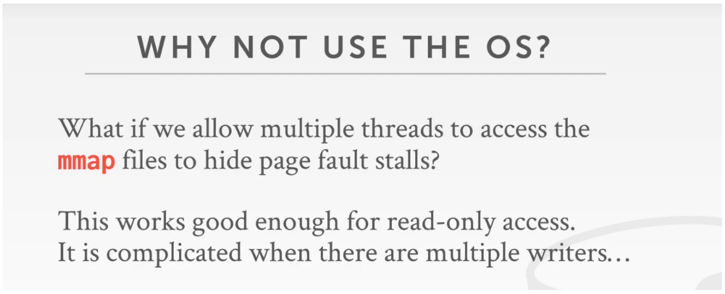
03–02

19.05–19.09
it doesn't understand any of the high–level semantics or what a query is, what data wants to read
它也无须理解任何高级语义，或者查询要做什么，以及要去读取哪些数据
19.09–19.11
right

19.11–
so we want to by going with virtual memory by going with memory mapped files, we're giving up controls or giving up knowledge that we have inside our database system over to the OS. that's blind and doesn't know anything
so 我们想要虚拟内存通过mmap来映射到文件，我们无须自己控制，无须绞尽脑汁在我们的数据库系统中自己来搞，交给操作系统就行了，无须关心它背后的任何事情

19.23–19.24
right

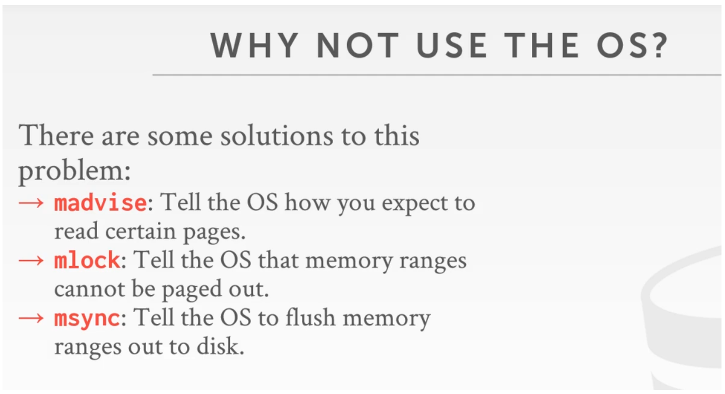
19.24–19.27
so if we're only reading data
So，如果我们只读取数据
19.27–19.31
there's a bunch of syscalls we can to mitigate some of these problems
那么我们就可以通过一系列系统调用来减少我们所遇到的一些问题
19.31–19.33
but if we start writing things
但如果我们开始写入某些东西
19.33–19.34
then it becomes problematic
那这就会变得很有问题


WHY NOT USE THE OS?

What if we allow multiple threads to access the mmap files to hide page fault stalls?

This works good enough for read-only access.
It is complicated when there are multiple writers...

19.34–19.40
because now the OS doesn't know that certain pages have to be flushed out the disk before other pages do
因为现在操作系统并不知道某些pages必须要在其他pages执行之前先从内存刷到磁盘中
19.40–19.44

again what we'll cover this later when we talk about logging and concurrency control
当我们讨论日志和并发控制时，再来介绍这个
19.44–19.47
but the OS sees yeah I need to write some data out
但OS表示我需要将某些数据写出（到磁盘）
19.47–19.48
means go ahead and write it out
意味着继续，写出到磁盘
19.48–19.50
it doesn't 'whether that was an okay thing to do or not
操作系统并不知道这么做ok不ok


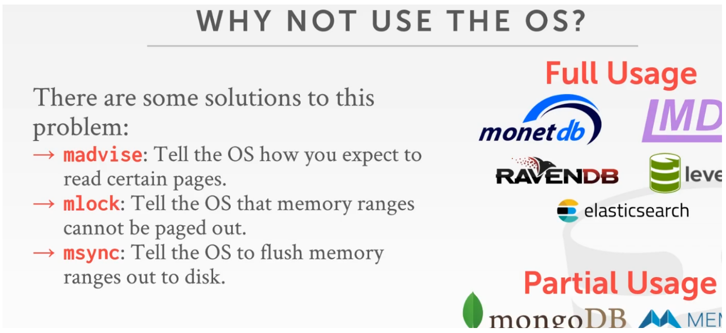
WHY NOT USE THE OS?

There are some solutions to this problem:

→ **madvise**: Tell the OS how you expect to read certain pages.
→ **mlock**: Tell the OS that memory ranges cannot be paged out.
→ **msync**: Tell the OS to flush memory ranges out to disk.

19.50–20.01
so you get around this by giving it hints, like using madvise to tell it how you're gonna access certain pages whether it's read sequential and random
so 你可以通过给它一些提示来解决该问题，例如使用madvise告诉它如何访问某些页面（无论是顺序读取还是随机读取）

how to prevent pages from beginning paged out
如何通过mlock阻止pages被回收
20.01–20.06
although you can lock doesn't prevent it from getting written out which again could to still be a problem
尽管你可以锁定但并不能阻止它被写出（到磁盘），这仍然可能是一个问题
20.06–20.08
and this is when you tell it to flush
这个 msync表示你在告诉它要将数据刷出到磁盘中
20.08–20.16
so I would say that memory map files of virtual memory sounds like a seductive thing we want to use in our database system
我想说的是虚拟内存中的mmap听起来好像是一种非常诱人的东西，搞得我们很想将它运用在我们的数据库系统中
20.16–20.19
and every year some student says why are we doing all this buffer pool thing
每年都有学生会问，为什么我们总做缓存池之类的东西
20.19–20.21
why can't we just let the os I do this for us
为什么不能让OS为我们做到这一点
20.21–20.23
and trust me you don't want to do this
相信我，你也不想让它来做这些

20.23–20.26
because it can be you have performance bottlenecks
~~因为你们可以会遇上性能瓶颈问题~~
因为你可能会在此处遇到性能瓶颈
20.26–20.27
and you'll have correct problems
你需要去解决这些问题


WHY NOT USE THE OS?

There are some solutions to this problem:
→ madvise: Tell the OS how you expect to read certain pages.
→ mlock: Tell the OS that memory ranges cannot be paged out.
→ msync: Tell the OS to flush memory ranges out to disk.

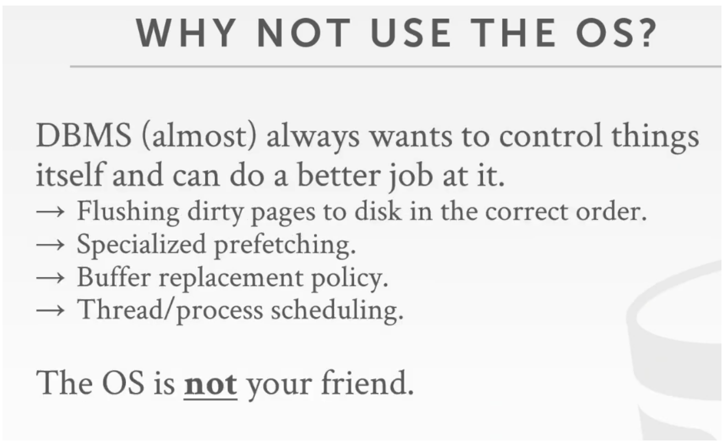
20.27–20.32
so there's not very many systems out there that actually use mmap
So，实际上并没有很多系统使用mmap
20.32–20.35
the most famous two are probably Monet DB and LM DB
最有名的两个可能就是Monet DB和 LMDB
20.35–20.38
level DB you ever heard of that from Google is another one
你以前所听过的谷歌的level DB则是另一个例子
20.38–20.42
elastic search is a search engine or the document store
elasticsearch则是一个搜索引擎或者说是文档存储系统
20.43–20.47
then Raven DB is a JSON database ~~at Israel~~
接着，Raven DB是一个JSON数据库
20.47–20.50
so all these guys use mmap
这些数据库都使用了mmap
20.50–20.55
but there's a bunch of extra stuff you have to do to prevent the OS from doing things that are incorrect
但你仍然需要做一些额外的事情来防止操作系统做一些错误的事情
20.55–20.59
or there's certain limitations or assumptions you have to make about what the os is allowed to do
或者你们必须做出一些限制或者假设来确保哪些事情操作系统可以做
~~20.59–21.01 砍掉不要~~
~~there's not right~~
~~这里并不正确~~
21.01–21.04
so this is like I mean there's a few more but there's not very many
看这里，是不是少了点什么，我的意思是应该有更多

21.04–21.06
so what’s missing here
这里缺了什么呢?
21.06–21.08
we’re missing all the major database systems
这里我们缺了所有的主流数据库系统

21.08–21.11
perhaps my sequel Oracle db2 sequel server
例如，MySQL，Oracle，DB2以及SQL server
21.11–21.12
none of those guys use mmap
这些数据库都没有使用mmap
21.12–21.13
cuz it’s a bad idea
因为这是一个糟糕的想法
21.13–21.14
because you’re giving up control
因为你们放弃了控制权
21.14–21.19
and the database can always do better then what the operating system could try to figure out
并且数据库能比操作系统所做的要来得更好
21.19–21.23
so there’s some systems that still use mmap in very limited cases
So，仍然有些系统在非常有限的情况下去使用mmap
21.23–21.26
this is actually out of ~~date MIT I~~ mean I talked to the guys last week
实际上，上个星期我还跟人讲过这个
21.26–21.28
memSQL got rid of mmap entirely
MemSQL已经完全摆脱了mmap
21.28–21.30
SQLite has a special engine
SQLite有一个特殊的引擎
21.30–21.35
you have to tell I want to use Mmap for some embedded devices that’s what actually you want to use
你必须告诉数据库你想在一些你所想使用的嵌入式设备上使用mmap

21.35–21.36
but default you don’t get this
但默认情况下，你不会用到这个

21.36-21.39
influx DB only uses this for like read-only caches
Influx DB只有在只读缓存上才使用mmap
21.39-21.42
but the example I always like to give it talk about is MongoDB
但是我一直喜欢谈论的例子是MongoDB
21.42-21.45
everyone here has heard of MongoDB before right
在座的每个人之前应该都已经听说过MongoDB了
21.45-21.47
that's a famous JSON database system
这是一个非常著名的JSON数据库系统
21.47-21.53
so when they first started their default storage engine or storage manager was using Mmap
当他们第一次开发他们的默认存储引擎或者存储管理器时，所用的就是mmap
21.53-21.56
and there's a bunch of crap they had to do to make that thing actually work
为了让这个引擎能够正常工作，他们做了很多无用功
21.56-22.00
but it was a it ~~was a super button pimp~~ was a big bottleneck for them
但对于它们来说，这是一个巨大的瓶颈
22.00-22.02
and then they raised put of money
接着，他们筹集了很多钱
22.02-22.06
and then the first thing they did was got rid of mmap and got it
他们首先干的事情就是摆脱mmap，并且他们做到了
22.06-22.09
you know bought this thing called wire tire which was a non-mmap storage engine
他们买了一个叫做WiredTiger的非mmap存储引擎
22.09-22.14
so if a map was a good idea， these guys had all the money in the world had some top engineers they could have figured it out
如果mmap是一个好的想法，那么这群土豪手下的顶级工程师肯定能将它证明
22.14-22.15
but it was just it became untenable
但这种想法站不住脚



WHY NOT USE THE OS?

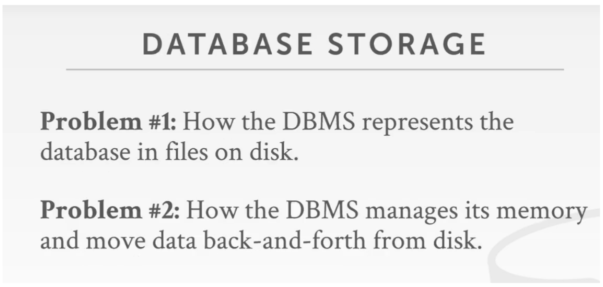
DBMS (almost) always wants to control things itself and can do a better job at it.
→ Flushing dirty pages to disk in the correct order.
→ Specialized prefetching.
→ Buffer replacement policy.
→ Thread/process scheduling.

The OS is **not** your friend.

22.15-22.17

so if I die
So，如果我死了的话
22.17-22.22 （我太难了？？？）
in this class and you want to have a memorial something just say Andy hated Mmap
上这门课的同学可能会记住某事，那就是Andy痛恨mmap
22.22-22.24
you can you can even publicly say these things
你们可以公开说这些事情
22.24-22.24
okay
这没关系，Ok
22.24-22.29
we're actually working a paper at paper
实际上，我们正在写一篇论文
22.29-22.31
and they let this year and actually proving that is a bad idea
实际上，这篇论文的内容就是为了证明mmap是一个糟糕的想法
22.31-22.31
all right
22.31-22.37
so the main takeaway I want you to get from this is that the database system is oh it can always do better
我想让你们从中得到的主要结论就是，数据库始终可以做得更好
22.37-22.40
it always knows exactly what the queries are trying to do
数据库总是很确定地知道查询要做什么

22.40-22.42
it knows what the workload looks like
它也知道工作负载是怎么样的
22.42-22.43
and therefore it can make the best decision
因此，它可以做出最佳选择
22.43-22.45
the operating system doesn't know anything
但是操作系统啥也不知道
22.45-22.48
it just sees a bunch of again reads and writes read and write calls
它只是看到了一些读写调用
22.48-22.54
so some of the things that we'll talk about maybe late in the semester that we can do if we're not using mmap
我们会在这学期后面点的时候去讨论些东西，即如果我们不使用mmap，我们能做什么
22.54-22.57
is like prefetching，better replacement policies，better scheduling
例如：预取，更好的替换策略，更好的调度之类的东西
22.57-23.00
again the OS is sort of general purpose pick up truck
操作系统就像是一辆通用货车

23.00-23.05 **
whereas we can tune our system look like a Porsche or Ferrari to be exactly what we want to do for application
然而我们可以像保时捷或法拉利那样调整我们的系统，使其完全符合我们的应用需求
23.05-23.09
so another main takeaway is that the operating system is not your friend
另一个主要想法则是，操作系统并不是你的朋友
23.09-23.11
we don't want to rely on up, we want try to avoid it as much as possible
我们并不想依赖操作系统，我们想试着尽可能的避开它
23.11-23.14
because it's gonna make decisions that could be hurtful to our database system
因为它可能会做出某些对我们的数据库系统有害的决策
23.14-23.16
so it's like a frenemy
因此，操作系统亦敌亦友
23.16-23.17
you need it to survive
你需要依靠它来存活
23.17-23.18
but ideally you don't want to talk to it
但理想情况下，你并不想和它说话
23.18-23.21
all right all right

DATABASE STORAGE

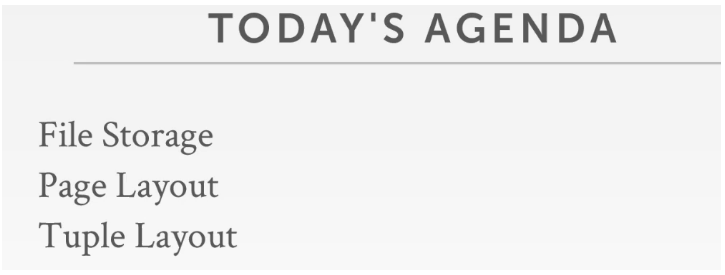
Problem #1: How the DBMS represents the database in files on disk.

Problem #2: How the DBMS manages its memory and move data back-and-forth from disk.

23.21-23.23
so for database storage this is we're gonna focus on today
我们今天的重点就是数据库存储这块内容
23.23-23.26
so there's two main problems we have to take care of
此处我们必须关心的问题主要有两个
23.26-23.30
the first is how we're gonna represent the data on files on disk
第一个问题是是我们如何表示磁盘上文件中的数据
23.30-23.35
and the second is that how we actually manage the Move memory back and forth right between the disk files and the buffer pool
第二个问题则是，我们实际该如何管理内存以及在硬盘间来回移动数据
23.35-23.40
so for this lecture today we're gonna focus on this problem
So，在今天这节课上，我们会重点讨论第一个问题
23.40-23.42
next class well stuff up in this problem

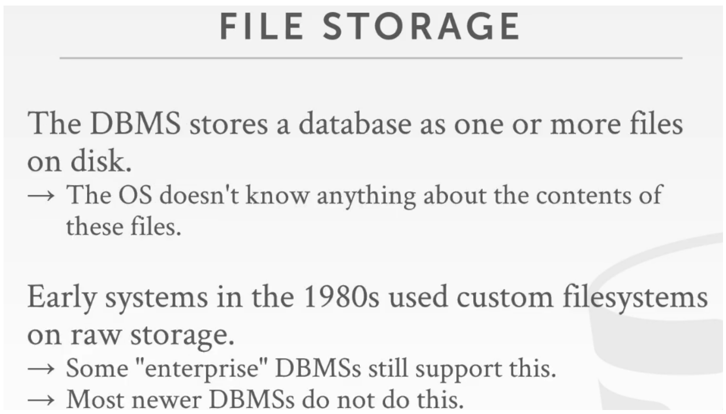
下堂课则会很好地解决这个问题
23.42-23.46
and then starting when we talk about buffer pools on Wednesday next week， we'll focus on the second problem
接着，我们会在下周三讨论buffer 缓存池，并且重点关注第二个问题
23.46-23.46
okay

23.46-23.48
all right
++

23.48-23.50
so today's lecture again we're gonna go to that first question
So，今天这节课我们会去讨论第一个问题

TODAY'S AGENDA

File Storage
Page Layout
Tuple Layout

23.50-23.54
how are we actually gonna represent the the database on files on disk
即我们实际该如何用磁盘上的文件来表示数据库
23.54-24.00
so we're first talk about how would organize the database across a sequence of pages
So，我们首先讨论如何在一系列页（pages）上组织数据库
24.00-24.04
now let's talk about how we're actually gonna store the pages inside those files
现在我们来讨论该如何将这些页存储在这些文件中
24.04-24.08
and then let's talk about what's actually the tuples look like inside those pages
然后，我们会去讨论这些页中的tuple看起来是什么样的
24.08-24.16
alright so we're gonna go sort of at a macro level and deep you know step down to you know inside the data that we're actually storing
So，我们从宏观的角度出来，然后逐步深入到我们所存储的数据内部
24.16-24.18
all right


FILE STORAGE

The DBMS stores a database as one or more files on disk.
→ The OS doesn't know anything about the contents of these files.

Early systems in the 1980s used custom filesystems on raw storage.
→ Some "enterprise" DBMSs still support this.
→ Most newer DBMSs do not do this.

24.18–24.22
so at the end of the day, the database is just a bunch of files on disk
So，到头来，数据库其实就是磁盘上的一堆文件
24.22–24.25
some system stored the database as one file
~~某些系统用一个文件来保存数据库~~
某些系统将数据库存储为一个文件
24.25–24.26
like SQLite does that
例如SQLite就是这么干的
24.26–24.31
for the first homework you download that DB file that’s the entire database and capsulated it in that single file
你第一个作业中的db文件，其实就是整个数据库，SQLite将它封装在一个文件里面了

24.31–24.35
most other systems however store things in across multiple files
然而，其他大部分系统会将这些东西分为多个文件来保存
24.35–24.39
so here a look at like the the data directory from I see bone Postgres
So，来看下Postgres中的数据目录
24.39–24.41
you’ll see a bunch of different directories and a bunch of files
你会看到一系列不同的目录以及一系列文件
24.41–24.41
right

24.41–24.45
you do this because you know databases could be very large like petabytes
你这样做是因为，数据库可能非常巨大，它里面可能有PB级别的数据量
24.45–24.50
and you don’t want to you know you don’t want to hit up the ~~faucet~~ faults~~ ~~and limitation of a file into the size of a file
你不会想要对这样么大的一个文件来做错误修复（将所有数据局限在这么大的一个文件里）

24.50–24.54
 so again the OS doesn’t know anything about what’s in these files
操作系统其实根本不知道这些文件里面有什么东西

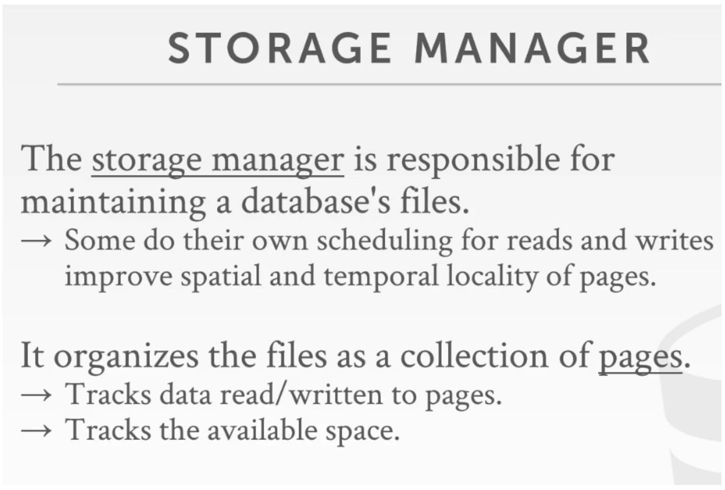
24.54–24.57
it’s just there’s a bunch of binary data to the operating system
对于操作系统来说，这只是一堆二进制数据
24.57–24.57
they’re not special
它们并没有什么特别之处
24.57–25.04
but the format for these data files are typically proprietary or specific to the database management system
但这些数据文件的格式通常情况下都是专属于某个数据库管理系统的
25.04–25.09
so meaning you can’t take a SQLite file plop it down inside a directory for MySQL
这就意味着，你无法将一个SQLite的文件导入MySQL中
25.09– 25.11
and I think MySQL is gonna not be able to read it
我觉得MySQL也没办法去读取该文件
25.11–25.14
right they’re always specialized to whatever the software is
这些文件通常是专属于某些特定软件的
25.14–25.22
so these files for databse we’re typically just gonna store them on top of the regular file system that the OS provides us
这些数据库文件我们通常会存放在操作系统提供给我们的文件系统中

25.22–25.26
ext3 ext4 whatever windows net has now I forget
windows系统用的可能是Ex3或者Ex4，具体是什么我忘了
25.26–25.33
right these are just and the OS who sees a bunch of files and we rely on the file system to provide us with basic readwrite api’s
我们基于操作系统的文件系统所提供的基本读写API来对文件进行读写
25.33–25.42
in the 1980’s, people did try to build database systems that use custom file systems on raw storage devices
在1980年代，人们试着在裸存储设备上构建使用自定义文件系统的数据库系统
25.42–25.48
so like say you plop down a new hard drive instead of formatting it and you know setting it up for NTFS or WinFS or xEt4
就好像是，你直接拿了一块新硬盘使用，而不是使用前先将它格式化并将它的格式设置为NTFS或者WinFs或者ext4

25.48–25.55
for you say screw all that is give me the raw storage device and I’ll manage what’s actually being stored in it myself
就好比说，直接给我裸存储设备，我自己来管理存储在上面的东西
25.55–26.03
some of the enterprise systems like that like enterprise meaning like high-end ones like Oracle db2 and sequel server will still do this

某些高端的企业级数据库系统，例如：Oracle，DB2和SQL server依然这么做（知秋注：有自己的文件管理系统）

26.03-26.09
 but most of the new database startups are anything you gave him that's come out in the last ten years or 15 years doesn't do this
但大部分近10年或者15年的新兴初创数据库企业并不会这么做
26.09-26.10
 right

26.10-26.15
because it's the engineering effort to make your own custom file system for your database system is not worth it
因为将工程师的精力都花在为你的数据库系统定制专属的文件系统，这并不值得
26.15-26.17
you get maybe like a 10% improvement
你在性能上可以得到10%左右的提升
26.17-26.21最后一个单词骂人被屏蔽了
but now you you know you're managing your own file system which is a big ~~chaos~~
但现在你知道，你要自己去管理你自己的文件系统，这真的是个很大的坑
26.21-26.22
 and it makes your thing less portable
这就大大降低了你东西的可移植性
26.22-26.27
cuz now you can't easily run it on Amazon and other other hardware providers
因为这样你就没办法将你的东西轻易地运行在Amazon服务器或者其它的硬件上了

STORAGE MANAGER

The storage manager is responsible for maintaining a database's files.
→ Some do their own scheduling for reads and writes improve spatial and temporal locality of pages.

It organizes the files as a collection of pages.
→ Tracks data read/written to pages.
→ Tracks the available space.

26.27-26.32
so what we're building essentially is now again what is called a Storage Manager
现在我们所要构建的东西，本质上来讲，它被称为存储管理器
26.32-26.34
sometimes also called the storage engine
有时也被称为存储引擎
26.34-26.37
and then this is the piece of the software at the component in our database system
它是我们的数据库系统中的一个组件
26.37-26.43
 that is responsible for maintaining our database files on disk
它负责维护我们在磁盘上的数据库文件

26.43-26.47
now we could do reads and writes
现在，我们可以进行读写操作
26.47-26.49
and let the OS schedule things
并让操作系统来进行调度工作
26.49-26.52
some of the more high-end database systems will actually have a shim layer above the file system
某些高端数据库系统实际上在文件系统之上会有一个shim层
26.52-26.54
~~right right~~
26.54-26.58
that does allows the the database to do some disk scheduling
它允许数据库去做一些磁盘调度

26.58-27.03
you do this like I know I have a bunch of threads writing to blocks that are close to each other
~~你这样做就像是通过一堆线程来对彼此邻近的区块进行写入~~
这就像是可以通过一堆线程来对彼此邻近的区块进行写入
27.03-27.06
I can maybe combine them together and do a single write request
我也可以将这些块合并，并做一次写入请求
27.06-27.08
right the OS can kind of do these things
操作系统可以做到这些事情
27.08-27.13
but again~~ it doesn't know exactly what's above ~~this doesn't know what the semantics of the query above it
但操作系统并不知道在此之上的查询语义到底是什么
27.13-27.15
most systems don't do this and
大部分数据库系统不会去做这个shim层
27.15-27.17
then for the project we'll be working on here we don't do this
在我们所做的项目中，我们也不会去做
27.17-27.19
it's typically for the higher ones
通常这是在高级课程中做的东西
27.19-27.20
yes
请问
27.20-27.38
yes a question is I said that most databases and split up the file the database files into multiple files
她的问题是，我之前说过大部分数据库会将数据库文件拆分为多个文件
27.38-27.41

because you don’t hit the the file size limit of the operating system
因为你这样就不会遇上操作系统上的文件大小限制的问题了
27.41-27.44
 is there any optimization for putting things in memory?
你想问的是，将东西放入内存时是否存在任何优化
27.44-27.55
 yes just a file just the file have a limit for the amount of the size it can be in memory
她想问的是放入内存的文件大小是否有任何限制

27.55-27.59
with virtual memory，no
如果是虚拟内存，那就没有这种限制
27.59-28.04
~~when we talk to us to~~ whatever the the swap size is what the OS could let you store
无论swap大小是多少，OS都会让你保存
28.04-28.08
 but it’s essentially limited the physical memory that’s available to you
但本质上来讲，操作系统限制了你所能使用的物理内存的大小
28.08-28.19
her question is would be better to have a single file
她的问题是，使用一个文件是否要来得更好
28.19-28.23
because ~~then you get the over~~ you get rid of the overhead of having multiple files
因为这样你可以摆脱使用多文件所带来的开销
28.23-28.25
what you meaning the overhead
你所说的开销是什么？
28.25-28.30
 like the inode that you’re that you have to find it go open a file or what
例如：inode，你必须找到它，以此来打开文件的这种开销，或其他什么开销
28.30-28.40
okay

28.40-28.42
so you’re talking like the metadata
So，你讨论的是元数据之类的东西
28.42-28.44
there I say minutes
这里我稍微讲下
28.44-28.45
if I have one file
如果我使用的是单文件存储这种情况
28.45-28.48
then I have one file name and I have one inode in my file system that points to it
那么，我就只有一个文件名，并且在我的文件系统中有一个指向它的inode
28.48-28.51
 if I have multiple files that I have multiple inode entries
如果我使用的是多文件这种情况，那么我就会有多个inode
28.51-28.55

and each one has their own file name much a meditator you know referencing it
每个文件都有它们自己的文件名，这些inode会指向它们
28.55-28.57
but like that’s what
但这看起来像是什么呢?
28.57-29.00
maybe a kilobyte of metadata it’s nothing right
可能是1kb大小的元数据之类的东西，它很小，并且微不足道

29.00-29.01
if your database is one petabyte
如果你的数据库是1PB大小

29.01-29.03
who cares that you have a bunch of file names
谁会去在意你有这么多的文件名呢
29.03-29.03
right

29.03-29.08
I think really large scales it doesn’t make a difference
我认为在大规模的情况下，这并不会造成什么差异
29.08-29.08
right

29.09-29.14
I think now for like for modern file systems it’s not really an issue anymore
我认为这对于现代文件系统来说，这并不再是一个问题了
29.14-29.19
like you can have like exabyte you know single files are exabytes
例如，你的单个文件可以是EB级别大小的
29.19-29.21
but thinking like in the 90s or early 2000s
但在1990年代，或者是2000年代早期
29.21-29.23
when like you were running like fat32
当你使用的是FAT32格式的文件系统
29.23-29.24
you can only have a 4 gigabyte file
你文件的最大只能为4GB
29.24-29.25
right

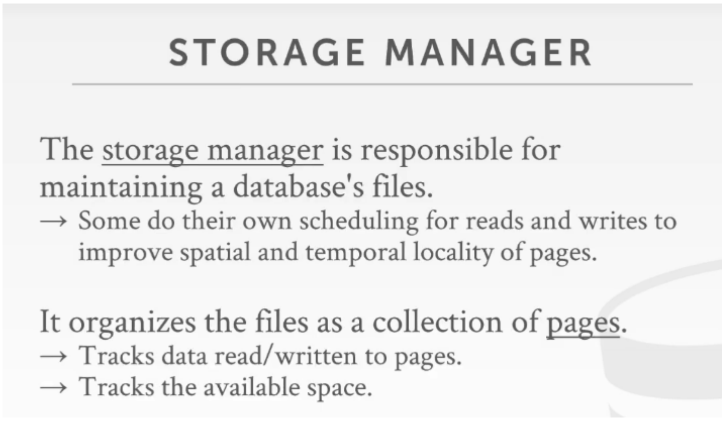
29.25-29.29
that’s back in the days, it mattered more，not so much now
~~时光倒流，现在这已经不再是个问题了~~
对于以前来讲，这很是问题，对于现在就不是了
29.29-29.30
but even then the metadata doesn’t matter

但对于元数据来讲根本就无关紧要（知秋注：元数据很小）
29.30–29.31
yes
请问
29.31–29.36
do operating system limit how many file like the process can create
操作系统是否会对进程所创建文件有数量限制
29.36–29.44
his statement doesn’t limit the number of files that you can have open is usually open file handles on them are things you can create
针对他所说的，这并不会限制文件创建的数量，你通常可能有打开你所创建的文件句柄数量的限制（知秋注：Linux对单一进程会有文件打开数量限制，ext3文件系统下单个目录里的最大文件数无特别的限制，是受限于所在文件系统的inode数）

29.44–29.45
 and therefore you have to have permissions to do this
因此，你必须要有权限，才能这么做
29.45–29.46
 that’s really yes

29.46–29.50
 and so if you go look at like the tuning guides or setup guides for a bunch of different database systems
So，如果你看下不同数据库系统的调试指南或者设置指南
29.50–29.56
they’ll talk about like tune this kernel parameter delight you have a bunch of you know this number inode or file handles open
你可以根据指南所说，来调整inode或可打开文件句柄的数量所针对的内核参数，直到你满意为止

29.56–29.57
absolutely yes

29.57–30.00
okay awesome
Ok，他的问题问的很好

STORAGE MANAGER

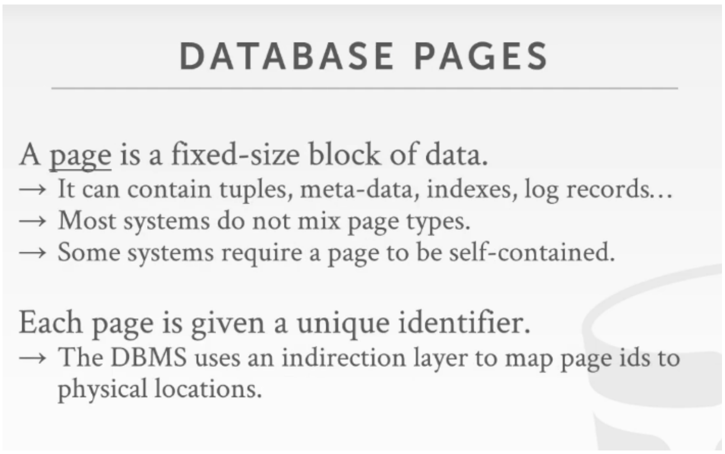
The storage manager is responsible for maintaining a database's files.
→ Some do their own scheduling for reads and writes to improve spatial and temporal locality of pages.

It organizes the files as a collection of pages.
→ Tracks data read/written to pages.
→ Tracks the available space.

30.00–30.02
 all right

30.02-30.03
again so we're trying to build a storage manager
我们想试着去构建一个存储管理器
30.03-30.07
and the storage manager is responsible for maintaining these files on disk
该存储管理器负责维护磁盘上的文件
30.07-30.09
and whether it's one file or multiple files it doesn't matter
不管是管理一个文件还是多个文件，都没问题
30.09-30.16
so now within these files, we're going to organize them as a collection of pages
我们会将这些文件组织为一个page的集合

30.16-30.21
and so our Storage Manager is going to keep track of all the reads and writes we're gonna do to these pages
So，我们的存储管理器将跟踪我们要在这些page上所执行的所有读取和写入操作

30.21-30.26 ！！！！！
as if you track a wood available space what space is available to us to store new data in our pages
这就像是我们在跟踪我们的页（pages）中还有多少空间可允许我们往里面存储新的数据

DATABASE PAGES

A page is a fixed-size block of data.
→ It can contain tuples, meta-data, indexes, log records…
→ Most systems do not mix page types.
→ Some systems require a page to be self-contained.

Each page is given a unique identifier.
→ The DBMS uses an indirection layer to map page ids to physical locations.

30.26-30.32
so a page is essentially just a fixed size chunk or block of data
本质上来讲，一个page就是一个固定大小的数据块
30.32-30.36
 that were just we're going to organize our file and you know into these chunks
我们将我们的文件组织为这些数据块
30.32-30.39
 so a page can contain anything
一个page能够保存任何东西
30.39-30.43
right it contained the actual tuples the database itself
它里面可以保存数据库里面的tuple

30.43-30.46
contain metadata indexes log records

它也能保存元数据，索引，日志记录之类的东西
30.46-30.48 *******
from the storage management perspective，it doesn’t really matter
从存储管理的角度来看，这并不重要
30.48-30.51
right but we always have to store things within a single page
但我们始终必须将东西保存在一个page
30.51-31.00
so now some database systems will require you that to have the page be self-contained
so，现在有些数据库系统会要求你的page是self-contained的

31.00-31.04
and what I mean by that is all the information you need to know how to interpret and comprehend
我的意思是，所有的信息你都需要知道该如何去解释以及理解
31.04-31.08 *****
the contents of a page have to be stored within the page itself
page的内容必须存储在page本身内
31.08-31.10
so let me give an example
这里我给出个例子
31.10-31.14
let’s say that I have a table and I have the table has 10 columns
假设我有一张表，表内有10列
31.14-31.15
they have different types
这些列有不同的类型
31.15-31.18
but I call create table and I create the table of different attributes
我创建的表具有不同的属性
31.18-31.22
so I could have the metadata about what’s in that table stored in one page
So，我可以将有关该表内容的元数据存储在一个page中
31.22-31.27
and then all the tuples for that that table stored in another page
该表的所有tuple则保存在另一个page中
31.27-31.30
so the problem is now if I have a disk failure
So，现在的问题是如果我遇上磁盘故障
31.30-31.32
like my my data center catches on fire
例如，我的数据中心发生了火灾
31.32-31.38
my disks melt and I lose that one page that tells me what what the layout of the the schema is
我的磁盘也烧了，并且我丢失了能告诉我数据库schema布局的那个page

31.38-31.42
now I don’t know how to easily interpret what the contents are of my tubule pages

现在，我就不知道该如何简单地去解释保存了我tuple的那个page上的内容有什么了
31.42–31.49
and so some systems like Oracle for example require all the metadata about how to say here's what's in that page has to be within the page itself
某些数据库系统，例如：Oracle就需要将描述该page中内容的所有元数据，和这些内容数据一起保存在该page中
31.49–31.51
so that way if you lose any other page
这种情况下，如果你丢失了其他任何page
31.51–31.52
it doesn't affect
这并不会影响什么
31.51–31.55
you know you lose one page it doesn't affect any other pages
即使丢了一个page，这也不会影响其他page
31.55–31.58
you think it's a bit overhead to this that seems crazy
你们可能认为这似乎有点疯狂
31.58–32.00
well they do it for disaster recovery
这些系统通过这个来做到灾难恢复
32.00–32.03
again so now again the machine catches on fire
假设，现在机器着火了
32.03–32.04
and you lose a bunch of pages
你丢失了许多page
32.04–32.11
you can you you literally open up a hex editor and try to reconstruct what the database was by looking one page at a time
你可以通过打开一个十六进制编辑器（恢复数据库神器），尝试通过通过一次查看一个page来试着重建数据库
32.11–32.14
and all the metadata you need about what's in that page is stored within itself
你所需要的所有元数据就保存在page里面
32.14–32.14
alright

32.14–32.23 *****************
so all the things that's important understand is that we're not going to mix different types types of data within a page
因此，所有需要了解的重要事情是，我们不会在page中混合使用不同类型的数据
32.23–32.26
there's some research systems that do this
有些研究系统会这么做

32.26–32.28
we could have you know one page have tuple data and log record data for our purposes here

我们可以在一个page上存放tuple数据和日志记录数据来供我们使用
32.28-32.31
 and most systems they don't do this
大部分系统不会这么做
32.31-32.33
it's like here's a page and only stores tuples
就比如说，这里有个page，但它里面只存了tuple
32.33-32.35
here's a page that only stores index information
另一个page上则只保存了索引信息
32.35-32.40
so now each page is going to be given a unique internal identifier
因此，每个page都会被赋予一个唯一的内部标识符
32.40-32.44
that the database system is gonna generate for us a page ID
数据库系统会为我们生成这些page ID
32.44-32.44
 right

32.44-32.47
and we're gonna have then now have an indirection layer
然后，我们会有一个indirection层
32.47-32.49
and this is would be a reoccurring theme when we talk about storage
当我们讨论存储时，这是一个会反复提到的东西
32.49-32.56
 we have an indirection layer that's gonna allow us to map a page ID to some location in a file at some all set
indirection层允许我们将一个page ID映射到某个集合中一个文件中的某个位置 （知秋注：其实就是记录一个相对位置，方便文件整体移动后，只要知道整体文件的初始位置，我依然可以通过该相对位置即page ID找到某个文件某个位置的数据所对应的page，要知道一个页得到大小是固定的，id数*页大小就找到了offset值）
32.56-32.58
right

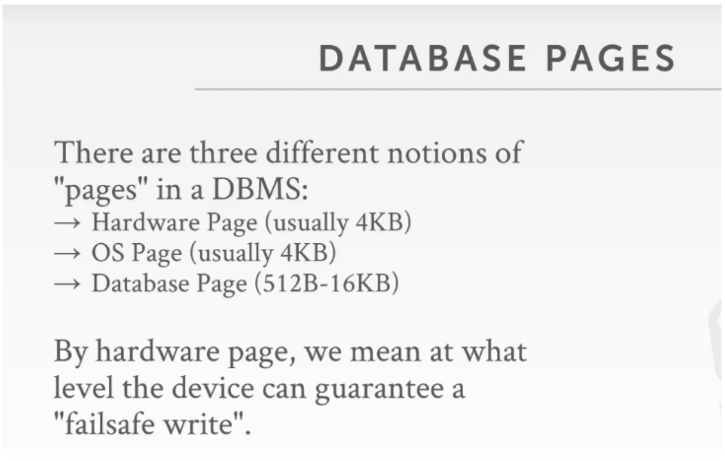
32.58-33.00
and we want to do this
我们想这么做
33.00-33.02
because now underneath the covers we can start moving pages around
是因为在内部我们能够移动page
33.02-33.05
you know if we start compacting the disk or or set another disk
如果我们对磁盘进行压缩，或者设置使用另一块磁盘
33.05-33.07
 and it doesn't change our page ID
这并不会改变我们的page id
33.07-33.09
cuz we have this this page directory

因为我们有page目录

33.09–33.11

say you want page 1 2 3 here's where to go find it

假设你想要page 1、2以及3，page目录就能告诉你哪里能找到它们



DATABASE PAGES

There are three different notions of "pages" in a DBMS:

→ Hardware Page (usually 4KB)
→ OS Page (usually 4KB)
→ Database Page (512B-16KB)

By hardware page, we mean at what level the device can guarantee a "failsafe write".

33.11–33.19

so there's a bunch of page concepts we need to talk about to put it in the context of how real computers work

因此，我们需要讲很多page相关的概念，以便将它们放在真实计算机的工作环境中

33.19–33.22

so at the lowest level

因此，在最底层

33.22–33.23

we have what's called a hardware page

我们有hardware page

33.23–33.29

this is the page API or page access level you get from the actual storage device itself

这是你从实际存储设备本身获得的page相关的API或page访问级别

33.29–33.32

I just know what the SSD or spinning disk hard drive exposes

我只知道SSD或者机械硬盘会暴露这些

33.32–33.34

this is typically four kilobytes

通常这只有4kb大小

33.34–33.37

then above that you have an operating system page

然后，在此之上，我们有操作系统page

33.37–33.41

and again that's as you as you take things out of the storage device and put it into memory

通过它，你可以从存储设备中取出数据，并放入内存中

33.41–33.42

right

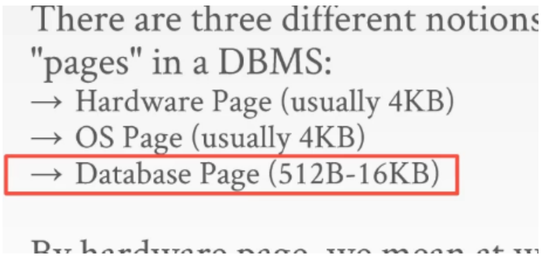
33.42–33.44

they represent that as an internal page as well

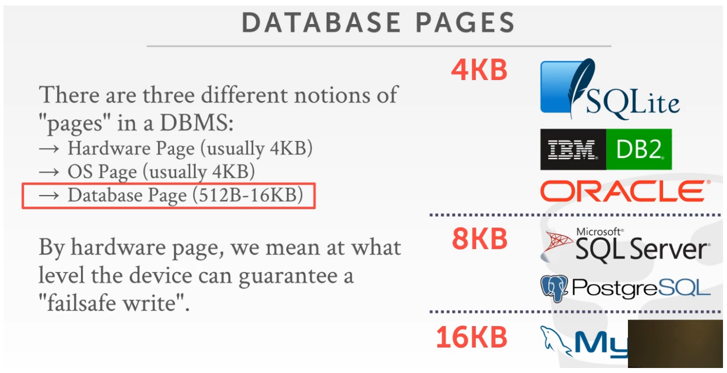
它们也将其表示为内部page

33.44–33.48

and that's typically usually four kilobytes by default in Linux and Windows
默认情况下，在Linux和Windows中，这通常使用了4kb大小
33.48–33.51
there's things like huge pages where you turn it
当然也有占用空间很大的page
33.51–33.57
you can take one gigabyte page to be broken up to massive four kilobyte horror pages
你可以将一个大小位1GB的page，拆分成无数个大小为4kb的page
33.57–33.59
 but for our purposes we don't care about
但出于我们的目的，我们并不在意这个
33.59–34.01
the thing we care about at the database page here
我们所关心的是此处的数据库page
34.01–34.01
right

34.01–34.05！！！！！！
and this is gonna bury between different systems
不同的数据库系统会有不同

There are three different notions
"pages" in a DBMS:
→ Hardware Page (usually 4KB)
→ OS Page (usually 4KB)
→ Database Page (512B-16KB)

34.05–34.08
 so at the low end at 512 bytes
最低是512 bytes
34.08–34.11
 that's like something like SQLite like an embedded system
就像像SQLite这种嵌入式的系统
34.11–34.14
 but then at the high end you'll have like 16 kb
然后高的话你可以有16kb大小的page
34.14–34.14
 that could be like MySQL
就像MySQL
34.14–34.17
 so different database systems do different things
So，不同的数据库系统做不同的事情
34.17–34.19
 and there's different trade-offs for all of these
所有这些数据库系统都有不同的权衡
34.19–34.20
all right



DATABASE PAGES

There are three different notions of "pages" in a DBMS:
→ Hardware Page (usually 4KB)
→ OS Page (usually 4KB)
→ Database Page (512B-16KB)

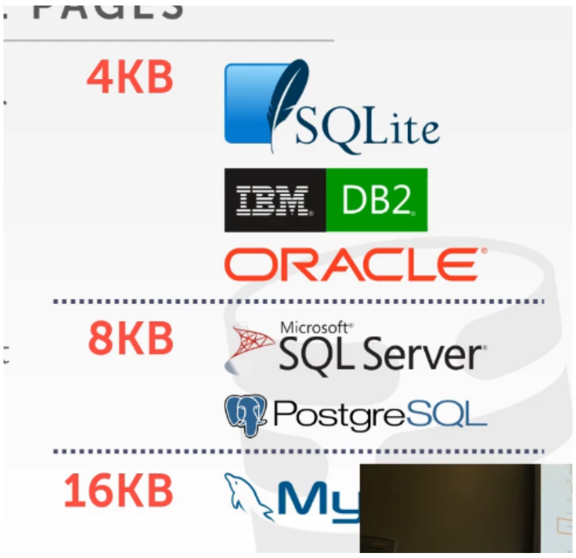
By hardware page, we mean at what level the device can guarantee a "failsafe write".

4KB
SQLite
IBM DB2
ORACLE
8KB
Microsoft SQL Server
PostgreSQL
16KB
My

34.20–34.25
the main thing we’re going to care about though is that the hardware page
我们所主要关心的是hardware page
34.25–34.30
is is the sort of the lowest level that we do atomic writes to the storage device
它是我们执行原子写入存储设备的最低底层的东西
34.30–34.32
and typically 4 kilobytes
通常是4kb大小
34.32–34.35
so what I mean my bad is say I need to modify a bunch of data
So，我的意思是，假设我需要区修改一些数据
34.35–34.45
the hardware can only guarantee that if I do a write and flush to the disk，it can only guarantee that at 4 kilobyte of time it’s gonna be atomic
如果我对磁盘进行 write和flush操作，存储设备只能保证每次写入4kb时是原子的
~~能保证每次写入是原子性操作~~

34.45–34.49
so what i mean by that
我这么说是什么意思呢?
34.49–34.51
so like if I say I need to write 16 kilobytes
就好比说，如果我需要写入16kb大小的数据
34.51–34.56
I could try to write the I say I tell the disk hey write 16 kilobytes for me
我想告诉磁盘为我写入16kb大小的数据
34.56–34.57
it might crash
它可能会故障崩溃
34.57–35.00
before you know if it writes the first 8 kilobytes
就比如，如果磁盘先写入8kb数据
35.00–35.03
then it crashes before writing the next 8 kilobytes
然后磁盘在写入下一个8kb数据前崩溃了
35.03–35.04
and then you come back
当磁盘恢复正常工作后继续
35.04–35.05

and now you have a torn right
现在，你就会得到一个分裂的数据
35.05–35.07
 you only see the first half and the second half
~~你只能看到第一部分和第二部分~~
你只能看到第一半和第二半的数据（知秋注：虽然都写入了，但这两段数据不连续了，写入16kb的操作不具备原子性，失败后不会回滚，也就是一份坏掉的数据）
35.07–35.10
because the hardware can only guarantee 4 kilobytes at a time
因为硬件一次只能保证4kb大小的数据没有问题

35.14–35.17 ? ? ? ?
all right this will come up this we’ll talk about this more later when we talk about logging and ~~commercial~~ concurrency（应该是发音的问题）
我们会在之后讨论日志和并发的时候，再去讨论这个问题
35.17–35.20
but this is something we need to be mindful of
但这是我们需要注意的东西
35.20–35.22
and again there’s different systems do different things
不同的系统会做不同的事情
35.22–35.29
 the high end systems like an Oracle sequel server and db2 you can actually tune it
实际上，像Oracle、SQL server和DB2之类的高端系统，你可以对它们进行调整


35.29–35.30
 so you say I want to start things there’s 4 kilobytes 8 kilobytes or 16 kilobytes
就比如说，你想将它们的page设置为4kb、8kb或者16kb大小
35.30–35.35
you can even vary say for index pages store much larger page sizes
~~你甚至可以这样做，让索引page的大小变得更大，以此来存储更大的page~~
你甚至可以这么做，将index pages存储的大小更大
35.35–35.36
and then data page is sort of smaller
然后让数据page变得更小

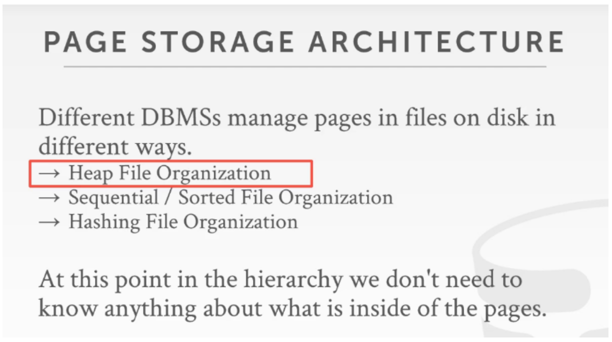
35.36–35.38

you can go crazy and do much different things

你也可以更加激进，并做些更加不同的事情

35.38–35.40

all right



PAGE STORAGE ARCHITECTURE

Different DBMSs manage pages in files on disk in different ways.

→ Heap File Organization
→ Sequential / Sorted File Organization
→ Hashing File Organization

At this point in the hierarchy we don't need to know anything about what is inside of the pages.

35.40–35.45

so now we want to talk about how we're gonna represent the the page storage architecture

So，现在我们想去讨论下我们该如何表示page 存储架构

35.45–35.47

so again there's different ways to do this

So，要再说一下，我们可以通过很多不同的方法做到这点

35.47–35.48

there's different trade–offs for this

这里面也有许多不同的取舍

35.48–35.53

the most common one it's going to be the heap file organization

最常见的方式是使用Heap File Organization

35.53–35.54

so we'll focus on that

这也是我们要关注的东西

35.54–36.01

but the thing to understand is that at this point at this lowest level in the storage manager we don't care about what's actually in our pages

但有件事情要理解的是，在存储管理器最底层的级别中，我们不用关心我们的page中到底有什么

36.01–36.03

we don't care whether this indexes data or tuple data

我们不介意里面放的是索引数据还是tuple数据

36.03–36.05? ? ?

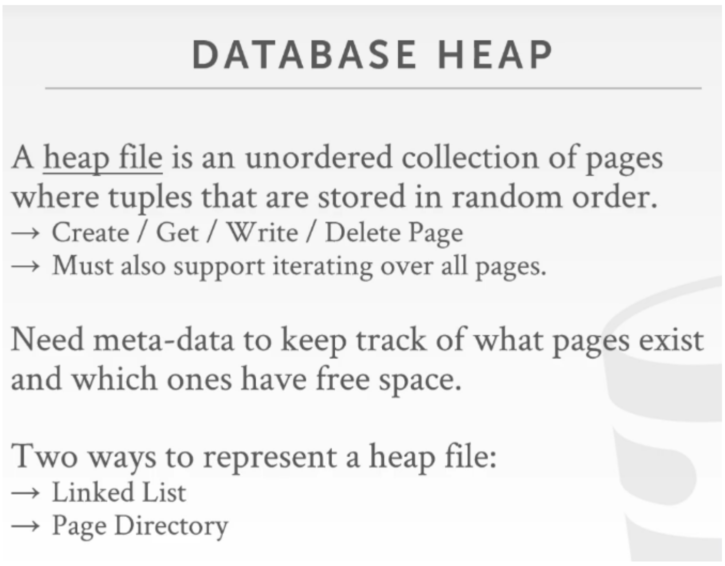
we don't care USS for a page

或是其他

36.05–36.09

we'll read that page or delete it

我们就可以对这个page进行read或delete操作



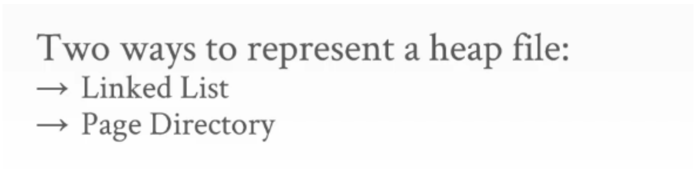
DATABASE HEAP

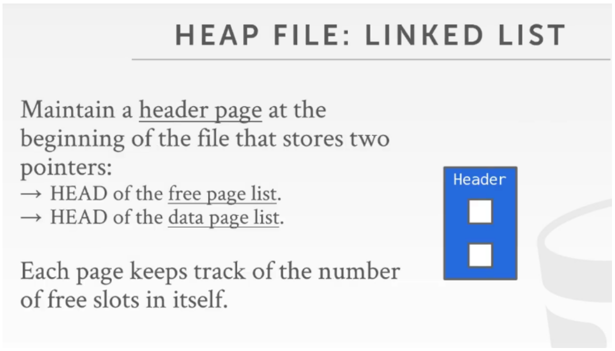
A heap file is an unordered collection of pages where tuples that are stored in random order.
→ Create / Get / Write / Delete Page
→ Must also support iterating over all pages.

Need meta-data to keep track of what pages exist and which ones have free space.

Two ways to represent a heap file:
→ Linked List
→ Page Directory

36.09-36.15
so database heap file is a unordered collection of pages
So，数据库中的heap文件是一个无序的page集合
36.15-36.19
where the tuples of the data can be stored in random order
即可以以随机的顺序把tuple数据保存在里面
36.19-36.25
so again the relational model doesn’t have any orderings
So，要说一下，关系模型并没有任何排序
36.25-36.31
if I insert tuples one by one I’m not guaranteed that they’re gonna be stored that way on disk
如果我一个接一个地插入tuple，我并不能保证它们是按照我插入的顺序保存在磁盘上的
36.31-36.34
because it doesn’t matter
这并没有什么关系
36.34-36.36
because I write sequel queries and that have no notion of ordering
因为在我写的SQL查询中并没有排序的概念
36.36-36.43
so the API we need to have again is be able to read and write and access pages at a time
因此，我们所需要的API必须能够对page进行读写和访问
36.43-36.49
as well as being able to iterate over every single page that we have，in case we need to do a sequential scan across the entire table
如果我们需要使用SQL来按顺序扫描整个表，那么我们也必须要有这种API，能够让我们遍历所拥有的每个page
36.49-36.55
we’ll have some additional metadata to keep track of what pages we have which ones have free space
我们会使用一些额外的元数据来跟踪我们有哪些page，哪些page中是空余空间
36.55-36.58
so that if we need to insert new data we know where to find a page to go ahead and do that
这样如果我们需要插入新数据时，我们就知道可以在哪个page上插入数据了

36.58–37.03
right and internally we can represent this heap file in a bunch of different ways
在内部，我们可以通过一系列不同的方式来表示heap文件
37.03–37.08
again at the lowest level we can organize these in pages
在最低层，我们将它们组织为page
37.03–37.11
and then within these pages we can represent them with different data structures
在这些page中，我们可以用不同的数据结构来表示它们

Two ways to represent a heap file:
→ Linked List
→ Page Directory

37.11–37.14
so let's first talk about doing linked lists
So，首先我们来谈论下链表这种方式
37.14–37.16
because that's sort of the dumb way to do this and nobody actually does this
这是一种愚蠢的方式，实际上也没人使用这种方式
37.16–37.17
but it exists
但确实可以用它来表示heap文件
37.17–37.20
and then we'll see the page directory way which is a better approach
之后，我们会去了解page目录这种方式，它是一种更好的方案

HEAP FILE: LINKED LIST

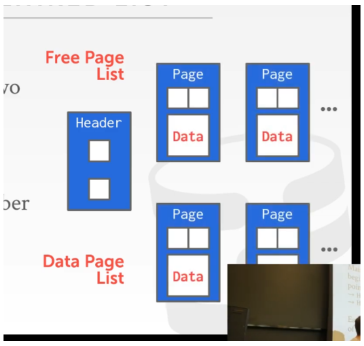
Maintain a header page at the beginning of the file that stores two pointers:
→ HEAD of the free page list.
→ HEAD of the data page list.

Each page keeps track of the number of free slots in itself.

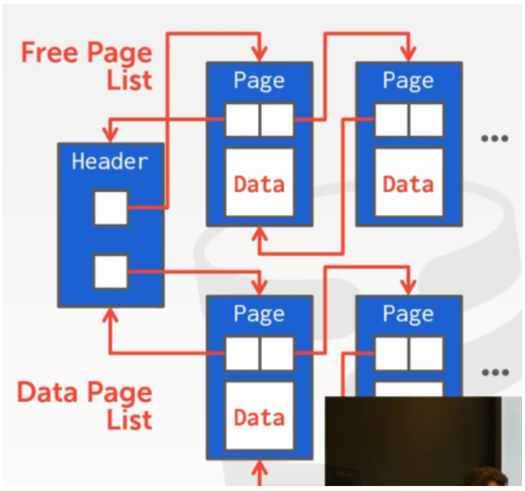
37.20–37.29
so the way we're gonna do this again the goal is we what we're trying to do here is we're trying to figure out within my file I have a bunch of pages
我们所试着要做的是，假设在我的文件中，我有许多page
37.29–37.34
what pages you know what where those pages exist，and what kind of you know whether they have data or not
在该文件中，存在着哪些page，哪些page里面保存了数据，哪些没有保存
37.34–37.36
where they have free space for me to store stuff
它们中是否有空余空间来让我保存数据

Free Page List
Header
Data Page List

37.36–37.43 ！！！！！！！！！！！！！
so in the header of this this heap file said that for this linked list we’re just gonna have two pointers
在heap文件的header中，我们的链表里面有两个指针
37.43–37.47
we have one pointer that says here’s the list of the free pages that I have in my file
我们使用一个指针来表示我文件中的空余page列表
37.47–37.51
and here’s a list of the the pages that actually have completely for occupied
那这个指针就表示一个已经被数据完全占据的page 列表



37.51–37.54
right and then again this is just a linked list
这就是一个链表
37.54–37.56
so it doesn’t matter where these pages are stored
这些数据放在哪都没关系
37.56–37.58
doesn’t matter whether contiguous or not
不用管这些数据是否是连续存放的
37.58–38.06
I just I now have just pointers and say hey here’s you know here’s the data that you know here’s here’s the first page my linked list that where they’re occupied
我现在只有这些指针，看这里（下面那有数据的page图示），这是被数据占据page链表的第一个page
38.06–38.07
and here’s a pointer to the next one
这里有指向下一个page的指针
38.07–38.11
if I want to say find me a page a free page like a store stuff

如果我想去想找一个page来保存些东西
38.11–38.13
I can follow the free page list look in here
我可以在这些空余page的列表中进行查找
38.13–38.17
and you know in traverse long until I find something that has enough space for what I want to store
遍历这些page，直到我找到一个有足够空间保存我想保存的数据的page


38.17–38.24
and because we need to go possibly an iterate in reverse order
因为我们可能需要以相反的顺序来遍历这些page
38.24–38.24
we need pointers on the way back as well
我们需要一个指向上一个page的指针（知秋注：也就是双向链表）
38.24–38.24
 yes
请问
38.24–38.28
 the question is why is the heap file unordered
他的问题是，为什么heap文件都是无序的
38.28–38.30
thinking at a high level
我们站在一个高级的层面来思考这个问题
38.30–38.36
like the data we’re storing does not need to be ordered as we insert it
我们所保存的数据无须按照我们插入时的顺序进行保存
38.36–38.38
 right

38.38–38.39
so if I insert like three tuples
如果我插入三个tuple
38.39–
I could insert I could in my page layout the actual inside the pages
我可以在page 布局中插入page 内部的实际内容
–38.47
 I could have tuple three tuple two two and one
我可以有tuple 3、tuple 2和tuple 1

38.47-38.49
 I’m not required to put them in order that are given
我无须将它们按照给定顺序排列
38.49-38.59
right so this question if you had to look for a particular page with these link list things
 So，他的问题是，如果我们要在这些链表中找到某个特定的page
38.59-39.03
 I have to reverse potentially entire link list if I want
我就必须根据我的需要来反转整个链表
39.03-39.04
one absolutely yes this sucks
没错，这很操蛋
39.04-39.06
this is a bad idea
这是一个糟糕的想法
39.06-39.07
and saintly yes
请问
39.07-39.13
his question is if it’s ordered you can always search faster
他的问题是，如果是有序排列，这样搜索速度会不会更快点
39.13-39.13
 right

39.13-39.15
so there’s different trade-offs
这里面有些不同的取舍
39.15-39.19！！！！！！
so I have no metadata to say where I have free pages
我并没有元数据来表示在哪里可以找到free pages
39.19-39.22
so I need to insert something now
现在，我想插入些东西
39.22-39.24
 where am I going to insert it
我要往哪里插入呢?
39.24-39.29
 now I’ve been do a sequential scan and look at every single page until I find one that has free space
现在，我可以进行循序扫描来查看每个page，直到我找到有空余空间的page为止（知秋注：比如，有些数据比较小，page剩余空间刚好可以容纳，有些page虽然有剩余空间，但不足以容纳这个数据，所以不确定到底是哪个page，需要遍历）
39.29-39.35
or in this approach here I’m not saying right at while this is the right way to do it I’m saying this is how this works
我并没有说这种方法是对的，我说的只是它是如何工作的
39.35-39.42
~~if I need~~ if I have this one here，then I get this go follow this pointer to find the first free page and see whether it has enough space for what I want to store

如果我在这里，接着我要从这个指针开始来查找第一个free page，并看看它是否有足够的剩余空间来存储我想存储的数据
39.42-39.44
 it’s a trade-off right
这是一种取舍
39.44-39.49
I can either go do you know almost binary search to find exactly the page that I want
我可以通过二分查找法来找到我想要的page
39.49-39.53
or I can just do I can do this linked list
或者我也可以通过链表的这种方式做到
39.53-40.02
statements you can maintain ordered sets of it free pages and fill pages
So，你的问题是，维护一个free page的有序集合，以此来填充page?
40.02-40.15
look he’s ever usually linked lists use a tree
他想说的是使用树的结构，而不是链表
40.15-40.16
 a linked list can still be ordered right
链表也能做到有序