Физические модели баз данных

Лекция 13

Часть 1

Часть 2

Общие принципы организации данных во внешней памяти в SQL-ориентированных СУБД:

- Хранение таблиц
- Индексы
  - B+-деревья
  - Хэширование
- Журнальная информация
- Служебная информация

- Строки в таблицах хранятся в неупорядоченном виде. При выполнении операций выборки, обновления и удаления СУБД должна отыскать нужные строки. Для ускорения этого поиска и создается **индекс**.
- На основе данных, содержащихся в конкретной строке таблицы, формируется значение элемента (записи) **индекса**, соответствующего этой строке.
- Для поддержания соответствия между элементом **индекса** и строкой таблицы в каждый элемент помещается указатель на строку. **Индекс** является упорядоченной структурой.
- Элементы (записи) в **индексе** хранятся в отсортированном виде, что значительно ускоряет поиск данных в индексе.

Индексы

Для создания индексов предназначена команда:

CREATE INDEX имя-индекса
ON имя-таблицы (имя-столбца, ...);

CREATE INDEX ON airports (airport_name);

Посмотрим описание нового индекса в БД “demo” в PSQL:

demo=# **\d airports**

...

Индексы:

...

"airports_airport_name_idx" btree (airport_name)

...

Обратите внимание, что имя индекса, сформированное автоматически, включает имя таблицы, имя столбца и суффикс idx.

Файловые структуры, используемые для хранения информации в базах данных

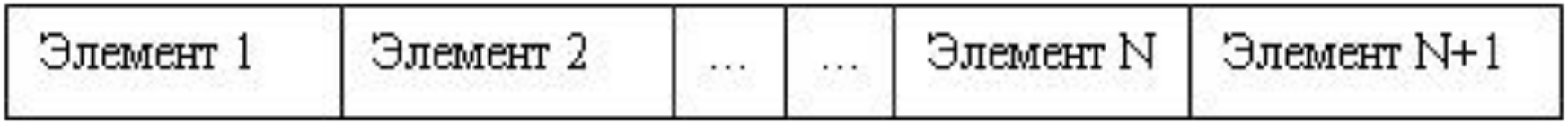
Файл как линейная последовательность записей

С точки зрения пользователя, ***файлом*** называется поименованная линейная последовательность записей, расположенных на внешних носителях.

В соответствии с методами управления доступом различают устройства внешней памяти с

- *произвольной адресацией* (магнитные и оптические диски)
- и устройства *с последовательной адресацией* (магнитофоны, стримеры).

Модель хранения информации на устройстве последовательного доступа



Система управления файлами (СУФ)

Файлы прямого доступа

Файлы с постоянной длиной записи, расположенные на устройствах прямого доступа (УПД), являются ***файлами прямого доступа***.

Для каждого файла в системе хранится следующая информация:

- имя файла;
- тип файла (например, расширение или другие характеристики);
- размер записи;
- количество занятых физических блоков;
- базовый начальный адрес;
- ссылка на сегмент расширения;
- способ доступа (код защиты).

Для файлов с постоянной длиной записи физический адрес размещения записи **NZ** с номером К может быть вычислен по формуле:

NZ = BA + (K - 1) * LZ + 1,где BA — базовый адрес, LZ — длина записи.

Файлы прямого доступа

Можно всегда определить адрес, на который необходимо позиционировать механизм считывания-записи, то устройства прямого доступа делают это практически мгновенно, поэтому для таких файлов чтение произвольной записи практически не зависит от ее номера.

Файлы прямого доступа обеспечивают наиболее быстрый доступ к произвольным записям, и их использование считается наиболее перспективным в системах баз данных.

Файлы последовательного доступа

Файлы с переменной длиной записи всегда являются файлами последовательного доступа. Они могут быть организованы двумя способами:

- Конец записи отличается специальным маркером.

	Запись 1	**X**	**Запись 2**	**X**	**Запись3**	**X**	

- В начале каждой записи записывается ее длина.

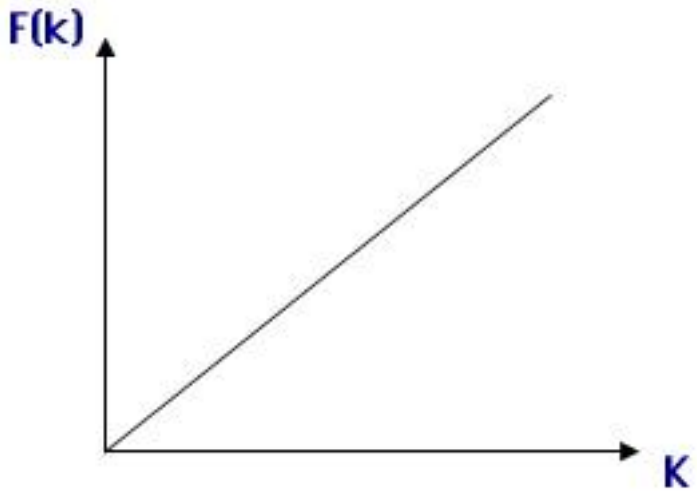
	LZ1	**Запись1**	**LZ2**	**Запись2**	**LZ3**	**Запись3**	

Файлы прямого доступа

- При организации в некоторых очень редких случаях возможно построение функции, которая по значению ключа однозначно вычисляет адрес (номер записи файла).

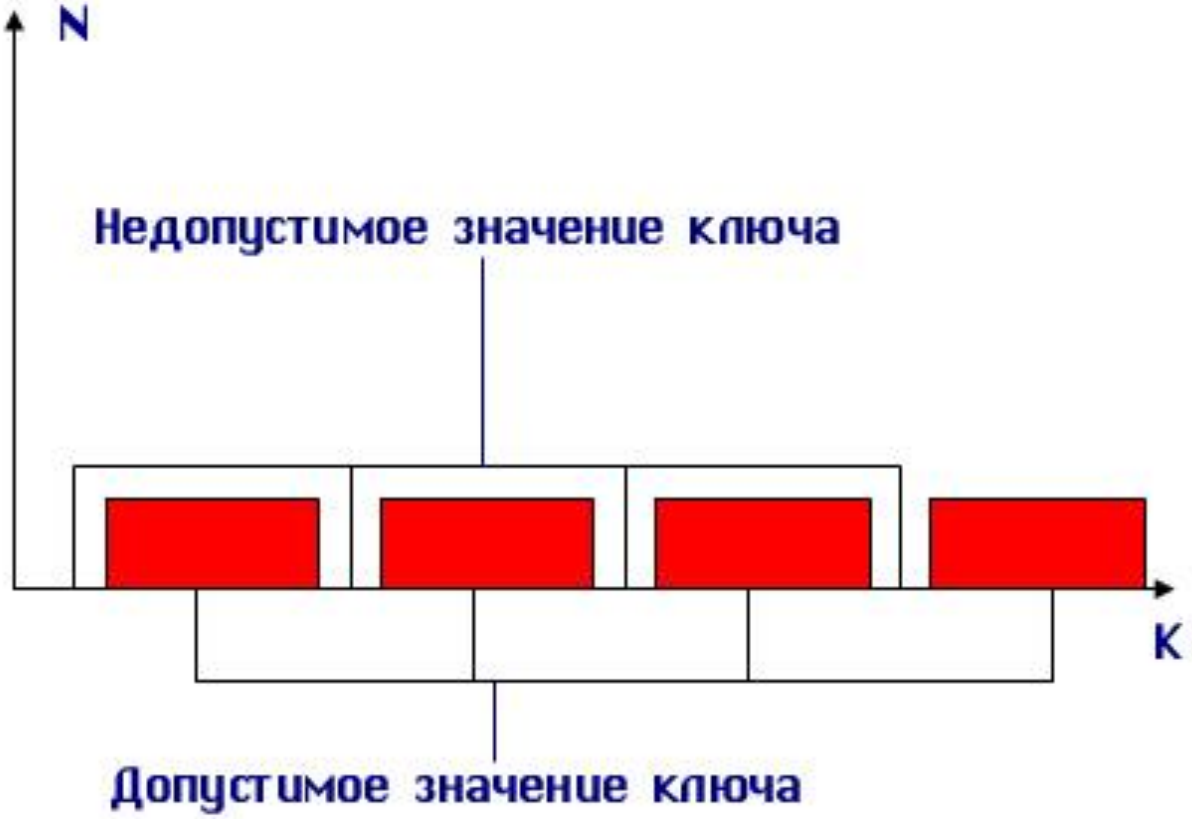
$$NZ = F(K),$$

где NZ — номер записи, К — значение ключа, F( ) — функция.



Хэширование

Часто бывает, что значения ключей разбросаны по нескольким диапазонам



В подобных случаях применяют различные методы ***хэширования*** (рандомизации) и создают специальные ***хэш- функции***.

Хэширование

Суть методов ***хэширования*** состоит в том, что мы берем значения ключа (или некоторые его характеристики) и используем его для начала поиска, вычисляем некоторую хэш-функцию h(k) и полученное значение берем в качестве адреса начала поиска.

- То есть мы не требуем полного взаимно-однозначного соответствия, но, с другой стороны, для повышения скорости мы ограничиваем время этого поиска (количество дополнительных шагов) для окончательного получения адреса.
- Таким образом, мы допускаем, что нескольким разным ключам может соответствовать одно значение хэш-функции (то есть один адрес). Подобные ситуации называются ***коллизиями***. Значения ключей, которые имеют одно и то же значение хэш-функции, называются ***синонимами***.

Стратегия разрешения коллизий с областью переполнения

необходимо принять два независимых решения:

1. выбрать хэш-функцию;
2. выбрать метод разрешения коллизий.

две достаточно распространенные стратегии разрешения коллизий

При выборе 1-й стратегии область хранения разбивается на 2 части:

- основную область;
- область переполнения.

Для каждой новой записи вычисляется значение хэш-функции, которое определяет адрес ее расположения, и запись заносится в основную область в соответствии с полученным значением хэш-функции.

Основная область:

Содержание записей	Ссылка на синонимы
Петров	1
Сахаров	3

Область переполнения:

Содержание записей	Ссылка на синонимы
Петров	2
Петровский	
Сахарочкин	

- Если вновь заносимая запись имеет значение функции хэширования такое же, которое использовала другая запись, уже имеющаяся в БД, то новая запись заносится в область переполнения на первое свободное место,
- В записи-синониме, которая находится в основной области, делается ссылка на адрес вновь размещенной записи в области переполнения.
- Если уже существует ссылка в записи-синониме, которая расположена в основной области, то новая запись получает дополнительную информацию в виде ссылки и уже в таком виде заносится в область переполнения.

Поиск произвольной записи

- При поиске записи также сначала вычисляется значение ее хэш-функции и считывается первая запись в цепочке синонимов, которая расположена в основной области.
- Если искомая запись не соответствует первой в цепочке синонимов, то далее поиск происходит перемещением по цепочке синонимов, пока не будет обнаружена требуемая запись.
- Скорость поиска зависит от длины цепочки синонимов, поэтому качество хэш-функции определяется максимальной длиной цепочки синонимов.
- Хорошим результатом может считаться наличие не более 10 синонимов в цепочке.

Удаление записи

- При удалении произвольной записи сначала определяется ее место расположения.
- Если удаляемой является первая запись в цепочке синонимов, то после удаления на ее место в основной области заносится вторая (следующая) запись в цепочке синонимов, при этом все указатели (ссылки на синонимы) сохраняются.

•Если же удаляемая запись находится в середине цепочки синонимов, то необходимо провести корректировку указателей: в записи, предшествующей удаляемой, в цепочке ставится указатель из удаляемой записи.

•Если это последняя запись в цепочке, то все равно механизм изменения указателей такой же, то есть в предшествующую запись заносится признак отсутствия следующей записи в цепочке, который ранее хранился в последней записи.

Организация стратегии свободного замещения

файловое пространство не разделяется на области

но для каждой записи добавляется 2 указателя:

- указатель на предыдущую запись в цепочке синонимов
- и указатель на следующую запись в цепочке синонимов.

- Отсутствие соответствующей ссылки обозначается специальным символом, например нулем.

1. Для каждой новой записи вычисляется значение хэш-функции, и если данный адрес свободен, то запись попадает на заданное место и становится первой в цепочке синонимов.

Организация стратегии свободного замещения

2. Если адрес, соответствующий полученному значению хэш-функции, занят, то по наличию ссылок определяется, является ли запись, расположенная по указанному адресу, первой в цепочке синонимов.
3. Если да, то новая запись располагается на первом свободном месте и для нее устанавливаются соответствующие ссылки: она становится второй в цепочке синонимов, на нее ссылается первая запись, а она ссылается на следующую, если таковая есть.
4. Если запись, которая занимает требуемое место, не является первой записью в цепочке синонимов, значит, она занимает данное место «незаконно» и при появлении «законного владельца» должна быть «выселена», то есть перемещена на новое место.
5. Механизм перемещения аналогичен занесению новой записи, которая уже имеет синоним, занесенный в файл. Для этой записи ищется первое свободное место и корректируются соответствующие ссылки: в записи, которая является предыдущей в цепочке синонимов для перемещаемой записи, заносится указатель на новое место перемещаемой записи, указатели же в самой перемещаемой записи остаются прежние.
6. После перемещения «незаконной» записи вновь вносимая запись занимает свое законное место и становится первой записью в новой цепочке синонимов.
7. Механизмы удаления записей во многом механизмам удаления в стратегии с областью переполнения

Индексные файлы

Несмотря на высокую эффективность хэш-адресации, в файловых структурах далеко не всегда удается найти соответствующую функцию, поэтому при организации доступа по первичному ключу широко используются *индексные файлы.*

В некоторых коммерческих системах индексными файлами называются также и файлы, организованные в виде инвертированных списков, которые используются для доступа по вторичному ключу.

Индексные файлы можно представить как файлы, состоящие из двух частей.

Это не обязательно физическое совмещение этих двух частей в одном файле, в большинстве случаев индексная область образует отдельный индексный файл, а основная область образует файл, для которого создается индекс.

Но нам удобнее рассматривать эти две части совместно, так как именно взаимодействие этих частей и определяет использование механизма индексации для ускорения доступа к записям.

Индексные файлы

- Сначала идет ***индексная область***, которая занимает некоторое целое число блоков, а затем идет ***основная область***, в которой последовательно расположены все записи файла.
- В зависимости от организации индексной и основной областей различают 2 типа файлов: **с *плотным индексом* и с *неплотным индексом.***
- Эти файлы имеют еще дополнительные названия, которые напрямую связаны с методами доступа к произвольной записи, которые поддерживаются данными файловыми структурами.
- Файлы с плотным индексом называются также ***индексно-прямыми*** файлами, а файлы с неплотным индексом называются также и ***идексно-последовательными*** файлами.
- Смысл этих названий нам будет ясен после того, как мы более подробно рассмотрим механизмы организации данных файлов.

Файлы с плотным индексом (индексно-прямые файлы)

В этих файлах основная область содержит последовательность записей одинаковой длины, расположенных в произвольном порядке, а структура индексной записи в них имеет следующий вид:

Значение ключа	**Номер записи**

Здесь значение ключа — это значение первичного ключа, а номер записи — это порядковый номер записи в основной области, которая имеет данное значение первичного ключа.

В индексных файлах с плотным индексом для каждой записи в основной области существует одна запись из индексной области.

Файл индексов (блок индексов) намного меньше блока данных и может поместиться в буфере основной памяти, что увеличивает скорость поиска записи

Все записи в индексной области упорядочены по значению ключа, поэтому можно применить более эффективные способы поиска в упорядоченном пространстве.

Бинарный поиск

Наиболее эффективным алгоритмом поиска на упорядоченном массиве является логарифмический, или бинарный поиск.

$$T_n = \log_2 N,$$

где N — число элементов.

Максимальное время доступа в количестве обращений к диску

$T_n = \log_2 N$бл.инд. + 1.

Пример организации файла с плотным индексом

Файлы с неплотным индексом (индексно-последовательные файлы)

Неплотный (*разрежённый*) индекс строится именно для упорядоченных файлов. Для этих файлов используется принцип внутреннего упорядочения для уменьшения количества хранимых индексов.

Структура записи индекса

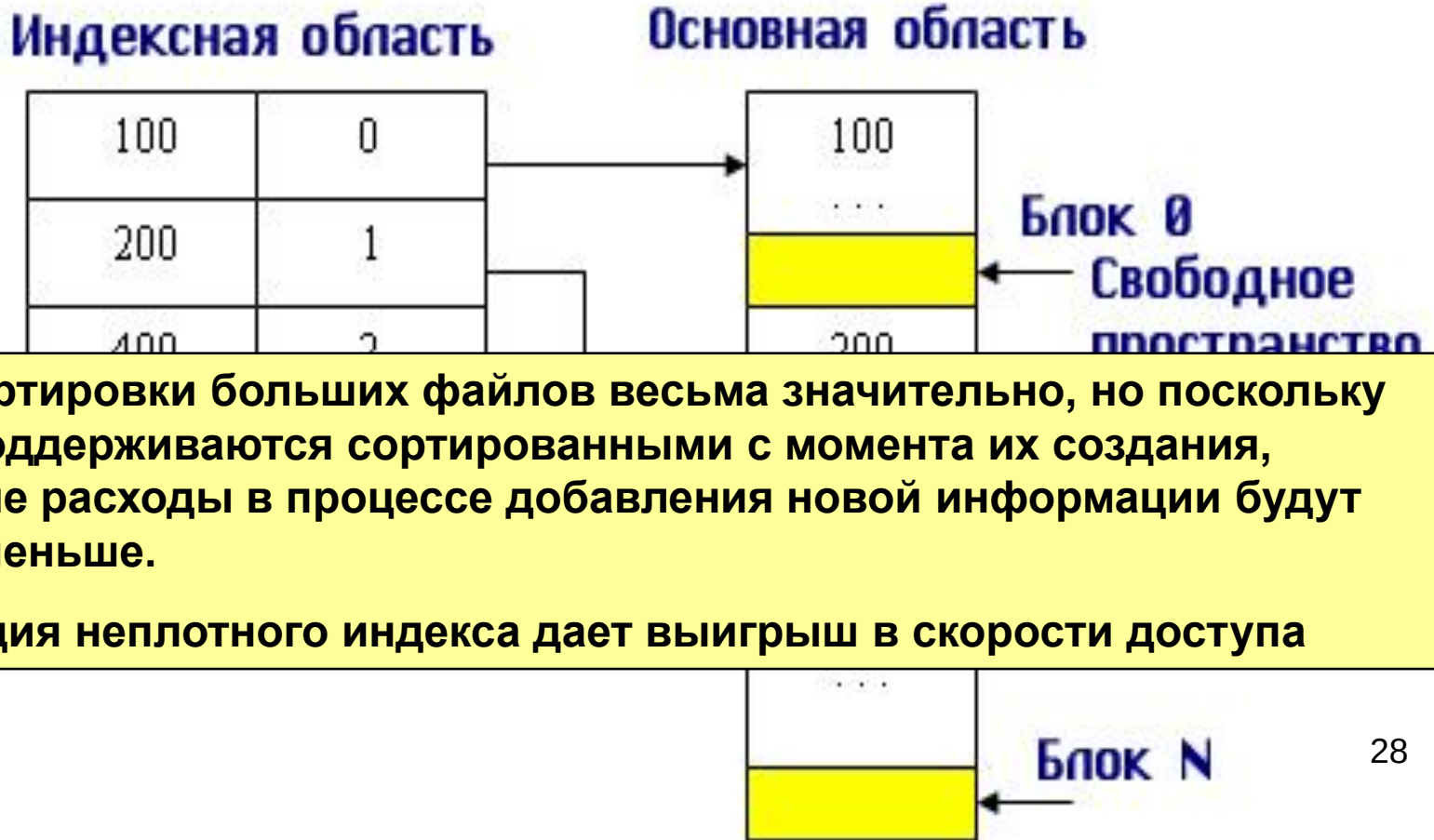
	Значение ключа первой записи блока	**Номер блока с этой записью**	

Разрежённый индекс (*sparse index*) характеризуется тем, что каждый ключ ассоциируется с определённым указателем *на блок* в сортированном файле данных.

Плотный индекс (*dense index*) в свою очередь отличается тем, что каждый ключ ассоциируется с определённым указателем *на запись* в сортированном файле данных.

Пример заполнения индексной и основной области при организации неплотного индекса

В индексной области мы теперь ищем нужный блок по заданному значению первичного ключа. Так как все записи упорядочены, то значение первой записи блока позволяет нам быстро определить, в каком блоке находится искомая запись. Все остальные действия происходят в основной области.



Время сортировки больших файлов весьма значительно, но поскольку файлы поддерживаются сортированными с момента их создания, накладные расходы в процессе добавления новой информации будут гораздо меньше.

организация неплотного индекса дает выигрыш в скорости доступа

Процедуры добавления и удаления новой записи

- Здесь новая запись должна заноситься сразу в требуемый блок на требуемое место, которое определяется заданным принципом упорядоченности на множестве значений первичного ключа.
- Поэтому сначала ищется требуемый блок основной памяти, в который надо поместить новую запись, а потом этот блок считывается,
- Затем в оперативной памяти корректируется содержимое блока и он снова записывается на диск на старое место.
- Должен быть задан процент первоначального заполнения блоков, но только применительно к основной области. В MS SQL server этот процент называется Full-factor и используется при формировании *кластеризованных* индексов. *Кластеризованными* называются индексы, в которых исходные записи физически упорядочены по значениям первичного ключа.
- При внесении новой записи индексная область не корректируется.

Организация индексов в виде B+-tree (B-деревьев)

- Построение B-деревьев связано с простой идеей построения индекса над уже построенным индексом.
- Если мы построим неплотный индекс, то сама индексная область может быть рассмотрена нами как основной файл, над которым надо снова построить неплотный индекс, а потом снова над новым индексом строим следующий и так до того момента, пока не останется всего один индексный блок.

Получим некоторое дерево, каждый родительский блок которого связан с одинаковым количеством подчиненных блоков, число которых равно числу индексных записей, размещаемых в одном блоке.

Количество обращений к диску при этом для поиска любой записи одинаково и равно количеству уровней в построенном дереве. Такие деревья называются *сбалансированными* (balanced) именно потому, что путь от корня до любого листа в этом древе одинаков. Именно термин «сбалансированное» от английского «balanced» — «сбалансированный, взвешенный» и дал название данному методу организации индекса.

B+-деревья

1. Корень либо является листом, либо имеет по крайней мере двух сыновей.
2. Каждый узел, за исключением корня и листьев, имеет от (m/2) до m сыновей.
3. Все пути от корня до любого листа имеют одинаковую длину.

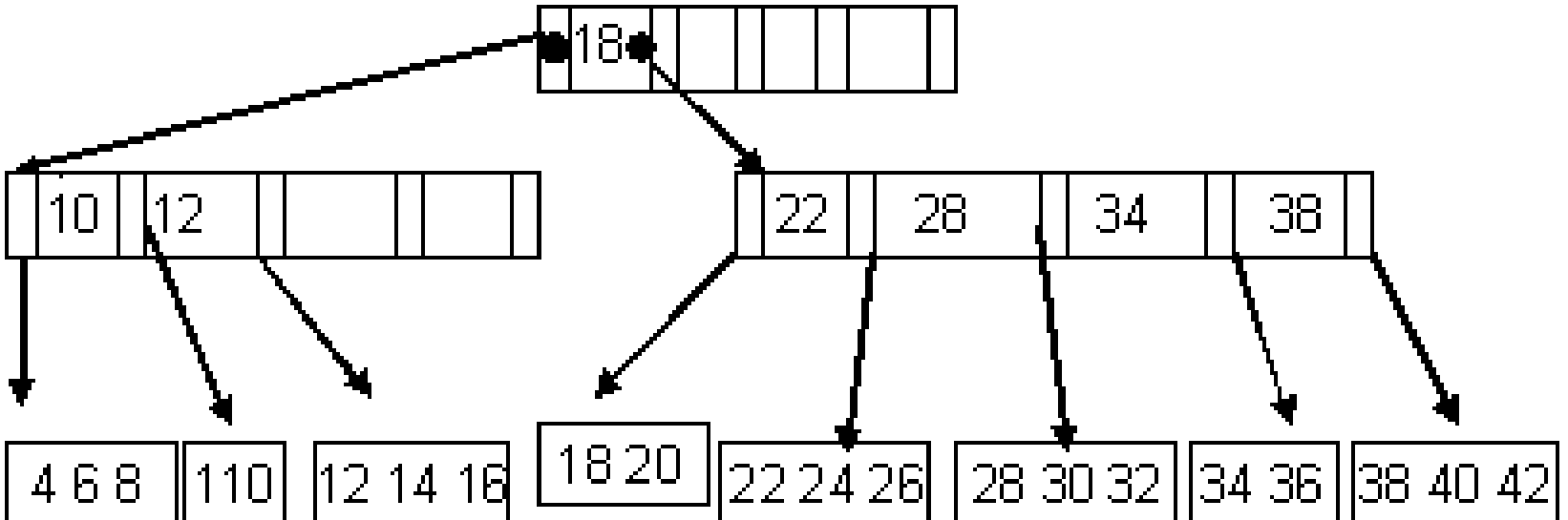
•B-дерево можно рассматривать как иерархический индекс, каждый узел в котором занимает блок во внешней памяти.

•Корень B-дерева является индексом первого уровня.

•Каждый нелистовой узел на B-дереве имеет форму $(p_0, k_1, p_1, k_2, p_2, ..., k_n, p_n)$, где p_i является указателем на i-го сына, $0 \le i \le n$, а k_i - ключ, $1 \le i \le n$.

•Ключи в узле упорядочены, поэтому $k_1 < k_2 < ... < k_n$. Все ключи в поддереве, на которые указывает p_0, меньше, чем k_1.

•В случае $1 \le i < n$ все ключи в поддереве, на которое указывает p_i, имеют значения не меньше, чем, k_i, и меньше, чем k_{i+1}. Все ключи в поддереве, на которое указывает p_n, имеют значения, не меньшие, чем k_n.



Построенное B-дерево

1 уровень -12500 блоков; 2 уровень -172 блока;
3 уровень -3 блока; 4 уровень -1 блок

Конкретный пример

Сравним время доступа при последовательном просмотре и при организации плотного индекса

Исходные данные:

1. Длина записи файла (LZ) — **128** байт.
2. Длина первичного ключа (LK) — **12** байт.
3. Количество записей в файле (KZ) — **100000**.
4. Размер блока (LB) — **1024** байт.

Рассчитаем размер индексной записи.
Для представления целого числа в пределах 100000 нам потребуется 3 байта, можем считать, что у нас допустима только четная адресация, поэтому нам надо отвести 4 байта для хранения номера записи,
тогда длина индексной записи будет равна сумме размера ключа и ссылки на номер записи, то есть:
LI = LK + 4 = 12 + 4 = 16 байт.

Количество индексных блоков

KIZB = LB/LI = 1024/16 = **64** индексных записи в одном блоке.

- Определим необходимое количество индексных блоков: **KIB = KZ/KZIB = 100000/64** = **1563** блока.
- максимальное количество обращений к диску при поиске произвольной записи:

  $\mathbf{T_{поиска} = \log_2 KIB + 1 = \log_2 1563 + 1 = 11 + 1 = 12}$ обращений к диску при организации плотного индекса .

Без индекса

Количество блоков, которое необходимо для хранения всех 100 000 записей, определим по следующей формуле:

- **KBO = KZ/(LB/LZ) - 100000/(1024/128) - 12500 блоков**.

Это означает, что **максимальное время доступа** равно **12500** обращений к диску.

Выигрыш существенный!

При плотном индексе

- Тдобавления = $\log_2 N + 1 + 1 + 1$.

Файлы с неплотным индексом

- T = $\log_2$KBO = $\log_2$12500 = 14 обращений к диску. Это существенно меньше, чем 12 500 обращений при произвольном хранении записей файла.

Если ранее ссылка рассчитывалась исходя из того, что требовалось ссылаться на 100000 записей, то теперь нам требуется ссылаться всего на 12 500 блоков, поэтому для ссылки достаточно двух байт. Тогда длина индексной записи будет равна:

- LI = LK + 2 = 12 + 2 = 14 байт.

Тогда количество индексных записей в одном блоке будет равно:

- KIZB = LB/LI = 1024/14 = 73 индексные записи в одном блоке

Определим количество индексных блоков, которое необходимо для хранения требуемых индексных записей:

- KIB = KBO/KZIB = 12500/73 = 172 блока.

Тогда время доступа по прежней формуле будет определяться:

- Тпоиска = $\log_2$KIB + 1 = $\log_2$172 + 1=8+1=**9** обращений к диску!

B-деревья

Построим подобное дерево для нашего примера и рассчитаем для него количество уровней и, соответственно, количество обращений к диску.

- **На первом уровне** число блоков равно числу блоков основной области, это нам известно, — оно равно **12 500** блоков.
- **Второй уровень** образуется из неплотного индекса, мы его тоже уже строили и вычислили, что количество блоков индексной области в этом случае равно **172** блокам. Над этим вторым уровнем снова построим неплотный индекс.
- Мы не будем менять длину индексной записи, а будем считать ее прежней, равной 14 байтам. Количество индексных записей в одном блоке нам тоже известно, и оно равно 73. Поэтому сразу определим, сколько блоков нам необходимо для хранения ссылок на 172 блока.
- KIB3 = KIB2/KZIB = 172/73 = **3** блока
- **Над третьим уровнем** строим новый, и на нем будет всего **один** блок, в котором будет всего три записи. Поэтому число уровней в построенном дереве равно **четырем**, и соответственно количество обращений к диску для доступа к произвольной записи равно **четырем**

Построенное B-дерево

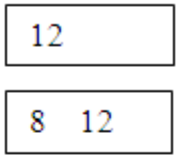
1 уровень -12500 блоков; 2 уровень -172 блока;
3 уровень -3 блока; 4 уровень -1 блок

Пример.

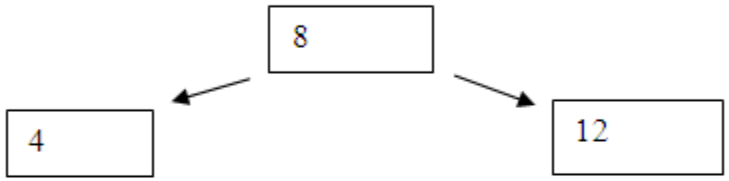
Пусть у нас есть дерево третьего порядка, с ключами 12, 8, 4, 9, 6, 13, 14, 16,100, 10.

Приходит ключ 12, его заносим в корень

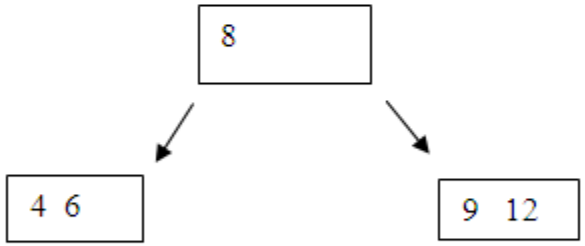
Приходит 8, 8 меньше 12



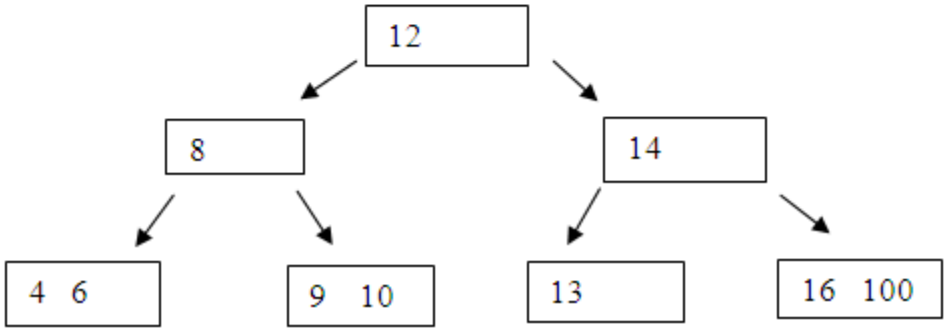
Приходит 4, места в корне нет, разбиваем на 2



Приходит 9, она больше 8, но меньше 12, 12 сдвигаем, перед ней записываем 9, приходит 6 она меньше 8, но больше 4, записываем ее после 4:



После того как придет 13, 14, 16, 100, 10 вид дерева будет следующим:



Задание для самостоятельной работы

Задание 1

- Постройте B-дерево для последовательности ключей, поступающих в следующем порядке: 20, 40, 10, 30, 15, 35, 7, 26, 18, 22, 5, 42, 13, 46, 27, 8, 32, 38, 24, 45, 25.

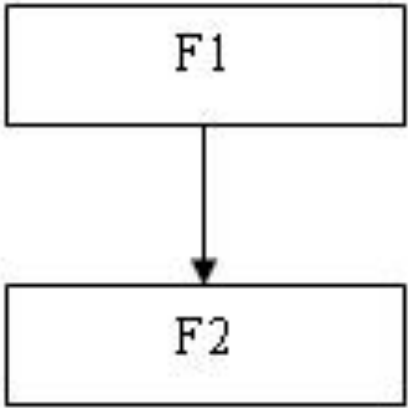
Задание 2

- Постройте многоуровневый индекс для файла, содержащего не менее 24 записей.

Моделирование отношений «один-ко-многим» на файловых структурах

Для моделирования отношений 1:M (один-ко-многим) и M:M (многие-ко-мно-гим) используется принцип организации цепочек записей внутри файла и ссылки на номера записей для нескольких взаимосвязанных файлов.

1. Моделирование отношения 1:M с использованием однонаправленных указателей



Основной файл - F1.

Ключ	Запись	Ссылка-указатель на первую запись в «Подчиненном» файле, с которой начинается цепочка записей файла F2, связанных с данной записью файла F1

Структура записи «подчиненного» файла – F2.

Указатель на следующую запись в цепочке	Содержимое записи

Пример

Если мы проведем интерпретацию данных связей на уровне предметной области, то можно утверждать, что преподаватель Иванов ведет предмет « Вычислительные сети» в группе 4306, «Моделирование» в группе 84305 и «Вычислительные сети» в группе 4309.

Номер записи	Ключ и остальная запись	Указатель
1	Иванов И. Н. ...	1
2	Петров А. А.	3
3	Сидоров П. А.	2
4	Яковлев В. В.	

F2 - список занятий, которые они ведут		
Номер записи	**Указатель на следующую запись в цепочке**	**Содержимое записи**
1	4	4306 Вычислительные сети
2	-	4307 Контроль и диагностика
3	6	4308 Вычислительные сети
4	5	84305 Моделирование
5	-	4309 Вычислительные сети
6	-	84405 Техническая диагностика

Алгоритм нахождения записей подчиненного файла

1. Ищется запись в «основном» файле в соответствии с его организацией (с помощью функции хэширования, или с использованием индексов, или другим образом). Если требуемая запись найдена, то переходим к шагу 2, в противном случае выводим сообщение об отсутствии записи основного файла.
2. Анализируем указатель в основном файле если он пустой, то есть стоит прочерк, значит, для этой записи нет ни одной связанной с ней записи в «подчиненном файле», и выводим соответствующее сообщение, в противном случае переходим к шагу 3.
3. По ссылке-указателю в найденной записи основного файла переходим прямым методом доступа по номеру записи на первую запись в цепочке «Подчиненного» файла. Переходим к шагу 4.
4. Анализируем текущую запись на содержание если это искомая запись, то мы заканчиваем поиск, в противном случае переходим к шагу 5.
5. Анализируем указатель на следующую запись в цепочке если он пуст, то выводим сообщение, что искомая запись отсутствует, и прекращаем поиск, в противном случае по ссылке-указателю переходим на следующую запись в «подчиненном файле» и снова переходим к шагу 4.

Алгоритм удаления записи из цепочки «подчиненного» файла

- **Шаг 1.** Ищется удаляемая запись в соответствии с ранее рассмотренным алгоритмом. Единственным отличием при этом является обязательное сохранение в специальной переменной номера предыдущей записи в цепочке, допустим, это переменная NP.
- **Шаг 2.** Запоминаем в специальной переменной указатель на следующую запись в найденной записи, например, заносим его в переменную NS. Переходим к шагу 3.
- **Шаг 3.** Помечаем специальным символом, например символом звездочка (*), найденную запись, то есть в позиции указателя на следующую запись в цепочке ставим символ «*» — это означает, что данная запись отсутствует, а место в файле свободно и может быть занято любой другой записью.
- **Шаг 4.** Переходим к записи с номером, который хранится в NP, и заменяем в ней указатель на содержимое переменной NS.

Задание для самостоятельной работы

Использование цепочек записей позволяет эффективно организовывать модификацию взаимосвязанных файлов.

Для того чтобы эффективно использовать дисковое пространство при включении новой записи в «подчиненный файл», ищется первое свободное место, т. е. запись, помеченная символом «*», и на ее место заносится новая запись, после этого производится модификация соответствующих указателей. При этом необходимо различать 3 случая:

1. Добавление записи на первое место в цепочке.
2. Добавление записи в конец цепочки.
3. Добавление записи на заданное место в цепочке.

Задание для самостоятельной работы:

Разработать алгоритмы добавления записи для всех трех случаев

Применение двойных указателей

- В **«основном файле»** один указатель равен номеру первой записи в цепочке записей «подчиненного файла», а второй — номеру последней записи.
- В **«подчиненном файле»** один указатель равен номеру следующей записи в цепочке, а другой — номеру предыдущей записи в цепочке. Для первой и последней записей в цепочке один из указателей пуст, то есть равен пробелу.

Пример

F1			
Номер записи	**Ключ и остальная запись**	**Указатель на первую запись**	**Указатель на последнюю запись**
1	**Иванов И. Н. ....**	**1**	**5**
2	**Петров А. А.**	**3**	**6**
3	**Сидоров П. А.**	**2**	**2**
4	**Яковлев В. В.**		

F2			
Номер записи	**Указатель на предыдущую запись в цепочке**	**Указатель на следующую запись в цепочке**	**Содержимое записи**
1	**-**	**4**	**4306 Вычислительные сети**
2	**-**	**-**	**4307 Контроль и диагностика**
3	**-**	**6**	**4308 Вычислительные сети**
4	**1**	**5**	**84305 Моделирование**
5	**4**	**-**	**4309 Вычислительные сети**
6	**3**	**-**	**84405 Техническая диагностика**

Инвертированные списки

- Достаточно часто в базах данных требуется проводить операции доступа по **вторичным ключам**.
- *Напомним, что **вторичным ключом** является набор атрибутов, которому соответствует набор искомых записей. Это означает, что существует множество записей, имеющих одинаковые значения вторичного ключа*.

- Для обеспечения ускорения доступа по вторичным ключам используются структуры, называемые ***инвертированными списками***, которые послужили основой организации ***индексных файлов*** для доступа по ***вторичным ключам***.

Инвертированные списки

Инвертированный список в общем случае — это двухуровневая индексная структура.

На первом уровне находится файл или часть файла, в которой упорядоченно расположены значения вторичных ключей.

Каждая запись с вторичным ключом имеет ссылку на номер первого блока в цепочке блоков, содержащих номера записей с данным значением вторичного ключа.

На втором уровне находится цепочка блоков, содержащих номера записей, содержащих одно и то же значение вторичного ключа. При этом блоки второго уровня упорядочены по значениям вторичного ключа.

На третьем уровне находится собственно основной файл.

Механизм доступа к записям по вторичному ключу

1. Ищем в области первого уровня заданное значение вторичного ключа
2. Затем по ссылке считываем блоки второго уровня, содержащие номера записей с заданным значением вторичного ключа
3. Далее уже прямым доступом загружаем в рабочую область пользователя содержимое всех записей, содержащих заданное значение вторичного ключа

Для одного основного файла может быть создано несколько инвертированных списков по разным вторичным ключам.
Организация вторичных списков действительно ускоряет поиск записей с заданным значением вторичного ключа.

Модификации основного файла

При модификации основного файла происходит следующая последовательность действий:

- Изменяется запись основного файла.
- Исключается старая ссылка на предыдущее значение вторичного ключа.
- Добавляется новая ссылка на новое значение вторичного ключа.

При этом следует отметить, что два последних шага выполняются для всех вторичных ключей, по которым созданы инвертированные списки. Такой процесс требует гораздо больше временных затрат, чем просто изменение содержимого записи основного файла без поддержки всех инвертированных списков.

Можно придерживаться следующей позиции: если база данных достаточно стабильна и ее содержимое практически не меняется, то построение вторичных индексов действительно может ускорить процесс обработки информации.

Общие принципы организации данных во внешней памяти в SQL-ориентированных СУБД

- Хранение таблиц
- Индексы
  - ✓B+-деревья
  - ✓Хэширование
- Журнальная информация
- Служебная информация

Модели физической организации данных при бесфайловой организации

- Файловая структура и система управления файлами являются прерогативой операционной среды, поэтому принципы обмена данными подчиняются законам операционной системы.
- По отношению к базам данных эти принципы могут быть далеки от оптимальности. СУБД подчиняется несколько иным принципам и стратегиям управления внешней памятью, чем те, которые поддерживают операционные среды для большинства пользовательских процессов или задач.
- СУБД взяли на себя непосредственное управление внешней памятью. При этом пространство внешней памяти предоставляется СУБД полностью для управления, а операционная среда не получает непосредственного доступа к этому пространству.

Организация System R. Используемая терминология

- Базовым понятием System R является понятие *таблицы*
  - приближенный к реализации аналог основного понятия реляционного подхода *отношения*
  - иногда, в зависимости от контекста, мы будем использовать и этот термин
- Таблица – это регулярная структура данных, состоящая из конечного набора однотипных записей – кортежей.
- Каждый кортеж одного таблицы состоит из конечного (и одинакового) числа полей кортежа, причем
  - *i*-тое поле каждого кортежа одной таблицы может содержать данные только одного типа, и
  - набор допустимых типов данных в System R предопределен и фиксирован

Основные понятия, цели и общая организация System R (5)
Используемая терминология (5)

- **Таблицы**, составляющие базу данных System R, могут физически храниться в одном или нескольких **сегментах**, которые проще всего понимать как **файлы** внешней памяти
- **Сегменты** разбиваются на **страницы**, в которых располагаются **кортежи таблиц** и вспомогательные служебные структуры данных – **индексы**
- Соответственно, каждый **сегмент** содержит две группы **страниц** – страницы **данных** и страницы **индексной информации**
- Страницы каждой группы имеют фиксированный размер, но страницы с индексной информацией меньше по размеру, чем страницы данных
- В **страницах** данных могут располагаться **кортежи более чем одной таблицы**

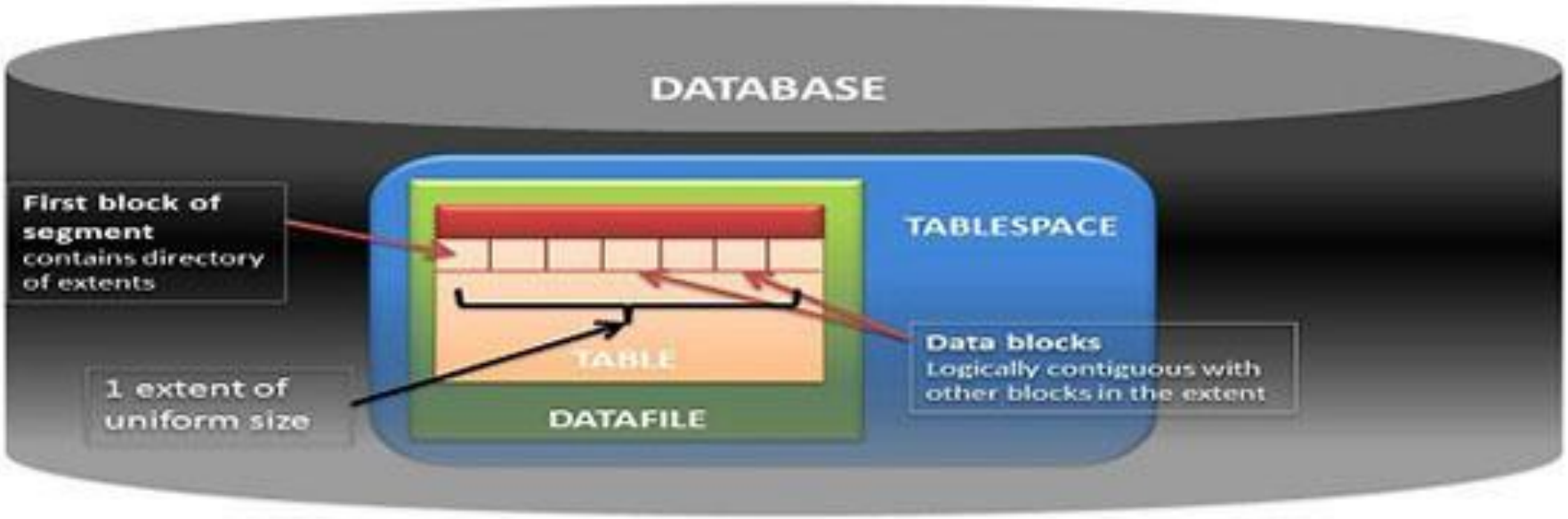
Схемы структуризации

- физическая, которая определяет хранимые данные
- логическая, которая определяет некоторые логические структуры, связанные с концептуальной моделью данных

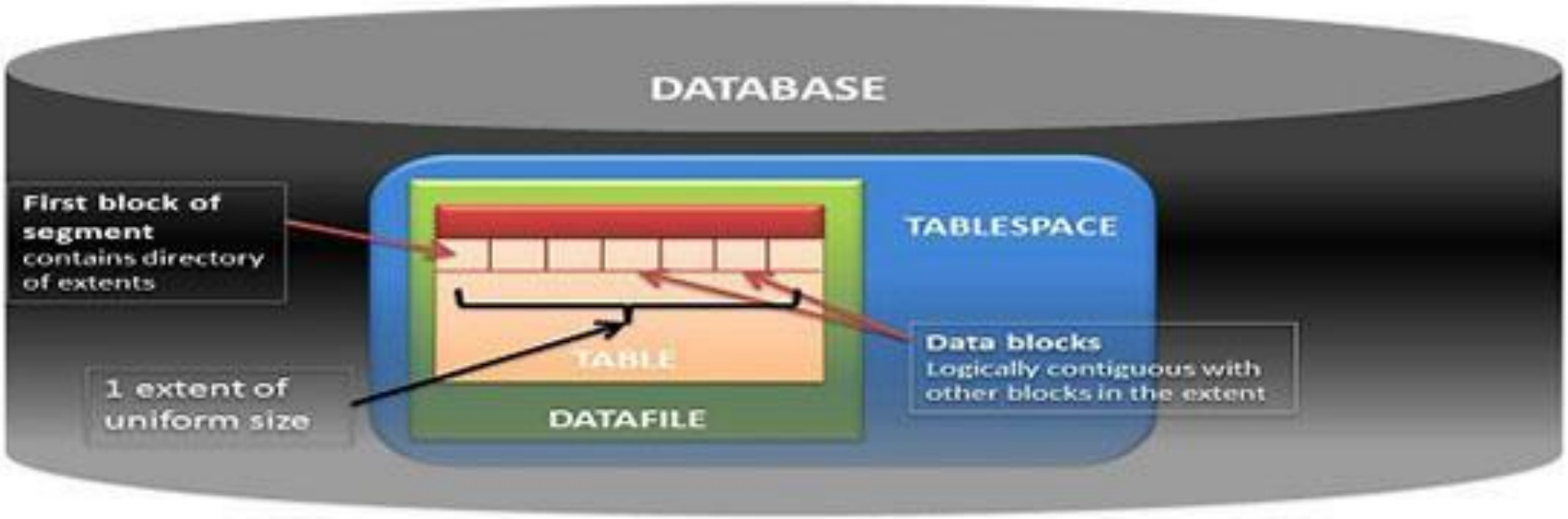
Классификация

- **Чанк (chank)** — представляет собой часть диска, физическое пространство на диске, которое ассоциировано одному процессу (on line процессу обработки данных).
- **Чанком** может быть назначено неструктурированное устройство, часть этого устройства, блочно-ориентированное устройство или просто файл UNIX.
- **Чанк** характеризуется маршрутным именем, смещением (от физического начала устройства до начальной точки на устройстве, которая используется как чанк), размером, заданным в Кбайтах или Мбайтах.
- При использовании блочных устройств и файлов величина смещения считается равной нулю.
- Логические единицы образуются совокупностью экстентов, то есть **таблица** моделируется совокупностью ***экстентов***.
- **Экстент** — это непрерывная область дисковой памяти.

Табличные пространства Oracle



Табличные пространства Oracle



Экстент

Для моделирования таблицы используется 2 типа экстентов: первый и последующие.

- Первый экстент задается при создании нового объекта типа таблица, размер задается при создании. EXTENTSIZE — размер первого экстента, NEXT SIZE — размер каждого следующего экстента.
- Минимальный размер экстента в каждой системе свой, в большинстве случаев равен 4 страницам, максимальный — 2 Гб.
- Новый экстент создается после заполнения предыдущего и связывается с ним специальной ссылкой, которая располагается на последней странице экстента. В ряде систем экстенты называются *сегментами*, но фактически эти понятия эквиваленты.
- При динамическом заполнении БД данными применяется механизм адаптивного определения размера экстентов.
- Внутри экстента идет учет свободных станиц.

Экстент и страницы

- Между экстентами, которые располагаются друг за другом без промежутков, производится операция ***конкатенации***, которая увеличивает размер первого экстента.
- ***Механизм удвоения размера экстента***: если число выделяемых экстентов для процесса растет в пропорции, кратной 16, то размер экстента удваивается каждые 16 экстентов (если размер текущего экстента 16 Кбайт, то после заполнения 16 экстентов данного размера размер следующего будет увеличен до 32 Кбайт).
- Совокупность экстентов моделирует логическую единицу — ***таблицу-отношение (tblspace).***

Экстенты состоят из четырех типов страниц:

- ***страницы данных,***
- ***страницы индексов,***
- ***битовые страницы***
- ***страницы blob-объектов***.

Страницы blob-объектов

Blob — это сокращение Binary Larg Object, и соответствует неструктурированным данным. В ранних СУБД такие данные относились к типу **Memo**.

В современных СУБД к этому типу относятся

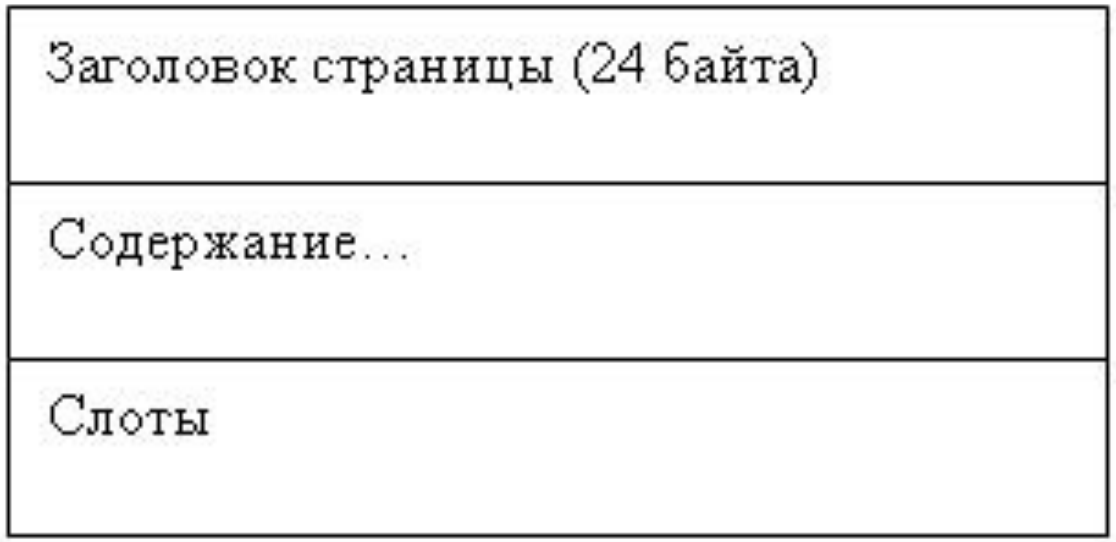
- неструктурированные большие текстовые данные,
- картинки,
- наборы машинных кодов.

Для СУБД важно знать, что этот объект надо хранить целиком, что размеры этих объектов от записи к записи могут резко отличаться и этот размер в общем случае неограничен.

Страницы данных

- Основной единицей осуществления операций обмена (ввода-вывода) является страница данных.
- Все данные хранятся постранично.
- При табличном хранении данные на одной странице являются однородными, то есть станица может хранить только данные или только индексы.

Обобщенная структура страницы данных



Структура страниц

- **Слот** — это **4-байтовое** слово, 2 байта соответствуют смещению строки на странице и 2 байта — длина строки.
- Слоты характеризуют размещение строк данных на странице. На одной странице хранится не более **255 строк**.
- В базе данных каждая строка имеет ***уникальный идентификатор*** в рамках всей базы данных,
- **RowID** — номер строки, он имеет размер 4 байта и состоит из номера страницы и номера строки на странице.
- Под номер страницы отводится **3 байта**, поэтому при такой идентификации возможна адресация

  к **16 777 215 страницам**.

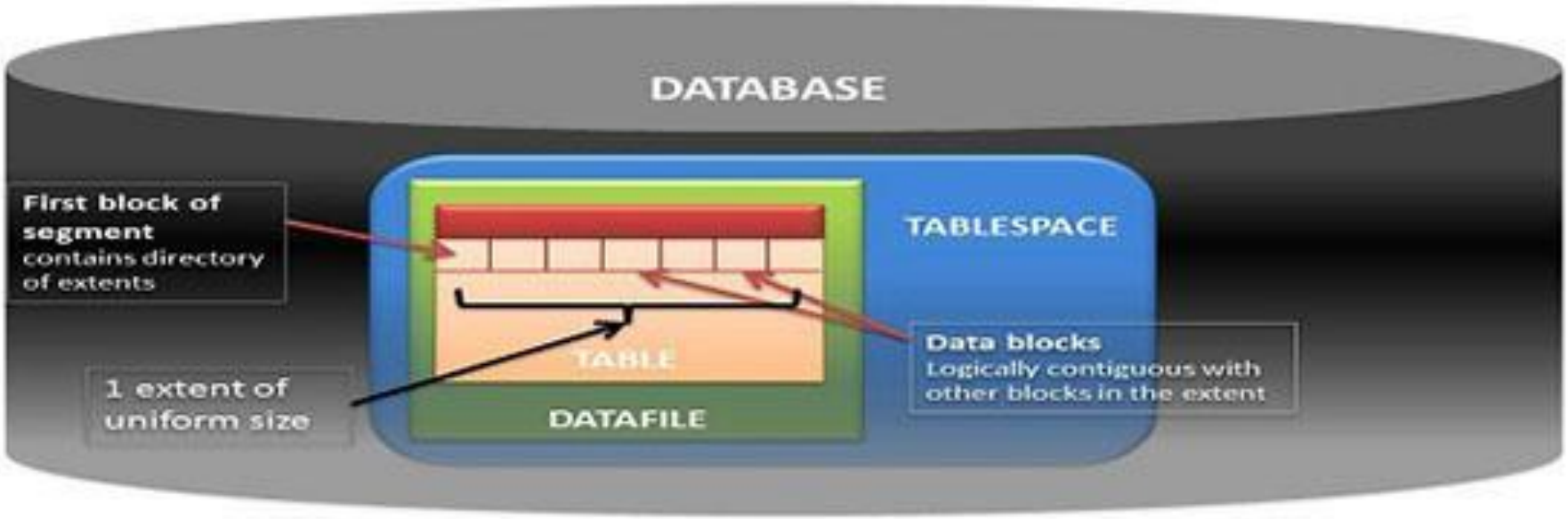
Структура страниц

- При упорядочении строк на страницах не происходит физического перемещения строк, все манипуляции происходят со слотами.
- При переполнении страниц создается специальный вид страниц, называемых страницами остатка.
- Строки, не уместившиеся на основной странице, связываются (линкуются) со своим продолжением на страницах остатка с помощью ссылок-указателей «вперед» (то есть на продолжение), которые содержат номер страницы и номер слота на странице.
- Страницы индексов организованы в виде B-деревьев.

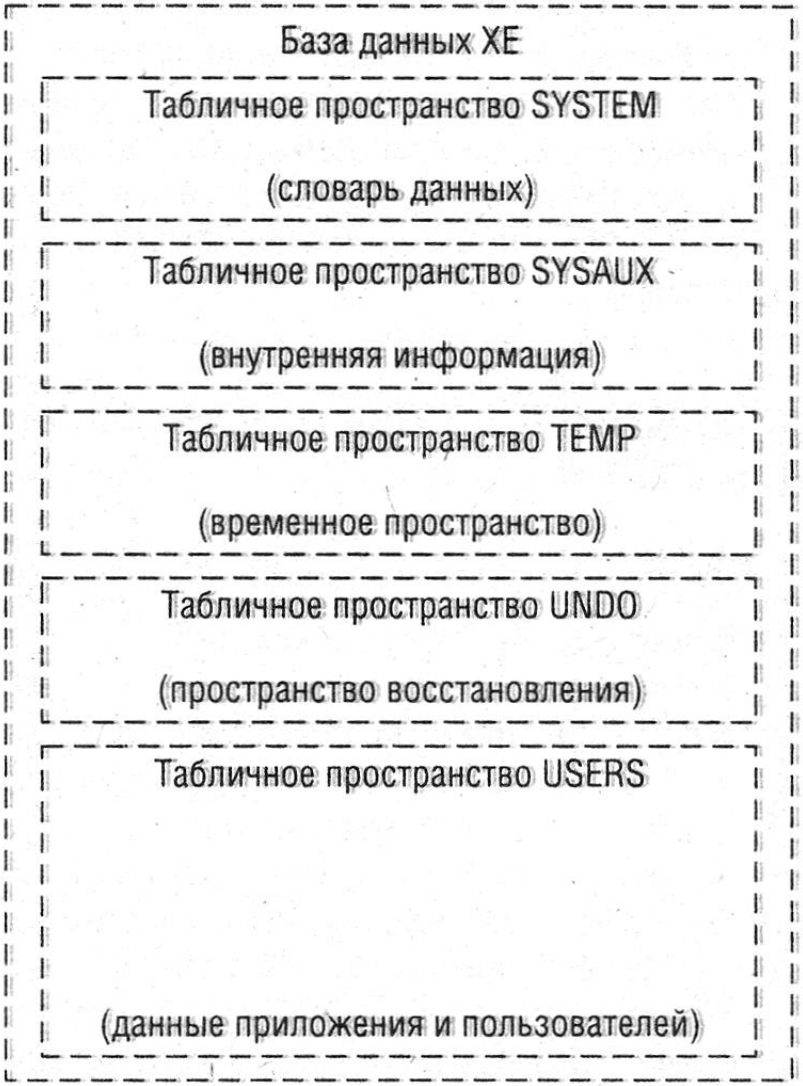
Битовые страницы

- Битовые страницы служат для трассировки других типов страниц.
- В зависимости от трассируемых страниц битовые страницы строятся по 2-битовой или 4-битовой схеме.
- 4-битовые страницы служат для хранения сведений о столбцах типа Varchar, Byte, Text, для остальных типов данных используются 2-битовые страницы.
- Битовая структура трассирует 32 страницы.
- Каждая битовая структура представлена двумя 4-байтными словами.
- Каждая i-я позиция описывает одну i-ю страницу.
- Сочетание разрядов в i-х позициях двух слов обозначает состояние данной страницы: ее тип и занятость.

Табличные пространства Oracle



В Oracle



Табличные пространства при хранении логически и физически отделяют друг от друга различные типы информации базы данных

В Oracle

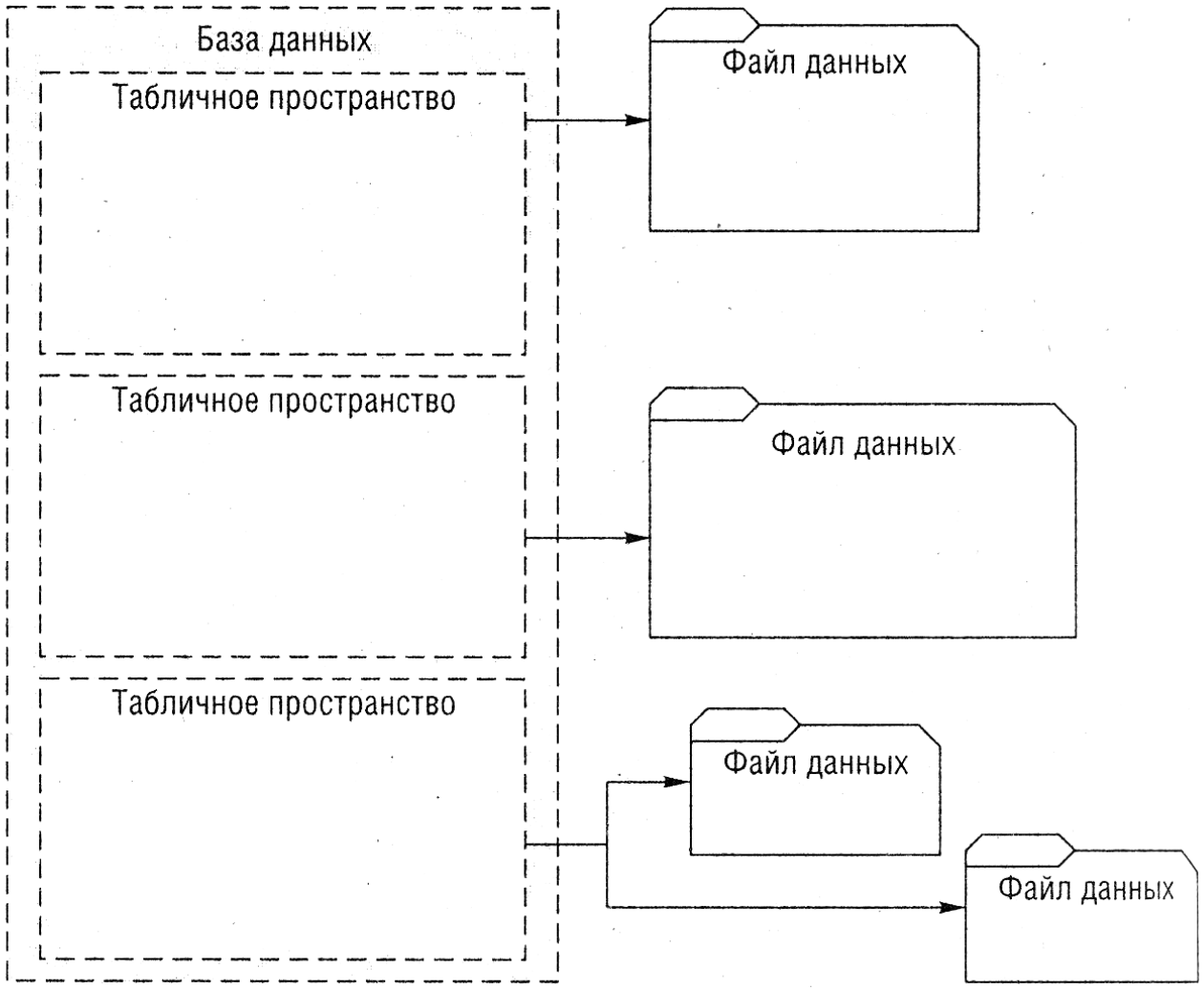
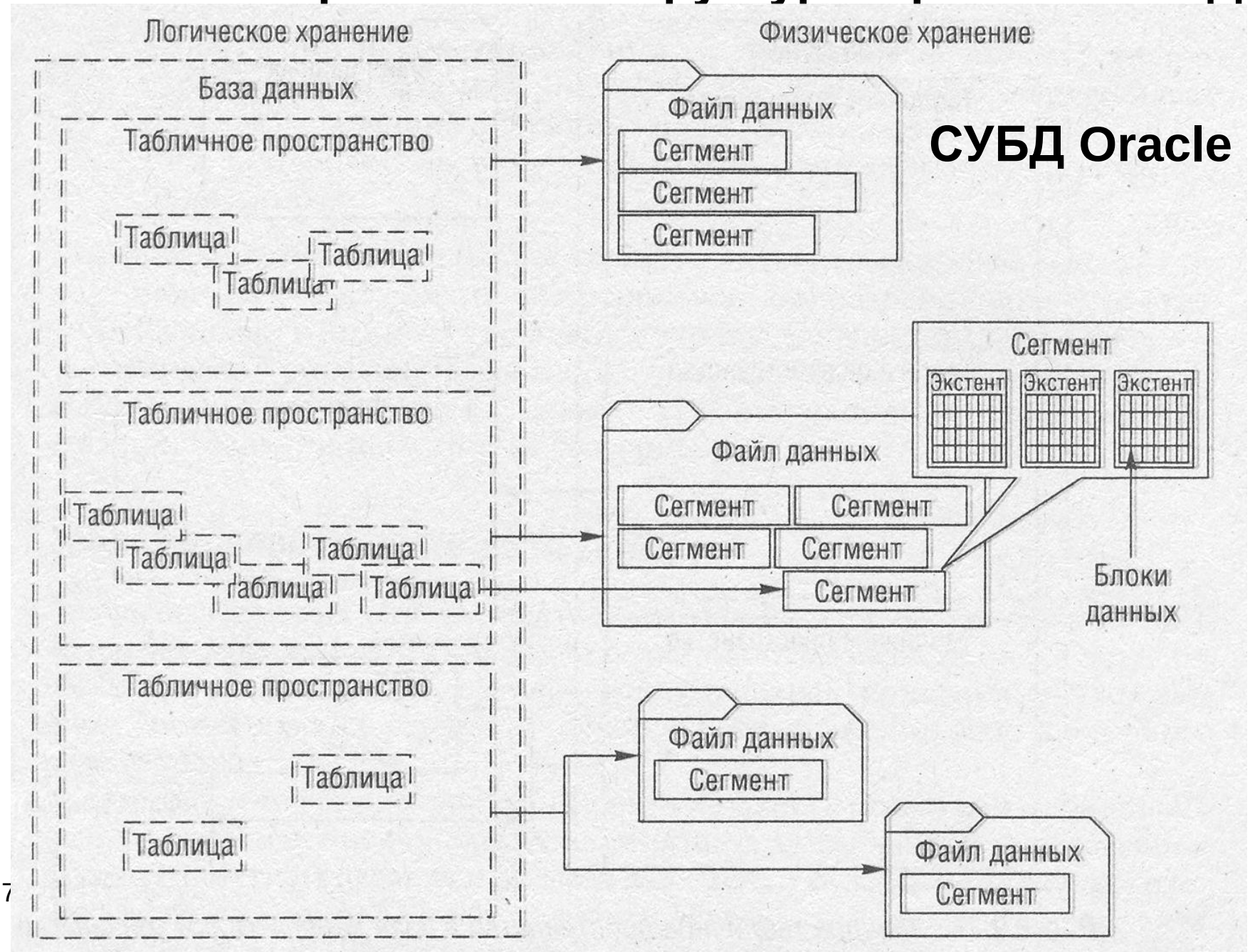


Рис. 8.2. *Каждое табличное пространство в базе данных Oracle физически хранит свои данные в одном или нескольких связанных файлах*

Логические и физические структуры хранения в БД



Структуры памяти

При обработке данных СУБД организует **специальные структуры**

- в оперативной памяти, называемые ***разделяемой памятью***,
- и специальные структуры во внешней памяти, называемые ***журналами транзакций***.

Разделяемая память служит для кэширования данных при работе с внешней памятью с целью сокращения времени доступа, кроме того, разделяемая память служит для эффективной поддержки режимов одновременной параллельной работы пользователей с базой данных.

Журнал транзакций служит для управления корректным выполнением транзакций.

Структура хранения данных для MS SQL

В версии 6.0 и 6.5 MS SQL Server

•Файлы операционной системы представляются как устройства для хранения БД (Device), устройства нумеруются.

•Сервер может управлять 256 устройствами.

•Главное устройство называется **MASTER**: на нем хранятся системные базы данных:

Master, **Model**, **Pubs**, **TempDb**.

•Устройство **Master** имеет номер 0 — ноль.

•Каждое устройство разбивается не более чем на 16 777 216 виртуальных страниц по 2 Кбайта (Virtual page), максимальный размер устройства 32 Гбайт.

•Первые 4 страницы устройства Master заняты под блок конфигурации (Configuration block) — там хранятся все параметры конфигурации сервера.

Структура хранения данных для MS SQL

На устройствах размещаются конкретные базы. На каждом устройстве может быть размещено несколько баз, но и одна база может быть размещена на нескольких устройствах.

Каждая страница БД имеет свой уникальный номер.

Физически используются 3 единицы хранения данных:

- страница;
- блок (extent)-16Кб из 8 следующих друг за другом страниц;
- единица размещения (Allocation Unit) — 512 Кб из 32 последовательных блоков (256 страниц).

При создании новой базы данных пространство для нее отводится единицами размещения. Минимальный объем базы данных равен 1 Мбайт, т.е. - 2 единицы размещения.

Страница

Страницы бывают пяти типов:

1. страницы размещения (Allocation page);
2. страницы данных (Data page);
3. индексные страницы (Index page);
4. текстовые страницы (Text/image page);
5. статистические страницы (Distribution page).

Любая страница имеет заголовок (32 байт).

Заголовок содержит:

- номер страницы,
- номера предыдущей и следующей страниц,
- идентификатор объекта — владельца страницы
- сведения о свободном пространстве на странице.

Страницы связаны в двунаправленный список.