Лекция 7. Этап физического проектирования. Основные структуры хранения и методы доступа к данным.

7.1 Цели и задачи этапа физического проектирования 1
7.2 Основные понятия физического хранения данных 1
7.3 Основные структуры хранения физических записей и методы доступа к ним 4
7.3.1 Последовательная структура хранения и последовательный доступ 4
7.3.2 Индексные структуры хранения и методы доступа 4
7.3.2.1 Плотные и неплотные индексы 7
7.3.2.2 Структуры типа Б-дерева 8
7.3.3 Хэширование 9
7.4 Поиск текстовых значений 10

7.1 Цели и задачи этапа физического проектирования

Этап физического проектирования состоит в отображении логической схемы базы данных (созданной, например, в СУБД ORACLE) во внутреннюю схему базы данных, которая поддерживается данной СУБД.
Как упоминалось в лекции о трехуровневой архитектуре систем баз данных, при описании внутренней схемы используется свой особый язык описания данных. Этот язык оперирует внутренними (хранимыми) записями.
Будем различать **физические записи** (т.е. записи в файле данных) и **логические записи** (т.е. например, строки реляционной таблицы, или записи в иерархической структуре).

Общую задачу этапа физического проектирования сформулируем так:

Как можно представить логические структуры данных в виде некоторой оптимальной структуры физического хранения значений данных?

Общая задача этапа физического проектирования разделяется на несколько проблем:
Проблема 1. Как найти нужную запись среди всех записей базы?
Проблема 2. Как организовать эффективный поиск записи или группы записей?
Проблема 3. Как организовать физическое размещение записей с максимальной плотностью?
Проблема 4. Как обеспечить минимальное время реакции системы базы данных?

7.2 Основные понятия физического хранения данных

Под **структурой хранения** понимаем любое упорядочение данных на диске. Можно организовать различные структуры хранения, обладающие различной производительностью и оптимальные для различных способов использования. Очевидно, что не существует некой идеальной структуры хранения, оптимальной для всех задач. Поэтому, часто в рамках даже одной СУБД поддерживается несколько разных структур хранения. Основным критерием оптимальности структуры хранения является **число дисковых операций ввода/вывода данных**.

Прежде чем обратиться к известным путям решения указанных проблем, рассмотрим взаимодействие СУБД и операционной системы для работы с физическими записями (см. Рис.7.1):

1. СУБД определяет искомую запись, и для ее извлечения запрашивается **диспетчер файлов.**

2. Диспетчер файлов определяет страницу, на которой находится искомая запись, а затем для извлечения этой страницы запрашивается **диспетчер дисков**.
3. Диспетчер дисков определяет физическое положение искомой страницы на диске и посылает соответствующий запрос на ввод/вывод данных.

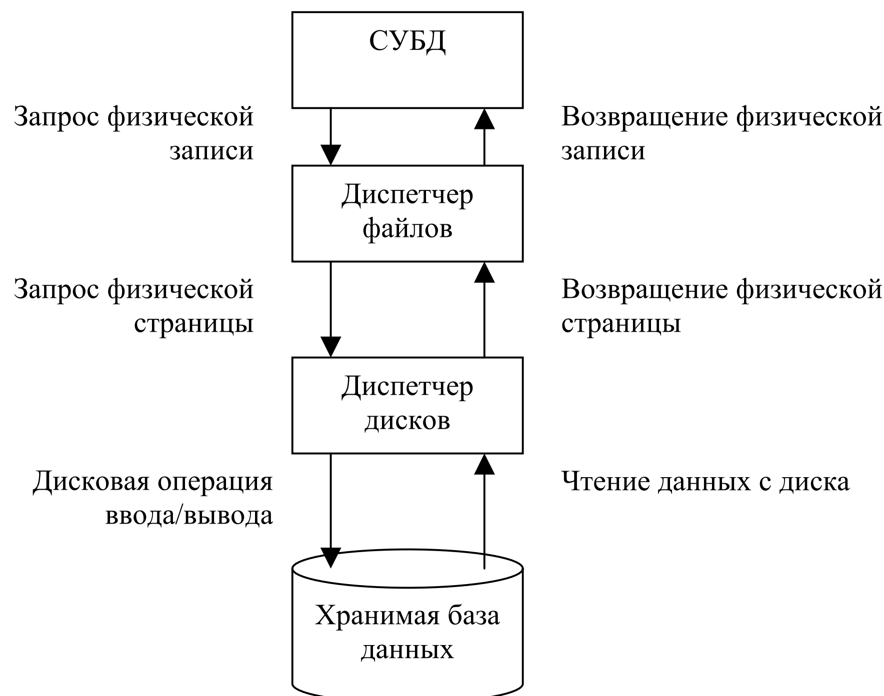


Рисунок 7.1 – Схема взаимодействия СУБД, диспетчера файлов и диспетчера дисков.

Рассмотрим компоненты взаимодействия при доступе к физическим записям подробнее.

Диспетчер дисков

Диспетчер дисков является компонентом операционной системы, с помощью которого выполняются все дисковые операции ввода/вывода. Для выполнения этих операций необходимо знать значения **физических адресов на диске**. Однако пользователю, т.е. диспетчеру файлов, не обязательно знать физические адреса. Вместо этого диспетчеру файлов достаточно рассматривать диск как **набор страниц** фиксированного размера, с уникальным **идентификационным номером набора страниц**. Каждая **страница**, в свою очередь, обладает уникальным внутри данного набора **идентификационным номером страницы**, причем наборы не имеют общих страниц. Соответствие физических адресов на диске и номеров страниц достигается с помощью диспетчера дисков. Главным преимуществом такой организации является изоляция программного кода, зависящего от конкретного устройства диска, внутри одного компонента ОС, а именно, внутри диспетчера дисков. В этом случае, все компоненты высокого уровня, в частности, диспетчеры файлов, могут быть аппаратно независимыми. Существует специальный набор страниц – **набор пустых страниц**, который содержит все имеющиеся свободные страницы. Введение в использование или освобождение страниц осуществляется диспетчером дисков по запросу диспетчера файлов.

Основные операции диспетчера дисков таковы:

- извлечь страницу *p* из набора страниц *s*;
- заменить страницу *p* из набора страниц *s*;

- добавить новую страницу в набор страниц *s* (т.е. извлечь одну страницу из набора пустых страниц и возвратить новую страницу с номером *p*;
- удалить страницу *p* из набора страниц *s* (т.е. возвратить одну страницу с номером *p* в набор пустых страниц;

Как правило, диспетчер дисков размещает или удаляет страницы в наборах не по одной, а целыми группами физически связанных страниц, или **блоками** (**экстентами; экстент – непрерывная область на диске**), например по 8 страниц сразу.
(например, для MS SQL Server 2000: 1 страница = 8 Кб, 1 экстент – 8 страниц)

Диспетчер файлов
При работе с диском как набором хранимых файлов диспетчер файлов использует все имеющиеся средства диспетчера дисков.

> **Файл** рассматривается как хранимый на диске набор однотипных записей.

В одном наборе страниц может содержаться несколько хранимых файлов.
Каждый хранимый файл имеет **имя** (**file name**) или **идентификационный номер** (**file ID**), уникальные в данном наборе страниц. Каждая хранимая запись в файле обладает **идентификационным номером записи** (**record ID, RID**), уникальным, по крайней мере, в пределах данного хранимого файла. Структура RID показана на Рис.7.2.

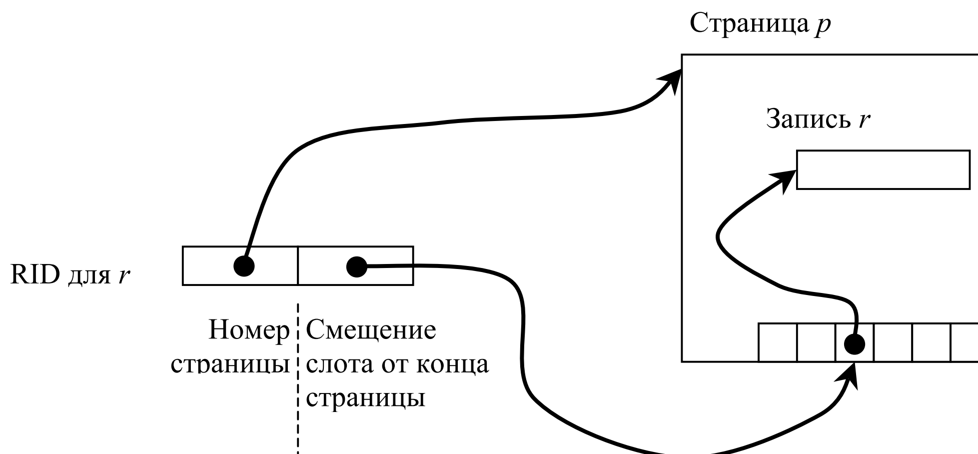


Рисунок 7.2 – Структура идентификационного номера записи RID

В дальнейшем, будем предполагать, что физический файл состоит из физических записей с RID, неизменными до тех пор, пока эти записи существуют.

Основные операции диспетчера файлов таковы:

- извлечь хранимую запись *r* из хранимого файла *f*;
- заменить хранимую запись *r* в хранимом файле *f*;
- добавить новую хранимую запись *r* в хранимый файл *f*;
- удалить хранимую запись *r* из хранимого файла *f*;
- создать новый хранимый файл *f*
- удалить хранимый файл *f*

7.3 Основные структуры хранения физических записей и методы доступа к ним

7.3.1 Последовательная структура хранения и последовательный доступ

Самый простой способ хранения записей – когда физические записи располагаются в файле упорядоченно (обычно по возрастанию) по значениям своих идентификационных номеров. Такая последовательность записей называется **физической**.

Часто под физической последовательностью записей понимают и последовательность, в которой записи упорядочены по значениям первичного ключа. Это связано с тем, что СУБД для идентификации логических записей использует первичные ключи, а не RID физических записей (которые, вообще говоря, и не являются частью компетенции СУБД), а каждой логической записи взаимно однозначно сопоставляется физическая запись.

Предположим, что физические записи в файле упорядочены по значениям первичного ключа.

Тогда для доступа к одной физической записи используются такие методы:

1. **Простой последовательный перебор** записей с проверкой ключа каждой записи.
2. **Блочный поиск**: считывается не каждая запись, а, например, каждая сотая запись, и сравнивается ее ключ с заданным значением ключа. Если в файле N записей, то используется разбиение на блоки по sqrt(N) записей.
3. **Двоичный поиск** («дихотомия»).
4. **Ключ эквивалентен адресу**. Этот способ адресации применяется, если физический адрес записи (или ее RID) равен значению первичного ключа этой записи, или вычисляется по простой формуле.

Основные характеристики поиска при использовании метода дихотомии таковы:

если

n – количество страниц, в которых хранится файл,

то

x – общее число доступов к страницам для нахождения страницы с искомой записью r вычисляется как:

$$x = \log_2(n) + 3$$

В общем случае, если файл упорядочен **не** по первичному ключу, а по другому полю, или, например, в **хронологическом порядке** (т.е. в порядке возрастания времени поступления записи), методы перебора записей являются неоптимальными. Поэтому чаще всего применяются другие методы доступа и структуры хранения.

7.3.2 Индексные структуры хранения и методы доступа

> **Индекс** (**индексный файл**) – это физический файл особого вида, в котором каждая запись состоит из двух полей – RID записи и значения некоторого поля записей физического файла данных.
> Файл данных, для которого существует индекс, называется **индексированным**.
> Поле индексированного файла, значения которого используются в индексе, называется **индексным полем**.

Индекс можно создать как по одному полю, так и по нескольким полям. Особенности работы с такими индексами рассмотрены в этом разделе ниже.

Пример 1.

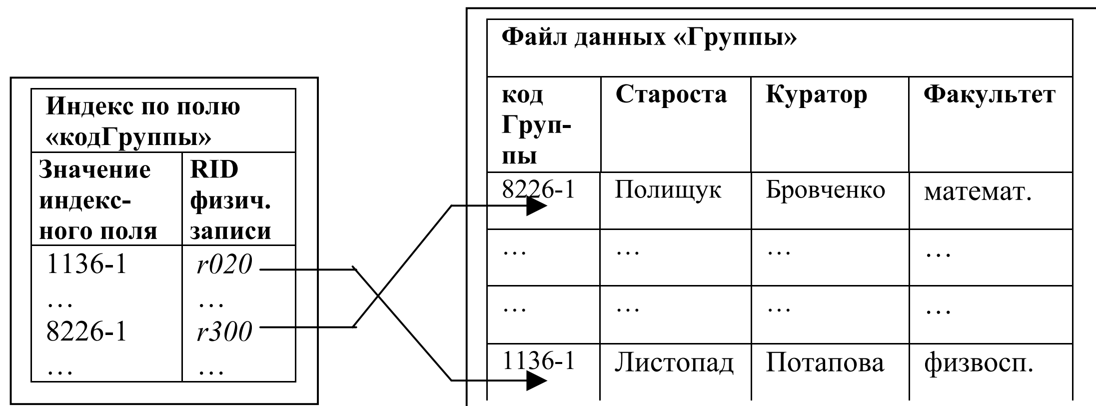


Рисунок 7.3 – Соотношение между файлом данных и индексным файлом

Основным преимуществом использования индексов является значительное ускорение выборки или извлечения данных. Действительно, вместо того, чтобы искать запись по значению некоторого поля, гораздо проще обратиться к индексу, созданному для этого поля и точно выйти на нужную запись.

Основным недостатком использования индексов является замедление процесса обновления данных. Например, при каждом добавлении новой записи в индексированный файл потребуется внести новую запись и в индекс, на что требуется дополнительное время.

Индексы можно использовать двумя разными способами:

1. Последовательный доступ

Здесь предполагается, что значения индексного поля в индексированном файле как-то упорядочены. Такой доступ к индексированному файлу используется для поиска набора записей из некоторого диапазона значений индексного поля. Например, для запроса на выборку всех записей о группах с номерами, начинающимися на цифры от 1 до 3, подойдет именно последовательный доступ.

2. Произвольный доступ

Здесь предполагается, что значения индексного поля в индексированном файле могут и не быть упорядочены (или упорядочены по каким-либо другим полям). Такой доступ к индексированному файлу используется для поиска записей по списку. Например: «Найти записи о группах с номерами 8220-1 и 8220-2».

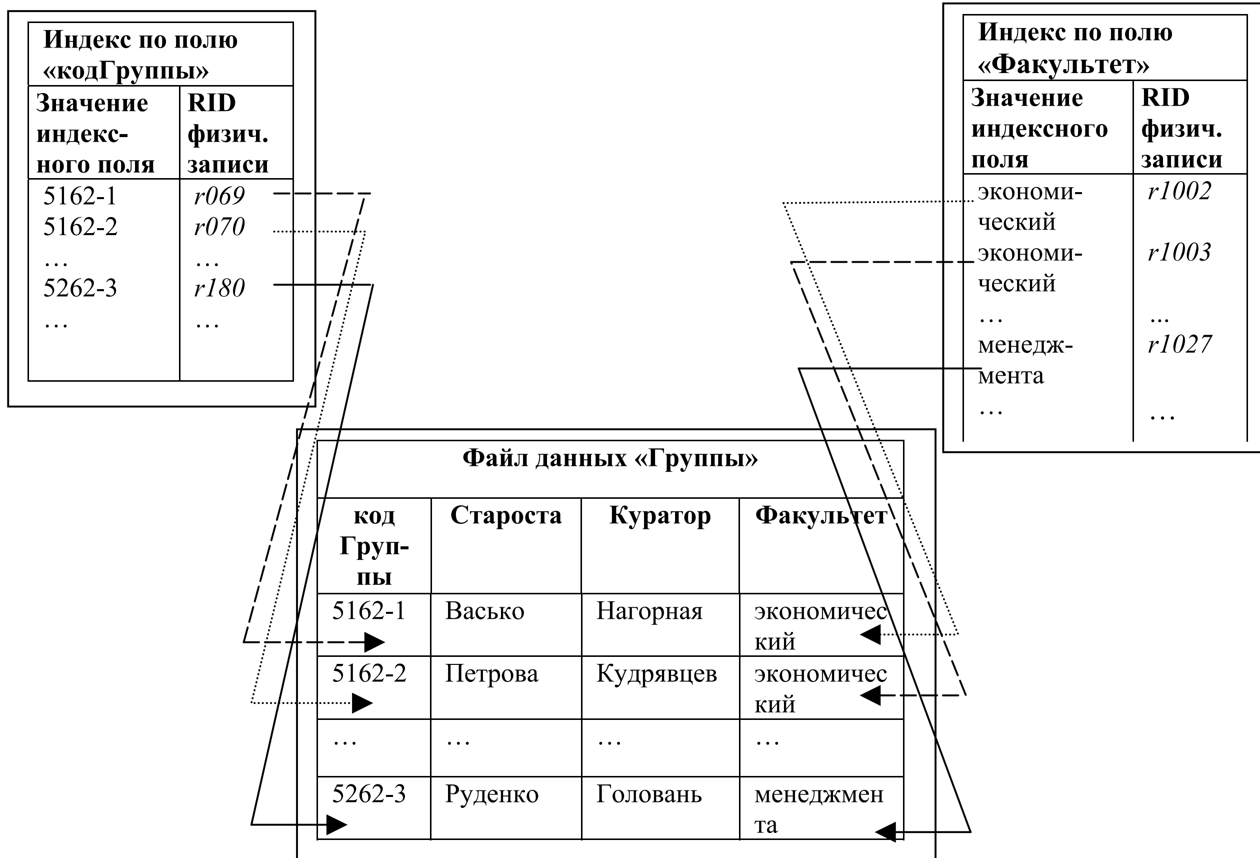
Индексы часто называют **инвертированными списками**, т.к. в индексе каждому отдельному значению индексированного поля соответствует список записей.

Хранимый файл может иметь несколько индексов.

Пример 2.

Предположим, таблица «Группы» имеет 2 индекса – по полю «кодГруппы» и по полю «Факультет». На рис.7.4 показана эта ситуация.

Тогда при ответе на запрос «Отобрать все записи о группах, номер которых начинается на цифру 5, а факультет - экономический», множество записей, полученных при просмотре индекса по коду группы (группы 5162-1, 5162-2 и 5262-3) и множество записей, отобранных в индексе по названию факультета (5162-1 и 5162-2) имеют только 2 общие записи, которые и являются ответом на запрос.



Комбинированный индекс – индекс, созданный не по одному полю, а по комбинации полей.

Пример 3.
Для таблицы «Группы» создан индекс по комбинации полей «факультет, кодГруппы» (см. Рис.7.5). Тогда ответ на запрос из примера 2 можно получить просмотром всего одного индекса (тогда как в примере 2 понадобилось два отдельных просмотра).

Индекс по комбинации полей «факультет, кодГруппы» может служит и для поиска по полю «факультет», поскольку все записи в комбинированном индексе расположены последовательно, по значениям поля «факультет». Для поиска же по полю «кодГруппы» таким комбинированным индексом пользоваться уже нельзя.

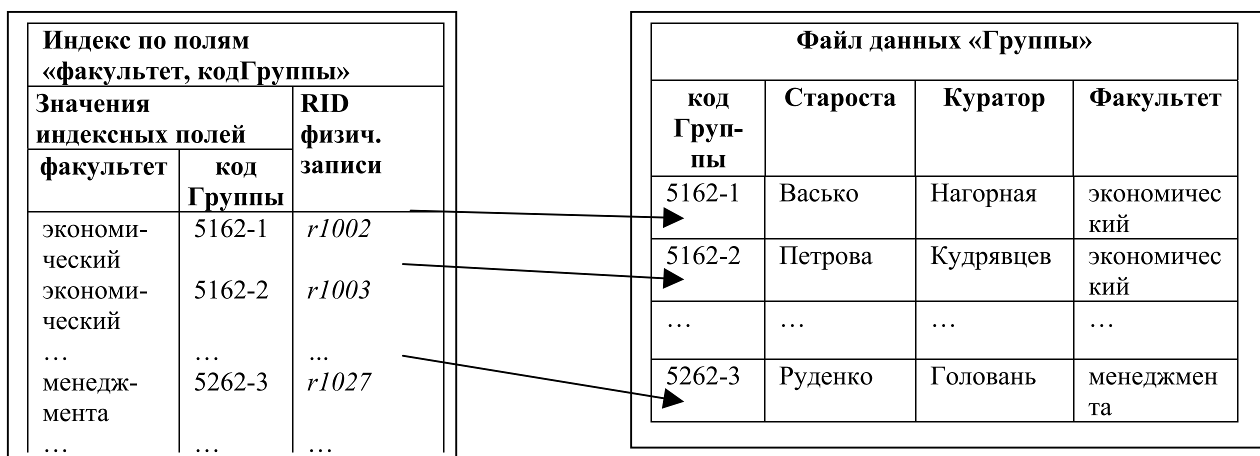


Рисунок 7.5 – Индекс по комбинации полей.

Вообще говоря, индекс на основе комбинации полей F1,F2,F3…Fn (в перечисленном порядке) может использоваться либо для отдельного поиска по полю F1, либо по комбинации полей F1,F2 (или F2,F1, но обязательно вместе), либо для F1,F2,F3 (и всех комбинаций из этих полей), и т.д. Но не может использоваться для поиска отдельно по полю F2.

Для N полей максимальное число индексов, достаточных для всевозможных запросов, равно:

$$C_N^n = \frac{N!}{n!*(N-n)!}$$

где n – наименьшее целое число, большее либо равное $N/2$.

Т.е. для таблицы из 4 полей достаточно создать $C_4^2 = \frac{4!}{2!*2!} = \frac{3*4}{2} = 6$ индексов, чтобы обрабатывать любые запросы по данной таблице.

7.3.2.1 Плотные и неплотные индексы

Данные в каждом физическом файле хранятся в физической последовательности на основе комбинации последовательности хранимых записей внутри каждой страницы и последовательности страниц внутри каждого набора страниц.

Предположим, что физическая последовательность записей в файле соответствует логической последовательности, заданной на основании некоторого поля, например, первичного ключа. Допустим, что по этому же полю выполнено индексирование. Очевидно, что в таком случае в индексе не нужно хранить указатель для каждой записи индексированного файла. Достаточно хранить в индексе указатель для каждой страницы, состоящий из максимального значения индексного поля для данной страницы и собственно номера данной страницы.

Индекс такой структуры называется **неплотным** (или **разряженным**), поскольку в нем содержатся указатели *не на все* записи индексированного файла. Все остальные индексы называются **плотными**.

Пример 4.
Пример использования неплотного индекса показан на рис.7.6.

Рисунок 7.6 – Неплотный индекс по полю «кодГруппы»

7.3.2.2 Структуры типа Б-дерева

Одним из наиболее важных и распространенных индексов является **структура типа Б-дерева** (**B-tree**). Эта структура позволяет работать с большими файлами данных и большими индексами. Дело в том, что чем больше файл индекса, тем больше времени требуется для просмотра этого файла в его физической последовательности, т.е. время, выигранное при использовании индекса, «съедается» за счет увеличения времени просмотра большого индекса. В этом случае поступают так: рассматривают индекс как обычный хранимый файл и индексируют его (т.е. создается индекс для индекса). Эту операцию можно повторять сколько угодно раз (обычно, она повторяется трижды). При этом индекс на каждом уровне будет *неплотным* по отношению к индексу предыдущего уровня (иначе не было бы смысла в его создании).

Рассмотрим существующий стандарт VSAM (Virtual Storage Access Memory, аппаратно-независимая структура доступа к памяти) на структуру типа Б-дерева.

Согласно стандарту, индекс состоит из двух частей: набор последовательностей и набор индексов.

Набор последовательностей – включает одноуровневый индекс для реальных данных, который обычно является плотным. Записи внутри индекса сгруппированы по страницам, а страницы связаны в цепочку так, чтобы *логическое* упорядочение на основе индекса осуществлялось внутри первой страницы согласно физической последовательности записей в первой странице, затем продолжилось внутри второй страницы согласно физической последовательности записей во второй странице, и т.д.. Таким образом, с помощью набора последовательностей можно осуществить быстрый последовательный доступ к индексированному файлу.

Набор индексов – обеспечивает быстрый доступ к набору последовательностей. Для этого набор индексов организуется в виде Б-дерева. на самом верхнем уровне такого индекса находится только один элемент структуры, который называется **корневым** (**root**).

Комбинация набора индексов и набора последовательностей называется структурой типа **Б-плюс-дерева** (**B+-tree**).

Каждая запись в файле из набора индексов состоит из **двух** значений индексного поля (причем первое значение меньше второго) и **трех** указателей на страницы: **левый указатель** ссылается на 1-ю страницу из набора последовательностей (или из набора индексов), содержащую значения, меньшие первого из двух значений; **правый указатель** ссылается на 1-ю страницу, содержащую значения, большие второго из двух значений; **средний указатель** ссылается на 1-ю страницу, содержащую значения, большие первого значения и меньшие второго значения записи.

Пример 4.
Простое Б-плюс-дерево показано на рис.7.7

Основные характеристики поиска при использовании сбалансированного Б-дерева таковы:

если

d – количество записей индекса в странице, $d>=2$

e – максимальное число записей индексированного файла в блоке, $e>=2$

n – количество записей в файле индекса

то

x – общее число доступов к страницам для нахождения страницы с искомой записью r вычисляется как:

$$x \le 1 + \log_d(\frac{n}{e})$$

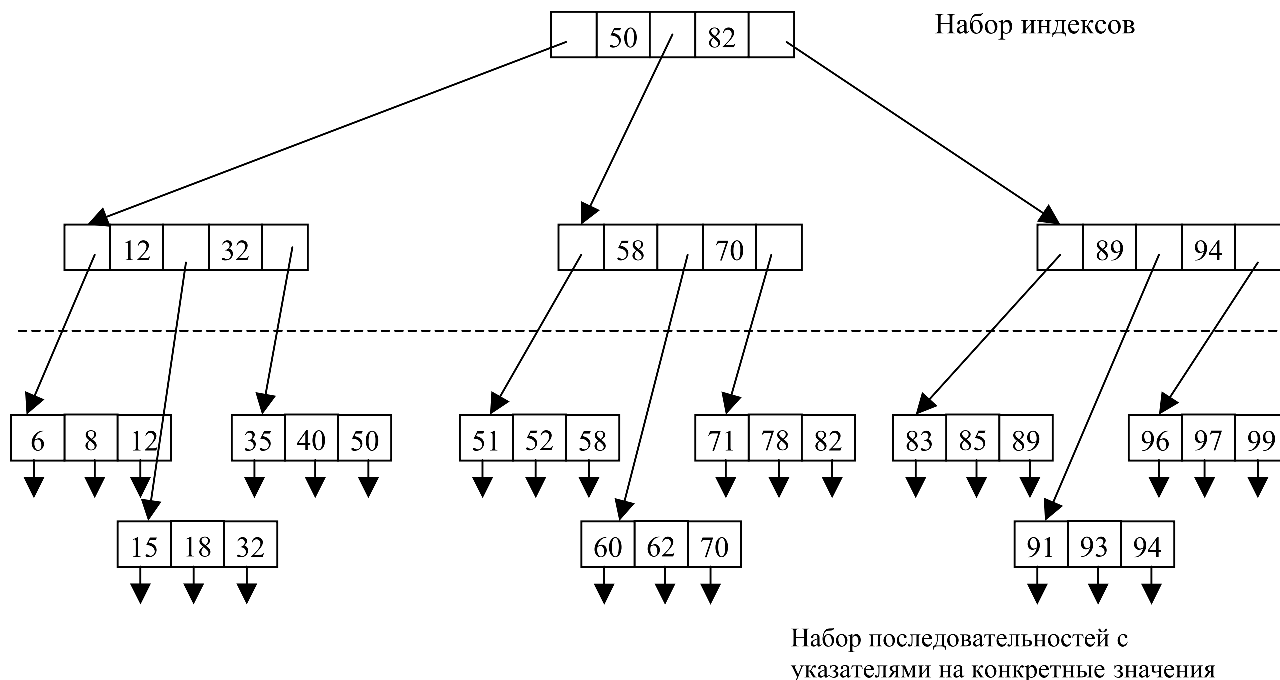


Рисунок 7.7 – Б - плюс–дерево.

7.3.3 Хэширование

Хэширование (от слова hashing - перемешивание) – это технология быстрого прямого доступа к физической записи на основе заданного значения некоторого поля (не обязательно ключевого).

Основные черты этой технологии таковы:

1. Каждая физическая запись базы данных помещается по адресу (RID-указателю или номеру страницы), который вычисляется с помощью специальной **хэш-функции** на основе значения некоторого поля данной записи, т.н.**хэш-поля**. Вычисленный адрес называется **хэш-адресом**.
2. Для сохранения записи СУБД вначале вычисляет хэш-адрес новой записи, а затем диспетчер файлов помещает эту запись по вычисленному адресу.
3. Для извлечения нужной записи по заданному значению хэш-поля в СУБД сначала вычисляется хэш-адрес, а затем диспетчеру файлов посылается запрос для извлечения записи по вычисленному адресу.

В общем случае этот метод похож на рулетку: шарик (запись) может попасть с некоторой вероятностью в одно из N гнезд рулетки (областей, где хранятся записи). Вероятность попадания записи в конкретную область зависит от количества областей, их размера, алгоритма преобразования ключа в адрес.

К недостаткам хэширования относятся такие:

1. Как правило, физическая последовательность записей внутри физического файла отличается от последовательности ключевого поля, а также любой другой логической последовательности.

2. При хэшировании могут возникать **коллизии**, т.е. ситуации, когда две или более различных записей имеют одинаковые вычисленные хэш-адреса.

Основные характеристики поиска при использовании хэширования:

x – общее число доступов к страницам для нахождения страницы с искомой записью r :

$$x \geq 3$$

7.4 Поиск текстовых значений

Проблема эффективного поиска документов по ключевым словам сегодня очень популярна, благодаря развитию Интернет, всемирной паутиныWeb, созданию автоматизированных баз патентной информации, электронных энциклопедий и словарей.
Запросы на поиск текста разделяются на:

1. Логические запросы. Например, найти документы, в которых слова «чемпионат», «футбол», «Аргентина» встречаются в одном предложении. Или другой пример: найти документы, в которых есть слова «Лукьяненко» **и** «новый роман»
2. Запросы по ключевым словам (keyword search). Например: запрос с такими ключевыми словами: «чемпионат, футбол, Аргентина» или «Лукьяненко, фантастика, новинка».

Варианты поиска ответов на запросы:

1. Полнотекстовое сканирование. Недостаток: подходит для малых баз документов.
2. Использование инверсии.

Каждый документ снабжается списком ключевых слов. Создается инвертированный файл ключевых слов, в котором ключевые слова упорядочиваются (например, по алфавиту), и для каждого слова создается таблица соответствия.(см. Рис. 7.3)

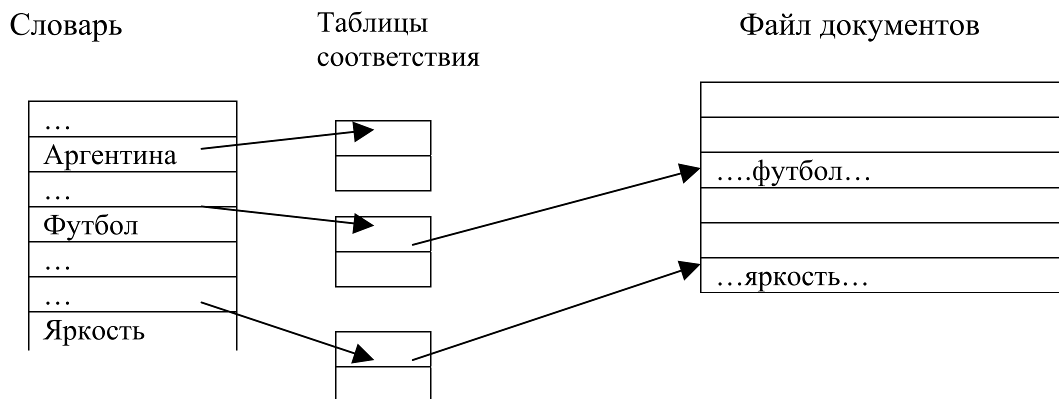


Рисунок 7.3 – Поиск документов с помощью инвертированных списков

На низком уровне, подобные словари и таблицы соответствия используют B-деревья или хэш-таблицы.
Недостаток: требуется много памяти и времени на обновление или реорганизацию словаря, если список документов часто меняется.

3. Файлы сигнатур.

Идея создания сигнатурных файлов состоит в том, чтобы создать «быстрый, но приблизительный» фильтр, с тем, чтобы быстро отсекать неподходящие по запросу документы (см. Рис. 7.4.)

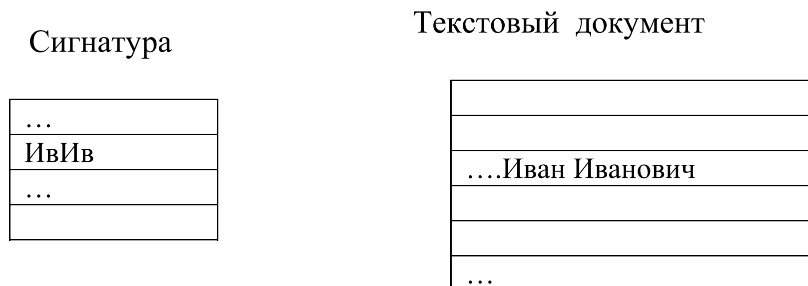


Рисунок 7.4 – Использование файла сигнатур документов.

На самом деле, каждому слову в тексте ставится в соответствие битовая строка (сигнатура слова), заданной длины. Битовые строки затем логически складываются (применяется логическое OR) и полученное значение составляет сигнатуру документа.