 1978, 3—26 (with comments pp. 17—30).
308. Sussenguth E. H., Jr., Use of Tree Structures for Processing Files, *Commun. ACM*, **6**, 5, 272—279 (May 1963).
309. Sutherland W. R., Distributed Computation Research at BBN, III BBN Tech. Rep. 2976, Bolt, Beranek and Newman, Dec. 1974.
310. Swartwout D., An Access Path Specification Language for Restructuring Network Databases, Proc. 1977 ACM/SIGMOD Int Conf. Manage. Data (ACM), 1977, pp. 88—101.
311. Swartwout D.E., Deppe M. E., Fry J. P., Operational Software for Restructuring Network Databases, Proc. 1977 Natl. Comput. Conf., Vol. 46, AFIPS Press, Arlington, VA, 1977a, pp. 499—508.
312. Swartwout D. E., Wolfe G. J., Burpee C. E., Translation Definition Language Reference Manual for Version IIa Translator, Release 3, Working Paper 77 DT 5.3, Data Translation Project, University of Michigan, Ann Arbor, MI, 1977b.
313. Taggart W. M., Jr., Tharp M. O., A survey of Information Requirements Analysis Techniques, *ACM Comput. Surv.*, **9**, 4, 273—290 (Dec. 1977).
314. Taylor R. et al., Database Program Conversion: A Framework for Research, Proc. 5th Int. Conf. Very Large Data Bases (ACM), 1979, pp. 299—312.
315. Teichroew D., Hershey E. A., PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems, *IEEE Trans. Softw. Eng.*, SE-3, 1, 41—48 (1977).
316. Teichroew D., Sayani H., Automation of System Building, Datamation, 25—30, Aug. 1971.
317. Teichroew D., Hershey E. A., III, Bastarche M. J., An Introduction to PSL/PSA, ISDOS Workshop Paper 86, University of Michigan, Ann Arbor, MI, 1974.
318. Teorey T. J., General Equations for Idealized CPU-I/O Overlap Configurations, *Commun. ACM*, **21**, 6, 500—507 (June 1978).
319. Teorey T. J., Das K. S., Application of an Analytical Model to Evaluate Storage Structures, Proc. Int. ACM/SIGMOD Conf. Manage. Data, 1976, pp. 9—20.
320. Teorey T. J., Fry J. P., Logical Database Design: A Pragmatic Approach, INFOTECH State of the Art Report, INFOTECH International, Maidenhead, Berkshire, England, 1978, pp. 357—383.
321. Teorey T. J., Fry J. P., The Logical Record Access Approach to Database Design, *ACM Comput. Surv.*, **12**, 2, 179—211 (1980). Corrigendum 12, 4, 465 (1980).
322. Teorey T. J., Oberlander L. B., Network Database Evaluation Using Analytical Modeling, Proc. 1978, Natl. Comput. Conf., Vol. 47, AFIPS Press, Arlington, VA, 1978, 833—842.

323. Toth K. C., Mahmoud S. A., Riordon J. S., Sherif O., The ADD System: An Architecture for Distributed Databases, Proc. 4th Int. Conf. Very Large Data Bases (ACM), 1978, pp. 462—471.
324. Towsley D., Chandy K. M., Browne J. C., Models for Parallel Processing within Programs: Application to CPU : I/O and I/O : I/O Overlap, *Commun. ACM*, **21**, 10, 821—831 (Oct. 1978).
325. Tozer E. F., Data Systems Analysis and Design, Proc. I Conf. European Coop Informatics, K. Samelson, ed., Lecture Notes in Computer Science, Springer-Verlag, New Work, 1976, pp. 193—224.
326. Tremblay J., Sorenson P. G., An Introduction to Data Structures with Applications, McGraw-Hill, New York, 1976.
327. Tuel W. G., Jr., Optimum Reorganization Points for Linearly Growing Files, *ACM Trans. Database Syst.*, **2**, 3, 32—40 (1977).
328. Turnburke V. P., Jr., Sequential Data Processing Design, *IBM Syst. J.*, **2**, 37—48 (Mar. 1963).
329. Uhrowczik P. P., Data Dictionary/Directories, *IBM Syst. J.*, **12**, 4, 332—350 (1973).
330. Verhofstad J. J. M., Recovery Techniques for Database Systems, *ACM Comput. Surv.*, **10**, 2, 167—195 (June 1978).
331. Wagner R. E., Indexing Design Considerations, *IBM Syst. J.*, **12**, 4, 352—367 (1973).
332. Wang C. P., Wedekind H. H., Segment Synthesis in Logical Data Base Design, *IBM J. Res. Dev.*, **19**, 1, 71—77 (1975).
333. Wasserman A. I., Information System Design Methodology, *J. Am. Soc. Inf. Sci.*, **31**, 1 (Jan. 1980).
334. Webster's New World Dictionary of the American Language, D. B. Guralnik, ed., William Collins and World Publishing Co., New York, 1974.
335. Wedekind H., On the Selection of Access Paths in a Data Base System, in Data Base Management, J. W. Klimbie, K. L. Koffeman, eds., North-Holland, Amsterdam, 1974, pp. 385—397.
336. Whitney V. K. M., A Study of Optimal File Assignment and Communication Networks Configuration, Ph. D. thesis, University of Michigan, Ann Arbor, 1970.
337. Wiederhold G., Database Design, McGraw-Hill, New York, 1977.
338. Wilens M. E., Volz R. A., Fry J. P. Interactive Database Design Laboratory, Tech. Rep. 78 DE 13, Database Systems Research Group, Graduate School of Business Administration, University of Michigan, Ann Arbor, MI, Apr. 1978.
339. Winkler A. et al., The Data Admdinistrator's Handbook, U. S. Air Force Academy Unclassified Rep. USAF-TR-76-1, National Technical Information Service, Springfield, VA, Jan. 1976.
340. Wirth N., Program Development by Stepwise Refinement, *Commun. ACM*, **14**, 4, 221—227 (Apr. 1971).
341. Wirth N., On the Composition of Well-Structure Programs, *ACM Comput. Surv.*, **6**, 4, 247—259 (Dec. 1974).
342. Wong E., Youssefi K., Decomposition — A strategy for Query Processing, *ACM TODS*, **1**, 3, 223—241 (Sept. 1976).
343. Yamamoto S., Tazawa S., Ushio K., Ikeda H., Design of a Generalized Balanced Multiple-Valued File Organization Scheme of Order Two, Proc. ACM/SIGMOD Int. Conf. Manage. Data, 1978, pp. 47—51.
344. Yang C., A Class of Hybrid List File Organizations, *Inf. Syst.*, **3**, 49—58 (1978).
345. Yao S. B., Evaluation and Optimization on File Organizations through Analytic Modeling, Ph. D. dissertation, University of Michigan, Ann Arbor, 1974.
346. Yao S. B., An Attribute Based Model for Database Access Cost Analysis, *ACM Trans. Database Systems*, **2**, 1, 45—67 (1977a).

347. Yao S.B., Approximating Block Acesses in Database Organizations, *Commun. ACM*, **20**, 4, 260—261 (1977b).
348. Yao A., On Random 2—3 Trees, *Acta. Inf.*, **9**, 2, 159—170 (1978).
349. Yao S. B., Optimization of Query Evaluation Algorithms, *ACM TODS*, **4**, 2, 133—155 (June 1979).
350. Yao S. B., De Jong D., Evaluation of Database Access Paths, Proc. ACM/SIGMOD Int. Conf. Manage.Data, 66—77, 1978.
351. Yao S. B., Das K. S., Teorey T. J., A Dynamic Data Base Reorganization Algorithm, *ACM Trans. Database Syst.*, **1**, 2, 159—174 (1976).
352. Yoa S. B., Navathe S. B., Weldon J. L., An Integrated Approach to Logical Database Design, NYU Symp. Database Design, May 1978, pp. 1—14.
353. Yeh R. T., Roussopoulos N., Chang P., Data Base Design — An Approach to Logical Database Design, INFOTECH State of the Art Report on Data Base Technology, INFOTECH International, Maidenhead, Berkshire, England, 1978a, 413—477.
354. Yen R. T., Chang P., Mohan C., A Multi-level Data Base Design Approach, Proc. COMPSAC 1978, Chicago, 1978b, pp. 370—375.
355. Yourdon E., Design of On-Line Computer Systems, Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 340—353, 515—542.
356. Yourdon E., Constantine L. L., Structured Design, Prentice-Hall, Englewood Cliffs, NJ, 1979.
357. Zloof M. M., Query by Example, Proc. NCC, 1975, AFIPS Press, pp. 431—438.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ[1]

[1]) Звездочкой отмечены номера страниц во 2-ом томе.

Оглавление

Часть V. Специальные вопросы проектирования

Приложение Б

Приложение В