05–01

05 – Buffer Pool s + Memory Management

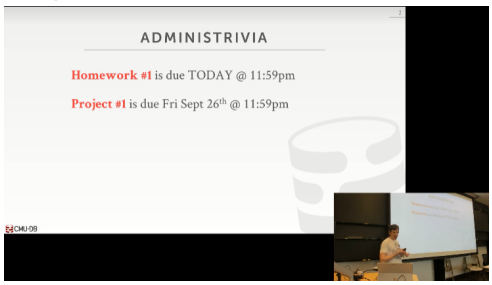
(CMU Databases Systems _ Fall 2019)

资料：https://15445 courses cs cmu edu/fall2019/notes/05–bufferpool pdf
Pdf: https://15445 courses cs cmu edu/fall2019/slides/05–bufferpool pdf

00:16 – 00:19
All right, hey done

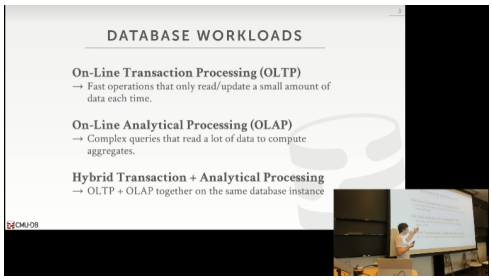
00:19 – 00:21
Grab

00:21– 00:23
Your wrap you look down,what's wrong
你今天看起来好丧，出什么问题了?
00:23 – 00:26
City women woman problems
感情问题?
00:26 – 00:36
What do you say women problems
What are your problems
不是？那是什么问题
00:36 – 00:43
They were saying your beats are too fresh and they can't handle it
他们说你的beats（节奏）太前卫（此处是反语贬低）了，他们无法认同
00:43 – 00:48
I'm not qualified to help ,I'm sorry okay
sorry，对此我无能无力



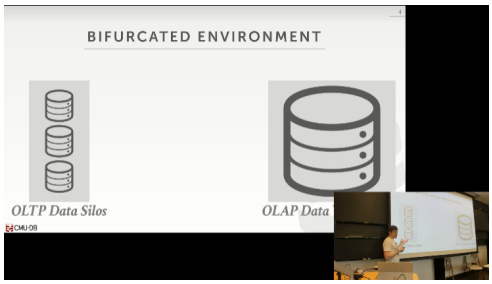
00:48 – 00:50
All right, so let's talk about databases system
言归正传，我们来讨论数据库系统吧
00:50 – 00:57

All right, so a quick reminder **Homework #1** is due tonight
So，提醒你们一下Homework 1今晚就截止了
00:58 – 01:01
And then **Project #1** is going out today
接着，我今天会把Project 1放出来
1.01–1.05
I'll sit at the end of the class there's actually the website now the source code is online
我会在这节课结束的时候放出来，实际上我已经将源码放在网上了
1.05–1.08
but I'll discuss what it is, what you're required to do today
但我会去解释下它是什么，以及你们今天要做什么
01:09 – 01:15
And then just like before you're submitted some great scope,and you know everything will be auto graded
和以前一样，你们将东西上传上去，然后网站会自动帮你们打分
~~01:16 – 01:19 (视频剪切问题，YouTube也是)~~
~~All right ,um I do want to spend some time talking ……~~
~~这句不要了~~

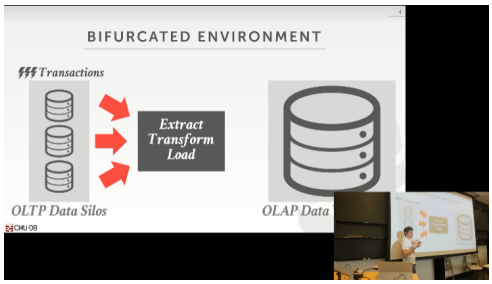


01:19 –01:23
~~Whatever lab workloads is where~~ after you've collected a bunch of data in the OLTP side

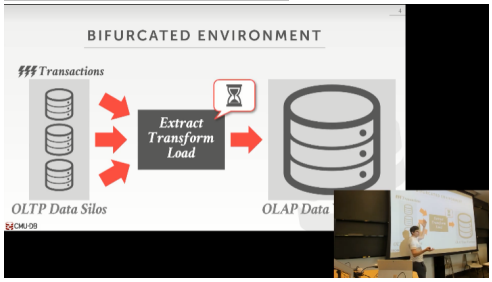
当你从OLTP处收集到一堆数据后
01:23 – 01:26
Now you want to start analyzing it to extrapolate new information
现在，我们想去开始对这些数据进行分析，以此推断出新的信息
01:26 – 01:30
Like people in the city of Pittsburgh are more likely to buy this kind of product
比如说，匹兹堡的人更喜欢买这种产品
01:30 – 01:36
I said you can use that information, then you know push information to the OLTP side to get people to do things you want them to do
你就可以将这个信息推送到OLTP处，让人们去做你想让他们做的事情（知秋注：人们会通过查询得到这条信息，然后你就可以迷惑对方了）
01:37 – 01:44
And then the **Hybrid Transaction Analytical Processing** HTAP workloads,this is sort of a new buzzword that Gartner invented a few years ago
接着就是HTAP （混合事务分析处理），它是几年前Gartner所发明的一个新流行词
1.44–1.47
basically describing these database systems that try to do both of them
基本上来讲，它所描述的是既做OLTP，又做OLAP的数据库系统



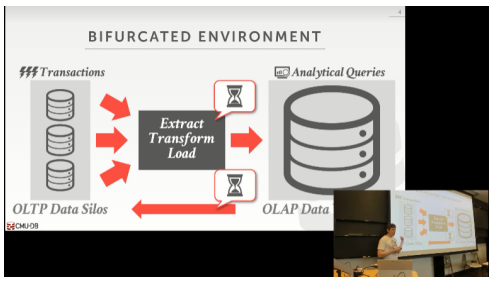
01:48 –01:52
So a typical setup you'll see often is like this
So，你们通常会见到这种标准设置
01:52 – 01:57
You'll have your front end OLTP databases and then you have your giant back–end data warehouse
你们会有前端OLTP数据库，以及后端大型数据仓库
01:57 – 1.59
So these are sometimes called *Data Silos*
这些有时候被称为Data silo(数据孤岛，即相互独立的数据存储区)
1.59–2.03
because you can do a bunch of updates into them sort of one database instance
因为你们可以对其中一个数据库实例进行一系列更新操作
02:03 – 02:07
Whether it's a single node or should be doesn't matter, because it's a single logical database
不管它是不是单个节点都没关系，因为它是一个单个逻辑数据库
02:07 – 02:10
And then you apply your changes here
然后，你将你的修改应用到此处的单个逻辑数据库（Data silo）上
2.10–2.14
but they don't really communicate with each other，each one is sort of an island by ltself
但它们彼此之间并不会真的交流，每个数据库自身都是一个孤岛



02:14 – 02:18
So then you can do what's called extract transform load or ETL
So，然后你就可以进行某种被称为ETL的操作（ETL是将业务系统的数据经过抽取、清洗转换之后加载到数据仓库的过程）

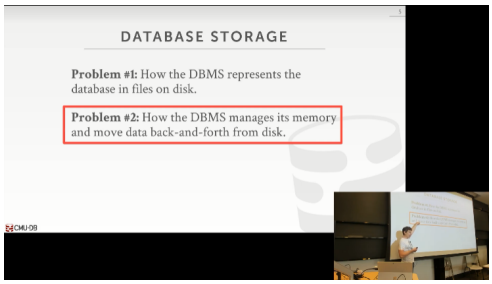


02:18 – 02:28
And this is sort of a the term you use is described taking data out of these front ends,cleaning it up processing it,and then putting it to the back end data warehouse
它所表达的意思是，我们从前端数据库中取出数据，将数据进行清洗处理，接着将处理后的数据传入后端数据仓库
02:28 – 02:30
So the example, I like to get for this is like Zynga
此处我想以Zynga为例（Zynga一家游戏公司）
2.30–2.34
the farmville people they buy a lot of gaming startups
Zynga收购了许多游戏初创公司，例如：FarmVille
02:35 – 02:36
And then when they buy them
然后，当他们买这些公司时
2.36–2.38
they all run their own front end OLTP database
他们会去运行他们自己的前端OLTP数据库
2.38–2.42
but then when they want to put it in their back–end giant data warehouse
但当他们想将这些数据放入他们的后端大型数据仓库时
02:42 – 02:45
So they can do analyze things to make you buy crap on farmville better
这样他们就能更好地分析出如何让你们在farmville上买东西
02:45 – 02:53
Right and so ,because say like in one database，the first name of a customer will be F name ,another database we you know F first underscore name
因为在一个数据库中，客户的First name是f开头，在另一个数据库中，则是f_
02:53 – 02:59
Right ,so it's the same concept or same entity just with different syntax and nomenclature
So，这两个是相同的实体，只是使用了不同的语法和命名法而已

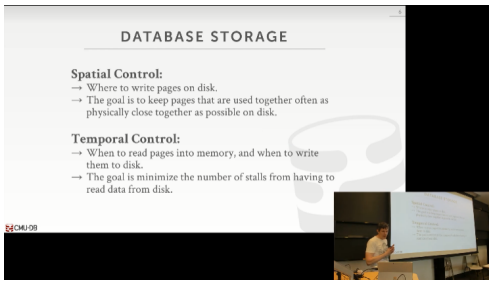


02:59 – 03:01
So this ETL process cleans all that up
So，ETL会对这些数据进行清洗处理
3.01–3.04
you shove it to your data warehouse, you do all your analytics here
你将它们推入你的数据仓库，并在那里进行所有的分析
03:04 – 03:07
And then whatever new information you have，you push it to the front
接着，不管你拿到什么新信息(分析后的)，你将它们推到前端OLTP数据库就行

03:08 – 03:12
All right, and when you see things like people that bought this item also bought this item
当你看到人们买了这个东西，又买了那个东西
3.12–3.14
that's ~~that~~ doing that on the OLAP side
这是在OLAP处进行的
03:14 – 03:17
And then they shove it to the front end to expose that through the OLTP application
~~接着，将数据推送到前端，并将其暴露给OLTP应用程序~~
接着，将数据推送到前端，并通过OLTP 应用程序进行对外暴露
03:18 – 03:24
So HTAP basically says, let's just also do some of the integral queries that we can normally only do on the OLAP side
So，HTAP基本上是在说，让我们也来做一些平常只能在OLAP端所做的
03:24 – 03:26 ！！！！！！！！
We can do it on the front end Data Silos
我们可以在前端的Data silo里面做这些
03:26 – 03:30
You still want this giant thing your giant data warehouse
我们依然想使用这个大型数据仓库
3.30–3.35
because you want to be able to look at all your Data Silos put together
因为我们想能够看到我们所有的Data silo放在一起的样子
03:35 – 03:38
But now instead of waiting for things to be propagated to the backend
但现在，不用等待将数据传播到后端
03:38 – 03:40
You can do some things on the front-end
我们就可以在前端来做些事情了
03:40 – 03:41
So that's basically what HTAP is
So，基本上来讲这就是HTAP所干的事情
03:41 – 03:45
So again this could be like your MySQL,does your Postgres ,MongoDB whatever you want
So，再次，这个你可以用MySQL，也可以用PostgreSQL或者MongoDB这类数据库来做，或者也可以选你想要的数据库来做
03:45 – 03:52
And then your back-end data warehouse would be Hadoop stuff,Spark,Greenplum,Vertica
接着，你的后端数据仓库可以用Hadoop，Spark，Greenplum或者是Vertica来做
3.52–3.57
there's large enterprise data warehouse systems ,Redshift or Snowflake your other cloud ones
这些都是大型企业级数据仓库系统，你们也可以使用RedShift或者Snowflake这些云端的数据库系统
03:57 – 03:58
Okay,so this is clear
Ok，So这样你们应该清楚了

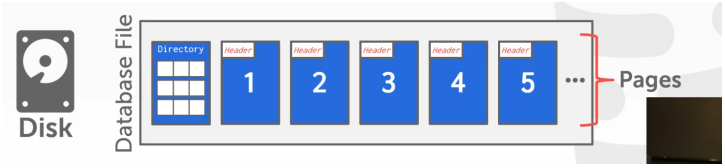


03:59 – 04:09
Okay, so the main topic today,we're talking about is now given that we've already spent two previous lectures on deciding how we're actually gonna represent the database in on disk
Ok，之前我们已经花了两节课时间来决定我们该如何在磁盘上表示数据库
04:10 – 04:18
Now we want to talk about what we actually do to bring the database from those files on disk the pages on disk and bring them into memory ,so that we can operate on them
今天我们想去讲的内容就是，我们该如何将磁盘中的数据库文件或page放到内存中，以便我们可以对它们进行操作
04:18 – 04:23
Right, so remember that we the database system can't operate directly on disk
So，要记住数据库系统无法直接在磁盘上进行操作
04:24 – 04:27
We can't do reads and writes without having to bring into memory first
我们没办法在不将它们先放入内存的情况下对这些数据进行读写
4.27–4.29
~~that~~ that's the von–Neumann architecture
这是冯诺依曼架构
04:29 – 04:34
Now there are some new hardware coming out you can push execution logic down to the disks
现在也推出了一些新的硬件，我们可以将这些处理逻辑推送到磁盘上
04:35 – 04:36
But we can ignore that for now
但现在，我们可以将它抛之脑后
04:37 – 04:40
So we're trying to figure out how do we want to bring that those pages in the disk
So，现在我们想试着弄清楚如何将这些page放回磁盘中
04:41 – 04:45 **********
And we want to do this and be able to support a database that exceeds the amount of memory that we have
我们希望做到这一点，并且想要能够去支持超出我们所拥有的内存容量大小的数据库
04:45 – 04:52
And we want to minimize the impact the slowdown or the problems of having queries tap that touch data on disk
我们想去最小化在磁盘上执行查询速度缓慢带来的影响
4.52–4.54
we want to make it appear as if everything's in memory
我们想让这些操作看起来就像在内存中执行那样

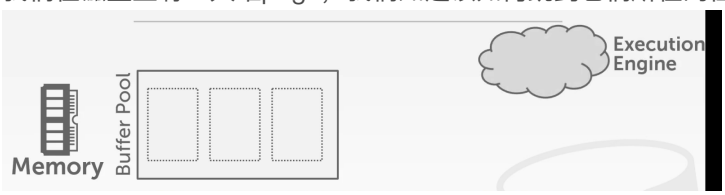


04:54 – 05:00
So another way to think a problem is also in terms of Spatial versus Temporal Control
So，我们也可以从空间和时间的管理上来思考这个问题
05:00 – 05:05
So Spatial Control is you know where are we physically gonna write this data on disk
空间管理指的是我们实际是在哪里将数据写入磁盘
05:06 – 05:11
Right, meaning like we know that these pages are can be used together often possibly one after another
这意味着，我们想要尽可能的将这些经常使用的page一个挨着一个的放
05:11 – 05:13
So when we write those pages out , we want to write them sequentially
So，当我们写出这些page时，我们想按顺序对它们进行写入
05:13 – 05:15
So that when we go read them again
So，当我们再去读取它们时
5.15–5.23
physically close to each other and we don’t have to do long seeks to find different spots on disk
它们是一个挨着一个放的，所以它们在物理位置上彼此靠近，这样我们就无须花很长时间在磁盘上找到这些不同的地方
05:23 – 05:24
We also care about temporal control
我们也会去关心时间上的管理
5.24–5.30
and this is where we make decisions about when do we read pages into memory, what time we do this
即我们会决定该在什么时候将pages读入内存
05:30 – 05:34
And then at some point we have to write it back out if it’s been written, if it’s been modified
接着，在某些时候，如果它们被修改了，我们必须得将它们写回磁盘
05:34 – 05:37
And we don’t make a decision of when we actually go ahead and do that
但我们不会对何时进行这种操作做出决定
05:37 – 05:42
And yeah this is the overarching goal of trying to minimize the number of stalls we have
~~这就是为了试着最小化我们所面对的烂摊子的总体目标~~
总体目标就是尝试最小化我们所面对的烂摊子
05:42 – 05:45
Because our queries try to read data that we didn’t have in memory

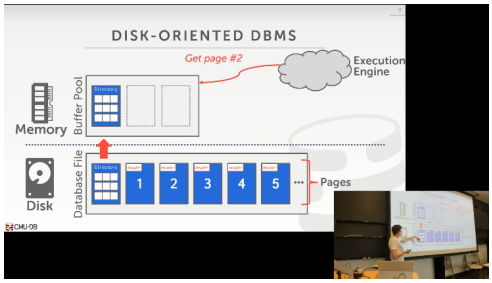
因为我们的查询会试着读取那些不放在内存中的数据
5.45–5.48
and we had it write up, you know that was out on disk we had to go fetch it
我们必须将它写下来，你们都知道，这些数据是放在磁盘上的，所以我们得去拿到它



05:48 – 05:53
So this is the overall architecture of the lower store manager,but I showed in the beginning
So，这就是我一开始所展示的底层存储管理器的整体架构
05:54 – 05:55
So we've sort of covered this part already
So，我们已经介绍过这部分了
05:55 – 05:59
So now we know how to have a database file or files on disk
So，我们知道如何在磁盘上保存数据库文件
05:59 – 06:01
We know how to represent the page directory to find the data we need
我们也知道如何表示page目录，以此来找到我们需要的数据
06:01 – 06:05
And then we have a bunch of pages sloted pages log structure pages ,it doesn't matter
接着，我们有一大堆page，不管是slotted page还是log-structured page，这都无所谓
06:05– 06:08
We have a bunch of pages on disk and we know how to jump to them to find them
我们在磁盘上有一大堆page，我们知道该如何跳到它们所在的位置，并找到它们

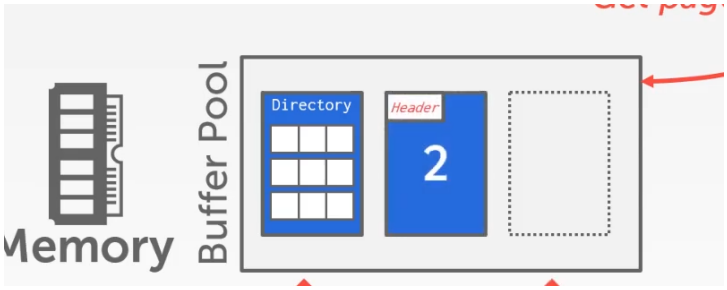


06:08 – 06:11
So now we're talking about this part up here at the Buffer Pool
So，现在我们要去讨论幻灯片上Buffer池这一部分



06:11 – 06:16
Right ,when something else in the system like the execution engine the thing executing queries comes along says,I want to read page two
当系统中的执行引擎想去执行查询时，它会说：我想去读取page 2

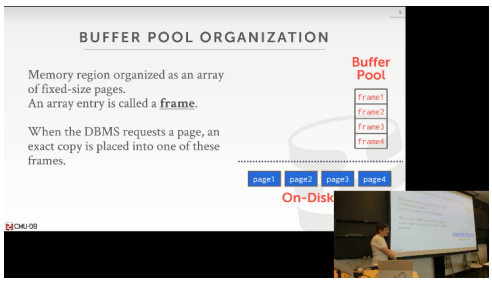
06:17 – 06:20
We got to know how to fetch the page directory into memory ,figure out what's in there
我们知道如何将page目录放到内存中，并弄清楚里面有什么



06:20 – 06:23
And then go find the page that we want and fetch that into memory
接着，从里面找到我们想要的那个page，并将它放入内存
06:24 – 06:29
And then the tricky thing is going to be ,if we don't have enough space now free memory to bring that page we need in
接着，棘手的事情出现了，那就是我们没有足够的空余内存来容纳我们需要的那个page
6.29–6.31
we have to make decision what page to write out
我们必须决定对哪一个page进行写出
06:31 – 06:35
So that's you know this is what we're trying to solve today
这就是我们今天要解决的问题
06:35 – 06:43
Right, and then the other parts of the system don't need to know or really care about what's in memory with not in memory
系统的其他部分无须去知道或者去关心哪些东西在内存里面，哪些东西不在内存里面
6.43–6.48
there's going to wait until you get the thing you need and give you back a pointer to let you do whatever it is that you wanted to do
它会等到你拿到你需要的东西，然后返回给你一个指针，以此让你做你想做的事情
06:48–6.49
Okay

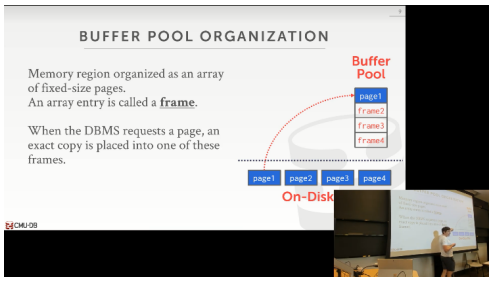
06:50 – 06:55 ！ ！ ！
So the things were talked about today is essentially just how to build what a Buffer Pool manager actually gonna do
So，今天我们要讨论的东西就是如何去构建一个Buffer池管理器（Buffer Pool Manager）
06:56 – 07:01
In some comedies the term Buffer Pool manager,some systems will call this a buffer cache it's the same thing
在有些场景下，某些系统会将Buffer池管理器叫做buffer缓存，它们是一回事

07:01 – 07:04
Right, it’s memory manage by the database system
它是由数据库系统管理的内存
07:04 – 07:12
Then we’ll talk about how we actually can do different policies that decide what pages we want to write out the disk, if you need to free up space
然后，我们会去讨论，当我们需要释放内存空间时，我们该如何使用不同的策略来决定我们想让哪些pages写出到磁盘上
7.12–7.16
additional optimizations We can do to minimize this impact
然后会讨论我们可以通过哪些额外的优化来最小化这种影响
07:16 – 07:21
and then we’ll finish up talking about two other pieces of the database system that may need memory ,Okay
然后，我们会结束讨论数据库系统中其他两个需要用到内存的部分，Ok



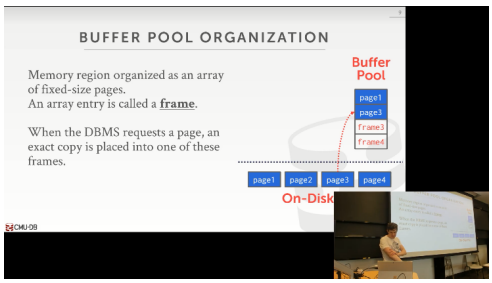
07:23 – 07:29
So again the Buffer Pool is essentially just a large memory region that we’re gonna allocate inside our database system
本质上来讲，Buffer Pool需要我们在数据库系统内部分配的一块很大的内存区域
07:29 – 07:30
We’re gonna call **malloc**
我们会去调用malloc
07:30 – 07:35
I want to get some chunk of memory,and that’s we’re gonna put all our pages that we fetch from disk
我想要拿到一些内存块，并将我们从磁盘中读取到的所有page放入里面
07:35 – 07:42
And so this is again ,this is all entirely managed by the database system other than having go to the operating system and ask for the memory
So，再说一遍，这段内存完全是由数据库系统来控制的，而不是操作系统来分配这些内存的
07:42 – 07:46
Right, we have to use **malloc** there’s ,we handed **malloc** allocate memory on our own
此处我们使用malloc，由我们自己来手动分配内存
07:46 – 07:47
So we know OS can provide us this
我们知道操作系统可以为我们提供这个
07:48 – 07:55
But then we’re gonna break up this memory region into fixed size or page size chunks called **frames**
但之后，我们将这段内存区域分成一个个固定大小的chunk，它被称为frame
07:56 – 08:01

And this is you know **frame** seems kind of unusual why don't I just say page or block or whatever
这里我为什么将它叫做frame，而不是把它叫做page或者是block或者其他名字呢
08:01 – 08:05******
There's so many different terms in database systems to roughly describing the same thing
数据库系统中有太多不同的术语可以粗略地描述同一件事
08:06 – 08:11
Frames correspond to slots in the or see I use the term slot when use that
Frame对应的是我们之前用的slot
08:11 – 08:16
Frames correspond to regions or chunks in the Buffer Pool memory region that we can put pages in
frame对应的是Buffer池内存区域中的区块或者Chunk，我们可以将page放在里面
08:16 – 08:20
Right, and we slot is the thing we put things into pages within for tuples
slot是我们在page中用来放置tuple（知秋注：slot存储的是对应的offset值，但其实对外来看，它指代了一段存储区域）
08:20 – 08:23
So for Buffer Pool , it's frames for on the page it'll be slots
So，对于Buffer池来说，它叫frame。对于Page来说，它就是slot
08:24 – 08:30
So what happens is when the database system calls makes a request and say I want a page
当数据库系统发出一个请求，表示我想要一个page时，会发生什么呢?

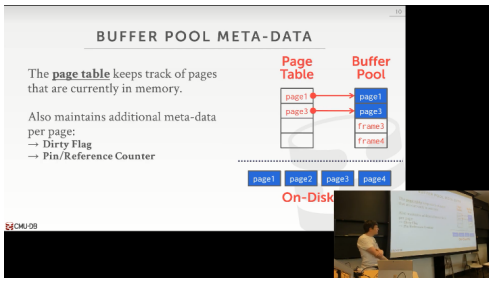


08:30 – 08:33
Right, we're gonna look to see whether it's already in our Buffer Pool
我们会去看我们的Buffer池中是否存在这个page
8.33–8.38
if not，then we go out in the disk make a copy of it fetch that put it into memory
如果不存在，那我们就从磁盘中拷贝一份出来，并将它放到内存中去
08:38 – 08:42
So this is a straight one–to–one copy, we're not doing any D civilization
这就是很简单的一对一拷贝，我们并没有做其他事情
08:43 – 08:44
All right, we can ignore compression for now
我们现在可以将压缩放在一边，暂时无视
8.44–8.49
but whatever however it's represented on disk is exactly how it'll be represented in memory
但它在磁盘中是如何表示的，那么它在内存中也是一样如此

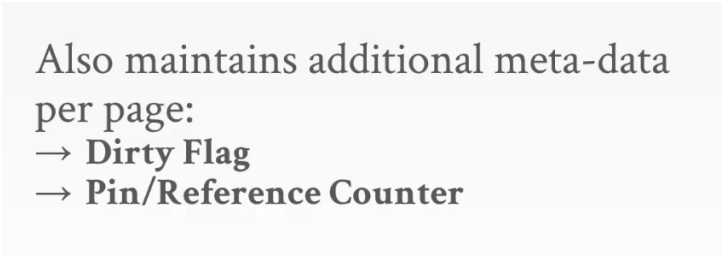
08:49 – 08:50
We're not doing any marshaling of the data
我们不会对数据做任何封装序列化处理
08:50 – 08:53
We just take it from the disk and put it directly in the memory
我们只是将它从磁盘中取出，然后直接放入内存



08:53 – 08:57
All right,we keep doing this row all the other pages that ~~that that~~ we may need
All right，我们会对我们可能需要的其他page也进行这种操作
08:58 – 09:01
Right, so the in my earlier example
So，在我之前的例子中
9.01–9.04
when I showed how the execution ender says,hey I want a page 2
当我展示执行时，我表示我想要page 2
09:05 – 09:09 ！ ！ ！ ！ ！ ！
Right, it manually you know buffer pool manager magic figured out what page two is
Buffer池会很神奇地告诉我们page 2是什么
09:10 – 09:14
So if we're just organizing these things as frames
So，如果我们将这些东西整理为frame
09:16 – 09:20
Pages can go in any order in the frames that they want,right
Page可以以它们想要的任何顺序放在frame中
09:20 – 09:28
In this case here even though it's page one page 1, 2, 3 ,in my Buffer Pool its page 1,3, that's not in the same order that it's out on disk
在这个例子中，在磁盘上我们有page1、2和3，但在我的Buffer池中的是page1和3，Buffer池中page的顺序和磁盘上的顺序并不一致
09:28 – 09:34
So we need an extra indirection layer above this to figure out ,if I want a particular page what frame has the one I want
So，在此之上，我们需要一个额外的indirection层，如果我想要某个特定的page，通过这个indirection层我就知道它在哪个frame中了
09:34 – 09:38
Because it's not going to match exactly the same order that it is on disk
因为它并不会完全匹配磁盘上的排列顺序



09:38 – 09:40
So this is the page table was
So，这就是page表
09:40 – 09:45
Page table is just a hash table, that's going to keep track of what pages we have in memory
Page表其实就是一个hash表，它用来跟踪我们在内存中有哪些page
09:45 – 09:50
And if you ask for a particular page ID ,it'll tell you what **frame** that is located in ,all right
如果我们想找一个特定的page，通过page表和page id，我们就可以知道这个page在哪个frame中
09:52 – 10:02
And so the database systems can have to maintain some additional metadata to keep track of what's going on with the pages that it currently has in its Buffer Pool
So，数据库系统必须维护一些额外的元数据，以此来跟踪当前Buffer池中page的发生了什么



10:03 – 10:06
So the first thing we got to keep track of is called the **Dirty Flag**
So，首先我们要跟踪的东西被称为Dirty Flag
10:06 – 10:11
And this is just a flag single bit that tells us whether the page has been modified since it's been read from disk
这个flag其实就是用来告诉我们，当我们从磁盘中读取到这个page后，这个page是否被修改
10.11–10.16
did some query，some transaction make a change to it
是否有查询或者事务对它进行修改
10:17 – 10:21
The other thing that I keep track of also is called a pin count or a reference counter
我想追踪的另外一个东西叫做Pin count或者说引用计数
10:22 – 10:30
And this is just keeping track of the number of threads or queries that are currently running that want this page to remain in memory
~~它用来跟踪希望该page保留在内存中的当前运行线程数或者是查询的数量~~
它用来跟踪想要使用该page的当前线程数量或者是正在查询该page的数量
10.30–10.33
meaning we don't want it written out to disk

这意味着我们并不想将该page写出到磁盘上(知秋注：还在被强引用使用，Java程序员可这么理解)

10:33 – 10:35

Right, it could be because I'm gonna update it

因为我可能会去对它进行更新

10:35 – 10:38

So I do my fetch, I go fetch the page, I need bring into my Buffer Pool

So，我拿到page，我需要将它放入我的buffer池

10:38 – 10:41

Then I'm gonna go ahead and modify

然后我对这个page进行修改

10:41 – 10:47

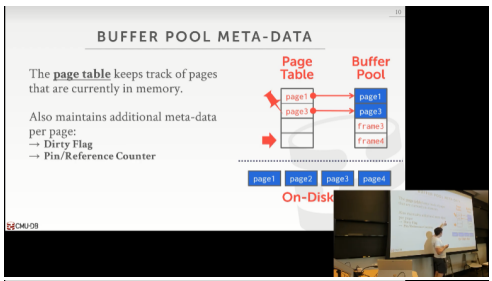
I don't want that page to get evicted or swapped out back at the disk in between the time it's been brought in and before I can actually do my update to it

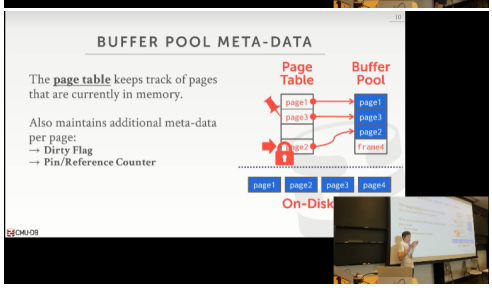
从该page被放入内存到我对它进行更新前这段时间内，我不想让该page被移除或者是交换回磁盘

10:48 – 10:54

There's also gonna prevent us from evicting pages that have not been safely written back to disk yet

这也将阻止我们去移除那些还未被安全写回磁盘的page





10:56 – 10.58

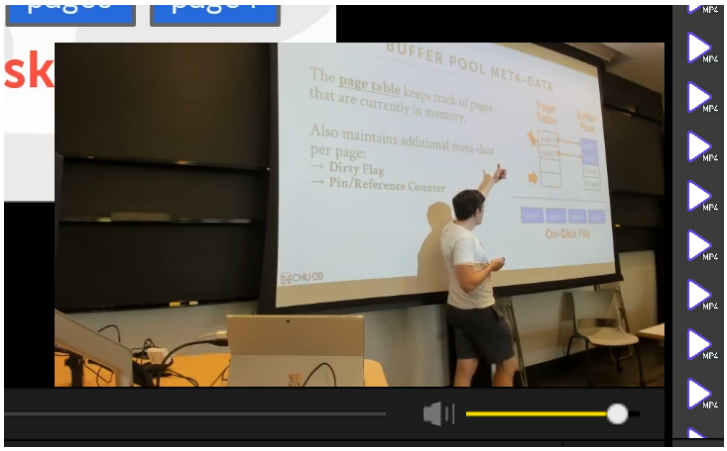
Alright, so again, so like I get pinned page and say

如图所示，我可以将该page固定住，并表示

10.58–11.03

I don't want this thing in to ever be removed from the Buffer Pool for now

当下，我不想让这个page从buffer池中移除掉



11:03 – 11:05

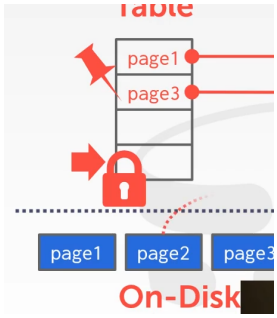
And then say I'm reading a page here

接着我表示，我正在从这里读取一个page

11:05 – 11:07

Sorry, I want to read a page that's not currently memory

抱歉，我想说的是，我想去读取一个不在当前内存中的page



11:08 – 11:11

I want to put a latch on this entry in the hash table

我想在hash表中这一项上加个latch(锁)

11:11 – 11:14

So that I can go fetch the page and then update the page what I point to it

这样我就可以去拿到这个page，然后更新这个我所指向的page

11:14 – 11:18

Right, and I have to do this, because multiple threads were running at the same time

我必须这么做，因为同一时间可能有多个线程在运行

11:18 – 11:21

I can't assume that I'm the only person, I'm looking at the page table

我无法保证只有我正在访问这个page表

11:21 – 11:25

So I want to rent somebody else from taking this this entry in my page table

So，我想将我的page表中的这一项也让别人来使用

11:25 – 11:28

And while I'm fetching the page that I need, they come and steal it from me and put something else in

当我正在接收那个我需要的page时，其他人会从我手中偷走这一项，并往里面放些其他东西

11:29 – 11:34

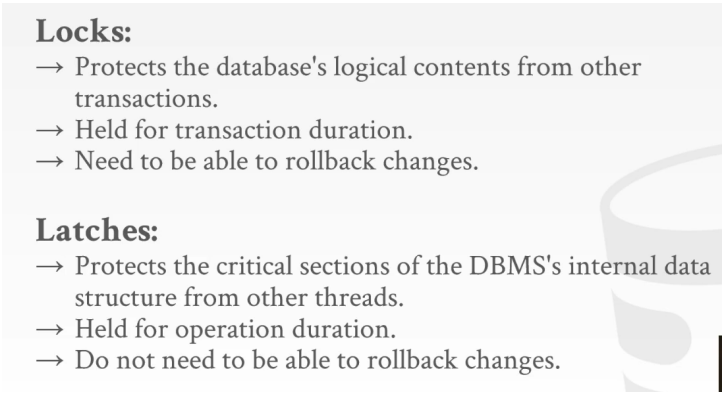
Alright, so again is theirs well see this as we go along later in the semester

Well，我们会在这学期内晚些时候看到这个

11:34 – 11:38

But there's a bunch of extra stuff we have to do to keep track of what pages have been modified

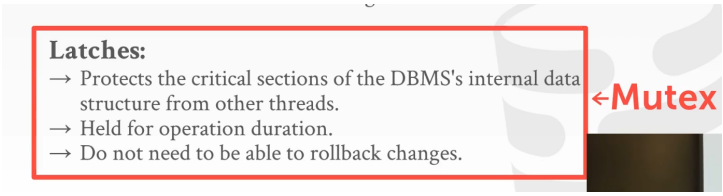
但我们必须做一些额外的事情，以此来跟踪哪些page被修改了
11:38 – 11:40
So the dirty bit is just sort of one piece of it
So，这个Dirty-Flag只是其中的一部分
11:40 – 11:44
We also need keep track of who actually made the modification
我们也需要去追踪是谁进行了这项修改
11:44 – 11:48
So because we want to write a log record to say ,here's the change that was made
So，我们想通过日志来记录做了哪些修改
11.48–11.51
we're gonna make sure that log records written first before our page is written
我们要去确保我们在先写完日志后，再去修改我们的page
11:51 – 11:54
This is another example,why mmap is a bad idea
这是另一个例子，它可以说明为什么mmap是一个糟糕的想法
11.54–11.59
because I can't guarantee the operating system is not gonna write my page out the disk before I want it to
~~因为我无法保证操作系统在我想让它这样做之前，不会将我的page写出到磁盘上~~
因为我无法保证操作系统在我想将page写到磁盘之前，不去做这件事情(知秋注：自己无法控制，OS可能会提前将page写出到磁盘)
11:59 – 12:02
Okay, that's it doesn't prevent you from doing that
Ok，它不会阻止你这样做
12:03 – 12:08
At least on FreeBSD can let you do this ,but windows and Linux don't much prevent this
至少在FreeBSD上，它允许你这样做，Windows和Linux上不会阻止这么做
12:09 – 12:11
Alright, so is this clear what we're trying to do here
So，你们应该明白我们此处试着要做的事情了
12:11 – 12:18
Right, basically managing our own memory ,but we're keeping track of how the transactions or queries are modifying the pages
基本上来讲，就是管理我们自己的内存，但我们也要去跟踪事务或查询是如何修改page的
12:18 – 12:27
And we have to protect ourselves and the page table to prevent anybody else ,and you know addicting things overwriting stuff, before ~~we're done with with~~ what we wanted need to do
在我们想做任何事之前，我们必须保护page表免受其他人污染或者是被其他人覆写里面的东西
12:29– 12:31
Any questions,okay
有任何问题吗？Ok，看起来没有



12:33 – 12:38
So I need to make a very important distinction now about the difference being **locks** and **latches**
关于Lock和Latch，我需要说下它们之间的非常明显的区别
12:39 – 12:43
So this will come up later on you have to do this for the first project as well
So，这会在之后出现，你们也会在你们第一个Project中遇到这个
12:43 – 12:45
If you're coming from an operating system background
如果你们之前有学过操作系统
12:46 – 12:50
in there word a lock is what we call a latch
以操作系统来看的话，所谓的lock，我们将之称为latch
12:50 – 12:57
So let me try both of them in the context of databases,and I'll see us to describe how they map into the OS world
So，我会试着以数据库的角度来解释它们两个，我会让你们看到它们是如何与操作系统对应起来的
12:58 – 13:03
So a lock in the database world is some higher-level logical primitive
So，在数据库的世界中，lock是某种更高级的逻辑原语
13.03–13.09
that's going to protect the contents of the database,the logical contents like a tuple,a table, a database
它会去保护数据库中的逻辑内容，例如：tuple，表以及数据库
13:11 – 13:16
All right, and the Transaction is gonna hold this lock for its duration and while it's running
事务会在运行的时候去持有这个lock
13.16–13.18
which means could be multiple queries
这就意味着可以有多个查询
13:18 –13:21
This could be you know multiple milliseconds or multiple seconds even
它们可能许多几毫秒或者甚至是几秒钟来完成
13.21–13.24
or even in minutes or hours if it's a really long running query
如果它真的是一个耗时很长的查询，那就可能得花几分钟或者几小时才完成
13:25 – 13:33

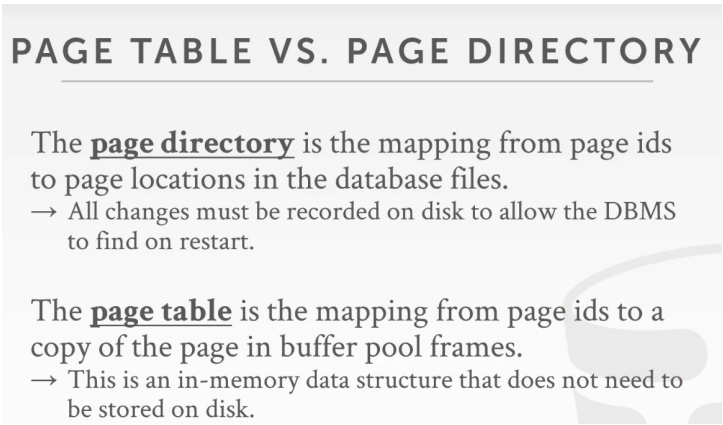
So in that word again this is something that database systems can provide to us and expose to you as like the application programmer
So，换句话讲，数据库系统可以为我们提供这些东西，它能够将这些暴露给我们这些应用程序开发人员
13:33 – 13:35
You even see what locks are being held for as you run queries
你甚至可以在你运行查询时看到持有的是什么lock
13:35 – 13:39
Latches are the low-level protection primitives
Latch是一种底层保护原语
13.39–13.43
that we use for the critical sections of the internals of the database systems
我们使用它来保护数据库系统内部的关键部分
13.43–13.48
like protecting data structure, protecting regions of memory
比如，保护数据结构和保护内存区域
13:50 – 13.54！！！
And so for these, these latches we're gonna hold for just the duration of the operation that we're making
So，在我们执行操作的期间内，我们会持有这些latch，用来保护某些东西
13.54–13.55
like if I go update my page table
比如说，如果我去更新我的page表
13.55–14.02
I take a latch on the entry on the location of that I'm gonna modify make the change and then I release the latch
我会在我要去修改的地方加上一个latch，修改完后，我会将它释放
14:04 – 14:08
All right and we don't need to worry about rolling back any changes in the same way we do for locks
~~All right，我们无须去担心回滚到修改应用前的情况，我们在使用Lock时，也是以相同的方式操作的~~
Alright，我们以同样的方式使用lock(知秋注：这里应该是latch)，我们无需去担心对改变进行的回滚操作
14:08 – 14:13
Because it's an internal thing or updating the physical data structure of the database system
因为它是一个内部的东西，它会去更新数据库系统的物理数据结构
14:13 – 14:16
I make the change and if I can't actually get the latch I want
在我进行修改时，如果我没能拿到我想要的Latch
14.16–14.19
then I did abort and don't worry about rolling back
那我就会终止操作，并且不需要担心回滚问题
14.19–14.20
yes
请问
14:20 – 14:22
Student:(提问)

14:22 –14:25
Okay so he says ,rolling back changes this will come later on we talk about concurrency control
Ok，他想问的是回滚修改这方面的问题，这会在我们之后讨论并发控制的时候提到
14:25 – 14:30
But basically say like I wanna take money out of my bank account and put it in your bank account
但基本上来讲，这就像是我从我的银行账户中取钱，然后将钱放入你的银行账户那样
14:30 – 14:34
So we take money out of my bank account ,but then the system crashes before I put the money in your account
So，当我从我的账号中取到钱后，在我要将钱打入你的账户时，系统崩溃了
14:34 – 14:36
I want to roll back the change I made to my account
我想将我的账号回滚到我对我的账号操作之前的样子
14.36–14.37
because I don't want to lose that money
因为我不想丢掉这些钱
14.37–14.39
that's not to me like that
我不想遇到这种情况
14:39 – 14:42
Right, this will discuss a whole lecture on concurrency control ,it's awesome trust me
So，我们会花一整节课来讨论并发控制，相信我，这块内容很棒
14:43 – 14:45
But for now the main thing we're focused on this thing here
但现在，我们主要关注的还是这块内容(Lock和Latch)



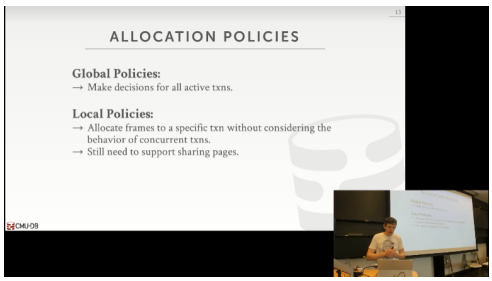
14:45 – 14:52
Right, so again in the operating system world this would allow should be something like a **Mutex**
So，在操作系统中，此处的Latch就像是它里面的Mutex
14:52 – 14:57
We're actually going to use **mutexes** in our database system to protect the critical sections of things
实际上，我们会在我们的数据库系统中使用mutex来保护其中的关键内容
14:57 – 15.00
So I will try to be very careful and always say latch when I mean latch
So，当我在讲Latch的时候就会使用Latch而不是其他词汇，我会谨慎措辞
15.00–15.04
but occasionally I slip up and we'll use lock
但偶尔我会翻车，我会将Latch讲成lock
15.04–15.08
but I said it's an internal thing we mean latch

但如果是内部的东西，那我们讲的lock就是latch
15:08 – 15:09
It’s also very confusing too
这也让人非常困惑
15.09–15.14
because the mutex implementation you would use to protect you for the latch is called a spin lock
因为我们要在latch中所使用的mutex实现被称为spin lock(自旋锁)
15:14 – 15:16
Alright, but it’s really you know this thing is not this thing
但你们要知道此物非彼物
15:17 – 15:19
ok all right
Ok



15:20 – 15:25
So the other think we want to make is the difference between the page directory and the page table
另一个我们要去区分的就是page目录和page表
15:25 – 15:32
So remember the page directory is what we’re going to use to figure out where to find pages in our files
So，要记住，page目录的作用是用来找到page在我们数据库文件中的位置
15:32 – 15:34
So we want page 123
So，假设我们想要page 123
15.34–15.40
it’ll tell us what file at what all offset or what’s ~~what~~ set of files have what we’re looking for
它会告诉我们我们要找的那个page在这些数据库文件中什么地方，page中所有offset值（即slot数组）有哪些
15:40 – 15:44
So all the changes we’re gonna make to the page directory have to be durable
So，我们对page目录做出的所有改变都必须持久化
15.44–15.45
they have to be written out the disk
它们必须被写到磁盘上
15.45–15.49
because if we crash, come back we want to know where to find the pages that we have

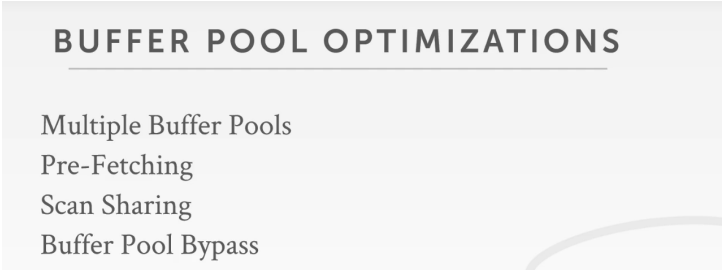
因为如果系统崩溃了，恢复后我们想要知道该在哪里可以找到我们拥有的page
15:50 – 15:53
The page table is an internal in memory map
page表则是内存中的内部映射
15.53–15.58
that just maps page IDs to where the frames of they are in the Buffer Pool
它将page id映射到它们在Buffer池中frame的位置
15:58 – 16:01
So this thing can be literally ephemeral
So，这个东西可以是暂时的
16.01–16.02
and we don't need to backup by disk
我们无须在磁盘上对它进行备份
16.02–16.07
because if we crash and come back our Buffer Pool is blown away anyway, so who cares
因为如果我们遇上系统崩溃，然后恢复后，我们的buffer池里的东西就灰飞烟灭了，So没人会去在意里面有什么
16:07 – 16:13
So this page directory has to be durable, the page table does not have to be
So，page目录必须持久化，但page表则无需这么做
16:13–16:18
And that means we just used whatever your favorite hash map or hash table implementation you want
这就意味着我们可以使用我们喜欢的hashmap或者我们想要的hashtable实现
16:18 – 16:22
Right for project 1 you're just used to be a std map that's fine
在Project 1中，你们也可以使用std::map，这没问题
16:23 – 16:26
Because again we don't have to worry about this thing being durable
因为我们无须去担心它是否要持久化
16:26 – 16:30
We have to make sure it's thread safe certainly, but not durable
我们必须确保它是线程安全的，它不需要是持久化的



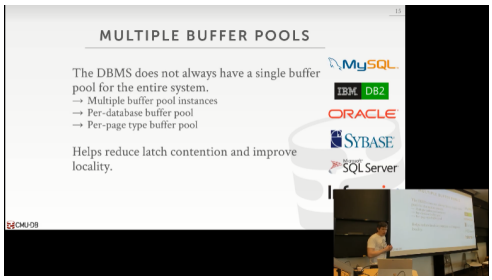
##
##
16:31 – 16:37
All right ,so now when we start talking about how we want to allocate memory in our database for the Buffer Pool
So，现在我们要开始讨论该如何为我们数据库中的Buffer池分配内存了
16:37 – 16:40
We will we start to think about this in two different ways
我们可以以两种不同的方式来思考这个问题

16:40 –16:44
So the first is that we can choose what I call a sort of global policies
我们可以选择的第一种方式，我将它称为全局策略(Global Policies)
16.44–16.51
where we're trying to make decisions that benefit the entire workload that we're trying to execute
即我们所试着做出的决策能够使整个我们要试着执行的worklaod都受益
16:51 – 16:55
We look at all the queries, all the transactions that are going on in the system
我们会去查看所有运行在该系统上的查询和事务
16:55 – 17:01
We try to say at this point in time what's the right thing I should do for choosing what should be in memory versus not memory
我们试图在此讲出我应该做的正确的事情，以选择该内容是否应该存储在内存中
17:02 – 17:09
An alternative is to use a local policy,we're on for each single query or each single transaction we're running
另一种方案就是使用局部策略，即针对每个单个查询或者单个事务来进行
17:10 – 17:14
We try to say what the best thing to do to make my one query,one transaction go faster
我们尝试讲出可以让我的一个查询，一个事务进行得更快的最佳方法
17.14–17.20
even though for the global system that actually, might be a bad a bad choice
即使对于整个系统而言，这实际上可能是个糟糕的想法
17:21 – 17:25
So the there's no one way that's better than another
So，我并没有说这种策略要比另一种策略要来得更好
17.25–17.29
obviously there's optimization you can do ,if you have a global view versus a local view
很明显，如果你知道全局的样子或者局部的样子，你就可以进行优化
17:28 – 17:33
But then for each integer query, you might be more tailored to what they want to do to make that run fast
但是，对于每个整数查询，你可能会去选择更适合它们的优化，以使其运行起来更快
17:33 – 17:37
So as we've seen a much of these examples as we go along for optimizations
So，正如我们在讲解优化这块内容时所看到的很多例子一样
17:38 – 17:41
The most systems will probably try to do accommodation of the two of them
大多数系统可能会试着尽量同时使用着两种优化
17:42 –17:45
What you'll be implementing for the first project is considered a global policy
我觉得你们在实现第一个Project时使用全局策略比较好
17.45–17.49
because there's just looking at ,you know what's the least recently used page and removing that
因为它只需要找到最近最少使用的page，将它移除即可
17.49–17.52
even though that make me bad for one particular query

即便它对于某个特定查询来说会变得很糟糕



17:54 – 17:58
All right ,so that basically all you really need to know about how to build a Buffer Pool
So，简单来讲，你们这群人真的需要去了解如何去构建一个Buffer池
17:59 – 18:02
Right it's just you have a page table that map's page IDs to frames
你需要有一个page表，它将page id映射到frame处
18.02–18.09
and then you look in the offset in the allocated memory, and that tells you here's the page that you were looking for
接着你根据我们所分配内存中的offset值，它会告诉你我们所查找的page的位置
18:09 – 18:11
It seems pretty simple right
这看起来真的很简单，对吧
18:12 – 18:21
So now we want to talk about how to actually make this thing be super awesome or super tailored for the application that we're trying to run or the work over trying to run inside of our database system
So，现在我们要讨论的是，如何针对我们要运行的应用程序真正使它变得超赞或超量身定制，或者如何使其在数据库系统内部运行
18:21 –18:25
And this is gonna allow us to do certain things that the operating system can't do
这就允许我们去做一些操作系统没办法做的事情
18.25–18.27
because it doesn't know anything about what kind of queries you're running
因为操作系统不知道我们要运行的查询是哪一种
18:27 – 18:30
It doesn't know what data they're touching,what are they're gonna touch next
它不知道它要去接触哪些数据，也不知道接下来要接触什么
18:30 – 18:36
All right ,so now we can talk about what we can do to make this thing do better than what sort of a naive scheme would do
So，现在我们要来讨论，我们该如何做才能比幼稚的scheme来的更好
18:36 – 18:42
So talk about how to handle multiple Buffer Pools ,prefetching the scan sharing and then the last one be Buffer Pool bypass
So，我们会去讨论如何处理多Buffer池、Pre–fetching(预取)、扫描共享以及Buffer Pool bypass



18:43 – 18:49
Okay, so in my example that I showed, I referred to the Buffer Pool as a single entity
Ok，So在我之前展示的例子中，我将Buffer池当做一个单个实体
18:50 – 18:51
And the database system has one Buffer Pool
数据库系统有一个Buffer池
18:52 – 18:55
In actuality you can have multiple Buffer Pools
实际上，我们可以有多个Buffer池
18:55 – 18:58
So you have multiple regions of memory you've allocated
So，我们可以分配多块内存区域
18:58 – 18.59
They each have their own page table
每个区域都有它们自己的page表
18.59–19.04
they each have their own been mapping to from page IDs to frame IDs or **frames**
它们每一个都有自己的一套page id和frame的映射关系
19:05 – 19:06
All right

19.06–19.08
and the reason why you want to do this is
我们想这么做的原因是
19.08–19.13
now, you can have for each Buffer Pool,you can actually have local policy for that Buffer Pool
实际上我们可以在每个Buffer池上使用局部策略
19.13–19.16
that's tailored for whatever is the data that you're putting into it
这样可以为你所放入的数据进行量身定制
19:16 – 19:22
You know so for Example, I could have a a single Buffer Pool for each table
例如，我可以让每一个表都有一个Buffer池
19:22 – 19:25
Because maybe some tables I'm doing a bunch of sequential scans
因为可能在有些表中我要进行一系列循序扫描
19.25–19.29！！！！！！！
and some tables I'm doing pointer queries or I'm jumping to single pages at a time
在某些表中我会进行指针查询（知秋注：其实就是索引查询），或者每次我要跳转到某个单个page上（知秋注：表中查询）

19:29 – 19:34
And I can have different cashman policies or different placement policies to decide based on the two workload types
我可以根据这两种workload类型来决定使用不同的替换策略
19:35 – 19:38
But I can't do that easily if it's a giant just a giant a Buffer Pool
但如果它是一个非常巨大的Buffer池，那我就没法轻易地这么做了
19:39 –19:42
Well, let's say I have it I can have a Buffer Pool for an index and Buffer Pool for tables
Well，假设我可以让一个Buffer池来处理索引，另一个Buffer池用来处理表
19:42 – 19:44
And then they have different access patterns
它们有不同的访问模式
19:44 – 19:46
And then I can have different policies for each of those
那么我就可以针对它们每个使用不同的策略
19:46 – 19:55
The other big advantage you also get is that it's gonna end up reducing latch contention for the different threads that are trying to access it
我们还可以获得另一个很大的优点，这样做将最终减少那些试图访问Buffer池的不同线程间争抢Latch的情况发生
19:55 – 19.57
Right, so when I do that look up in the page table
So，当我在page表中进行查找时
19.57–20.00
I have to take a latch on the entry that I'm looking at
我必须在我所查看的那一项上面加一个latch
20:00 – 20:02
As I go find the frame that has the data that I want
当我找到那个存放着我想要的数据的frame时
20.02–20.04
and I'll make sure that nobody else swaps that out
我要确保没人会将它交换出去
20:04 – 20:09
But you know from the time ,I do the lookup from the time ,I go get the page that I want
即从我查找它，到我拿到我想要的那个page时，没人去动它
20:09 – 20:14
And so that means that I could have a bunch of threads call contending on the same latch
这就意味着我会遇上一堆线程争抢同一个latch的情况
20.14–20.16
that could they're all accessing the same page table
它们会访问同一个page表
20:17 – 20:19
So no matter how many cores, I have on my brand–new machine
So，不管我全新的电脑上有多少个Core
20:19 – 20:21
I'm not getting good scalability
我在可扩展性这块并没有得到好的提升
20.21–20.25

because everything's contended on these critical sections
因为在这些关键部分，任何东西都会被争抢
20:26 – 20:28
But now if I just have multiple page tables
但现在如果我有多个page表
20.28–20.34
each thread ,you know they could be accessing different page tables at the same time
每条线程就能在同一时间访问不同的page表
20:34 – 20:36
And therefore they're not contending on those latches
因此，它们就不会去争抢这些latch
20.36–20.38
and now I get better scalability
这样，我就得到了更好的可扩展性
20:38 – 20:40
Now still could be still bottlenecks on the disk feed
现在，磁盘可能依然是性能瓶颈
20.40–20.41
which is always a big problem
它一直是个大问题
20.41–20.46
at least internally now, I'm not worried about them you know trying to all acquire the same latch
至少现在，我不会去担心这些线程都去获取同一个latch