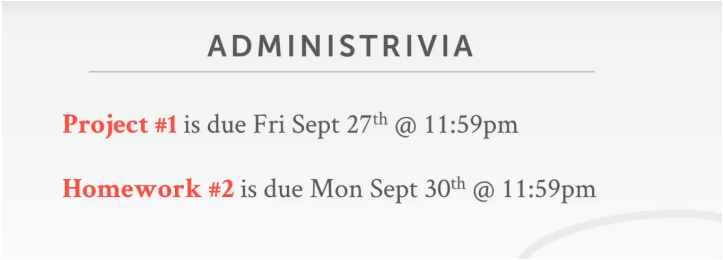
06-01

06 Hash Tables

(CMU Databases Systems _ Fall 2019)

00:16 – 00:17
Alright, let's get started
好了，让我们开始吧
00:18 – 00:21
Okay, let's uh again give it up for DJ Drop Tables
Ok，让我们把时间交给我们的DJ Drop Tables
00:22 – 00:24
Thanks boys, how was your weekend
你周末过得怎样
00:25 – 00:29
Good you know I was recruiting what TLC, but I don`t have a DJ jobs
还行吧，我最近被TLC雇佣了，但我干的并不是DJ的工作
00:29 – 00:30
Don`t have a DJ jobs
没能拿到DJ工作么
0.30-0.31
that's hard right
这太难了，对吧
0.31-0.36
actually I found out Salesforce has in the lobby, in the main building San Francisco
实际上，我知道SalesForce在旧金山的主楼的大堂里面
00:36 – 00:38
They have a DJ every morning
每天都会有一个DJ在那
0.38-0.39
and it rotates
打碟
00:39 – 00:42
oh do you want to figure out, how to get you that job or what
你想知道该怎样拿到那份工作不?
00:43 – 00:44
okay yeah I guess you gonna do
Ok，我想你可以的
00:45 – 00:49
okay so I think most people are at the TOC today
Ok，我觉得今天大部分人都在TOC

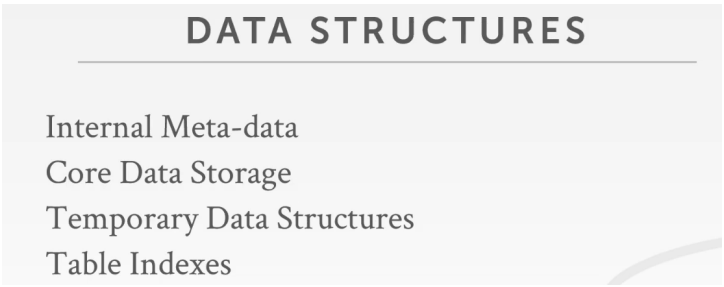
0.51–0.54
Because this is one of my favorite lectures **hash tables** right
因为hash table是我最喜欢讲的一节课
00:54 – 00:57
So we have a lot to discuss, let's get right through it
随着我们深入，我们会去讨论大量的内容



00:57 – 01:06
So real quickly and then reminders for what's on the docket for you guys what's due ,project #1 is due next week on Fri at 27th at midnight
So，快速提醒你们一下，Project 1 你们要在下周五午夜12点前交出来
01:06 – 01:14
And then homework #2 which will be releasing later today, that'll be due the 30th the Monday after the project
然后，Homework 2我们今天会放出来，你们要在30号那天交出来
01:15 – 01:17
So any quick high-level questions about project #1
So，对于Project 1，你们有任何逼格比较高的问题吗?
01:21 – 01:22
say again to what
麻烦再说一遍
01:23 – 01:25
The question is when with the autograder be released
So，他的问题是自动打分器什么时候放出来
1.25–1.28
I mean so it's live on great scope now,you can submit things today
它现在已经在GreatScope上线了，你今天就可以往上面提交东西
01:30 – 01:33
But we're not giving you the source code for the test, obviously,because that's means that the grade
但我们不会将测试源码也交给你们，因为这事关分数问题
01:34 – 01:35
Yes sir it should be live
我觉得它现在应该上线了
01:35 – 01:39
if you submit it doesn't work please post on Piazza
如果你提交了你的项目进行测试，但测试不奏效的话，请到Piazza上发帖
01:40 – 01:42
Okay,and the other high level questions
Ok，还有其他高逼格的问题吗?
01:45 – 01:55
Okay, so we're at now for the course is that we've spent the first couple weeks ,again starting at the bottom of the stack of a database system architecture and walking our

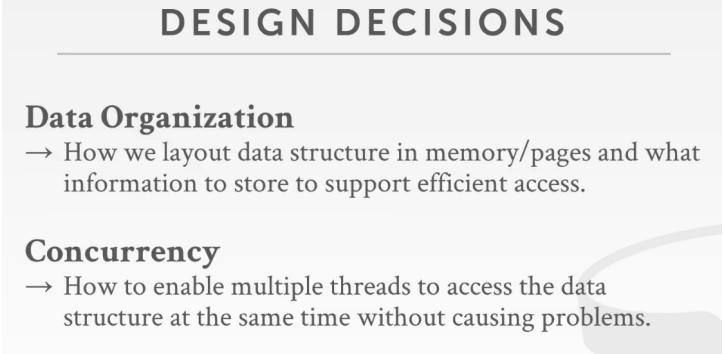
way up
Ok，目前为止我们已经花了两周时间从数据库系统架构的底层讲起，并逐渐向上进行探索
01:55 – 01:59
So we've discussed how to store data on disk the pages on disk
So，我们已经讨论了该如何往磁盘上存储数据，以及如何在磁盘上表示page
01:59 – 02:03
Then we talked about how to bring those pages into memory and our buffer pool or buffer cache
接着，我们讨论了该如何将这些page放入内存以及buffer pool或者buffer cache
02:04 – 02:09
And having a policy decide when it's time to evict something and how to pin things when I do writes
并且通过一种策略来决定何时将page从buffer pool中移除，以及当我们在做写操作时，该如何锁住东西
02:10 – 02:14
So now we're going above the buffer pool manager and we start talking about access methods
So，现在我们要来讨论下buffer pool管理器之上的东西，即Access methods
02:14 – 02:24
So these an access method is a way we're going to you know get essentially read or write the data that in our database that's stored in the pages that are stored out on disk
本质上来讲，Access method 是一种我们用来对数据库数据进行读或写的方式，数据库是存放在存储在磁盘上的page中的
02:25 – 02:36
So today we're talk about ,today we begin of a of a set of lectures, we're going to do on data structures that we're gonna maintain internally inside the database system
So，从今天开始，在接下来的几堂课上，我们会去讨论数据库系统内部所维护的数据结构
02:36 – 02:41
And we're gonna split it up between two discussions between hash tables and and order preserving trees
我们会将数据结构拆开来讲，一节课讨论hash table，另一节课讨论order-preserving tree
02:42 -02:44
So each of them have different trade-offs
So，它们每个在设计上都有所取舍
2.44-2.46
if you because you've taken an algorithms course by now
因为如果你们上过算法课
02:46 – 02:48
So you understand the implications for both of these
那么你们就明白了它们两者之间的含义了
02:48 – 02:52 ****
But we're gonna describe what matters to us in the context of database systems
但是我们会基于数据库系统的背景下描述对我们来说重要的事情
02:53 – 03:00
Because just because you have a tree versus a hash table maybe understand how to do proofs on it ,or write algorithms to interact with it
因为假设如果你们有一个tree和hash table，你们可能知道该如何去证明它们是tree还是hash table，或者也明白该如何通过算法和它们进行交互
03:00 – 03:04

Now let's talk about what happens when we actually put it inside a database system and actually try to use it
现在，我们来讨论下当我们将这些数据结构放入数据库系统中并试着使用时，会发生什么呢?



03:05 – 03:12
So data structures are used all throughout the database management System,right for a variety of purposes
出于许多不同的目的，在数据库管理系统中，数据结构无处不在
03:12 – 03:20
So one thing we've talked about so far we turn how to use data structures for maintaining the internal metadata about what's in our database
So，目前为止我们已经讨论过一件事了，那就是我们该如何使用数据结构来维护我们数据库中的内部元数据
03:20 – 03:23
When we talk about there being a page table or a page directory,
当我们在讨论page表或者page目录时
3.23–3.33
and that was a hash table to do lookups between a hash id or ,sorry a page to a page ID to a frame, or a page ID to some location on disk
其实它们就是一个hash table，我们通过传入一个page id，能找到对应的frame，或者是传入一个page id，能找到磁盘上所对应的位置
03:35 – 03:39
The next thing we can use them for is actually just the core data storage of the database itself
~~接下来，我们能使用到这些数据结构的地方就是数据库自身的核心数据存储~~
接下来，我们可以将它们用于数据库自身的核心数据的存储
03:40 –03:43
So what I mean by that is instead of as having on order to heap a bunch of pages
So，我的意思是，这样我们就无须去维护一堆page的顺序了
03:44 – 03:50
We can actually organize them at a higher level to be a hash table or a B+ tree or tree data structure
实际上，我们可以站在一个高级层面，将它们组织为一个hash table或者是B+ tree或者其他树形结构
3.50–3.55
and have the the values in the data structure actually be tuples
在这些数据结构中的值实际上就是tuple
03:55 – 03.57
So this is very common in a lot of systems
So，这在许多系统中都很常见
3.57–4.01
like like memcache for example essentially is a giant hash table

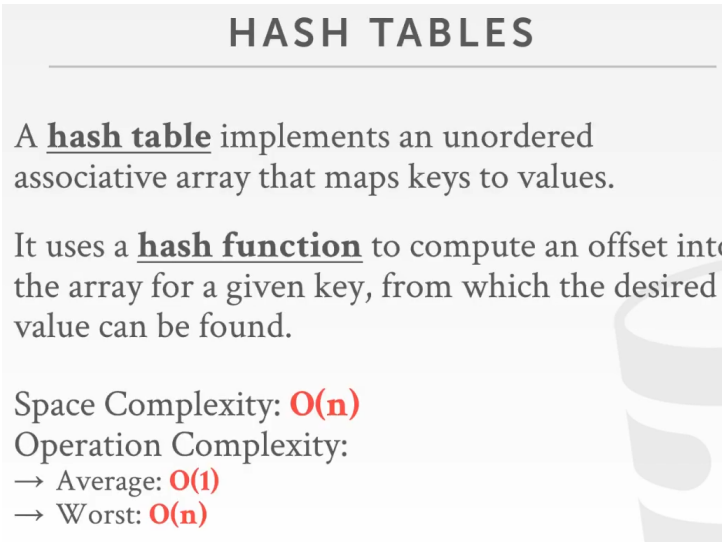
比如，memcache本质上来讲就是一个超大的hash table
04:01 – 04:04
Or MySQL innodb engine is just a B+ tree
或者，MySQL的innodb引擎使用的也是B+ tree
4.04–4.09
where they store the tuples themselves inside of the side of the leaf nodes of the tree
它们将tuple存储在B+ tree的叶子节点上
04:09 – 04:12
We can also use data structures to maintain temporary data
我们也可以使用数据结构来维护临时数据（知秋注：临时数据大家想成缓存就ok了，当然需要一个数据结构来对这些数据进行存储管理）
04:12 – 04:14
So this would be like if we're running a query
如果我们执行一个查询
4.14–4.17
and we need to compute something very efficiently
我们需要去高效地计算某些东西
04:17 – 04:19
We could build a data structure on the fly
我们可以在运行时构建一个数据结构
4.19–4.21
populated with whatever data we need
放入我们所需的任何数据
4.21–4.23
,finished executing the query,
完成执行查询
4.23–4.26
and then this throw away that data structure and be done with it
然后将这个数据结构丢弃，它的使命就已经完成了
04:26 – 04:31
And the last one that's why you you're most familiar with is using these data structures for table indexes
我们所要讲的最后一个，同时也是你们最为熟悉的那就是，把这些数据结构用在表索引（table index）上
04:32 – 04:36
Right, essentially building a glossary over keys inside of our tuples
本质上来讲，就是使用我们tuple中的key来构建一个词汇表（想成我们书中的目录就对了）
04:36 – 04:40
And allows how to do you know quick lookups to find individual elements that we want
这样允许我们快速找到我们想要的单个元素
4.40–4.43
rather than having to do a sequential scan throughout the entire database
这样就无须使用循序扫描整个数据库来找到我们想要的数据了
04:43 – 04:47
So for all these purpose again you need good data structures to do all these things
So，出于这些目的，我们就需要良好的数据结构来做些事情



04:49 – 04:50
So the things we want to care about
So，我们想去关心的事情是
4.50–4.52
how we design our data structures
那就是我们该如何设计我们的数据结构
4.524.54
is the two following things
它需要满足以下两点
04:55 – 04.57
So the first is we care about what the data organization is
So，我们所关心的第一件事就是数据结构是怎样的
4.57–5.06
we need how are we gonna represent the key value pairs are the elements of the data that we're storing in either in memory or on pages that were storing on disk
我们需要知道，我们该如何对我们存放在内存或者是磁盘中page上的这些（key/value键值对）元素进行表示
~~表示key/value pair这些数据元素，我们将它们存放在内存或者是磁盘中的page上~~
05:06 – 05:08
And we do this in an efficient way
我们要以一种高效的方式做到这点
5.08–5.18
that can support fast reads and writes without having to your major overhaul ,or maybe restructuring of the entire data structure every single time
~~在无须对数据结构进行大改或者是每次要重构整个数据结构的情况下，它支持快速读写~~
即在无须对数据结构进行大改或者是每次要重新转换整个数据结构的情况下，支持快速读写（知秋注：Java程序员都知道，我们在面向对象编程时，每次都会将表中一行数据转换为一个数据结构POJO）
05:19 – 05:25
The second issue is that how we're going to allow multiple threads to access our data structure, or multiple queries access to the our data structure
第二个问题就是，我们该如何让多个线程或多个查询去访问我们的数据结构
05:26 – 05:31
At the same time without causing any physical violations to the internal representation of the data
~~与此同时，它们不会对数据的内在表示造成任何物理上的问题~~
与此同时，在此环境下，该数据结构表示的数据并不会在物理存储层面出现问题（知秋注：多线程下内存中数据访问修改可能出现的问题，对多线程编程熟悉的都懂）
05:32 – 05:32
So what I mean by that is
我这么说的意思是

5.32–5.38
we don’t want to have maybe one thread update a memory address while another thread it’s reading that address
我们不希望这种情况发生，即当一个线程正在读取该内存地址上的数据，然而另一个线程对这个内存地址上的数据进行更新
05:39 – 05:42
And then they see some torn white or some corrupt version of that address,
然后，它们就会看到该地址上的数据是有问题的（脏数据）
05:42 – 05:46
And now that points to some invalid page or some invalid memory location
这会指向某些无效的page或者是某些无效的内存位置
5.46–5.49
where we end up producing incorrect results
这会导致我们最终生成错误的结果
05:49 – 05:54
So we’ll see how we actually handle this we’ll talk a little bit long as we go along today
So，今天我们会去花点时间来讨论实际该如何处理这些东西
05:54 – 06:01
But we’ll spend a whole lecture while discussing how to do concurrency control inside of indexes inside of these data structures
但我们之后会花一整节课时间来讨论该如何在这些数据结构内部进行并发控制
06:01 – 06:05
But for our purposes today but you sort of simplify the discussion just assume we only have a single thread
但出于我们今天的目的，我们会讨论的稍微简单些，我们假设我们只有一个线程
06:06 – 06:08
And because this is going to matter later on also to when we talk about transactions
因为这对于我们之后要讨论的事务非常重要
06:08 – 06:14
Because the type of things we’ll talk about here we’ll use latches to protect the physical data structure
因为我们能使用此处所讨论的latch来保护物理数据结构
6.14–6.19
that prevents from again reading invalid memory addresses or invalid page locations
它能防止我们去读取某些无效的内存地址或是无效的page位置
06:19 – 06:23 ******
There’s also a higher level concept of what’s the logical correctness of our data structure
关于数据结构的逻辑正确性还有一个更高层次的概念
6.23–6.27
that we need to care about as well and that’ll come later on in the semester
我们也需要去关心这点，我们会在这学期稍后一段时间去讨论它
06:27 – 06:31
So it says what I mean by that is to say, I have an index, I delete a key
我的意思是说，假设我们一个索引，然后我删掉了一个key
06:31 – 06:34
If I come back， my thread comes back and tries to retrieve that key again
如果我的线程回过头来试着查找这个key
06:35 – 06:36
I shouldn’t get it, because I know it’s been deleted

我应该拿不到这个key，因为我知道它已经被删除了
06:37 – 06:39
Even though the physical bit still may be there,
即使该key的physical bit依然在那个位置上
6.39–6.43
because I I'll do some background garbage to clean up later on
因为我之后会在后台进行垃圾回收
06:43 – 06:46
But logically my key should be gone even though physically it's not
但从逻辑上来讲，这个key应该GG了，即使从物理上讲它还在那里
06:46 – 06:48
So that this topic is very complicated
So，这个课题其实非常难
06:48 – 06:50
And so we'll touch on a little bit today,
So，我们今天会稍微接触下
6.50–6.57
but mostly care about the physical integrity of the data structure rather than the logical one
但我们最关心的还是数据结构的物理完整性，而不是逻辑上的

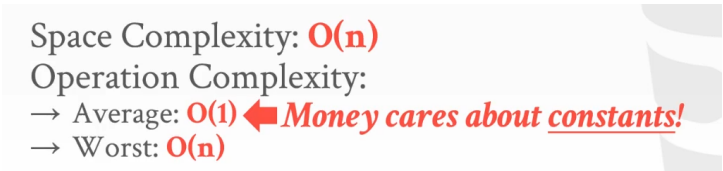


06:57 – 07:01
Okay, today again we're going to focus on hash tables
Ok，今天我们要讲的重点是hash table
07:02 – 07:11
So a hash table is a abstract data type that we're going to use to provide a unordered associative array implementation API
So，hash table是一个抽象数据类型，我们通过它来提供无序的关联数组实现API

07:12 – 07:16
And all that means that we're able to map arbitrary keys, to arbitrary values
这就意味着，我们能够将任意的key映射到对应的值上面
07:17 – 07:20
All right, there's no ordering to this thing like we're gonna see in trees
在hash table中并没有顺序这个说法，但我们会在树形数据结构中看到这点
07:21 – 07:25

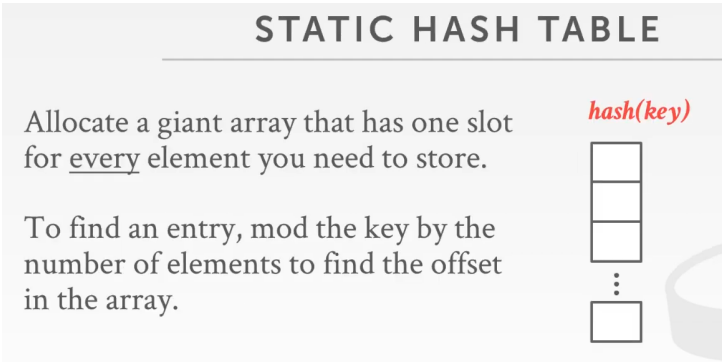
And so the way we're gonna be able to do this these fast look ups to find elements that we want
So，我们能够在hash table中快速查找我们想要的元素
07:25 – 07:29
It`s that we're gonna use a hash function, that's gonna take in our key
我们会使用一个hash函数，并将我们的key作为参数传入
07:29 – 07:34
And then compute some offset in some way to some location in my array
然后以某种方式计算出数组中某个位置的offset值
07:35 – 07:39
And that's gonna tell me either exactly the element looking for
这就会告诉我，我所要找的元素的具体位置在哪
7.39–7.45
,or I can roughly look around close to by where I land after I use my hash function to find the thing that I'm looking for
或者当我使用我的hash函数找到我所要找的数据后，我也可以大致看下周围的东西
07:45 –07:48
So the hash function isn't always gonna get us exactly where we want
So，hash函数并不会一直让我们精确地找到我们想要的东西（知秋注：因为会产生hash碰撞）
07:48 – 07:50
But at least get us in the right location
但至少它能让我们找到正确的位置
7.50–7.53
and we know how to then look around to find the thing that we are looking for
然后我们知道该如何在周围找到我们想要的那个数据
07:54 – 07:57
So again so this none of this should be new you should all take an algorithms class
So，这些东西并不是什么新知识，你们应该在算法课上都学过了
07:58 – 08:03
So the space complexity in the worst case of a hash table is is Big O(n)
So，hash table中空间复杂度最糟糕的情况是O(n)
8.03–8.06
that means that we for every single key we want to store
这就意味着对于每个我们想去存储的key
08:06 – 08:10
We least have one one entry for it in our hash table
在我们的hash table中至少有一个与它对应的entry
08:10 – 08:12
So that allocate that amount of memory amount of space
So，我们会给它分配一定量的内存空间
08:13 – 08:15
The operational complexity is interesting
接着，我们所感兴趣的是操作上的复杂度
8.15–8.18
because on average we're gonna get o one lookups,
因为从平均上而言，我们查找的复杂度是O(1)
8.18–8.24
meaning we in one step in constant time you can find exactly the thing that we're looking for

这意味着在常数时间内，我们只需一步就能准确地找到我们想要的那个数据
08:24 – 08:28
Worst case scenario and we'll see why this happens when we in a few seconds
最糟糕的情况指的是，我们需要花一定时间才能找到我们想要的那个数据
08:29 – 08:31
The worst case scenario will get Big O(n)
最糟糕情况下的操作复杂度就是O(n)
8.31–8.38
meaning we'll have to do a sequential scan or a linear search to find to look every single possible key to find that the key that we're looking for
这意味着我们必须进行循序查找或者是线性搜索来对每个可能的key进行查找，以此找到我们所查找的key（知秋注：key通过hash都碰撞到一起，放在一个数组或者队列或者链表中了，遍历查找想要的那个key）
08:39 – 08:40
So you may be thinking alright this this is great
So，你们可能觉得这会很棒
08:41 – 08:46
Any hash function or any hash table will do ,because I'm always gonna get O(1) for the most part
任何hash函数或是hash table都会做到这点，因为大部分情况下，复杂度都是O(1)



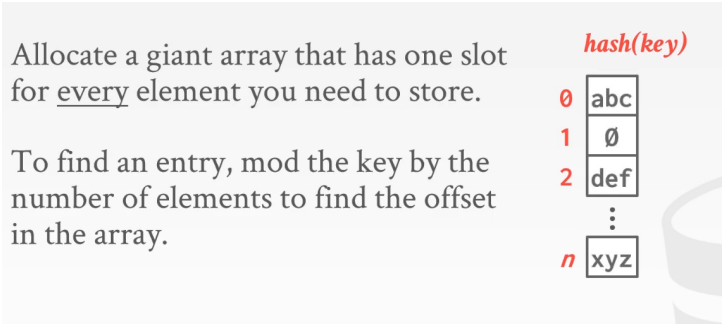
08:46 – 08:53
In practice even though this is super fast, in the real world where money's involved constant factors actually matter a lot
实践表明，尽管它是非常之快的，但有句老话讲得好，时间就是金钱
08:53 – 08:56
And so we'll see this when we just look at functions
So，当我们在看这些函数时，我们会看到这个
08:56 – 09:00
Right, hash functions what we, you know sometimes it'll be it would still be super fast
hash函数有时依然会超级快
09:00 – 09:04
But there'll be some hash function, that'll be twice as fast or three times as fast as other hash functions
但某些hash函数要比其他hash函数快2到3倍
09:05 –09:07
So you may say all right for one hashing who cares
So，你们可能会这样说，一个hash函数而已，谁会在意
09:07 – 09:09
But if now I'm hashing a billion things
但现在如果我对10亿数据进行hash处理
9.09–9.13
and my crappy hash function takes a second slower than the fastest one
对于一条数据来讲，我的垃圾hash函数所花的时间要比最快的那个慢1秒
09:13 –09:16
Now that's I'm spending a billion seconds to do this lookup

现在，我就需要多花10亿秒去进行查询了
09:16 – 09:21
So when there's real money involved when we're looking at large-scale the constant factors actually matter
当我们在面对这种超大规模基数的时候，我们要付出大量的金钱（知秋注：比如商业交易，每一秒都是大量的金钱消耗，这里原文中的constant factor如果为1，也就是一条查询多浪费1秒，那么一个亿*1那就真的有问题了）
09:21 – 09:26
We take your algorithms class there's like, O(1) we don't care about anything else the constants don't matter
我们通过 O(1)这种级别的算法来使得我们无须担心
9.26-9.28
in our world it does
在我们数据库的世界中，这很重要
09:29 – 09:32
All right, so let's look at the most simplest hash table you could ever build
So，我们来看下你们所能构建的最简单的hash table是怎样的吧



09:33 – 09:38
Right, and all it is is just a a giant array which is now like a big chunk of memory
它其实就是一个巨大的数组，它看起来就像是一大块内存
09:38 – 09:45
And then we're gonna say that every single offset in our array corresponds to a given element
接着，我们会说我们数组中每个offset位置都对应了一个给定的元素
09:46 – 09:51
And so for this to work we're gonna assume that we know exactly the number of keys are gonna have ahead of time
So，为了让这个能正常使用，我们会假设我们提前已经知道了key的具体数量
09:51 – 09:56
And we know exactly what there with the reduced we shouldn't their values are with what their actual values are
我们也知道这些key所对应的实际值是什么
09:57 – 10:00
Right, so now to find any key in my hash table
So，现在，我要在我的hash table中找某一个key
10:01 – 10:06
I just take a hash on the key mod it by the number of elements that I have
我通过对key进行hash，即我将该key与所有的元素数量进行取模操作（知秋注：如果该key对应这数组的下标）
10:07 – 10:09

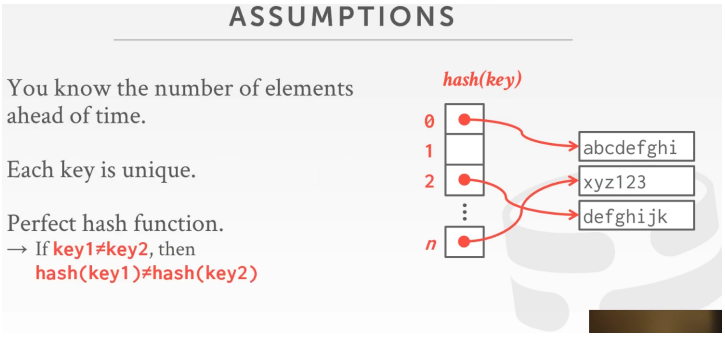
And then that's going to get me to some offset,
然后，我就会得到它对应的offset值
10.09–10.12
and this is exactly the thing that I'm looking for
这也正是我正在找的那个东西



10:15 – 10:16
So let's look at and see how this works
So，我们来看下它的工作原理是怎么样的
10:16 – 10:21
So let's say that and we have three keys abc Ø def xyz
So，假设我们有三个key，即abc，def以及xyz
10:21 – 10:25
So again I can just take this thing ABC hash it,
So，我可以通过对abc进行hash处理
10.25–10.31
and then that'll tell me I know I'm at offset zero is exactly the thing I'm looking for
然后hash函数告诉我在offset值为0的位置上，有我所想要找的东西
10:31 – 10:35
So this is not exactly what our hash table is actually could look like
So，这并不完全是我们hash table实际上看起来的样子
10.35–10.37
this is just storing the original keys
它里面只是保存了这些原始的key
10:37 – 10:44
in practice what we're gonna need to have is actually store pointers to where the original you know some other location where that original keys is located
实际上，我们需要去保存一些指向这些原始的key所在位置的指针
10:44 – 10:46
Again think of this like a table index
你们将它当做了一个表索引
10:46 – 10:49
I don't want to store the keys maybe in might my hash table,
我不想将这些key保存在我的hash table中
10.49–10.51
I want to store it a pointer to where the key is found
我想将它保存为指针，指向key所在的位置
10:53 – 10:59
All right ,so what are some problems to be sometimes we made with this kind of hash table
So，对于这种hash table而言，我们有时候会遇上的某些问题是什么呢?
11:03 – 11:04

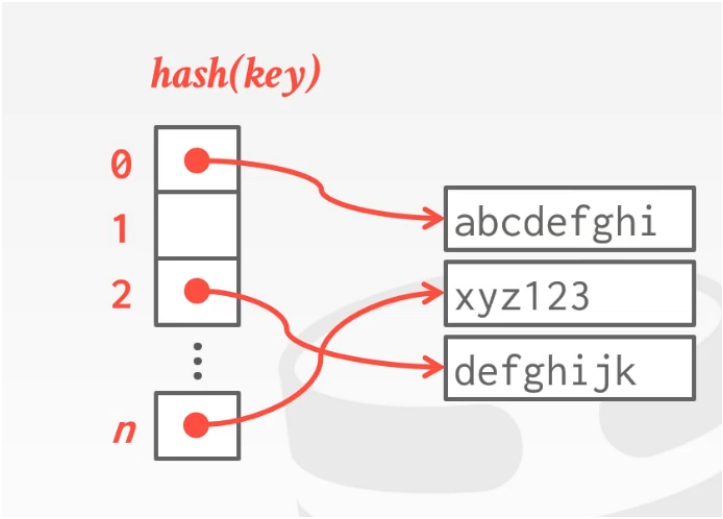
Yes, in the back
后面的那位同学，请问
11:06 – 11:07
Correctly,
没错
11.07–11.11
you say that we know the number elements ahead of time in the first place that's one
我们提前知道了hash table中元素的数量，这是其中一点
11.11–11.12
what's the second assumption
第二个假设是什么呢?
11:17 – 11:19
Is all the values are near each other in the cache
在cache中的所有值是否都是挨个放在一起?

11.19–11.20
for this purpose that doesn't matter here
出于此处的目的，这并不重要
11:24 – 11:25
But he says there's no collision between keys
他表示在key与key之间没有碰撞（collision）
11:26 – 11:26
So what is a collision
So，碰撞（collision）是什么?



11:29 – 11:30
He says the hashing the same slot ,exactly, right
他想说的是，我们所得到的hash后的结果指向了同一个slot，没错，确实这样
11:31 – 11:36
So this really simple hash table this is actually the fastest hash table you could ever possibly build
这种简单的hash table实际上是你们所能构建的最快的hash table
11:36 – 11:39
But you have to make these assumptions in order to make to work
但为了能让它工作，你们必须做出这些假设
11:39 – 11:43
Right, so the first is that as he said we need to know exactly the number of elements that we had ahead of time
So，首先正如他所说，我们需要提前知道元素的数量
11:43 – 11:47
So we know exactly how many slots we want to allocate in our array,
So，我们就知道了我们想在我们的数组中分配多少个slot

11.47–11.51
and in practice that's not always going to be the case
实际上，这并不是什么问题



11:51 – 11:56
All right, if I'm building a I'm using my hash table as a hash index on a table,
如果我使用我的hash table作为一张表的hash index来使用
11.56–11.59
when I create the table, I don't have any data in there in the first place
当我创建这个表时，我并没有放任何数据
11:59 – 12:01
And as I started inserting things
当我开始插入数据时
12.01–12.03
then the number of slots I need actually grows
接着，我实际所需要的slot的数量就会增加
12:05 – 12:08
The other assumption that we mixed was that, we said every hash or every key is unique
另一个我们需要作出的假设就是，我们说过每次hash的结果，或者是每个key都是唯一的
12.08–12.11
and that's what he's saying that there's no collision
这就是他所说的没有碰撞（collision）
12:11 – 12:12
So we're assuming that every time we hash it
So，假设我们每次对key进行hash处理
12:12 – 12:16
It's always gonna land into a unique slot for that one key
它始终会落到属于该key所对应的唯一的slot处
12.16–12.22
and only that key ever to be able to exactly find the thing that we're looking for
只有通过这个key，我们才能准确地找到我们所要找的东西
12:22 – 12:24
And so because we know all the keys ahead of time
So，因为我们提前知道了所有的key
12:25 – 12:27
And because we know that they're unique
因为我们知道这些key是唯一的
12.27–12.29

when we hash them
当我们对这些key进行hash时
12.29–12.32
, this is using what is called a perfect hash function
我们使用的这种被称为完美的hash函数
12:32 – 12:36
So a perfect hash function is like this if theoretical thing it exists in the research literature
So，实际上完美hash函数是一种存在于研究文献中的一种理论上的东西
12:37 – 12:38
But in practice nobody actually does this,
在实际生活中，实际上没人去这样做
12.38–12.40
because it's impractical you can't actually do this
因为这是不切实际的，实际上我们无法做到这点
12:41 – 12:43
And a perfect hash function just means that
完美hash函数指的是
12.43–12.46
if I have two keys that are not they're not equivalent,
如果我有两个不相等的key
12.46–12.52
then whatever hash I generate for them is also not gonna be equivalent
接着，我不管对它们进行怎样的hash处理，它们hash后的结果也不会相等
12:52 – 12:55
So for every unique key I generate an exactly unique hash value
So，对于每个唯一的key来说，我就会生成一个唯一的hash值
12:56 – 13:01
And again you can't actually do that there's no magic hash function that exists today that can guarantee this
再说一遍，实际上你没法做到这个，因为现在并不存在任何能保证唯一结果的神奇hash函数
13:01 – 13:08
The way you would actually implement a perfect hash function is actually use another hash table to map a key to another you know the hash value
实际上，你们所实现完美hash函数的方式是通过另一张hash table将一个key映射到另一个hash值上面（知秋注：Java程序员可以思考下concurrentHashmap，第一次hash是为了分区，第二次hash是为了确定具体位置，但也不能保证完美）
13:08 – 13:11
Which is kind of you know stupid cuz now you have a hash table for your hash table
虽然我们知道这样很蠢，因为你要为你的hash table再搞一张hash table
13:11 – 13:14
It's so nobody actually does this in practice
So，在实战中实际上没人会这么做
13:15 – 13:17
So the thing that we're gonna talk about today is,
So，我们今天要讨论的事情就是
13.17–13.25
how do we actually build a hash table in the real world to not have to make these assumptions, and be able to use them in a database system

在现实生活中，在不用管这些假设的前提下，我们实际该如何构建出一个hash table，并且能将它们运用在数据库系统中

13:25 – 13:27
So when people say, I have a hash table
So，当人们说，他有一个hash table时
13:28 – 13:32
They essentially mean it's it's a data structure comprised of two parts
他们本质上讲的是，一种由两部分组成的数据结构
13:32 – 13:34
The first is the hash function
第一部分是hash函数
13.34–13.42
which is a way to take any arbitrary key, and map you know a map it to a integer value in a smaller domain
该函数将任意的key映射到一个较小范围的interger值上面
13:42 – 13:46
Right, so I can take any string and the integer and you float ,doesn't matter
So，我可以使用任意字符串，整数（Integer），浮点数（float）之类的作为key，具体是什么都没什么关系
13:46 – 13:47
I throw it to my hash function ,
我将这个key丢给我的hash函数
13.47–13.55
and it's gonna produce either a 32–bit or 64–bit hash unique hash value integer or not unique, so I hash hash integer
接着，该函数就会为我生成一个32位或64位长度的唯一hash值，也可能不是唯一的，总之就是个hash数字
13:56 – 14:04
So there's gonna be this big trade off and and what kind of hash function we're gonna use between how fast it is and the collision rate
在我们所使用的hash函数中，在它的速度和hash碰撞率之间存在了巨大的取舍
14:04 – 14:08
Because again if we have different keys mapped to the same slot,
因为如果我们将不同的key映射到了同一个slot上
14.08–14.09
that's a collision
这就会导致一次hash碰撞
14.09–14.11
and now we have to deal with that in our hashing scheme

现在，我们就必须在我们的hashing scheme中处理这个问题
14:12 – 14:14
So what's the fastest hash function I could ever build
So，我所能构建的最快的hash函数是什么呢?
14:16 – 14:16
What that
能再说一遍吗?
14:18 – 14:19
It`s mod prime number even faster
他说的是，用质数来取模，在速度上甚至会更快点
14:23 – 14:23
What that
你想说啥
14:24 – 14:26
He said that value itself
得到的结果就是它自己
14.26–14.26
you're close
你的说法很接近正确答案
14.26–14.27
but what does that mean,
但这是什么意思呢?
14.27–14.30
if I have a string how do I return back that value and then put it into my slot
如果我有一个字符串，我该如何返回它的值，并将它放进我的slot中呢
14:32 – 14:32
Even faster
甚至更快
14:36 – 14:37
There's bits of memory to give it if it's a large string
如果它是一个很大的字符串，拿到它的内存位二进制表示
14:40 – 14:42
He said mod if there's a mod nur,
使用二进制运算取模
14.42–14.43
yes
请问
14:43 – 14:46
Constant, one, right
使用常量 1
14:46 – 14:49
No matter what key you give me I return back the number one
不管你给我什么key，我返回给你的始终是数字1（知秋注：任何key模1都为0，Andy又调皮了）
14:49 – 14:50
That's gonna be super fast
这样做肯定超级快
14.50–14.53
because that's gonna be on the stack, that's gonna be impossibly fast

它不可能不快
14:53 – 14:54
But your collision rate is BBBBBB
但你的碰撞率绝对会炸的
14:55 – 14:57
Because it always goes to the same slot
因为这样做会导致所有的key都映射到同一个slot上
14:58 – 15:00
So in the other end of spectrum is that perfect hash function
So，与之相反的就是完美hash函数
15:00 – 15:03
But I said I need I need another hash table to make that work
但我说过，我需要另一个hash table才能让它奏效
15.03–15.04
max like the worst case scenario
就像是那种最糟的情况那样
15:04 – 15:08
So my collision rate is is zero ,but that's the slowest
So，我的碰撞率是0，但它的速度是最慢的
15:08 – 15:09
So we want something in the middle
So，我们想在两者之间取得平衡
15:10 – 15:11
Okay

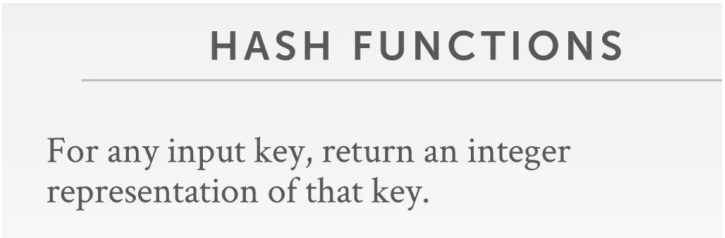
15.11–15.12
all right
##
15.12*–15.14 ！！！！！！！！
so the next piece is the hashing scheme
So，我们接下来要讲的东西是Hashing scheme
15:14 – 15:19
The hashing scheme is essentially the the mechanism or procedure we're going to use,
when we encounter our collision in our hash table
本质上来讲，hashing scheme是当我们在我们的hash table中遇上hash碰撞时，我们用来处理这种问题的一种机制或者步骤
15:20 – 15:26
Right, so again there's this trade–off between memory and and compute
So，再说一遍，在内存和计算之间存在了取舍
15.26–15.28
which is the classic trade–off in computer science
这是计算机科学中一个经典的取舍问题
15:28 – 15:35
So if I allocate an impossibly large you know slot array like you know two to the 64 slots
如果我分配一个非常非常庞大的slot 数组，里面有2的64次方个slot
15:35 – 15:37
Because that's all the memory you have them on my machine,
因为这是你机器上的全部内存
15.37–15.40

then my collision rate is gonna be practically zero
那么我的碰撞率几乎为0
15:40 – 15:42
Of course, now I can't do anything else in my database ,
当然，现在我没法在我的数据库中做其它事情了
15.42–15.45
because I've used all my memory for my hash table that's barely even full
因为我已经将我的所有内存都给了我的hash table，几乎没有任何空余空间了
15:45 – 15:47
But my collision rate is gonna be amazing
但我的碰撞率就会非常惊人
15:48 – 15:51
If I have a slot array of size 1 my clear rates can be terrible
如果我的slot array的大小为1，那么我的碰撞率就会很可怕
15:53 – 15:56
And if I have to do a bunch of extra instructions to deal with those collisions
那么我需要做一些额外的指令来处理这些碰撞问题
15:56 – 15:59
But my storage overhead is is the minimum
但我的存储开销是最小的
15:59 – 16:01
So again, there's this we want to be sort of in the middle here
So，再说一遍，我们想在两者之间取得平衡
16.01–16.06
we want to balance the amount of memory, we're using or amount of storage, we're using for a hash table
在我们使用一个hash table时，我们想在我们所使用的内存量和存储空间取得平衡
16.06–16.11
with the extra instructions you have to do when we have a collision
当我们遇上碰撞问题时，我们需要通过一些额外的指令来解决它们

16:12 – 16:17
All right, so today we're gonna focus on again the the the with we start beginnings are still about hash functions
So，我们今天首先依然要谈的重点就是hash函数
16:18 – 16:22

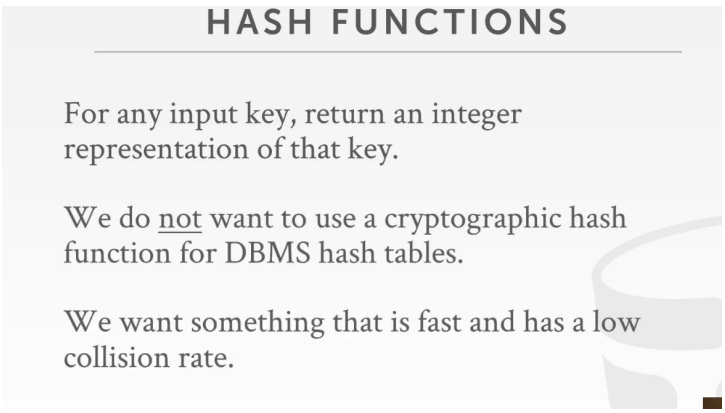
Just get to show you what what hash functions are out there the modern ones that people are using
我会向你们展示有哪些hash函数，以及人们现在使用的是哪些
16:22 – 16:24
And then we're talk about two type of hashing schemes,
然后我们会去讨论两类hashing scheme
16.24–16.26
the first is static hashing
首先要讲的是static hashing scheme
16.26–16.31
is where you have an approximation of what the size of the keys are trying to store the key set
你可以大致估算出我们所试着要保存的key的集合的大小

16:31 – 16:32
And then we'll talk about dynamic hashing
接着，我们会讨论dynamic hashing
16.32–16.37
where you can have a hash table that can incrementally grow without having to reshuffle everything
我们有这样一个hash table，它可以在不需要将所有存储条目重新打乱再做一次hash的情况下进行扩容
16:38 – 16:43
Again the combination of a hash function and hashing scheme is what people mean when they say I have a hash table
当人们说他们有一个hash table时，他们所说的其实就是由一个hash函数和hashing scheme所结合的产物



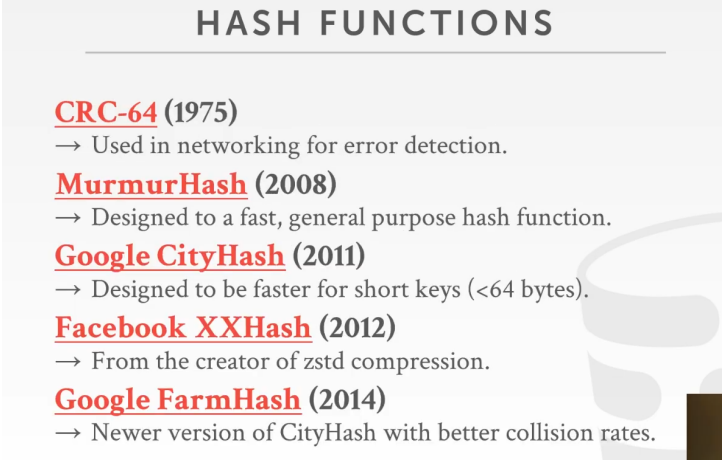
16:45 – 16:57
Alright, so again a hash function is just this really fast function that we want to take any arbitrary byte array or any arbitrary key, and then spit back a 32 bit or a 64–bit integer
So，hash函数其实就是一个速度很快的函数，我们将任意的byte array或任意的key传入函数，然后它就会返回一个32位或64位长度的integer
16:58 – 17:01
So can anybody name a hash function maybe one they've used before.
So，你们有人能说出一个hash函数的名字吗，可以是你们以前用过的
17:02 – 17:06
He says sha what sha–256, that's one can name another one
他说的是SHA–256，这是一个，有人能说下别的吗?
17:07 – 17:09
Yes, md5 perfect, all right
MD5，不错
17:09 – 17:10
This is actually a great example

这实际上是一个很棒的例子
17:10 – 17:13
So he said sha-256,he said md5
So，他们一个说了SHA-256，另一个人说了MD5
17.13-17.19
sha-256 is a cryptographic hash function that's actually reversible
sha-256是实际上可逆的加密hash函数
17:19 – 17:21
Right, it's a public/private key thing
它是一种使用了公钥/私钥的东西
17:21 – 17:23
So given a key I can hash it
So，对于一个给定的key，我可以对它进行hash
17.23-17.26
and then I know how to take that key and reverse and get back the original value
然后，我知道该如何将这个key变为原来的值
17:26 – 17:32
He said md5 which takes any arbitrary key and fist back a 32 character unique hash
他刚说了MD5，我们可以将任意的key传入它的hash函数，然后会返回给我们一个32位的唯一hash值
17:32 – 17:37
That in it's not supposed to be reversible it is now cuz people cracked it
它本来应该是不可逆的，但现在因为有人破解了它，所以可逆
17:37 – 17:39
But that's something where it's of one way hash,
但这是一种hash的方式



17.39-17.41
hisses hisses of reversible hash
这是一种可逆的hash
17:41 – 17:46
So in our database system we do not care about cryptography, for when we do we're doing hash tables
So，在我们数据库系统中，当我们在做我们自己的hash table时，我们并不在意加密性
17:47 – 17:52
Now that's you know you you can encrypt the data when you store it on on disk or on your public cloud infrastructure
要知道，当你们将数据保存在磁盘或者是你们的公共云设备上时，你们可以对数据进行加密
17:52 –17.56
But when we're doing our hash join or and building our hash table,

但当我们在进行hash join操作，或者是构建我们的hash table时
17.56–17.58
we're not gonna care about cryptography
我们并不会去在意加密性
17.57–18.02
we're not gonna care about leaking information about our keys
我们不在意泄露我们key的相关信息
18:02 – 18:06
Because we're just trying to build this hash you know build our hash tip hashing data structure
因为我们只是试着去构建hash型数据结构
18:06 – 18:08
So we're not gonna use something like like sha–256,
So，我们不会去试着使用诸如SHA–256之类的东西
18.08–18.11
because one we don't care about the cryptographic guarantees it provides
因为我们并不在意它所提供的加密性保证
18:11 – 18:13
It's also super slow so we're not gonna use it at all
并且它的速度也超慢，因此，我们根本不会去用它
18:14 – 18:17
MD5 is a one–way hash
MD5是一种one–way hash（单向散列）
18.17–18.23
and that's something we could use for a hash function we don't,because it's super slow we'll see other ones that are that are faster
我们可以将它作为我们的hash函数，但我们不会这样做，因为它还是非常慢，稍后我们会去介绍写其他比这更快的hash函数
18:23 – 18:25
And it's also sweet one way, but people have rainbow tables to reverse it
它也是不安全的，因为人们可以通过彩虹表对它进行破解
18:26 – 18:28
So that has doesn't we have good cryptographic guarantees
So，它也不能为我们提供良好的加密保证
18:29 – 18:35
All right,so again we care about something that's fast, we care something that has a low collision rate
So，再说一遍，我们所关心的是速度和碰撞率



18:36 – 18:42
So this is just sort of a list of some of the hash functions that are people are using today
So，幻灯片上所列的是人们现在所用的部分hash函数
18:43 – 18:49
So CRC is used in the networking world it was originally you know a Menten in 1975
CRC是用在网络世界的一种hash函数，它在1975年被人发明出来
18:49 – 18:52
I don't remember whether it was 32 bits or 16 bits back then
我不记得当时它是32位还是16位加密了
18:52 – 18:56
But now if you want to use CRC there's a 64 bit version and you would use something like that
但现在如果你使用CRC，它现在有64位版本，你们会使用某种和它类似的东西
18:57 – 19:04
So again this will produce something with a reasonable collision rate and,but it's gonna be super super slow
So，它所生成的数据的碰撞率虽然合理，但是它的速度非常非常慢
19:04 – 19:05
So we nobody actually does this in practice
实际上，现在在实战中，我们没人会去使用它

19:06 – 19:12
So this is sort of MurmurHash sort of again from a database perspective this enters the era of modern hashing functions
接着要说的是MurmurHash，从数据库层面来讲，它的诞生进入了现代hash函数的时代
19.12–19.14
and these are the ones that we're gonna care about
这也是我们所关心的一种hash函数
19:14 – 19:17
So murmur hash came out in 2008,
So，MurmurHash在2008年诞生
19.17–19.22
it was just some dude on the internet posted up his general–purpose hashing code on github
网上有位大兄弟将他的通用型hash代码发到了Github上
19:22 – 19:24
And then people picked it up and then started using it
接着，人们看到了这个，然后开始用起了它
19:26 – 19:30
Google then took murmur hash in in the early 2010's,
谷歌在2010年早期采用了Murmurhash
19.30–19.34
modified it to be a faster on shorter keys
并对其修改，使得长度更短的key可以获得更快的速度

19:34 – 19:36
And then they released something called Cityhash
然后，谷歌放出了某种叫做CityHash的东西

19:37 – 19:38
And then later on in 2014
然后，在2014年时
19.38–19.41
they've modified this again to have FarmHash
他们又对此进行修改得到了FarmHash
19.41–19.44
that has a better collision rate than CityHash
它的碰撞率要比CityHash来的更低
19:44 – 19:47
So farm hashes Cityhash are pretty common in some systems
So，FarmHash和CityHash在某些系统中很常见
19:48 – 19:54
What is now considered to be the state of your art and the fastest, and has the best collision rate in four hash functions today
So，在这如今的四个hash函数中，哪个函数的速度最快，并且碰撞率最低呢?
19:54 – 19:56
It`s actually Facebook's xx hash
实际上是FaceBook的XXHash
19:56 – 19.59
And not the the original one 2012
我们说的并不是2012年那个原始版本
19.59–20.01
there's a xxhash 3,
现在已经有了XXHash3
20.01–20.06
that is actively under active under active development now ,I think it came out in 2019
它现在依然还处于开发中，我觉得它会在2019年推出
20:06 – 20:10
So right now this is the fastest and has the best collision rate of all these hash functions
So，XXHash是这些hash函数中速度最快，碰撞率最低的那个
20:10 – 20:13
So if you're building a database system today, you want to be using xx hash
So，如果你们现在想构建一个数据库系统，那么你们会想去使用XXHash
20:14 – 20:18
So again we don't care so much how this is actually implemented
So，我们不会在意它实际上是如何实现的
20:18 – 20:21
Right, I don't this is not an algorithms class, I don't care about the internals are
我给你们上的这节课并不是算法课，我并不在意内部是怎么做的
20:21 – 20:25
Again,all I care about is how fast it is ,and what the collision rate is
我所关心的是它的速度有多快，它的碰撞率是怎么样的
20.25–20.30
,and there's benchmarks to measure and the quality of the collisions rates all these algorithms
我们会对这些算法进行测试，看下其中的碰撞率是怎么样的