


04-04

DECOMPOSITION STORAGE MODEL (DSM)

The DBMS stores the values of a single attribute for all tuples contiguously in a page.
→ Also known as a "column store".



CMU DB

01:00:06 – 01:00:09

Right, so again going back to our example here

So, 回到我们这里的例子

01:00:09 – 01:00:11

So this is what it looks like as a row store

So, 这里所展示的是行存储的样子

01:00:11 – 01:00:14

But so say now we just take every single column

但现在, 我们将每个列拿出来

01:00:15 – 01:00:18

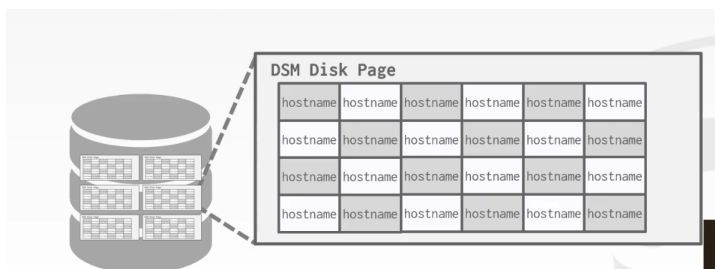
And we're going to split that up and now then within a single page

我们将它拆分开来, 并放在单个page内

1.00.18-1.00.20

we have just the data for that column

我们只有该列的数据



01:00:21 – 01:00:22

So here's all the host names together

So, 这就是所有hostname的数据了


1.00.22-1.00.26

and we had the same thing where user ID lastlogin and the other attributes for this table here

当然, userID, lastLogin以及这张表上的其他属性也是以同样的方式进行保存的

DECOMPOSITION STORAGE MODEL (DSM)

The DBMS stores the values of a single attribute for all tuples contiguously in a page.
→ Also known as a "column store".

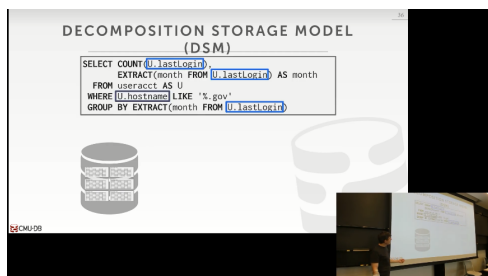


CMU DB

01:00:27 – 01:00:28

alright so forth like that

以此类推



01:00:29 – 01:00:32

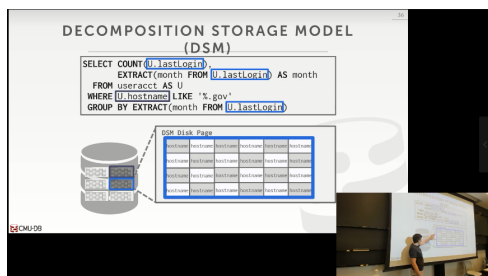
So now I come back to this query we had before

现在，我们来回看下我们之前这个查询语句

01:00:32 – 01:00:35

So the first thing I need to do is do my where clause on **hostname**

首先我需要做的事情就是对where子句中的hostname进行处理



01:00:36 – 01:00:42

So now I just need to know all I have to do is go bring in the **hostname** page wrong color but ignore that

So，现在我需要做的就是拿到hostname所在的page，这里我颜色标错了，但不要在意

01:00:42 – 01:00:50

I just bring the **hostname** page in, I can then rip through that quickly say look at every single **hostname** and do my ~~my~~ predicates

我拿到hostname所在的page（并放入内存中），然后对每个hostname进行扫描以及条件判断

01:00:50 – 01:00:53

Now I have a bunch of tuples that matched

现在，我就得到了一系列匹配的tuple

01:00:53 – 01:00:58

So then I go back and bring in the bring in the lastLogin page,

接着，我将lastLogin所在的page放入内存

1.00.58–1.01.01

and just jump to the locations that I need within that

然后跳到这个page中我所需要数据的那个位置

1.01.01–1.01.05

and get the lastlogin information that I want to produce my answer

并拿到我想用来生成我答案的那个最后登录的信息

并拿到我想用来生成我答案的lastlogin信息

01:01:05 – 01:01:12

So say in a real simple case here that one lastlogin data is one page, the hostname is another page

So, 在这个简单的例子中, 这里用了一个page专门放lastLogin的数据, 另一个page用来专门放hostname的数据

01:01:12 – 01:01:14

So before I had to scan all the pages

So, 和之前我必须扫描所有pages相比

1.01.14–1.01.16

and this one I do only have to scan two

这个我只需要扫描两个pages即可

01:01:16 – 01:01:19

again think of in extremes if I'm talking about billions of pages

考虑下极端情况, 如果我有数十亿个page

01:01:20 – 01:01:22

All right, then that's a big difference

那这就会有很大的不同

1.01.22–1.01.23

Yes

请问

01:01:23 – 1:01:27

Students:(提问)

1:01:27 – 01:01:29

So this question is are we storing the primary key with each columns

So, 他的问题是我们是否会将主键和每个列一起保存

1.01.29–1.01.33

your real question is,how do I figure out I had the hostnames that match

你的问题是, 我该如何弄清楚这些hostname所对应的是哪个tuple

1.01.33–1.01.37

how do I then go look up in the last login column and figure out how they match

当我在lastLogin列中查找时, 该怎么让它们和tuple对应

01:01:37 – 01:01:41

Next slide,perfect ,and any questions

问的好, 下一个, 你们还有任何问题么

01:01:43 – 01:01:46

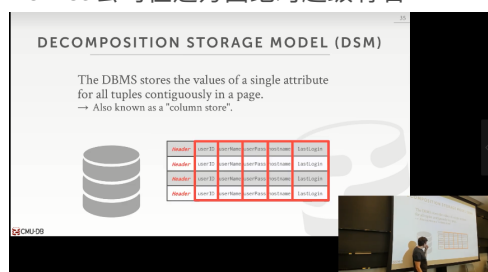
So the blows are other stuff we can do with this that we're not gonna cover in this class

So, 我们还可以通过其他方法来做到这一点, 但我们在课堂上并不会去介绍

01:01:46 – 01:01:52

But the most of other advantages you can get,and actually if you come to the vertica talk in two weeks,Vertica is super famous for this

如果你们参加两周后由Vertica公司所举办的讲座, 你们能够从他们的讲座中收获到很多, Vertica公司在这方面绝对超级有名



01:01:53 – 01:02:04

So with the row stored model, all the values within or the attributes within the tuple, they're all you know roughly different domains

在行存储模型中，对于tuple内的所有属性，粗略的来讲，它们都有不同的领域（知秋注：每个属性所占空间可能都不一样）

01:02:04 – 01:02:09

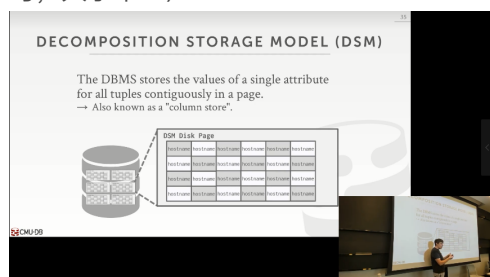
Right this is gonna be a **username**, this is gonna be **hostnames**, it's me **lastLogin**, we're just gonna look a timestamp

比如说，这里是username，那里是hostname，这边是lastLogin，我可以在这里面看到时间戳

01:02:09 – 01:02:11

Right, it's all sort of jumbled together

它们都纠缠在一起，看起来杂乱无章（知秋注：所占空间每一块对于每一行数据来说往往都不均匀，大小不一）



01:02:13 – 01:02:17

And so if I can then pack them all this data together that are the same column

So，如果我可以将同一列的数据都打包放在一起

01:02:17 – 01:02:20

Now that's one of compression techniques I can do

我可以通过一种压缩技术来做

01:02:21 – 01:02:24

Because I know they're gonna be all the the same type

因为我知道它们的类型都一模一样

01:02:24 – 01:02:30

Right, so ~~let's say~~ let's say that I'm storing temperatures of the room

假设，我们要去保存房间温度的数据

01:02:30 – 01:02:35

And you know it's ~~it's~~ 70 degrees now, maybe 70 point one, seven seventy point two,

比如说，70度，70.1度或者70.2度

1.02.35–1.02.36

like it's not gonna fluctuate that much

这些温度并没有太大波动

1.02.36–1.02.40

instead of storing ~~that~~ that's the full temperature every single time

这样我们无须每次都去保存完整的温度

01:02:40 – 01:02:45

What right is shorter a small Delta of what the base temperature was when we first started taking measurements

我们可以将我们第一次记录的温度作为标准，然后每次记录新的温度数据和该标准的差值即可

01:02:46 – 01:02:51

And now I don't need to store the entire value all over again,I just store you know that's smaller representation

现在, 我就无须去保存完整的值了, 我只需要保存这些更小的值即可

01:02:52 – 01:02:57

I think I mean think of like you know if you run like **gzip** or **snappy** or whatever your favorite compression algorithm is

你们可以去使用Gzip、Snappy或者其他你喜欢的压缩算法

01:02:57 – 01:03:01

You can't compress an **mp3** really well,because there's already sort of compressed

你们无法去压缩一个mp3文件, 因为它其实已经被压缩过了

01:03:01 – 01:03:02

But if it's a text file

但如果它是一个文本文件

1.03.02–1.03.05

that you can compress the hell out of that

那么你们就可以对其进行压缩

1.03.05–1.03.06

because there's gonna be a bunch of characters repeating over and over again

因为在这里面有大量重复的字符存在

01:03:07 – 01:03:11

So if you have repeated values in your attribute

如果在你们的属性中有重复的值存在

1.03.11–1.03.14

then you can compress the hell out of it and get much better performance

那么你们就可以将其压缩, 以此获得更好的性能

01:03:15 – 01:03:16

So now when I go want to go to a read

So, 现在当我想去进行读取时

1.03.16–1.03.23

again,with every page fetch instead of maybe getting a thousand tuples,I could get like ten thousand tuples,because in compressed form

原本每个page上只能放1000个tuple, 但因为现在进行了压缩, 那么一个page上我可能放10000个tuple

01:03:24 – 01:03:30

And some systems actually can operate directly on compressed data without you without getting uncompressor which is the big win

实际上, 在没有解压的情况下, 某些系统可以直接对压缩数据进行操作, 这是一种很大的优势

01:03:32 –01:03:34

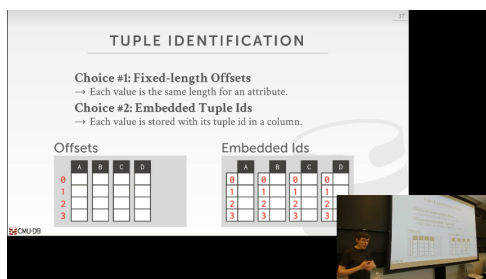
Okay,we don't cover we're not gonna cover compression in this class

Ok, 在这门课中, 我们并不会去介绍压缩相关的东西

1.03.34–1.03.38

we spent a whole lecture in the advanced class, but I'm happy to talk about more about it if you want

我们会在高级课程中花一整节课的时间去讲解这方面的内容, 但如果你们想听的话, 我也很乐意去多谈论些这方面的事情



01:03:40 – 01:03:47

Okay, so now the his question is how do I figure out I had a match in one page, how do I find a match in another page

Okay, 他的问题是我该怎样从一个page中找到一个匹配项, 怎样从另一个page中找到一个匹配项

01:03:47 – 01:03:52

So in general there's two approaches, but everyone pretty much does the first one

So, 通常情况下, 我们有两种方案, 但许多人都会选择第一种

01:03:53 – 01:03:55

So the first choice is to have fixed-length offsets

So, 第一种方案是使用固定长度的偏移值

01:03:56 – 01:04:03

So that means that for every single value in a column, it's always going to be a fixed length

这就意味着, 对于一列中的每个值来说, 它们的长度始终是固定的

01:04:04 – 01:04:06

So again think simple a 32-bit integer

稍微思考下, 假设是32位的integer

01:04:07 – 01:04:09

So all these are going to be each of these values to be 32 bits

So, 这些数据的长度都是32位

01:04:10 – 01:04:11

So now if I have a match say

So, 如果现在我要去进行匹配

1.04.11–1.04.15

in this column at offset 1

在这一列,

这一列在offset值为1的地方 (开始)

1.04.15–1.04.18

and I'm again need to find the corresponding tuple in this column

我现在需要在这个列中找到对应的tuple

1.04.18–1.04.21

I know that say this comes also 32 bits

我知道这个值的长度也是32位

01:04:21 – 01:04:23

And I can just do a simple arithmetic and say

我可以进行一些简单的算术运算

1.04.23–1.04.27

I want offset 1 times the size of each attribute

我想让offset值为1处的东西乘以每个属性的容量???

我可以将offset 1*每一个属性的size

1.04.27–1.04.29

and then I know exactly where I need to jump to

然后我就知道我需要跳到的位置在哪了

1.04.29–1.04.35

or translate that to the row ID,or the the page number and slot number that has the data that I'm looking for

或者将它翻译为row id, 或者是page number和slot number这样的东西, 这样我就知道我正在寻找的数据在哪了

01:04:35 – 01:04:37

So ~~that~~ this is probably the most standard approach

So, 这可能是最标准的方案

01:04:37 – 01:04:42

Of course now the tricky thing is say,well what if I have a bunch of strings that are varied length field

当然, 我们也会遇上些棘手的事情, 那就是如果我有一大堆字符串, 这些字符串的长度都是可变的

01:04:42 – 01:04:48

Then you get into like alright can I compress it to a fixed length field or kind it's padded outs as always fits in whatever the max size is

然后, 你们可能会去想, 我们能否将它压缩为一个固定长度的数据, 或者是对字符串进行填充, 让它的长度变成我们所允许的最大长度

01:04:48 – 01:04:50

Different database systems do different things

不同的数据库系统做不同的事

1.04.50–1.04.53

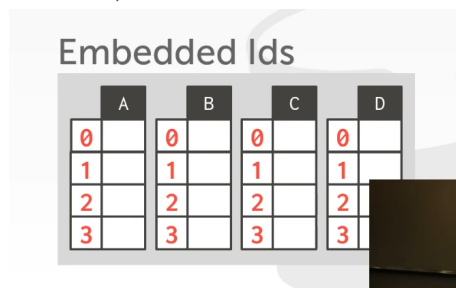
but overall ~~this is~~ this is the most common approach

但总而言之, 这是最常见的方法

01:04:54 – 01:05:01

The other approach which I forget there's like one system that does this which I think is a bad idea,they might have gotten rid of it,but I forget who it is

此外还有另外一种方法, 有某个系统使用了这种方法, 虽然我忘了它的名字, 但我觉得这是种糟糕的想法, 它们现在可能已经不用这种方法了



01:05:02 – 01:05:10

Where you actually store for each value in the column,you store a the primary key or an identifier for it

在这种方法中, 对于列中的每个值, 我们会为每个值都保存一个主键或者是标识符

01:05:10 – 01:05:14

So then you say alright,I'm at it for the column one,I'm looking at tuple 1

然后, 我们想在Column A中找到tuple 1

1.05.14–1.05.16

and I want to get to tuple 1 and column B

我想拿到Column B中的tuple 1

01:05:16 – 01:05:24

I have another map eagerly to do a lookup and say, how to go find the offset location for that particular tuple in this column

根据这个列中的这个tuple，我该如何找到这个tuple所在的offset值

我有另一个map急需要去查找，我该如何找到这个列中指定tuple所在的offset位置（知秋注：你要确定key和value两个一起的大小，然后再计算对应的offset值）

01:05:25 – 01:05:27

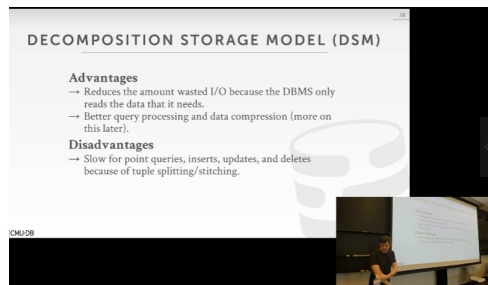
Of course obviously this has huge storage overhead

Of course，很明显我们能看出，这会造成非常大的存储开销

1.05.27–1.05.36

because you're storing this for there's you know this extra 32 bit or 64 that value or our ID for every single value which is wasteful

因为我们得为每个值都得用额外的32位或者64位空间来保存它们的id，这就是种非常浪费的做法了



01:05:37 – 01:05:40

All right, all right

1.05.40–1.05.47

so the advantages of the column store is that we can reduce amount of waste I/O for these OLAP queries

列式存储的好处在于，当我们进行OLAP查询时，我们可以降低这种垃圾I/O的数量

1.05.47–1.05.50

because we're only reading the the bearment amount of data we actually need

因为我们只会去读取我们实际所需要的数据量

01:05:50 – 01:05:52

We're not bringing in things we're never gonna need it all

我们不会去读取哪些我们根本不需要的东西

01:05:54 – 01:05:55

Will get better compression

这样就能更好的压缩数据

1.05.55–1.05.59

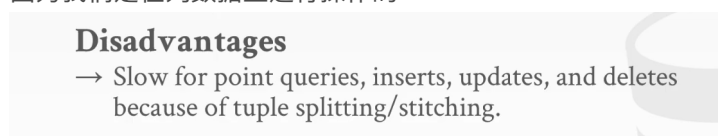
will give it a bet better query processing which we will cover and a few more lectures

这样在查询处理上的性能就会更好，关于这点我们会多花几节课来介绍

01:05:59 – 01:06:02

Because we know we're operating on columnar data

因为我们是在列数据上进行操作的



01:06:04 – 01:06:09

The the downside is obviously that for anything that needs access a single tuple

这种做法的缺点也很明显，那就是在读取一个tuple的时候就很慢

01:06:09 – 01:06:11

It becomes more expensive

这种代价就变得很高

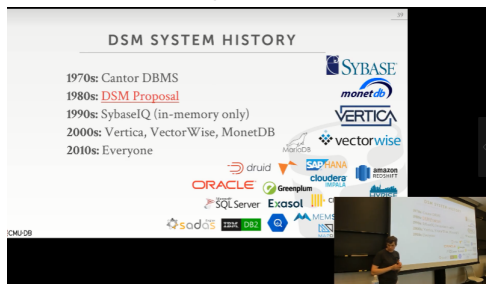
1.06.11–1.06.19

because now you essentially need to put together the the tuple from the different columns back together, whereas in the row store it's all just in one location for you 因为你就必须得把不同列的数据拼在一起，这样你才能拿到一个完整的tuple，然而在行存储的情况下，这些数据都是在一个tuple里，直接拿就行

01:06:20 – 01:06:24

And anytime you update or in sort of delete this, becomes more expensive, because again,because it you get to split it all up

你在任何时候对tuple进行更新或者删除，代价也会更加高，因为你将列都分开了



01:06:25 – 01:06:29

So I would say that column stores are not a new idea

So，列式存储其实并不是什么新的想法

01:06:30 – 01:06:31

They go back to the 1970s

这得从1970年代说起

1.06.31–1.06.41

there was like the Swedish military division built this thing called Cantor which essentially was ,they didn't call it a database system,because they used different language back 1970s

瑞士军方当时构建了一个叫做Cantor的系统，他们并没有将它称作数据库系统，因为在1970年代时，他们使用了别的语言来编写的这个系统

01:06:41 – 01:06:46

But if you go sort of read between the lines it at his essence it is a column store database System

但如果你们仔细看的话，它其实就是个列式存储数据库

01:06:46 – 01:06:51

It was never released never made public was this it was the only this internal project 它并没有被放出来，也并没有开源，因为它是一个内部项目

01:06:51 – 01:06:54

But that's the first known implementation of a column store

但它是第一个已知的列存储数据库的实现

01:06:55 – 01:07:00

The 1980s there was a paper that describes the decomposition storage model and more for more details to say

在1980年代，有一篇论文提到了列存储模型，并且里面提到了更多的细节

1.07.00–1.07.05

you know what's the short format look like, what are the implications of having this storage model

比如说，短格式（short format）的样子是什么样的，这种存储模型的意义是什么

01:07:06 – 01:07:12

One of the probably most famous commercial implementation among the first commercial implementations was this thing called **SybaseIQ**

注：Sybase IQ是Sybase公司推出的特别为数据仓库设计的关系型数据库，IQ的架构与大多数关系型数据库不同，特别的设计用以支持大量并发用户的即时查询。

可能最著名也是第一个商用列存储数据库实现应该是Sybase IQ

01:07:12 – 01:07:22

It was an **in-memory** columnstore that **Sybase** released as an accelerator for their regular row store database system, sort of have had have you actually worked at HTAP – and in sync

这是一种内存型列存储数据库，Sybase将它作为他们普通的行存储数据库系统的加速器放出，这样他们就可以进行HTAP(混合事务/分析)处理并且具备同步的能力

01:07:23 – 01:07:25

And there never really got big adoption

但它并没有广泛采用

1.07.25–1.07.30

because again it was sold as a add-on to the rows store database rather than a standalone thing

因为它被当做行存储数据库的插件来卖，而不是作为一个独立的产品

01:07:30 – 01:07:33

But it was the 2000s when the column store stuff really took off

但到了2000年代，列存储数据库才真正起飞

1.07.33–1.07.37

Vertica again was founded by Michael Stonebraker the guy admitted **Postgres** and **Ingress**

Vertica是由Michael Stonebraker所创立，他曾参与PostgreSQL和Ingress这些数据库的研发

01:07:37 – 01:07:44

That was his company that got bought by HP, **VectorWise** is a in-memory version of **MonetDB**

注：MonetDB是一个开源的面向列的数据库管理系统。MonetDB被设计用来为较大规模数据（如几百万行和数百列的数据库表）提供高性能查询的支持。

之后他的公司被HP收购了，VectorWise是MonetDB的内存版

01:07:44 – 01:07:48

MonetDB they out of Europe it's academic project ,but still around today

MonetDB以前是欧洲的一个学术项目，但现在到处都有在用它

01:07:48 – 01:07:52

I see he's asserted the first sort of columns store systems that were ever made in 2000's

它是在2000年代出现的第一个列存储数据库系统

01:07:52 – 01:07:58

But then it quickly became obvious ~~that~~ this is the right way to build data systems to for analytics

但之后列存储很快就变得非常出名，因为这是构建用于数据分析型数据库系统的正确方式

01:07:58 – 01:08:02

So pretty much everyone now has their own column store system

So很多公司现在都有了他们自己的列存储系统

01:08:03 – 01:08:06

And actually I wanted to give a demo of Vertica today, I couldn't get it running

实际上今天我想给你们演示下Vertica, 但我没法运行它

01:08:06 – 01:08:12

I did get the **MonetDB** column store working, and **stephannie** column store doesn't mean it's actually good

不过MonetDB还是可以运行的, 但使用列存储不代表实际上它就很好

01:08:13 – 01:08:16

So there's a bunch of stuff we'll cover as we go along for query optimization in **query** execution

但随着我们介绍查询执行中的查询优化时, 我们会介绍很多这方面的内容

1.08.16–1.08.19

just because your columns or doesn't mean you're magically gonna go faster

列存储并不会让你执行起来变得更快

01:08:20 – 01:08:25

I was actually able to get Postgres to be the column store for analytical queries

实际上, 我能够让PostgreSQL在做分析型查询时使用列存储进行

01:08:25 – 01:08:32

Because you know of how you actually queries, how you actually look at the data and what the query plan looks like

因为你们知道, 我们实际如何进行查询, 如何查看这些数据, 以及这些查询计划看起来是什么样子的

01:08:32 – 01:08:35

So the bunch of stuff we have to do it that we'll cover throughout the semester

So, 这里面我们需要做的事情有很多, 我会在这个学期中向你们介绍

1.08.35–1.08.40

that you have to that you want to do if, you know dirt if you're a column store that not everyone does

如果你们想做列存储的话, 那你们可能做, 但不是所有人都必须做的

01:08:40 – 01:08:43

Okay, so any question about column stores

Ok, 你们有人对列存储还有问题吗?

01:08:43 – 01:08:46

So if you go off and leave graduate from CMU

So, 如果你们从CMU毕业了

1.08.46–1.08.51

and you want to do analytics and some was like let's do it on post grass, but it's a row store, don't do that

你们想去做分析类的工作, 某些人可能会想去使用PostgreSQL去做, 但它是行存储型数据库, 请不要去霍霍它

01:08:51 – 01:08:57

Right, there's enough columns store systems that are out there that will they do you want to look at

明明有那么多我们可以使用的列存储数据库系统在那里

1.08.57–1.08.58

they're not cheap though

尽管它们并不便宜

1.08..58–1.08.59

at least for the commercial ones

至少对于商用系统来说是这样的

1.08.59–1.09.01

but there's some decent open-source ones

但当然这些数据库系统也有些是正经开源的

01:09:02 --1.09.04

Okay all right cool

1.09.04–1.09.10

so ~~the~~ the main takeaways from this is that as we show

因此，我们的主要收获是，正如我们所展示的那样

1.09.10–1.09.20

the underlying representation of the storage of the database is not something we can

just sort of put in our storage manager and not expose to any other part of the system

对于数据库存储的底层表示并不是那种我们可以放入存储管理器中的东西，并且不暴露给系统的任何其他部分

01:09:21 – 01:09:24

As we go out the rest of the semester, you'll see that a lot of times

在我们剩下的学期中，你们会经常看到这个

01:09:24 – 01:09:26

I'll say like Alright, this is the way to do it if you're a row store

我会说，如果你们使用的是行存储，那么就使用OLTP

1.09.26–1.09.29

this is the way to do it if you're column store

如果你们使用的是列存储，那就使用OLAP

01:09:29 – 01:09:34

And that's because again we if we know the data system knows more about what it's

actually doing what ~~what~~ the data looks like,

因为如果数据库系统知道它实际要做什么，数据看起来是什么样的

1.09.34–1.09.40

it's gonna make better decisions and better design choices and in order to get ,you know more efficient execution

数据库系统就会去做出更好的判断以及更好的设计选择，以便于能够更有效的执行查询

01:09:41 – 01:09:44

The other thing to also remember to basically for OLTP

我们要记住的另一个东西就是OLTP

01:09:44 – 01:09:46

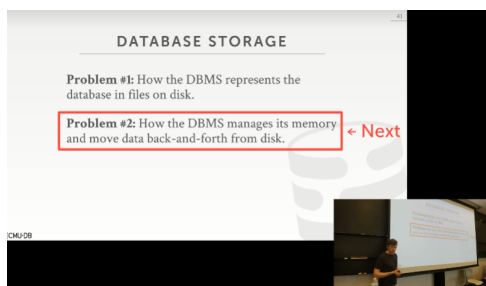
You want to use a row store 。 for OLAP, you want to use a column store

在OLTP中，我们会想去使用行存储。然而在OLAP中，我们会想去使用列存储

01:09:46 –01:09:53

But these this simple rule will carry you out through the rest your life and make your life easier

这些简单的规则会伴你一生，并且让你的职业生涯更加轻松



01:09:53 – 01:10:01

Alright, so now the last two classes we covered the this problem here how to actually represent the data in the database

在前两节课中，我们讨论了这个问题，即如何在数据库中表示数据

01:10:01 – 01:10:10

So now on starting on Wednesday, we'll talk about what do we add we actually bring the data in, and bring them to memory and manage that

接着在周三，讨论了如何将数据放入内存，并对它们进行管理

1.10.10–1.10.10

yes

请问

01:10:11 – 01:10:13

Students:(提问)

01:10:13 – 01:10:15

This question is there any good reason to do a mix of the two

他的问题是，有没有很好的理由来将两者混合在一起

01:10:15 – 01:10:19

So we actually built our database system that did a mix of the two

实际上在构建我们的数据库系统时（知秋注：这个课上cmu所开发的这个数据库），确实将它们两个混合在了一起

01:10:20 – 01:10:23

We threw that away and started over, because it's a bad idea

后面，我们将这种想法抛弃，并从头开始，因为它是个糟糕的想法

1.10.23–1.10.24

it was too much engineering overhead

它在工程上消耗太多时间了

01:10:24 – 01:10:33

There are some database systems will give you both they'll expose like some MySQL for example you can say ,create this table in it and it's a row store or create this other table on its a column store

有些系统会向我们提供这两种。例如，在MySQL中，我们在创建表时可以告诉它用行存储，创建另一张表时告诉它使用列存储

01:10:34 – 01:10:39

And they have essentially two separate storage managers, two separate execution engines to operate on them

本质上来讲，它们有两种独立的存储管理器，以及两种独立的执行引擎来对它们进行处理

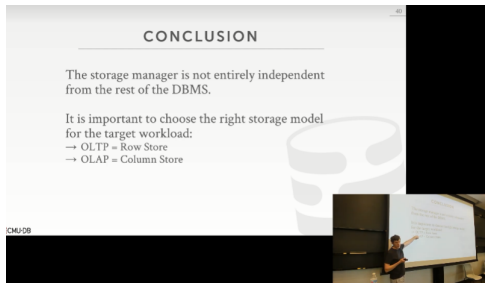
01:10:40 – 01:10:44

So those are sort of called hybrid storage systems hybrid databases systems

So，这种系统被称为混合存储系统，或者说混合数据库系统

01:10:45 – 01:10:48

We were all in I thought that was a good idea, I think it's a bad idea now
有人觉得这是种好想法，也有人觉得这种想法很糟糕



01:10:51 – 01:10:56

For in-memory we actually can do, we think we can do fast enough transactions on a column store

对于内存型数据库，我认为我们在列存储上能够足够快的执行事务处理

1.10.56–1.10.57

for disk it's a little more complicated

对于磁盘来说，就要稍微复杂点了

01:10:59 – 01:11:06

So there are systems that do both they're not they didn't really take off as much

虽然有些数据库对于OLTP和OLAP都支持，但真正牛逼飞起的没几个

01:11:06 – 01:11:07

So usually you see things like

So，通常你们会看到这样的东西

1.11.07–1.11.10

you could have a single interface

你们有一个简单的接口

1.11.10–1.11.16

where they have you write one query and then underneath it covers it figures out what you want to go the row store column store aside

当你写了一个查询时，在它内部它会去弄清楚你想用的是行存储还是列存储

01:11:17 – 01:11:18

There's ways to do that

有多种方式可以做到这点

1.11.18–1.11.24

but having a single server for architecture that can manage both I think is rough

但单引擎架构很难同时做到这两点

01:11:24 – 01:11:26

Students:(提问)

01:11:27– 01:11:32

He says why don't we store two copies of the same data ,great think of it extremes my database is one petabyte

他想问的是，为什么我们不将同一份数据保存两份，问得好，我们考虑下极端情况，假设我的数据库数据有1PB

01:11:34 – 01:11:39

So well I'm slides ready ,but like I can easily find

虽然我的幻灯片已经放完了，但是我还是能很轻松的回答你的问题

01:11:39 – 01:11:41

But I can cover this next class

但我可以在下节课讲这个

1.11.41–1.11.49

but basically with what people do is you have your front end OLTP systems and that's running MySQL ,MongoDB or whatever you want

但基本上来讲，人们在前端OLTP系统中所运行的是MySQL、MongoDB或者其他你们想用的数据库系统

01:11:49 – 01:11:53

And then you stream the data out over time to a back-end data warehouse

接着，你把你的数据传输给后端数据仓库

01:11:53 – 01:12:01

And then you basically can prune out ~~the latest data on~~ the old data on the OLTP side when you know you don't need it anymore

然后你基本上就可以将OLTP 这块的老的数据给修剪掉了，当它们不再被需要的时候

01:12:01 – 01:12:03

So you see this in like eBay ,

So，你可以在eBay之类的网站上看到这个

01:12:03 – 01:12:07

eBay only retains the last 90 days of auctions

eBay只保留最近90天的拍卖

1.12.07–1.12.08

and after that, they print it out

在此之后，它们将它打印出来

1.12.08–1.12.12

and that's because they want to keep the OLTP side nice and trim and fast

之所以这么做是因为它们希望保持OLTP这边运行足够好，减少不必要的数据库可以运行更快

01:12:12 – 01:12:17

But then they still retain everything else in the backend data warehouse would they do all the analytics to figure out what people are buying what when what they're doing

但之后，他们依然将所有数据保存在数据仓库内，这样他们能够对这些数据进行分析，并弄清楚人们买了些什么，以及人们做了什么

01:12:18 –01:12:20

that's the standard setup everyone does

这是所有人所做的标准设置

01:12:21 – 01:12:27

Right, and whether or not that's like you Know MySQL plus vertical like two separate database installations

比如，我们可以通过MySQL+Vertica这两个独立的数据库来做到

1.12.27–1.12.33

or whether it's a single hybrid database like splice machine can do this or MySQL could do this

或者使用单个混合型数据库，Splice machine可以做，MySQL也能做

01:12:33 – 01:12:37

Depends on what you want,you know how much money you have what you're willing to do

这就取决于你有多少钱，你想要做什么了

01:12:37 – 01:12:49

I think that what we found for our own system is that building having a super single Storage Manager try to manage both of these things was a bad idea among other things

我觉得如果我们要来构建一个试着管理这两种东西的单存储管理器，这会是一个很糟糕的想法

01:12:51 – 01:12:56

Okay, someone brought up testing last time and I really want to spend time to talk about that

Ok, 上次有人提到了测试这方面的事情，我也真的很想花时间来谈论这个

01:12:56 – 01:12:58

But I wouldn't have any time today

但我今天确实没什么时间了

01:12:58 – 01:13:03

But again next class we'll start talking on the buffer pool and hopefully, we talk about testing a little bit at the end okay

但下节课，我们会去讨论buffer池，希望我们在那节课末尾时有时间来谈论点测试方面的事情

01:13:03 – 01:13:06

~~And the other questions hit it~~

这句不要了