03-03

14

00:53:00,380 --> 00:53:01,0

00:53:01,280 --> 00:53:04,019这是个
14488047

40.17-40.19
so you could say all right say I delete all the tuples from this page
你可以这么说，我删除了这个page上所有的tuple
40.19-40.21
and it's been its page two
它已经变成了page2
40.21-40.23
and this is page one， this is page three
这是page1，这是page3

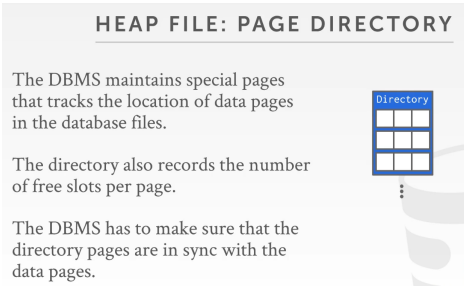
40.23-40.24
so I could insert it in between these two guys
因此，我可以在这两个page之间插入
40.24-40.24
sure
没错
40.24-40.38
order like we make two different or one on the page ideally but again

-40.46
I think your your the page ID is just like it all set in it
我认为page id已经在里面设置好了
40.46-40.51
right it's not there's no sort of a logical thing built in top of the heap file
在heap文件里面，并没有内置什么逻辑上的东西
40.51-40.56
look let's we take this offline， if you dont understand it

如果你不理解它的话，我们课后再讨论这个
40.56–40.58
this question over here or
那边的人是有问题吗?
40.58–40.59
okay

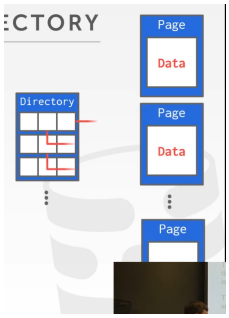
40.59–41.00
let's keep going
我们继续

41.00–41.02
if you have questionswe talked about it further
如果你还有问题，我们之后讨论
41.02–41.05
I mean the main opinion this is a bad idea
我的主要想表达的是这是一个bad idea
41.05–41.05
nobody does it
没有人会做么做
41.05–41.08
so I don't want to dwell on it too much
因此，我不想在它上面浪费过多时间
41.08–41.11
what people typically do is having a page directory
人们通常的做法是使用一个page目录

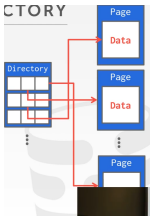


41.11–41.15
and for this one it's a we have a page now
对于这个，假设我们现在有一个page
41.15–41.22
again in the header of our file that's gonna maintain the a mapping from page IDs to they're all set
在我们文件的header中，它里面维护了page id和它们所处位置的映射关系
41.22–41.27
and then we can also maintain some additional metadata in this directory
然后，我们也可以在这个目录中维护某些额外的元数据
41.27–41.31
to say hey here's the amount of free space that's available to me in a particular page
假设在这个page上有一些我可以使用的空闲空间
41.31–41.34
so now when I want to go say I want to insert some data
现在，当我想去插入一些数据时

41.34–41.37
I don’t have to go you know scan the that list
我无须去扫描这个列表
41.37–41.39
I can just look my page directory and find everything that I need
我只需在我的page directory 中查找到我需要的东西（知秋注：比如这个directory中的每一个小格中不仅有对应page所在位置，也包含了它剩余空间信息）



41.39–41.43
right so if my pages are just ordered sequentially like this
如果我的page像这样按顺序排列



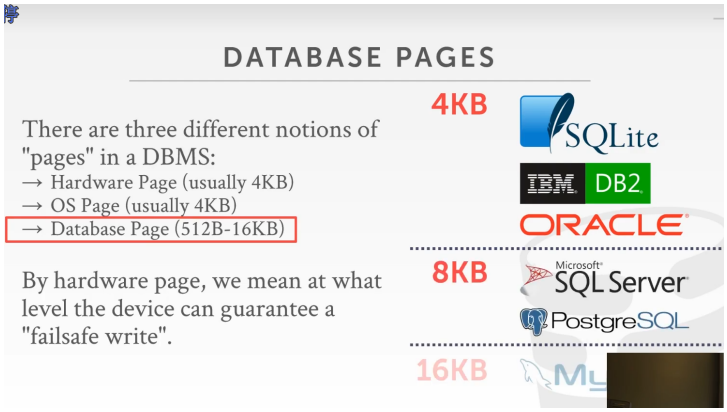
41.43–41.46
and then this is just a mapping to with they’re located
就像这样来映射到这些page所在的位置

++
41.46–41.54
right so the important thing about this going back to what we talked about the the atomic rights for Hardware
so 接着回到之前我们谈及的关于Hardware的原子性操作这个很重要的事上面
41.54–41.56
so now I have much metadata
现在，我有许多元数据
41.56–42.00
that’s a summarization of what’s in my actual pages themselves
它其实就是我实际page中内容的概括
42.00–42.02
and I have to keep it in sync
而且我必须保持同步
42.02–42.03
but I actually can’t guarantee that
但实际上我无法保证这点
42.03–42.09
because the hard can’t guarantee that I can write two pages of exactly the same time
因为很难保证我可以同时写两page数据
42.09–42.12
so let’s say that I delete a bunch of data here and my page

假设，我删除这里的一些数据和page
42.12–42.16
and then I update I want to update my page reference
接着我想去更新下我的page引用
42.16–42.17
and say oh I have this amount of free space
并表示，我有一定量的空闲空间
42.17–42.20
I made a bunch of data write that out
我想在这里写入一定量的数据
42.20–42.24
and then before I can update my page directory and write that out, I crash
接着，在我可以更新我的page目录并写入数据之前，系统崩溃了
42.24–42.26
so now I come back online
当我重新上线后
42.26–42.28
and say oh this I think this page is full
我发现这个page空间已经满了
42.28–42.30
and therefore I can't write any data to it
因此，我没办法往它里面写入任何数据
42.30–42.32
but I know it's actually not
但我知道，实际并不是这样
42.32–42.32
right

42.32–42.33
in reality it's not
实际上并不是
42.33–42.38
So you could say all right well when I boot back up, I'll just scan through all my pages and figure out what's actually really there
当我启动备份时，我只需扫描我的所有page，就能弄清楚page里面实际上都有什么内容
42.38–42.41
but now again think in the extreme
但现在思考下极端情况
42.41–42.42
if I have one gigbyte the data
如果我有1GB数据
42.42–42.46
then that's gonna take forever or sorry one petabyte of data that's gonna take forever to actually do this
抱歉说错了，如果是1PB数据的话，那么这种操作就会一直做下去，永远也停不下来
42.46–42.55
so the bunch of mechanisms will talk about later on or how we can maintain a log and initial metadata in sort of special special files
因此，我们之后会谈论一些机制，即如何在一些特殊文件中维护日志以及初始元数据

42.55–42.59
so that if we crash to come back we know how to reconstruct what's inside all these things
即使系统崩溃故障了，我们也知道该如何重建数据库里的东西
42.59–43.02
I think it is just a hash table
我认为它就像hash表那样
43.02–43.03
to say I want page 1 2 3
假设我想要page1,2以及3
43.03–43.04
here's where to go find it
我就可以从page目录中找到它们
43.04–43.06
and I just can get it
而且很容易就能找到
43.06–43.09
yes yes
请问
43.09–43.10
each page has the same size yes
每个page的大小都是一样的
43.10–43.15
what do you mean sorry
抱歉，你想表达什么?
43.15–43.20
these questions what is the size of a page in this world
这些问题所说的一个page的大小



43.20–43.23
right so this goes back through this diagram here
我们来回看下这里的图
43.23–4327
they do different systems do different things
不同的系统做了不同的事情
43.27–43.28
right
43.28–43.35
~~if fails afe is like~~ you know we can write four kilobytes and because the hardware guarantee that's atomic

我们每次可以写入4kb数据，因为硬件可以保证我们的写入操作是原子性操作

43.35–43.39
but now i need to write you know say my pages themselves are four kilobytes
但现在我需要让我的page的大小都是4kb

43.39–43.43
but i need to update one page they clean up clear out a bunch of data
~~但我需要去清除某个page上的一些数据~~
但我需要去更新一个page，清除它上面的一些数据
43.43–43.44
update the page directory
并且更新page目录
43.44–43.46
and say all right that page you've been cleared out
我们就可以说我们已经清理完这个page了
43.46–43.49
I can't guarantee the right both of those pages atomically
我无法保证在这两个page上的操作是原子性的
43.49–43.52
I can write one crash before I write the second one
在我对第二个page进行写入操作前，我在写入第一个page时可能系统就崩溃了
43.52–43.58
okay

43.58–44.01
all right so again

44.01–44.05
this is what the pages are gonna look like inside the yet certain files
这就是page在实际文件中的样子
44.05–44.14
so he says why do some pages you why do some data systems used larger pages
他想说的问题是，为什么某些数据库系统使用的是空间更大的page呢?
44.14–44.17
there's trade–offs
这其中有一些权衡
44.17–44.20
so internally inside my database system
在我的数据库系统内部
44.20–44.28！！！！！！！！！！！！！！！！
I have to have this page directory in memory mapping pages to some location either memory or on disk

我通过内存中的page目录将page映射到内存或者磁盘上的某个位置
44.28–44.31
but now if I can represent a larger amount of data with one page ID
但现在如果我使用一个page id来表示一个更大量的数据
44.31–44.34
then that size of that table goes down
然后，表格的所占用的大小就会变小（知秋注：固定容量下，一个id表达的数据量越大，所需要的id数也就越少）
44.34–44.40
think of this is like in the TLB the translation lookaside buffer inside on the CPU
我们可以将之看作是CPU中的TLB(页表缓存)来看待
44.40–44.43
if I am trying to match all a bunch of pages
如果我试着去匹配所有的page
44.43–44.45
but it's my page tables get really large
我page table会变得非常的大
44.45–44.47
and now I'm gonna have cache misses
那我就会出现cache misses的现象（知秋注：page id表示的数据范围太小，没办法全部在高速缓存中hold住）
44.47–44.52
so by you can represent more data in you know with few number of page IDs
So，你可以通过更少的page id来表示更多的数据
44.52–44.55
furthermore going back to talk about the difference being random and Sequential access
此外，我们回过头来在讨论下随机访问和循序访问之间的区别
44.55–44.58
so now if I can write out contiguously you say four kilobyte pages to represent a 16 kilobyte database page
So，现在如果我能连续写出4个4kb的page，以此来表示一个16kb的数据库page
44.58–45.09
when I do a read I just read all that sequentially and bring it in， now I'm getting potentially more useful data that I need
当我进行读取时，我会依次读取所有内容并将它们组合在一起，现在我获取到我所需要的更有用的数据
45.09–45.13
but again it makes doing writes more expensive
但这让写入操作的代价变得更高
45.13–45.17
look at that now at this stage a bunch of crap ahead of time to prevent myself from getting torn writes
看看现在，在写阶段要提前进行一堆烂糟逻辑，以防止自己写数据出问题（知秋注：就像前面说的数据写一半写满一个page出问题了，后一半数据还没来得及写，要保证该数据的原子性和完整性，就要多做很多工作）
45.17–45.19
so there's pros and cons for both of them
So，两者都有优点和缺点

45.19–45.22 ****
and this is why again the commercial systems allow you to tune them in different ways based on what your application wants to do
这就是为什么商用数据库系统允许你根据应用程序想要做的事情以不同的方式对其进行调整的原因

45.22–45.26
let's go question
问题环节，各位请问
45.26–45.26
yes
请问
45.26–45.39
so so his question is for self–contained pages would that solve this particular issue here
So，他的问题是，如果使用self–contained page，这是否能解决这个问题

45.39–45.39
no
并没有
45.39–45.47
so self–contained pages would mean like the contents of that inside the page I have all the metadata that I need for it
self–contained page指的是我们所需要的所有元数据都在这个page上
45.47–45.53
I still have to have a page director to tell me where to find that page, if I want page 1 2 3 or 4 5 6
如果我想找到page 1、2、3或者4、5、6，那么我依然必须要通过page目录来告诉我在哪才能找到我要的那个page
45.53–45.58
so it's not entirely self–contained at the higher levels in the system
从系统的更高层面来讲，这并不是完全self–contained
45.58–46.00
it's unavoidable at the bottom level
这在底层是不可避免的
46.00–46.02
so what we're at here
So，我们此处有什么呢?
46.02–46.06
might again we have any talk about what's actually inside this the page
可能我们需要再次来讨论下page里面实际有什么
46.06–46.10
but within the page directory，we can't guarantee that self–contained
但在page目录内，我们无法保证这是self–contained的
46.10–46.11
okay it doesn't make sense
这并没有什么意义
46.11–46.14
right yes

请问

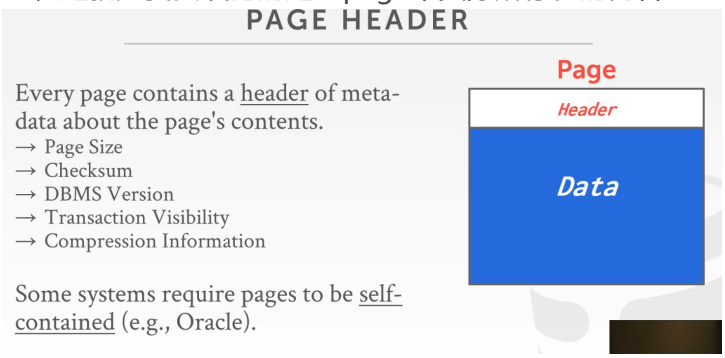
46.14–46.34
yeah so this question is there any way to guarantee that the if a crash happens, when you come back, you can identify that the crash happened
他的问题是，当系统崩溃了，当你恢复后，能否通过某种方式来定位崩溃发生的原因

46.34–46.35
yes
确实有
1274
46.35–45.36
so you knew checksums
So，你们应该知道checksum这个东西
45.36–46.42
right so say that I my database page is d3 pages here
假设我的数据库里面有3个page
46.42–46.44
in the header of the first page
在第一个page的header里面
46.44–46.44
I'll put a checksum
我会放一个checksum
46.44–46.54
and say all right the next from my starting point here, the next three pages the checksum should be like a CRC or md5 should be this amount
~~从此处这个起点开始，接下来三个page中的checksum的CRC或者md5值应该是这个数~~
从此处这个起点开始，接下来三个page的checksum应该就像是一个CRC或者md5这样的一个数
46.54–46.57
so I come back online and I after a crash
当我从故障中恢复过来后
46.56–47.00
I would look and say oh the last page when I went to compute the checksum doesn't match
接着，在我去查看最后一个page中所计算出的checksum时，发现它的值和预定的值并不匹配
47.00–47.02
because this thing didn't get written out
因为数据并没有被写进去
47.02–47.04
so therefore I would do I have an error
因此，我就会得到一个报错
47.04–47.04
right

47.04–47.10
and then we'll talk about logging and no in a second
接着，过会我们会去谈论下日志方面的内容
47.10–47.12

but like you can log the operations you do to modify the pages
你可以通过日志的形式记录下你修改page时所进行的操作
47.12–47.15
and that's essentially what the databases worries about mostly
从本质上来讲，这也是数据库所最为关心的一部分
47.15–47.19
 alright cool
Cool

47.19–47.22
so that's all I'll talk about what that actually looks like inside these pages
So，这就是我要谈论的这些page中实际所存在的内容了



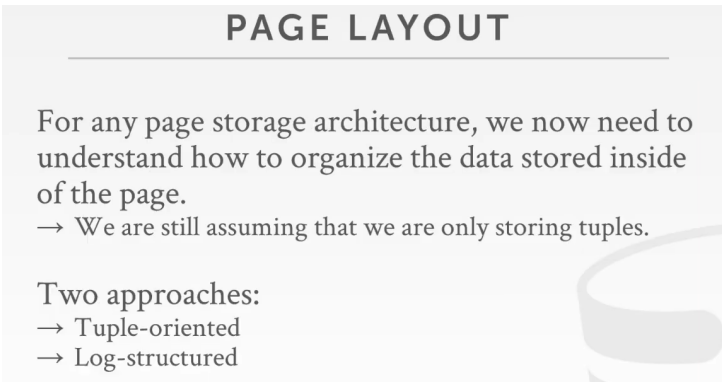
47.22–47.26
again every page is gonna have a header
每个page中都有一个header
47.26–47.36
and it's sort of what he asked about，we're gonna have information what's the size of the page the checksum what database version or the version of the software wrote this data out
这也就是之前那位同学所问的内容，在header里面，我们有page大小，checksum，DBMS版本之类的东西

47.37–47.41
what could happen is people can you know data is companies put out new releases
数据库公司会将新的发行版信息放在其中以供人们获取

47.41–47.43
Postgres puts out new releases every single time
Postgres每次都会发布新版本
47.43–47.45
 you know the page layout may change
它的page layout 就有可能发生改变
47.45–47.51

so when you want to upgrade, you want to know am i looking at pages that are created by the new software, the old software
当你想要升级的时候，你就会想去查看这些page是由新版软件所创建的还是由旧版软件创建的
47.51–47.53
and I can have different code paths to interpret them
我可以通过不同的代码来对它们进行解释
47.53–47.58
if you're doing compression like the dictionary compression or like lz4 gzip
如果你想进行压缩，例如使用字典压缩或者Lz4和gzip之类的方法进行压缩
47.58–47.59
you can store information about that
你可以通过这种压缩方式来保存数据
47.59–48.02
we all talked about this in the semester
我们会在这个学期里面讨论这个东西
48.02–48.08
but it's also information about you know what transactions or what queries modify this data and whether other queries allowed to see it
也会讨论关于事务，数据的查询，修改和查询权限之类的东西
48.08–48.09
again

48.09–48.13
and then we've already talked about the issue of a maybe self–contained
我们回到一个已经讨论过的问题：self–contained

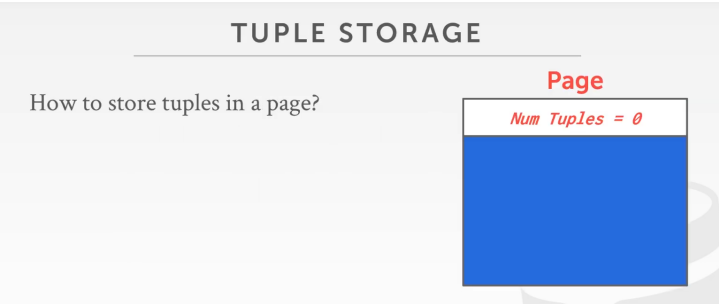


48.13–48.14
right

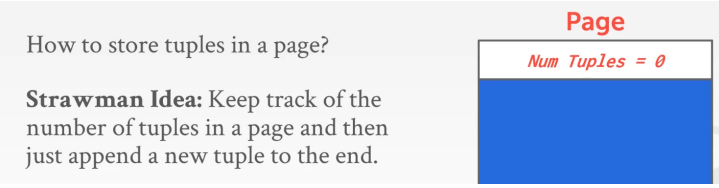
48.14–48.18
so now within a page we can represent data in two different ways
在一个page中，我们可以通过两种不同的方式来表示数据
48.18–48.22
so we can do this as a sort of a tuple oriented approach
So，我们可以通过面向tuple的方式来表示数据
48.22–48.24
and I'll explain what that means the next slide
我会在下一张幻灯片上解释这是什么意思
48.24–48.28
or we can do a log structured approach
我们还可以采用一种log–structured 的策略

48.28–48.31
so again it's within a page now
So，再次，在一个page中
48.31–48.36
assuming of a page directory to tell us how we need to get to that page，if we want you know a particular page of one two three
假设，如果我们想要某个特定的page，例如，page 1、2和3，那么page目录就会告诉我们该如何找到我们需要的那个page

48.36–48.39
now we're talking about what does it look like when you look inside the page
现在，我们来谈论下当我们在看page内部的时候，它里面是什么样的
48.39–48.41
what do you actually the data set I'm actually going to see
在page内部，我们实际会看到怎样的数据集合



48.41–48.45
so for this one let's just assume that we're storing tuples
So，在这个例子中，假设我们保存的是tuple

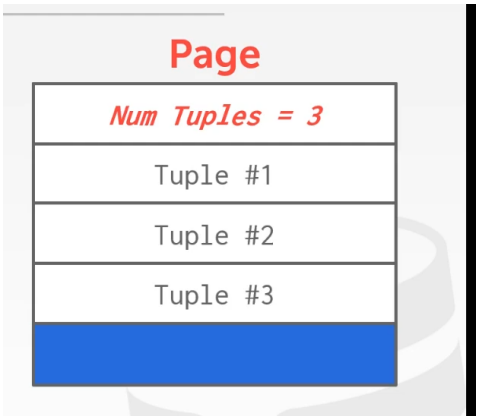


48.45–48.55
and let's say a really simple case, here a really simple strawman main idea is that, in our page all we're gonna do is just insert tuples one after another
在这个简单的例子中，我们有一个非常简单的strawman（稻草人） idea，那就是在我们已有的tuple后面再接着插入一个新tuple
48.55–48.59
right start from the beginning，we have a little header space and say here's the number of tuples that we have
我们从这个page的上面开始讲起，在它上面我们有一个很小的header空间，这里面存放了我们所保存的tuple数量

48.59–49.03
so we know what offset that we want to jump to， if we want to insert a new one
So，如果我们想插入一个新tuple，那我们就知道我们想要跳转的偏移量是多少

49.03–49.07
but it's super simple we just insert one at a time
这非常简单，因为我们一次就插入一个tuple
49.07–49.08

right



49.08–49.12

~~so say a third start~~ three tuples assuming they're all fixed length

假设我们有3个tuple，它们的长度都是固定的

49.12–49.14

every single time I insert one

每次我插入一个tuple的时候

49.14–49.17

I just jump to the next off free offset looking and then update the counter

我只需找到下一个空闲的偏移量处插入该tuple，并更新计数器

49.17–49.20

so this is a bad idea

这其实是一个糟糕的注意

49.20–49.21

why

为什么呢?

49.21–49.27

perfect

回答的漂亮

49.27–49.28

so he says, yeah if you delete a tuple

这位同学表示，如果你删除一个tuple

49.28–49.29

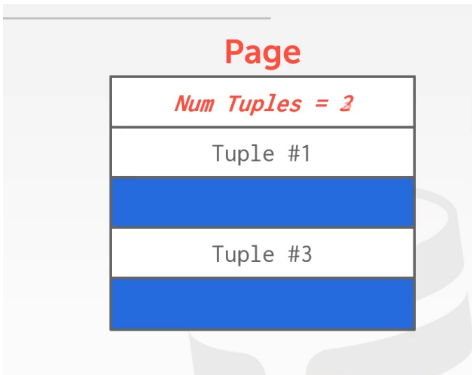
you have to move everything

那你就得移动所有的tuple

49.29–49.31

well not necessarily right

Well，这其实并不是必须这么做

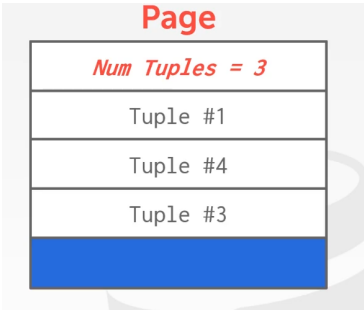


49.31–49.32

I could just do this
我可以这样做
49.32–49.33
all right

49.33–49.34
free the space up
释放掉原来tuple所在的空间
49.34–49.38
he says the external fragmentation
这位同学表示这会产生外部碎片
49.38–49.41
well why can't I just insert in there
Well，为什么我不能在这里面插入呢?
49.41–49.43
what's that?
你们想吐槽啥?
49.43–49.49
right so I made the assumption that their fixed length size，but he's absolutely right
虽然我假设这些tuple都是固定长度的，但这位同学说的没错
49.49–49.54
so this works great fixed everything's fixed Lane could I just shove it the new one where the old one was
通过将之移到一个新的空间来取代这个老的空间就可以很好的工作了（知秋注：也就顺带做到了压缩，懂Java的童鞋可以联想下GC）

49.54–49.55
but it's not fixed length
但如果它并不是固定长度的
49.55–50.00
then this slot may actually you know this location may not be big enough for what I would insert
那么你想插入的那个位置可能就没有足够的空间去保存那个tuple

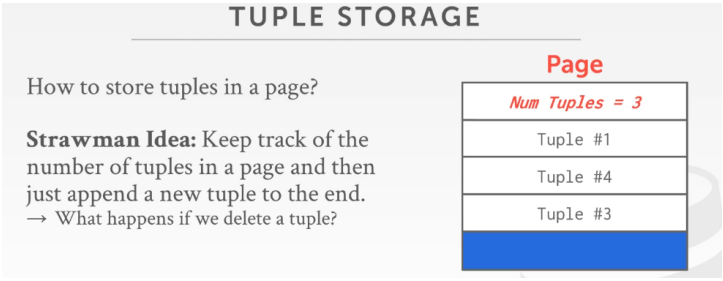


50.00–50.02
and now I got to try to put it down in here
现在，我试着将它放到下面
50.02–50.03
right

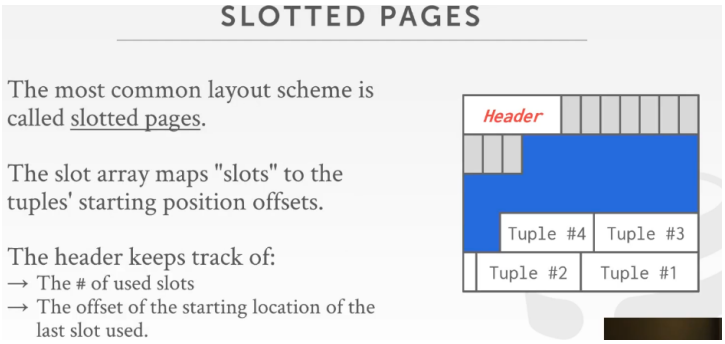
50.03–50.07

so so that's one issue
So这就是其中一个问题
50.07–50.11
I mean the other issue too is like every single time I need to go say I delete this thing
另一个问题是，每次当我需要去删除某个tuple的时候
50.11–50.14
I either maintain metadata at the top and
我需要去维护顶部的元数据
50.14–50.18
tell me hey here's a location in this page or you can write some data
元数据会告诉我们在这个page上的某个位置，我们可以写入数据

50.18–50.23
or I got a sequentially scan and look at every single tuple to figure out where I can go
或者我通过对这个page进行循序扫描，然后找到我能插入tuple的位置



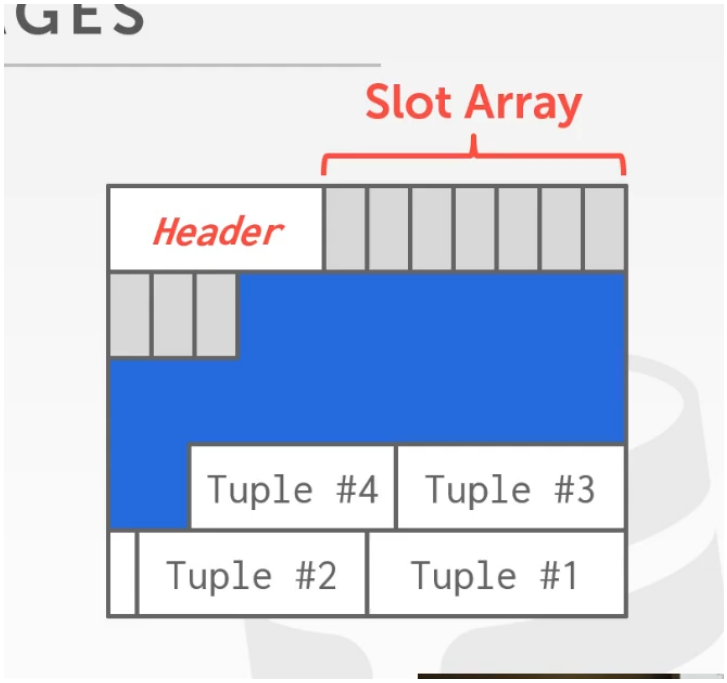
50.23–50.24
right so this sucks
So这很操蛋
50.27–50.26
nobody does this
没人会这么干



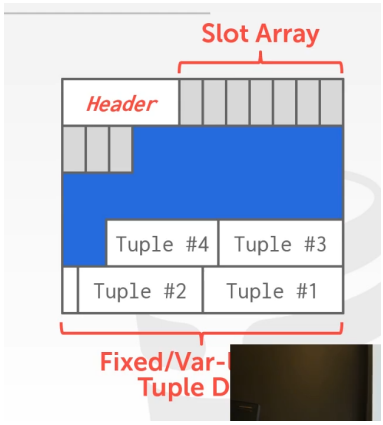
50.26–50.30
instead what you do is what it calls slotted pages
相反，我们要用的东西被称为slotted page
50.30–50.32
so this is the most common scheme
这是一种最常见的scheme
50.32–50.37
that every disk in a database system will use
数据库系统会在它所用的所有磁盘上都使用这种scheme
50.37–50.39
the exact details of how they're going to represent these pages are we slightly different
数据库在表示这些page的具体实现细节上都有所不同

50.39–50.41
but at a high level this is what everyone does
但从高级层面来讲，所有的数据库系统都是这么做的
50.41–50.44
so the way things could work we're always gonna have our header
So，我们可以通过我们的header来做到这些
50.44–50.49
the header it can store that basic metadata about checksum or you know access times and things like that
header里面能够保存基本的元数据，例如checksum或者访问时间之类的东西

50.49–50.54
and then we're gonna have to sort of regions of data we want to store
接着，我们必须有能够保存数据的区域



50.54–5056
at the top would have what's called a slot array
在顶部我们有一个称为slot数组的东西



50.56–51.00
and the bottom we're actually going to have the actual the data we want to store
底部的空间则是我们用来保存我们想保存的数据

51.00–51.03
these in again we're assuming we're doing tuples here

假设，我们在这里对我们的tuple进行操作
51.03–51.05
so then this one can be fixed length or variable length tuples
So，这里我们所保存的tuple可以是固定长度或者是可变长度的
51.05–51.07
it doesn't matter
我们并不在意这里面是可变长度还是固定长度的
51.07–51.14
so what the slot array basically is a mapping layer from a particular slot to some offset in the page
So，本质上来讲，slot数组是将一个特定的slot映射到page上的某个偏移量上
51.14–51.18
Well， that's the starting location of the particular tuple that you want
即根据这个偏移量，你能找到你想要的那个tuple
51.18–51.20
right

51.20–51.23
and the reason why we want to have this indirection layers
我们之所以想要这个indirection层的原因是
51.23–51.28
because now we can start moving these within a page we can move these tuples around any way that we want
因为我们现在就可以将这个page内的tuple移动到我们想要的任何地方

51.28–51.32
again the upper levels of the system don't care
再说一遍，上层系统并不在意这档子事
51.32–51.37
~~right they can always~~ you know the record it's gonna the page ID and the slot number
你们要知道，一条记录的位置是由page id和slot number来一起确定的
51.37–51.41
 and all i need to do is move these things around and just update the slot array
我所需要做的就是移动tuple，并更新slot数组
51.41–51.43
and say here's where you're actually pointing to
告诉这个slot数组，你实际应该指向tuple移动后的位置
51.43–51.48
and the way we're gonna fill up the page is that the slot array is gonna grow from the beginning to the end
我们填充page的方式是从前往后对slot数组进行填充
51.48–51.51
and the data is gonna grow from the end to the beginning
然而数据则是从后到前进行填充
51.51–51.55
and at some point we'll reach in the middle where we can't store any new information
在某些时候，我们的数据占用了该一半大小的page，我们再也无法存入任何新信息了
51.55–51.56
and then that's what we say that our page is full

这就是我们所说的page已满
51.56–52.02
so yeah this means that there could be a small little gap in the middle
这就意味着，在中间部分可能存在了部分空隙
52.02–5203
where we can’t store anything
这些空间太小，所以我们没办法存任何东西
52.03–52.07
but that’s you know because we wanted to support very length tuples
但你们知道我们想去支持可变长度的tuple存储
52.07–52.08
we have to we have to do this
所以，我们不得不这么做
52.08–52.10
all right

52.10–52.16
we could do what’s called a vacuum or compaction we could just scan through and reorganize defragmentation
我们可以进行一种称为vaccum的操作(Postgres中的一个操作，用于整理数据库)或者压缩，也可以对数据库进行扫描并整理碎片
52.16–52.20
and in our file systems we could do that in the background the database system we can do it
我们可以在我们的文件系统后台这么做，也可以在数据库系统中进行这些操作
52.20–52.24
but for our purposes, this is what we end up
但这就是我们最终要完成的目标
52.24–52.25
with yes
请问
52.25–52.31
good point
观点不错
52.31–52.38
so his question is are we assuming here that the within a page we could have tuples from different tables
So，他的问题是，我们假设能够在一个page内保存不同表的tuple
52.38–52.40
in practice nobody does that
在实际操作中，没有人这么干

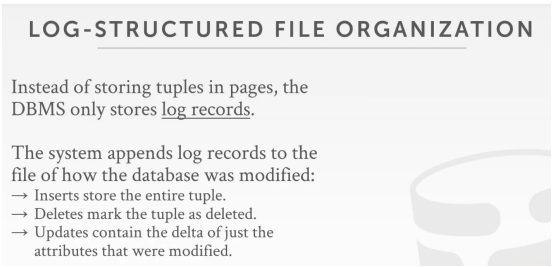
52.40–52.45
because you would have to maintain some metadata, say this is from tuple want a table one this is from table two
因为你就不得不去维护元数据，比如你就得说明，这个tuple来自表1，那个tuple来自表2
52.45–52.49
we’ll see at the end that there is a way to there’s some systems that do do this
我们会在这节课的最后看到有些系统确实有这种操作

52.49–52.51
but in general nobody does this
但一般情况来讲，没人会这么做
52.51–52.54
like if you open up SQLite、Postgres or whatever you call create table
如果你在SQLite，Postgres或者其他数据库中创建表时
52.54–52.57
it'll create pages and only tuples from those tables will go in those pages
数据库就会去创建page，这些表中只有这些表的tuple会保存在这些page中
52.57–53.01
it's good question what will come to that in a second
这是个好问题，我们稍后会看到
53.03–53.04
so these are tuple oriented pages
So，这就是面向tuple的page
53.04–53.08
and at the end day we're trying to store tuples inside these pages
在最后的课程我们会试着去往这些pages中存入tuples
53.08–53.10
and so you know when I do an insert I do an update
当我进行插入或者更新操作的时候
53.10–53.16
I want to find you know I take the contents of the tuple and just write it out in its entirety in this page here
我想要将拿到的tuple的内容，整体写入到这个page中
53.16–53.27
in next class, we'll talk about for really large data like if you have a tree that's like you want a store like a you know a video file in the database
在下节课中，我们会去讨论大型数据的存储，例如，我们想将一个视频文件保存在我们的数据库中之类的例子
53.27–53.28
don't do that
请不要这么做

53.28–53.30
what's fine why later
我会在之后解释原因

53.30–53.33
but like for that case here，you couldn't store this because it won't fit in a single page
但对于这种例子来说，你没办法把视频存在数据库里面，因为单个page根本放不下它
53.33–53.38
so you have some extra metadata some pointers to say here's the pages that have the rest of the data that you're looking for
So，你就必须通过一些额外的元数据和指针来表示我们所要查找的剩余部分的数据的page所在位置
53.38–53.41
but in general we want us pack in the entire tuple in a single page
但一般来讲，我们想将一整个tuple放在一个单个page中

53.41–53.48
because now when we go access that, you need to access to the tuple it's one page read to go get it and not a bunch of different ones
因为当我们想去访问这个tuple的时候，它就在这个page上，那我们直接读取就行，而不是分散在多个page上
53.48–53.49
again
53.49–53.51
we'll break that assumption next class
在下节课中，我们会打破这个假设
53.51–53.54
but for our purposes here it's fine
但对于我们此处的目的而言，这没什么问题



53.54–54.06
so another way to store data and pages is it's our question
你有问题是吗？请讲
54.06–54.09
so his question is say I move the third tuple here
So，他的问题是，我将这第三个tuple移动到此处
54.09–54.10
right

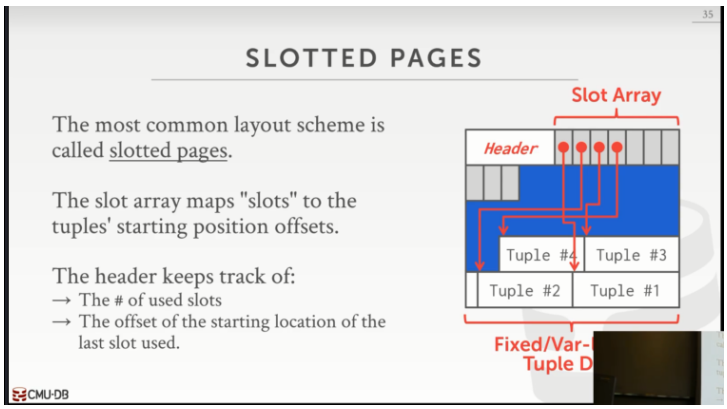
54.10–54.11
what happens
这会发生什么呢?

54.11–54.15
well it depends I'll give it a minute in the class
这得看情况来说了，我稍后会在课上讲这个问题
54.15–54.17
some systems will actually compact it before it writes out the disk
某些系统在将数据写出到磁盘时，实际上它们会对数据进行压缩
54.17–54.19
some systems will just leave a gap here
也有些系统会在page中留下些空隙
54.19–54.21
and then if it gets full
如果page的容量满了的话
54.21–54.23
and you say oh I have some free space maybe I try to do compaction
你会说，诶，这里还有点空闲空间，我们可以应该试着压缩下数据
54.23–54.25
yes

请问
54.28–54.36
yes this question is the slot is pointing to the starting position of the tuple
他想表达的是slot指向的是tuple的起始位置

54.36–54.47
~~the question is what does a point~~ so the question is what is the ordering of the storage address within the slots
他的问题是这些slot的存储地址的顺序是怎么样的

54.53–54.56
not sure what you mean
我不太懂你在说什么
54.56–54.57
like I have say this is 4 kilobytes
我说过这个是4kb大小
54.57–55.02
I want to store a 1 kilobyte tuple for tuple 1
我想保存一个1kb大小的tuple 1
55.02–55.06
so from starting from the offset，I jump to the 1 kilobyte
So，我以这个偏移量为起点，我跳到1kb的位置
55.06–55.08
and then my slot right points of that
我想要找的slot就在此处
55.08–55.13



right yes
请问
55.13–55.21
the question is if I delete one tuple in the middle
你的问题是，如果我删除page中间位置的一个tuple
55.21–
again I see tuple 3 what do I do it up above nothing miss on array?
就好比这个tuple 3来说，如果我按照刚才说的做了，上面这个Slot Array的对应slot会指向一个空（nothing）？

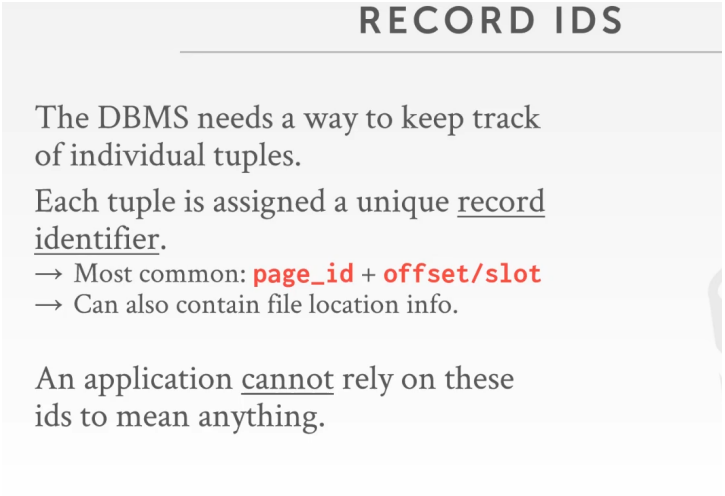
55.30–55.33

yes so again the header could different systems do different things
再说一遍，不同的系统做不同的事情
55.33–55.39
the header could have a bitmap and say you know here's the slots that are empty that you could put point something in
在header中有一个位图，通过它你可以知道哪些slot是空的，这样你就能够在它里面放些东西
55.39–55.42
or I just squential scan and read it
或者我可以进行循序扫描，就可以读到哪些slot是空的
55.42–55.43
right it doesn't matter
right 这没什么的

55.43–55.54
the key thing though I think is that the other parts of the system don't know and don't care that where I'm actually physically stored
我觉得关键事情在于系统的其他部分不知道也不在意我数据保存的物理位置在哪
55.54–55.59
like it first so for tuple 1 right this comes out of the slot here
比如说，tuple1是放在这个slot里面
55.,59–56.11
right so in the upper part of the system it would say Oh tuple one you find it in the page one two three, at slot zero slot one depending on what your starting offset is
系统的上层部分表示我要找tuple1在page123中，那么这个tuple是在slot 0或slot 1处找到，具体是slot 0还是1取决于你的起点偏移量（知秋注：数据存储的位置相对于开始位置的偏移量，因为slot存储的是偏移量）

56.11–56.15
so now no matter how I reorganize my page and move tuple 1 around
现在，不管我如何整理我的page，也不管我怎么去移动tuple1
56.15–56.19
I know that I always want to go to the first slot to find where it's actually located
我知道我始终想去第一个slot处找到这个tuple1，这也是它实际所在的位置
56.19–56.21
and now if I reorganize
现在如果我整理page的话
56.21–56.23
I don't have to update my indexes on to update anything
我无须去更新我的索引，也不用去更新任何东西
56.23–56.26
and this is sort of what the page directory is trying to do as well
这也是page目录所试着做的事情
56.26–56.33
so no matter where I move the pages on the file either on disk or different look you know on the network
So，不管我是将文件中的page移动到磁盘上还是网络上
56.33–56.37
other parts of the system don't care where it actually got moved to
系统的其他部分都不会关心这个page实际移动到了哪里

56.37–56.39
because ~~I have a page once~~ you know I have the page ID
因为你们知道我有page id
56.39–56.42
I can only use the page directory to find where it's actually main stored
我只能通过page目录来找到它实际所保存的位置
56.42–56.49
right these indirection layers avoid having to have updates propagate through all the parts of the system
这些indirection层避免了这些位置更新信息传播到系统的其他上层部分（知秋注：上层只需要知道一个page id就可以了，可以思考下GC，Java只保持对象间的引用关系就好，至于在对象到底存在内存哪个位置，随着GC进行，是会发生变化的，此处的page ID就好比是维护引用关系一样）
56.49–56.50
yes
请问
56.57–57.02
his question is how do I know that tuple one is stored in in slot one
他的问题是，我该如何知道tuple1就是保存在slot 1中的



57.14–57.17
so this is always in the last slide
So，这一般是最后一张幻灯片
57.17–57.18
but let's talk about it now
但我们现在就来讲下这个问题
57.18–57.24
so the way we identify tuples is through these record IDs or tuple IDs
我们识别tuple的方式是通过这些record id或者tuple id来做到的
57.24–57.35
and it's essentially a unique identifier to say here's the logical location or a logical address of a tuple
本质上来讲，它是一个唯一标识符，用来表示一个tuple的逻辑地址

57.35–57.38
it's a blend of logical and physical
它是一种逻辑位置和物理位置的结合
57.38–57.42

but usually the page ID and like the offset or the slot
但通常情况下，我们是使用page id加offset值或者slot来进行表示
57.42–57.44
so all the parts of the system that we want to address tuple one
对于我们系统的所有部分来讲，我们想要去找到tuple 1的位置
57.44–57.47
right they don’t know what tuple 1 is
但它们并不知道tuple 1是什么
57.47–57.48
they just know I have a page ID and a slot number
它们只知道我有一个page id以及一个slot number
57.48–57.52
and so I go to the page directory and say I want page 1 2 3
So，我跑去page目录那里，并表示我想要page123
57.52–57.54
the page directory says oh it’s in this file and this offset jump to that page
目录就会表示它在这个文件中的这个offset值所在的地方
57.44–57.57
then I get to that page
然后，我就得到了这个page
57.57–57.59
now I say oh I want slot 1
现在，我表示我想要slot 1
57.59–58.01
I look at my slot array
于是我在我的slot数组中进行查找
58.01–58.03
and that tells me where in that page you can find the data that I want
它就会告诉我，我可以在page中的哪个位置找到我想要的数据
58.03–58.11
so other parts of the system like the indexes log records and other things they’re going to address tuples through these record IDs
So，对于dbms系统的其他部分，例如索引，日志或者其他东西来说，它们可以通过record id来定位tuple的位置

~~58.11–58.15~~
~~that’s separate from the page~~

~~1563~~
~~00:58:22,250 --> 00:58:34,810~~
~~like as well yes me it like~~

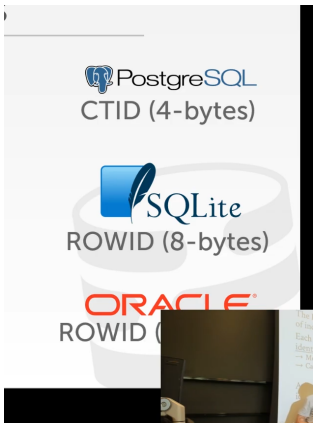
58.35–58.35
I’ll give a demo
我会用一个demo来向你解释
58.35–58.44
~~hope let’s make myself like~~ say I want to ~~find find that the salary~~ find the the student record or the professor record the name Andy
假设我想去找到某个名字叫Andy的学生或者教授的记录

58.44–58.46
I look in the index on the name
我会去看下索引上的名字
58.46–58.49
and something to say oh there's a professor named Andy
我表示，Oh，我看到这里有一个叫Andy的教授
58.49–58.53
and he has a record ID of page one two three offset slot one
他的record id是 page 123，偏移值是slot 1
58.53–58.55
that's what index gives me
这就是索引所给我的信息
58.55–58.57
so then I say it go to the page directory
接着，我跑到page目录那里
58.57–58.57
okay

58.57–59.01
Gav where do I find page one two three you go get it for me
并对它表示，我在哪里能找到page123，请你帮我找一下
59.01–59.02
it goes and gets it
page目录就会帮我找到它，并告诉我这个page的位置
59.02–59.03
now I have the pointer to the page
现在我就得到了指向这个page的指针
59.03–59.05
and say oh I want slot one
并表示，我想要slot 1中的数据

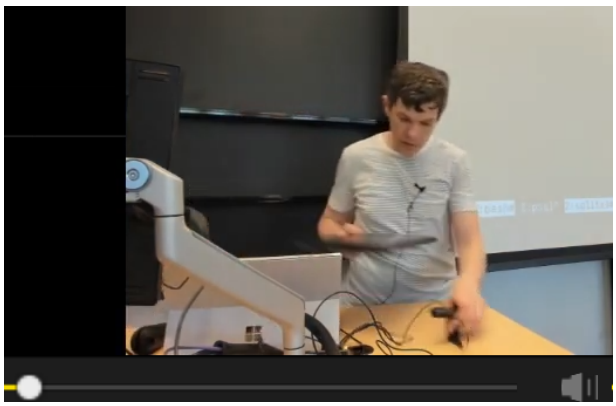
59.05–59.09
I look at my slot array and that tells me what offset the jump to that page to find that need
于是我就在我的slot数组查找我所需要的数据在该page的偏移值是多少，然后跳到那个位置
59.09–59.13
right and so different database systems do different things
So，不同的数据库系统做不同的事情
59.13–59.17
the most common approach is the page ID and the slot number the offset
其中最常见的方法就是通过page id和偏移值slot number来确定tuple的位置
59.17–59.24
and again advantage of this is that if I start moving data around either moving the page around or moving data within the page itself
这种做法的优势在于如果我移动数据，不管是移动page也好，还是在这个page内移动数据也罢
59.24–59.27
the index and all the other crap doesn't have to get updated
我们无须去更新这些索引和其他的一些东西
59.27–59.30

because they’re still looking at page one two three offset one
因为它们依然会去page123和偏移量1的地方查找数据
59.30–59.36
so okay now let me I’ll give a demo explain some more detail
So，我通过一个demo来解释下细节
59.36–59.38
so in different database systems do different things
So，不同的数据库系统做不同的事情



59.38–59.43
like in Postgres it’ll be 4 bytes or equals 10 bytes，Oracle 10 bytes there’s a bunch of extra metadata that they store
例如，在Postgres中，CTID的大小是4byte。然而，在Oracle中的ROWID使用了10byte来保存一系列元数据
59.43–59.45
and sequel Lite is 8 bytes
在SQLite中的ROWID则是8byte

==========
59.45–59.47
so let’s give a demo
So，我们来看个例子
59.47–59.49
because that’s always fun
因为它总是很有趣



59.50–59.53
because I hate typing on the surface
因为我非常讨厌在surface上打字
59.53–59.56
and use my other laptop
算了，我还是用我另一台笔记本吧

~~将它们组合在一起到我所~~实际情况来~~所在~~