1. ***Физическая организация данных. Формат записи***

Будем считать, что на физическом уровне информация хранится в файле, который содержит записи с идентичным форматом.

*Формат записи* – список имен полей, каждое поле занимает фиксированное число байт и имеет фиксированный тип. Запись состоит из значений каждого поля.

Над файлом требуется выполнить следующие типичные операции:

* включить запись;
* удалить запись;
* модифицировать запись;
* найти запись, удовлетворяющую заданным условиям;

1. ***Эффективность физической организации***

Эффективность физической организации определяется эффективностью реализации этих 4-х операций.

Основная операция с внешней памятью – это передача блока из внешней памяти в оперативную память и наоборот. Поэтому, если говорить о быстродействии различных алгоритмов доступа к данным, то следует подсчитать число блоков, которые должны быть прочитаны в основную память или записаны из неё. Это число и будет представлять оценку быстродействия алгоритма. ff

1. **Модель организации внешней памяти.**

Файловая система разделяет внешнюю память на логические блоки равного размера (кластеры). Файл хранится в одном или нескольких блоках. В каждом блоке могут храниться несколько записей (запись может занимать несколько блоков). Внутри блока может быть пространство, незанятое какими-либо записями. Мы предполагаем, что файловая система устанавливает соответствие между именем файла и адресами блоков, которые образуют этот файл.

Можно считать, что запись так же имеет адрес, который можно рассматривать либо как абсолютный адрес её первого байта, либо как адрес блока, содержащего запись, + смещение записи внутри блока.

Для того, чтобы найти нужную запись необходимо:

* прочитать блок, который её содержит;
* найти запись внутри блока.

При таком подходе запись внутри блока можно перемещать, при этом указатель на запись (фактически указатель на содержащий её блок) не изменится. Уменьшается вероятность «зависания указателя» – когда «указатель» указывает на место, где записи на самом деле нет.

1. **Закрепленные и незакрепленные записи.**

Записи становятся *закрепленными*, если где-либо в базе данных могут существовать (хранится) указатели на них.

При закреплении записей мы не можем перемещать записи об этих объектах, иначе указатели на них «зависнут», т.е. не будут указывать на данные, на которые они указывали первоначально.

При закреплении записей не может быть реализован общий случай организации файла, при котором запись удаляется (например, при включении других записей или удалении записей). Так как, сложно найти, где именно в полной базе необходимо откорректировать указатели на перемещаемые записи.

1. **Организация файлов в виде «кучи».**

Наиболее очевидный подход к хранению записей файлов заключается в последовательном размещении их в необходимом числе блоков. Часто предполагается, что запись не может перекрывать границу блока (часть блока теряется). Эту организацию иногда называют *«кучей»*.

Блоки, используемые для кучи, могут быть связаны в цепочку указателями (в конце блока указатель на следующий блок). При другой организации выделяют отдельный блок (блоки) в котором хранят список адресов блоков, образующих кучу.

Чтобы включить запись, её необходимо поместить в последний блок, если в нём имеется место, или получить новый блок из OS, если места в последнем блоке больше нет.

Удаление может осуществляться установкой бита удаления (перед каждой записью или в начале блока – пояснить).

Для поиска нужной записи (удовлетворяющей заданным условиям, например, при заданном значении ключа) в общем случае требуется просмотреть все записи файла (в «среднем» половину записей файла). Если число блоков, образующих файл велико, то это такой поиск требует большого количества доступов к внешней памяти и, следовательно, приводит к увеличению времени доступа.

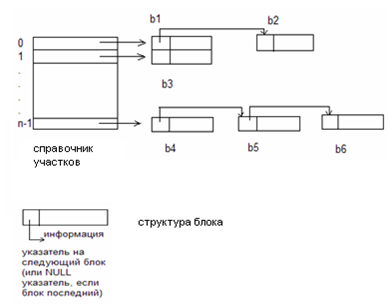
1. **Основной принцип организации хешированных файлов.**

Основная идея, лежащая в основе организации файлов с хешированным доступом, состоит в разделении записей файла между участками, каждый из которых содержит один или более блоков памяти. Для любого файла, хранимого таким образом, существует хеш-функция (h), которая использует в качестве аргумента значение ключа, и вычисляет целое число то нуля до некоторого максимального значения N.

Пусть v есть значения ключа. Тогда значение h(v) есть номер участка, в котором должна находиться запись с этим значением ключа, если она вообще присутствует в файле.

При выборе функции h желательно, чтобы h(v) принимала все её возможные значения (от 0 до N) примерно с равной вероятностью, когда v принимает всевозможные допустимые значения ключа.

1. **Выбор функции хеширования.** 
   1. Интерпретируем значение ключа как последовательность бит, сформированную путем конкатенации значений всех полей ключа. Эта последовательность имеет фиксированную длину, поскольку каждое поле имеет фиксированную длину.
   2. Делим последовательность бит на группы, состоящие из фиксированного числа бит. (из 16 бит). Последнюю группу при необходимости дополняем нулями.
   3. Складываем полученные группы бит как целые числа.
   4. Делим полученную сумму на число участков и используем остаток от деления как номер участка.
2. **Схема организации хешированного файла.**



1. **Организация поиска в хешированном файле.**

Задача: найти запись с ключом **v**. (**v–** одно поле или список полей в фиксированном порядке – тогда **v** конкатенация полей).

Вычислим h(**v**) получая номер участка i. Прочтём справочник участков и найдем адрес первого блока участка i. Читаем первый блок, анализируем субблоки на совпадение ключа. Если найдено – поиск успешен – конец. Если нет – читаем следующий блок данного участка. Если его нет – поиск закончен неуспешно.

1. ***Модификация в хешированном файле*.**

Задача: изменить одно или несколько полей записи с ключом **v.**

Найти запись с ключом **v**. Если не найдена – аварийный останов.

Найдена запись:

*среди модифицированных полей, есть хотя бы одно, которое входит в ключ:* необходимо удалить найденную запись из базы и затем добавить измененную;

*среди модифицированных полей нет ни одной, входящей в ключ:* необходимо изменить запись в ОП и записать модифицированный блок по месту (туда же).

1. ***Включение записи в хешированном файле*.**

Задача: добавить запись с ключом **v**.

Найти запись с ключом **v** (вычисляется номер участка, куда надо добавить запись). Если запись найдена, аварийный останов. Иначе:

Найдем первый свободный субблок среди блоков участка h(**v**) (этот субблок может быть в середине, если ранее было удаление с освобождением субблока, т.е. записи не закреплены). Адрес этого субблока можно запомнить при поиске ключа v. Помещаем запись в данный субблок (записываем блок на место). Если свободного места нет, то получаем свободный блок из OС. Организуем цепочку с последним блоком участка h(v), затем пишем блок на диск

1. ***Удаление в хешированном файле*.**

Задача: удаление записи с ключом **v**.

Найти запись с ключом **v**. Если запись не найдена, то аварийный останов. Иначе:

Если запись найдена, то:

если записи не закреплены →бит свободен/занят перевести в состояние свободен (при следующем добавлении этот субблок будет использован);

если записи закреплены, тогда бит свободен/занят не изменяеть, а бит удалён установить в состояние 1– удалён.

Если записи не закреплены, то для удаления возможно использовать следующий алгоритм: последнюю запись файла переписать на место удаленной, а последний субблок освободить.

1. **Проблема реорганизации. Анализ временных характеристик хеширования.**

Для каждой операции поиска, модификации, включения, удаления требуется: одно обращение для чтения справочника участков (если весь справочник не помещается в память) + число доступов, которое не превышает число блоков в данном участке. При поиске в среднем просматривается половина блоков участка. Для всех операций, кроме поиска, требуется ещё записать блок во внешнюю память.

Если каждый участок состоит ровно из одного блока (т.е. лучший случай) для поиска требуется 2 обращения. Для остальных доступов – 3 обращения. В каждом участке имеется не менее одного блока. Чтобы среднее число блоков в участках в превышало единицу, необходимо, чтобы число участков было бы примерно равно числу записей в файле, деленному на число записей в блоке.

Для повышения скорости доступа, при росте числа блоков в участках, потребуется реорганизация файла. Её достаточно просто провести, если ввести два ограничения:

1. При вычислении функции хеширования от ключа v получают большое число, гораздо большее, чем число участков. Полученное число делят на число участков, остаток от деления – номер участка.
2. При реорганизации число участков n умножают на некоторое фиксированное с (обычно с = 2).

Если мы удвоим число участков, то все записи участка i будут попадать в участки i или i+n. И в эти участки не попадут никакие записи других участков.

1. **Индексированные файлы.**

Допустим, что записи в файле отсортированы по значениям этого ключа (обратить внимание на сортировку дат dtos() и строк – лексикографическая сортировка). Для нескольких полей: сортировка по первому ключу, при равных первых – сортировка по второму и т.д.

В этом случае (файл отсортирован) для поиска можно использовать идею поиска в телефонной книге (словаре). Вместо просмотра всех записей будем анализировать только первую запись на каждой странице. Если искомый ключ превышает ключ этой записи, то возможно, что искомая запись находится на предыдущей странице. Если даже для последней страницы искомый ключ меньше, чем первый на этой странице, то следует проанализировать записи этой последней страницы.

Аналогично организован доступ к файлу. Пусть мы имеем отсортированный файл с информацией, назовем его главным файлом. Создадим для главного файла второй файл– так называемый разреженный индекс, состоящий из пар (значение\_ключа, адрес\_блока).

Пара (v,b) появляется в файле индекса, если первая запись в блоке главного файла с адресом b имеет значение ключа v. Записи файла индекса подобно обыкновенному (информационному) файлу с двумя полями: ключ и номер блока. Записи файла индекса отсортированы по значению ключа и не являются закрепленными, т.к. нигде в другом файле нет ссылки, ни на какую запись файла индекса. Вероятно, что с файлом индекса, как и с обычным (информационным) файлом следует выполнить операции включения, модификации, удаления.

Кроме того, необходима новая функция (вместо поиска): для данного ключа v1, найти такую запись ((v2 ,b2)| (v2 ≤ v1) Ù ((следующая (v3,b3) | (v3 >v1) Ú (v2,b2) –последняя в файле)))

В этом случае говорят, что v2 покрывает v1 (это значит, что если запись с ключом v1, содержится в файле, то она может содержаться только в блоке b2, у которого первая запись имеет ключ v2).

1. **Поиск в индексе.**

Задача (для файла индекса): найти в файле индекса запись (v2,b2) такую, что v2 покрывает ключ v1.

Решение: в простом случае (мало записей в индексе) – линейный поиск (условия применения – весь индекс в основной памяти).

Но и в этом случае имеется выигрыш, т.к. если в блоке основного файла можно записать *k* записей и известно. что одна запись в индексе организуется на один блок основного файла, то в файле индекса в *k* раз меньше записей. Кроме того, обычно записи индекса короче, чем записи основного файла и в один блок индекса помещается больше записей (появляется вероятность размещение всего индекса в оперативной памяти).

Лучшая стратегия поиска в файле индекса – использование двоичного поиска. При данной стратегии на каждом шаге количество блоков (при поиске записи (v2,b2) – покрывающих ключ v1) индекса, содержащих запись (v2,b2), сокращается вдвое.

Таким образом, если в индексе n блоков, то не боле чем за log2(n+1) чтений будет прочитан блок индекса, содержащий запись (v2,b2). В практике часто вместо оценки log2(n+1) используют оценку log2n. С учетом доступов к основному файлу общее число доступов – 3+ log2n (3 складывается из одного чтения справочника индекса, одного чтения блока файла и одной записи блока файла на диск).

1. **Поиск записи (сортированный файл с незакрепленными записями и индексом).**

**Поиск записи** (найти запись с ключом v1):

в индексе найти запись (v2,b2), где v2 покрывает v1;

прочитать блок b2 в оперативную память и искать запись с ключом v1 в оперативной памяти каким-нибудь методом (двоичным или линейным). Если найдено, то успешное окончание. Иначе – неуспешный поиск.

При поиске проверять биты свободен/занят, удален/не удален

1. **Модифицирование записи (сортированный файл с незакрепленными записями и индексом).**

**Модифицирование** (изменить запись с ключом v):

найти запись с ключом v1, если её нет то выход, неуспешное окончание;

запись найдена. Если в ключ входит хотя бы одно из модифицированных полей, то удаление записи из внешней памяти и добавление новой записи. Если среди модифицированных полей нет ни одной записи, в которую входит ключ, то изменить и записать на прежнее место.

1. **Включение записи (сортированный файл с незакрепленными записями и индексом).**

**Включение** (добавить запись с ключом v1):

если новую запись надо добавить в 1 блок, т.е. v1 меньше, чем первая запись первого блока. Добавить запись в первый блок и изменить ключ в первой записи индекса;

если v2≤v1 , иb2 не первый блок, тогда: если v2<v1 (запись не будет первой) добавить запись в блок b2 , индекс не менять ( в начало среднего блока запись никогда не добавится ). Если v2=v1, то запись с ключом v1 уже есть, аварийный останов.

1. **Удалить запись (сортированный файл с незакрепленными записями и индексом).**

**Удалить** (запись с ключом v1):

найти запись с ключом v1. Если запись не найдена, то ошибка;

запись найдена, то удаление производится со сдвигом влево, начиная с записи следующей за удаленной. Установка бита свободен/занят последней записи в состояние свободен. Если удаляется первая запись в блоке, то корректируется индекс. Если после удаления блок полностью пустой, то вернуть блок в OS и удалить соответствующую ему запись из индекса. Если она последняя и единственная, то блок главного файла вернуть в OS и удалить первую запись из индекса

1. **Организация сортированных файлов с закрепленными записями**

Если записи закреплены, то невозможно поддерживать их внутри блока в отсортированном порядке. Нет возможности даже гарантировать, что записи предыдущего блока предшествуют записям последующего блока (относительно некоторого ключа). В этом случае каждый блок главного файла, на который имеется ссылка в индексе, играет роль первого блока в цепочке блоков (аналогично участку в хешированных файлах). При включении записей в участок добавляются новые блоки, которые объединяются в цепочку. Кроме того, создается еще один пустой участок, в который будут включаться записи, предшествующие первому участку. В этом случае индекс никогда не изменяется. Первые записи каждого блока первоначального файла определяют распределение записей по участку. Это распределение остается неизменным пока участки не достигнут больших размеров и потребуется реорганизация файла.

1. **Инициализация (сортированный файл с закрепленными записями и индексом).**

**Инициализация:** процедура формирования файла индекса по первоначальному главному файлу называется – инициализацией.

Для инициализации отсортируем файл и распределим его записи по блокам. При этом в каждом блоке выделим некоторое свободное пространство (несколько свободных субблоков), которые при добавлении записи исключат на некоторое время рост цепочек в участках главного файла. Кроме того, предусмотрим один пустой блок в участке, предшествующем первому, куда будем помещать записи при расширении файла, и которые будут предшествовать записям первоначального файла. Создадим индекс для полученного файла, в том числе и для пустого блока. Запись в индексе для этого блока должна содержать номер «пустого» блока и не содержать ключа.

**Поиск** (найти запись с ключом v1 ):

Найдем запись индекса, значение ключа которой покрывает требуемое значение ключа v1. Если v1 меньше, чем первое значение ключа в файле индекса (заметим, что первичный ключ находится только начиная со второй записи индекса), то требуемой записью индекса является первая запись. По указателю на блок (из индекса) читаем первый блок цепочки (участка). Чтобы найти ключ v1 последовательно читаем блоки данного участка, соединенные в цепочку.

**Модификация** ( изменить запись с ключом v1 ):

Выполняется аналогично предыдущей организации файла.

**Включение** (добавить запись с ключом v1 ):

Поиск участка (v2,b2) по ключу v1. Последовательно прочитать блоки участка, с целью найти свободное место. Если свободного субблока нет, то получить из OS пустой блок и добавить его в конец участка (пояснить). Включим новую запись в новый блок.

**Удаление (**удалить запись с ключом v1):

Найти запись с ключом v1 . Если запись не найдена, то ошибка, аварийное завершение. Если запись найдена, то установка бита удален в состояние «1» (запись удалена). Бит свободен/занят оставить в состоянии «занят».