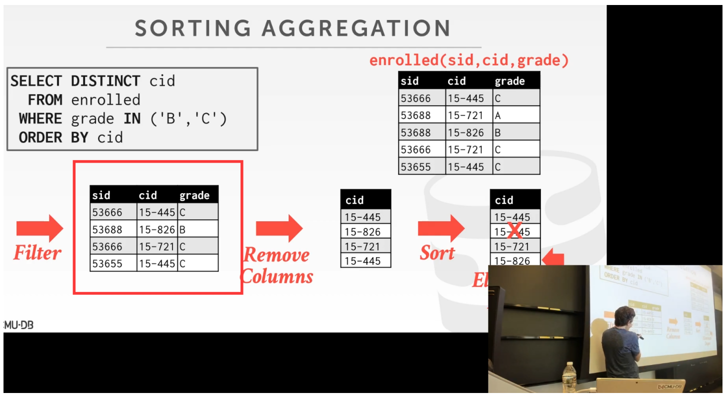
10-03

41:03 – 41:08
So in order for me to go get the data I need to do you know whatever the sorting I want to do
So，为了能让我拿到用来进行我想做的任意排序的所需数据
41:08 – 41:11
I gotta go get the entire I mean the entire record
我会去拿到整个record
41.11–41.12
because that’s gonna be packed together in a single page
因为它们被一起打包在一个单个page中
41:13 – 41:14
But if I can do a projection
但如果我能做projection （知秋注：即Select要查的那几个属性映射的数据）
41.14–41.18
i can strip out all the columns I don’t need or the attributes, they don’t need
我就可以剥离出所有我不需要的列或者属性
41:18 – 41:22
And then now when I’m doing my sorting ,I’m not copying around a bunch of extra Data
接着，当我进行排序时，我并不会去复制这些额外的数据
41:22 – 41:26
So my sort of sort of simple examples and was related to her question
So，我这个简单的案例其实是和她的问题有关联的
41:27 – 41:30
The you know what a might be actually passing around
这里实际可能是什么呢?
41.30–41.37
 it could be the record ID ,it could actually be the entire tuple itself, depending on how I want to materialize things
这里可能是record id，也可能是整个tuple自身，这取决于我想怎么实现
41:37 – 41:39
So the projection here allows me to throw away columns I don’t need
So，projection允许我将不需要的列给扔掉
41:40 – 41:42
So now when I’m doing my sorting
So，现在当我进行排序时
41.42–41.48
I’m copying things that just related to what’s needed for the rest of the query plan
我只复制查询计划剩下部分所需要的数据
41:49 – 41:49
Yes
请讲

42:12 – 42:14
So his question is
So，他的问题是
42.14–42.18
I mean it’s not really this query, you’re talking like a count query
你说的并不是这个查询，而是查询count
42:18 – 42:19
So this question is
So，他的问题是
42.19–42:30
the the grade column is has a fixed domain， meaning, it’s a B C D or E ,I don’t think see me a S does it in this place,right
这里的grade列是一个固定的字段，它里面的值是A，B，C，D或者E，这里应该没有S
~~42:29 – 42:30~~
~~You have an completes, right~~

42:30 – 42:31
But it’s fixed
但它是固定的
42.31–42.34
oh there’s another S one that’s whatever
它这里面可能还有一个S，但不管了
42:34 – 42:39
The problem is when I go putting your grades, I can’t tell whether you’re an undergrad or graduate student
现在问题在于，当我去填你们的分数的时候，我没法说出你是一个本科生还是一个研究生
42:39 – 42:42
So I’m like oh this student got you know did awesome to get an A+
但我会这么说，这个学生很牛逼，拿到了一个A+的成绩
42:42 – 42:43
But then it throws an error
但之后这里就抛出了一个错误
42.43–42.50
because they’re an undergrad, undergrads can‘t get A+, unless you’re ECE which I think you can,it’s a nightmare, but anyway
因为他们是本科生，本科生不可能拿到A+。除非你是ECE专业的学生，那么我觉得你可以拿到A+，虽然这算是一种噩梦对你们来说
42:50 – 42.52
So his question is all right
So，他的问题是
42.52–43.01
so couldn’t I have some kind of sid table that has a tally that keeps track of every single time I inserted a tuple with one of these values
~~So，这里我能否通过一个sid表来跟踪每次我插入tuple时它里面的值?~~
So，我有一张sid表，我是否可以有一个东西用来跟踪往表中插入的每一个tuple,
43:01 – 43:04
I’m trying to maintain a counter number I increment that counter by one
我会试着维护一个counter，每插入一次，counter值就加一
43:14 – 43:15

Within a page
是在一个page内
43:25 – 43:29
All right, so what he’s saying is
So，他所说的是

SORTING AGGREGATION

```
SELECT DISTINCT cid
  FROM enrolled
 WHERE grade IN ('B','C')
 ORDER BY cid
```

enrolled(sid,cid,grade)

sid	cid	grade
53666	15-445	C
53688	15-721	A
53688	15-826	B
53666	15-721	C
53655	15-445	C

Filter

sid	cid	grade
53666	15-445	C
53688	15-826	B
53666	15-721	C
53655	15-445	C

Remove Columns

cid
15-445
15-826
15-721
15-445

Sort

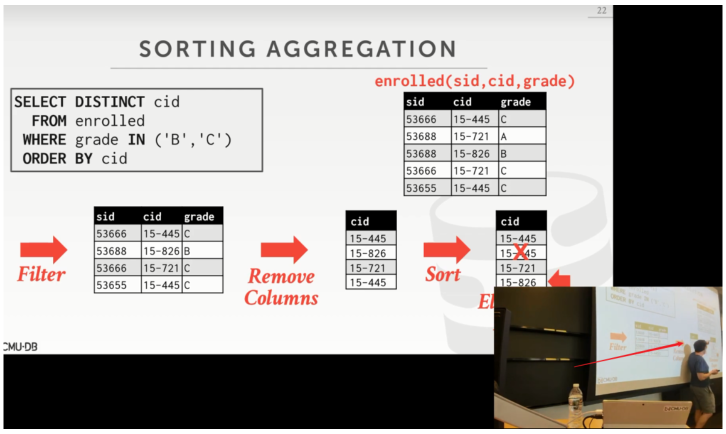
cid
15-445
15-445
15-721
15-826

El

MU-DB

43.29–43.31
say this was stored in a page
他表示，这张表是存储在一个page内的
43:33 – 43:35
This this little example here is in one page
因为这是一个小例子，所以它是放在一个page内的
43:36 – 43:37
And then for the grade column
接着，对于这个grade列来说
43.37–43.41
 I could keep track of the Min and Max value ,so this case B or C
我可以去跟踪它里面的最大值和最小值，在这个例子中是B和C（这个范围是我们自己定的）
43:42 – 43:45
So now if I’m say I’m looking for all people that have the grade A
So，假设现在我想去找到所有成绩为A的人
43:45 – 43:47
If I get to that page
如果我拿到这个page
43.47–43.50
and I look say oh why it’s only between B and C
为什么上面的人成绩只有B和C呢?
43.50–43.52
because nobody has an A in that page
因为在这一个page上，没有人拿到A
43.52–4.357
I don’t mean even bother looking at the column that’s what you’re saying, right
我甚至都不需要去看这一列了，这是不是就是你要说的?
43:57 – 44:02
Okay, I I think we are talking about the same thing what you’re describing called zone maps
Ok，我觉得我们讨论的是一回事，你所描述的东西叫做Zone Maps
44:02 – 44:08
Right, well we will talk about this I think next week or this week I forget when
Well，我们会在这周或者下周来讨论这个，具体看时间来定
44:08 – 44:14

But basically there’s a way to keep track of yourself on auxiliary data structure on the side that he looked at that first
基本上来说，你可以通过设定一个辅助数据结构来让你知道这是否需要在这个page上去遍历
44:15 – 44:17
And then you check the page, yes
然后，你就会去检查这个page
44:17 – 44:18
So that’s a zone map
So，这就是一个Zone map
44.18–44.24
you could or could not be in the same page you could have a separate page ,but within the page
Zone map可以在同一个page中，也可以在多个分离的page中，但在这个page中
44:24 – 44:29
But it’s basically a precomputed information to say the data you here’s the range of data that could possibly exist for each attribute
但简单来讲，这里预先计算好的信息会告诉你每个属性可能存在的数据范围是多少
44:29 – 44:33
And you refer to that first and make decisions whether you didn’t even go further
然后，你会先通过它看一看，然后再决定是不是要进行更进一步操作
44:33 – 44:38
Yes, so those are called zone maps, they’re called pre, Oracles called zone maps
So，这些叫做Zone maps，在Oracle中是这么叫的
44:38 – 44:46
Tremendous a call a pre computed pre-compute materialized aggregation sometimes, different systems do different things
有时这也叫做 pre-compute materialized aggregation，不同的系统有不同的叫法
44:46 – 44:48
But that does exist, we’ll cover that later, yes
这个属于我们要讲的范畴，之后我们会对它进行介绍
44:55 – 44.55
That’s an index,
这是一个索引
44.55–44.58
right, that’s what it index does
这是索引所做的事情
44.58–45.02
you talk about how my something more fine like like not an index, I think right
我觉得你更多讨论的并不是索引方面所做的事情
45:03 – 45:05
Yeah, that’s a zone map when you’re describing is in index
这其实是Zone map所做的事情，但你把它当作索引来描述了
45:08 – 45:13
And again the the beauty of a declarative language like SQL is that
拿SQL来讲，这种声明式语言的美妙之处在于
45.13–45.15
 I write My SQL query like this
比如我写了这样一个SQL查询
45:15 – 45:17

I don’t know whether I’m using zone maps
我不清楚我是否要去使用Zone Map
45.17–45.19
I don’t know whether I’m using an index
我也不知道我要不要去使用索引
45.19–45.20
I don’t care
我并不清楚这些
45.20–45.24
 the databases will figure out what’s the best strategy for me to go find the data that I want
数据库会帮我们弄清楚查找我想要的数据的最佳策略
45:24 – 45:31
Right, so just trying to try never move is is crap quickly as possible, that’s the whole goal of all this
它会尽可能快的放弃那些无须去看的策略，这就是它的目标所在
45:34 – 45:37
Alright, so that was a change about zone maps we’ll cover that later
我们会在后面去涉及zone maps

22

SORTING AGGREGATION

```
SELECT DISTINCT cid
  FROM enrolled
 WHERE grade IN ('B','C')
 ORDER BY cid
```

enrolled(sid,cid,grade)

sid	cid	grade
53666	15-445	C
53688	15-721	A
53688	15-826	B
53666	15-721	C
53655	15-445	C

Filter

sid	cid	grade
53666	15-445	C
53688	15-826	B
53666	15-721	C
53655	15-445	C

Remove Columns

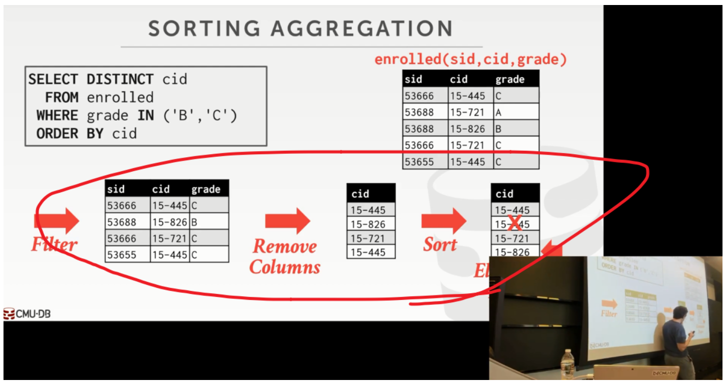
cid
15-445
15-826
15-721
15-445

Sort

cid
15-445
15-445
15-721
15-826

El

CMU·DB
45:39 – 45:41
The main point I want the main takeaway from this was
我们从此处可以学到的重点是
45.41–45.45
if I’m sorted， I do one pass and I can eliminate the duplicates
如果我花了一轮时间来排序，那么我就可以消除重复项
45:46 – 45:48
All right, in this example here
在这个例子中
45.48–45.52
 this worked out great for us, because the output need to be sorted on a course ID
这种做法对我们来说很棒，因为我们需要根据course id来对输出进行排序



SORTING AGGREGATION

```
SELECT DISTINCT cid
  FROM enrolled
 WHERE grade IN ('B','C')
 ORDER BY cid
```

enrolled(sid,cid,grade)

sid	cid	grade
53666	15-445	C
53688	15-721	A
53688	15-826	B
53666	15-721	C
53655	15-445	C

Filter

sid	cid	grade
53666	15-445	C
53688	15-826	B
53666	15-721	C
53655	15-445	C

Remove Columns

cid
15-445
15-826
15-721
15-445

Sort

cid
15-445
15-~~445~~
15-721
15-826

45:52 – 45:59

So I was it was two for one, I did my sorting, because that's the output I needed ,but then I'm also in the sort order I need for my output

so 我需要对这两步得到的结果进行排序，得到的输出就是我想要的结果

46:00 – 46:00

Right

46:01 – 46:02

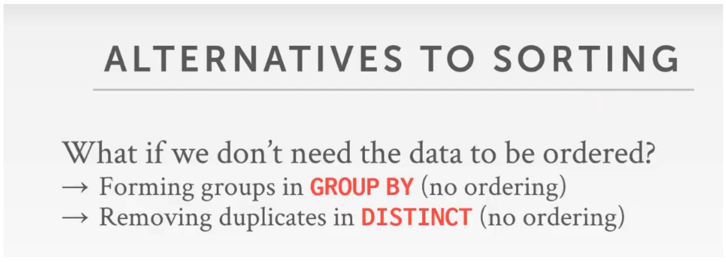
So in this case here

So，在这个例子中

46.02–46.06

doing a sorting based aggregation is a definite win for us

这就是我们完成基于这个聚合函数所做的排序



ALTERNATIVES TO SORTING

What if we don't need the data to be ordered?
→ Forming groups in **GROUP BY** (no ordering)
→ Removing duplicates in **DISTINCT** (no ordering)

46:06 – 46:08

But in many cases

但在许多例子中

46.08–46.09

we don't actually need the output to be sorted

实际上，我们并不需要排好序的输出结果

46:11 – 46:11

Right

46:12 – 46:14

So again you still can use sorting for this

So，你依然可以对输出结果进行排序

46.14–46.18

like you can do for group by and and and doing distinct stuff

这里，你可以使用GROUP BY，也可以使用DISTINCT

46:18 – 46:20

But if you don't need to be sorted

但如果你不需要排序

46.20–46.21

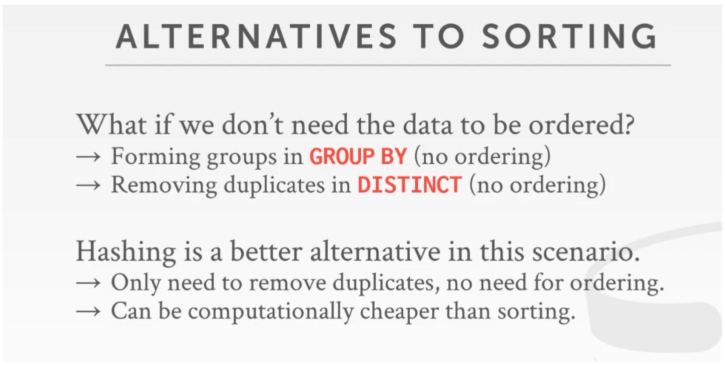
then this actually might be more expensive

那么这实际上代价会更加昂贵

46.21–46.24

because again the sorting process itself is not cheap

因为它自身排序过程所付出的代价并不低（知秋注：这种方式下，GROUP BY与DISTINCT内部也是会进行排序操作的，如果事先做好排序了，也就不需要进行它们内部这些排序操作了）



ALTERNATIVES TO SORTING

What if we don't need the data to be ordered?
→ Forming groups in **GROUP BY** (no ordering)
→ Removing duplicates in **DISTINCT** (no ordering)

Hashing is a better alternative in this scenario.
→ Only need to remove duplicates, no need for ordering.
→ Can be computationally cheaper than sorting.

46:25 – 46:27

So this is where hashing can help us

So，对此场景，hashing能帮助我们

46:29 – 46:36

So hashing is a way for us to be able to sort of again another divide and conquer approach

So，hashing是另一种分治的方法

46.36–46.38

where we can split up the data set

通过它，我们可以对数据集进行拆分

46.38–46.45 ******

and guide the tuples or the keys that were examining to particular pages

并将正在检查的tuple或key引导到特定page中

46:45 – 46:48

And then do our processing in memory on those pages

然后，在内存中对这些page进行我们想要的处理

46:49 – 46.55

But again hashing removes all all locality all any sort ordering

但hashing这种方式会移除所有的局部性，以及排序顺序

46.55–46.58

because it's taking any key and do you know doing some hash function on it

因为它会拿到key，然后对该key进行hash处理

46.58–

 and now it's going to jump to some random location

接着，它就会跳到某个随机位置

47:01 – 47:03

So this works great

So，这种方式很棒

47.03–47.03

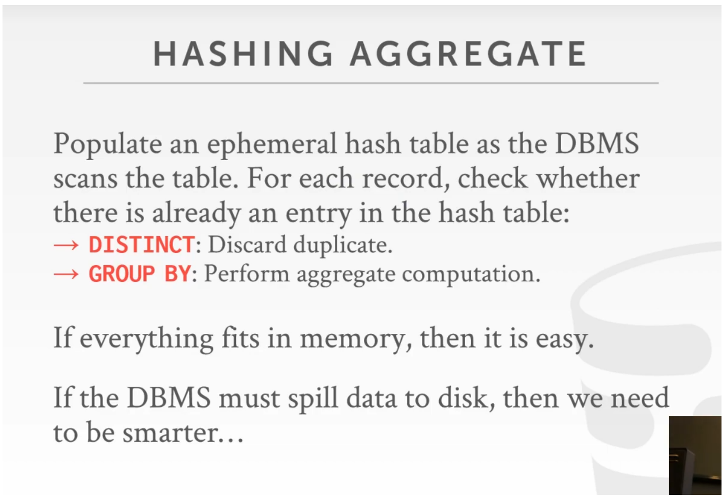
if we don't need sorting

如果我们不需要排序

47.03–47.06

 we don't think don't need things to be ordered

那么，我们就不需要让这些数据是有序的了


HASHING AGGREGATE

Populate an ephemeral hash table as the DBMS scans the table. For each record, check whether there is already an entry in the hash table:
→ DISTINCT: Discard duplicate.
→ GROUP BY: Perform aggregate computation.

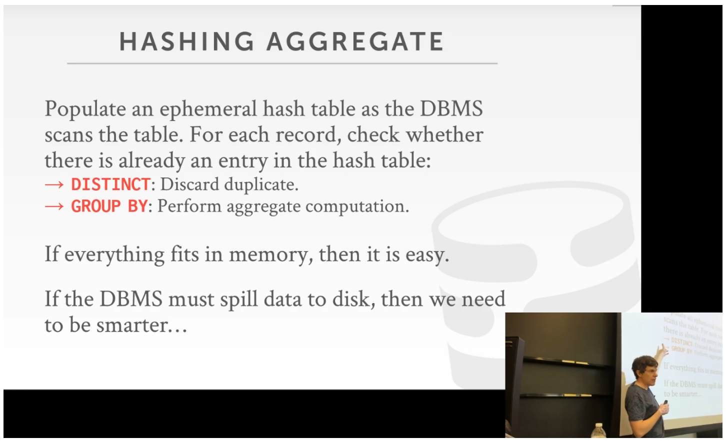
If everything fits in memory, then it is easy.

If the DBMS must spill data to disk, then we need to be smarter…

47:07 – 47:09
So the way we can do a hashing aggregate is
So，我们进行hashing聚合操作的方法是
47.09–47.17
we're gonna populate an ephemeral hash table, as the database system of some scans the table or scans whatever our input is
~~当DBMS对表进行扫描时，我们会把这些输入填充到一个临时的hash table~~
我们会将从DBMS对表进行扫描所得到的输入，填充到一个临时的hash table中
47:17 – 47:22
Hey and then say we you know when we do our lookup
当我们进行查找时
47.22–47..24
depending on what kind of aggregation we're doing
取决于我们所做的聚合操作类型
47:25 – 47:28
If we do an insert and the key is not there, then we populate it
如果在插入一个key时，key并不在里面，那么我们就将它填充到这个临时hash table中去
47:28 – 47:30
If it is there
如果key在那里的话
47.30–47.35
then we may want to modify it ,or modify its value to compute whatever the aggregation that it is that we want
那么，我们可能会想去对它进行修改，或者对它的值进行修改，以此来计算出我们想要执行的聚合操作的结果
47:35 – 47:37
Right, for distinct
在DISTINCT中
47.37–47.40
it just a hash see whether it's in there
它是通过hash的方式来看这个key是否在里面
47.40–4742
if it's is,then I know it's already I it's a duplicate
如果它在里面，那么我就知道这是一个重复的key
47.42–47.44
so I don't bother inserting it
So，我不用再将它插入了

47:44 – 47:45
For the group by queries
在用到GROUP BY的查询中
47.45-47.47
 for the others other aggregations
以及其他聚合操作来说
47.47-47.52
 you may have to update a running total and what we'll see an example of this
你们可能会去更新RUNNING_TOTAL，之后我们会看到关于这个的一个例子
47:52 – 47:56
So this approach is fantastic if everything fits in memory
So，如果所有数据都能放在内存中进行处理，那么这种方法就很棒
47:58 – 48:00
So the key thing I'm saying up above
So，我之前就说过一个很重要的事情
48.00-48.03
 I'm saying it's an ephemeral hash table not an emery hash table
我说过，它是一个临时（ephemeral ）的hash table，它并不是一个永久的hash table
48:03 – 48:05
So ephemeral or transient means that
So，ephemeral或transient（短暂）意味着
48.05-48.08
 this is a hash table I'm gonna build through my one query
我所构建出的这个hash table只用于这个查询中
48:09 – 48:11
And then when that query is done I throw it all away
当这个查询结束的时候，那我就可以把它给扔了
48.11-48.13
I'm gonna do this for every single query
我可以对每个查询都做这种事情
48:14 – 48:16
I said that we said in the very beginning
我在一开始的时候就说了
48.16-48.18
we use data structures in different ways of the database system
我们会在数据库系统的各个方面使用数据结构
48.18-48.20
so there's the example of a transient data structure
So，这就是一个关于临时数据结构的案例
48:20 – 48:23
I need it for just my one query, I do whatever I want then I throw it away
我只会在一个查询中使用这个临时数据结构，当我做完我要做的事情，我就可以将这个数据结构扔了
48:24 – 48:25
So if everything is in memory
So，如果所有东西都在内存中
48.25-48.26
the hash tables fantastic
那么使用hash table就会很棒

48.26–48.31
because it’s O(1) lookups to go update things
因为它在查找和更新这方面的复杂度是O(1)



HASHING AGGREGATE

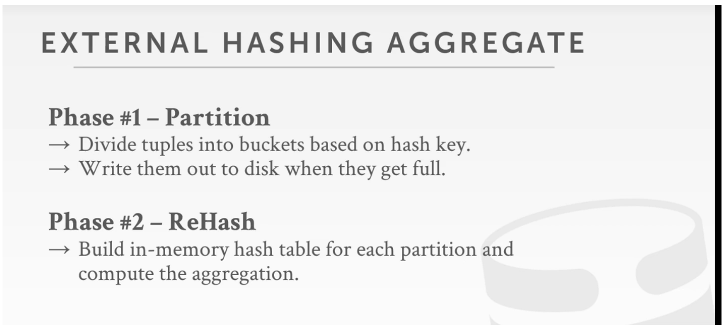
Populate an ephemeral hash table as the DBMS scans the table. For each record, check whether there is already an entry in the hash table:
→ DISTINCT: Discard duplicate.
→ GROUP BY: Perform aggregate computation.

If everything fits in memory, then it is easy.

If the DBMS must spill data to disk, then we need to be smarter…

48:31 – 48:33
Right, in this case you’re also not doing deletes
在我这个幻灯片上的例子中并没有做删除操作
48:34 – 48:36
Right, it’s just inserting things or updating things
这里只涉及了插入和更新操作
48:37 – 48:40
If we need to spill a disk though
如果我们需要将数据溢出到磁盘
48.40–48.41
now we’re screwed
那我们就完蛋了
48.41–48.44
because now that this random randomness is gonna hurt us
因为这种随机性对我们来说很糟糕
48:44 – 48:48
Because now I’m jumping around to different pages or blocks in my hash table
因为我现在要跳到我的hash table中的不同page或者是block中
48.48–48.50
and each one could be incurring an I/O
每跳转一次可能都会引起一次I/O
48:51 – 48:53
So we want to be a bit smarter about this
So，对此，我们需要点更加聪明的处理方式
48.53–48.58
and trying to maximize the amount of work we can do for every single page we bring into memory
我们要试着最大化我们对每个放入内存中的page所做的工作量
49:00 – 49:02
So this is what external hashing aggregate does
So，这就是external hashing聚合操作所做的事情
49:03 – 49:08
And it’s again the high level is the same way as the same technique that we did for external merge sort
从高级层面来看，这和我们在外部归并排序中用到的是相同的技术
49.08–49.11

it’s a divide and conquer approach
这是一种分治策略

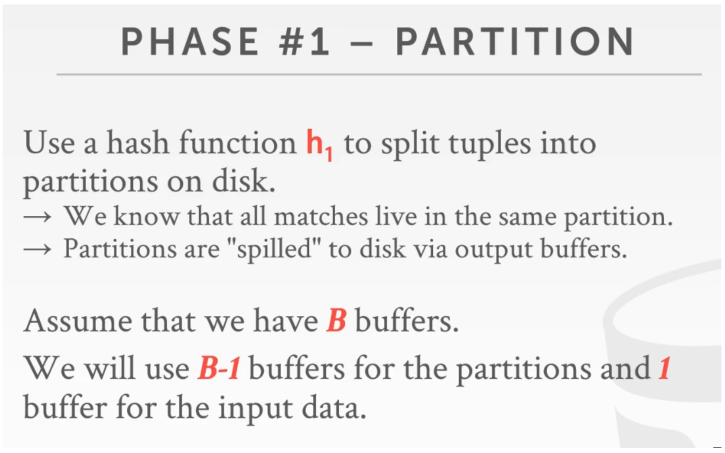

EXTERNAL HASHING AGGREGATE

Phase #1 – Partition
→ Divide tuples into buckets based on hash key.
→ Write them out to disk when they get full.

Phase #2 – ReHash
→ Build in-memory hash table for each partition and compute the aggregation.

49:11 – 49:13
So the first thing we’re to go through to take a pass through our data
So，首先我们要做的就是传入我们的数据
49:14 – 49:16
And we’re to split it up into a partition into buckets
然后，我们将数据拆分开来，并放入一个个bucket中
49.16–49.26
where so that all the tuples that are either the same that all tuples that are the same had the same key will land in the same partition
所有具有相同key的tuple都会被放在同一个分区中

49:27 – 49:30
And then we go back through in the second phase
然后，我们进入第二个阶段
49:30 – 49:32
And now for each partition
现在，在每个分区中
49.32–49.33
 we’re gonna build an in–memory hash table
我们会去构建一个内存中的hash table
49.33–49.38
that we can then do whatever it is that the aggregation that we want to do
然后，我们就可以进行我们想做的任何聚合操作了
49:38 – 49:39
Then we produce our final output
然后，我们生成出我们的最终输出结果
49.39–49.42
throw that that a memory hash table away
并将这个内存中的hash table给扔掉
49.42–49.43
 and then move on to the next partition
接着就去处理下一个分区
49:46 – 49:48
Ok, we’re maximizing the amount of sequential I/O that we’re doing
Ok，这样我们就最大化了我们所做的循序I/O的工作量
49:49 – 49:52
And for ~~every single page we~~ every single I/O we have to do to bring something into memory
在进行每次I/O的时候，我们必须将数据放入内存

49:53 – 49:57
Then we do all the work we need to do on that one page before we move on to the to the next ones
接着，在我们移动到下一个page之前，我们会在当前该page上做所有我们需要做的事情
49:58 – 50:00
So we never again never we never have to backtrack
So，我们永远不需要进行回溯（backtrack）


PHASE #1 – PARTITION

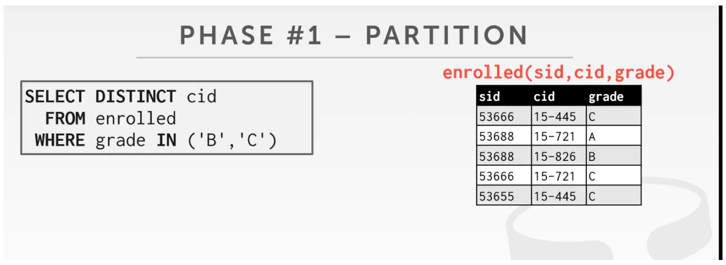
Use a hash function h_1 to split tuples into partitions on disk.
→ We know that all matches live in the same partition.
→ Partitions are "spilled" to disk via output buffers.

Assume that we have *B* buffers.

We will use *B-1* buffers for the partitions and *1* buffer for the input data.

50:02 – 50:03
So let’s go through these two phases
So，我们来看下这两个过程
50:04 – 50:05
So in the first phase again
So，在第一个阶段中
50.05–50.07
 what we’re trying to do is we’re going to split the tuples up into partitions,
我们所试着做的事情就是将这些tuple拆分到不同的分区中去
50.07–50.10
that we can then write out the disk as needed
根据需要，我们可以将其写到磁盘中
50:11 – 50:13
So we’re gonna use our first hash function it’s just to split things up
So，我们要去使用我们的第一个hash函数来将数据进行拆分
50.13–50.19
,and again we use murmurhash ,city hash, xx hash 3 whatever it doesn’t matter
我们可以去使用Murmurhash，CityHash，XXHash 3，但不管用什么都无所谓
50:19 – 50:21
And so the reason why we’re doing this is that
So，我们之所以这样做的原因是
50.21–50.25
because our ~~hash table~~ hash function is deterministic
因为我们的hash函数是确定性的
50:25 – 50:29
meaning the same key will always be given the same hashed value output
意味着，对同一个key进行hash所得到的hash值永远是一样的
50:29 – 50:33
that means that tuples that have the same key will land in the same partition
这意味着具有相同key的tuple会落在同一个分区
50:34 – 50:39

And we don’t need to hunt around for other parts of the the tablespace at the table to find the same key
我们不需要去表中其他的表空间去寻找具有相同key的tuple
50.39-50.43
 they’re always gonna be in our one partition
这些具有相同key的tuple始终会在同一个分区中
50:43 – 50:47
Our partitions can just spill to disk using the buffer manager, when they get full
当这些分区存满了之后，我们可以通过buffer管理器，将它们写出到磁盘上
50:47 – 50:52
So so we have a page that we’re storing the the current partition data
So，我们通过一个page来保存当前分区中的数据
50.52-50.53
 when that gets full
当这个page满了
50.53-50.56
 we just write that out to disk and start filling in the next page
我们会将这个page写出到磁盘上，并开始填充下一个page
50:57 – 50.58
So in this case here
So，在这个例子中
50.58-51.00
we’re gonna assume we have B buffers
我们假设我们有B个buffer
51.00-51.03！！！！
 and we’re gonna use B-1 buffers for the partitions
然后，我们使用B-1个Buffer用来保存分区内容
51.03-51.05
 and at least 1 buffer for the input
并使用至少1个Buffer来保存输入
51:07 – 51:09
So I’m gonna bring in one page from my table
So，我会从我的表中拿到一个page
51:09 – 51:12
And I’m going to ask sequential scan on that page look at every single tuple
然后，我会对该page进行循序扫描，以此来查看每个tuple
51:12 – 51:16
And then it’s gonna write it out to B-1 partitions
接着，将它们写出到这B-1个分区中
51:17 – 51:21
Alright, because you need to have at least 1 buffer in memory for each partition
因为对于每个分区来说，你必须至少使用内存中的1个buffer来保存它
51.21-51.21
yes
请问
51:25 – 51:32
So if say I’m doing I’m gonna doing a group by on the course ID
So，如果我在Course id上使用Group By



PHASE #1 – PARTITION

```
SELECT DISTINCT cid
  FROM enrolled
 WHERE grade IN ('B','C')
```

enrolled(sid,cid,grade)

sid	cid	grade
53666	15-445	C
53688	15-721	A
53688	15-826	B
53666	15-721	C
53655	15-445	C

51.32–51.35

here next slide

看这个幻灯片

51:35 – 51:39

I'm doing a group by on the course ID, I'm doing aggregation

我在Course id上进行了Group By，然后做聚合操作

51:40 – 51:44

So I'm gonna hash this course ID for every single tuple

So，我会对每个tuple的course id进行hash处理

51:44 – 51:47

If I had the same course ID, it's gonna Lane that in the same partition, so it's gonna live there

如果是相同的course id，我就会让这些具有相同course id的tuple都放在同一个分区中

51:48 – 51:50

Right, reside live stored

就放在同一个分区中

51:51 – 51.55

And then that way when I want to go now do that in this case the duplicate elimination,

在这个例子中我想做的是去重

51.55–51.56

when I come back the second time,

当我再回来的时候

51.56–52.02

 I know that the the tuples that go have the same key has to be in the same partition

我知道具有相同key的tuple会被放在同一个分区之中

52:02 – 52:04

There's not gonna be some other random place

它们不可能会放在其他的随机位置

52:06 – 52:07

His question is

他的问题是

52.07–52.11

~~it's partition to page no~~ partition would be like it's a logical thing

这个分区其实看起来有点像是一个逻辑上的东西

52.11–52.16

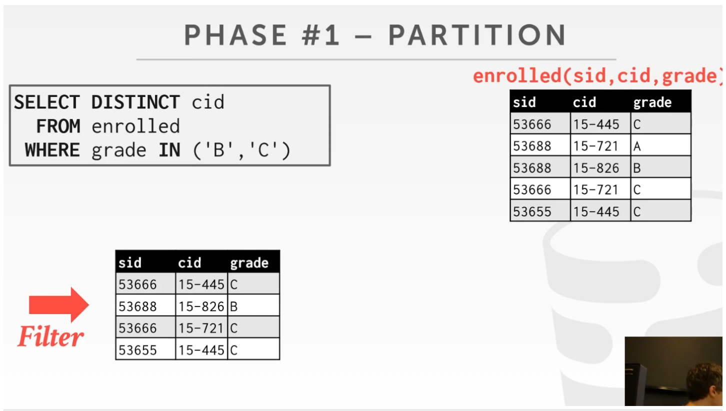
take the hash value modded by the number of partitions and that were you write into

我们对key进行hash处理（通过使用分区的数量来对key进行取模）

52.16–52.18

 and these partition can have multiple pages

这些分区可以使用多个page



PHASE #1 – PARTITION

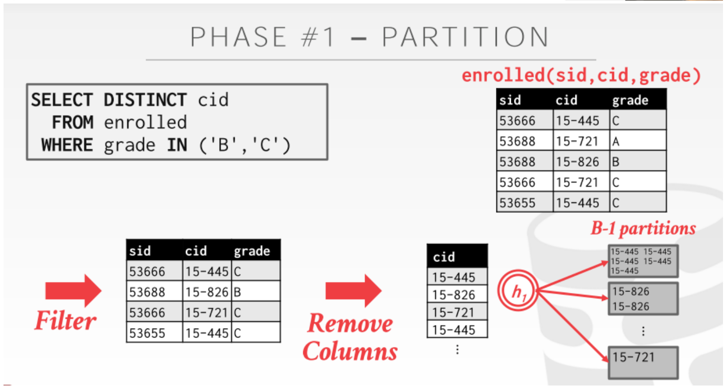
```
SELECT DISTINCT cid
  FROM enrolled
 WHERE grade IN ('B','C')
```

enrolled(sid,cid,grade)

sid	cid	grade
53666	15-445	C
53688	15-721	A
53688	15-826	B
53666	15-721	C
53655	15-445	C

Filter

sid	cid	grade
53666	15-445	C
53688	15-826	B
53666	15-721	C
53655	15-445	C



PHASE #1 – PARTITION

```
SELECT DISTINCT cid
  FROM enrolled
 WHERE grade IN ('B','C')
```

enrolled(sid,cid,grade)

sid	cid	grade
53666	15-445	C
53688	15-721	A
53688	15-826	B
53666	15-721	C
53655	15-445	C

Filter

sid	cid	grade
53666	15-445	C
53688	15-826	B
53666	15-721	C
53655	15-445	C

Remove Columns

cid
15-445
15-826
15-721
15-445
⋮

52:21 – 52:25

Alright, so again we do our filter do as we did before, we remove our projection columns

So，在此我们进行以前做过的过滤操作，将其他列移除掉

52:26 – 52:31

And then now we take our all the output of here, we’re gonna run it through our hash function

然后将这里我们过滤出的值用第一个hash函数进行hash处理

52:31 – 52:34

And we write it out to the partition pages

然后我们将数据写入到它所对应的分区中的page里面

52:35 – 52:36

So in this case here，I’d B–1

So，在这个例子中，应该有B–1个分区

52.36–52.40

so say there’s like four or five ,I’m showing three here

So，这里应该有4或5个分区，但我只展示了3个

52:40 – 52:46

So all the 15–445 keys land here,all the 15–826 land here ,at 15–721 lend here

So，所有key为15–445的数据会落在这里，15–826会落在第二个分区，15–721会落在第三个分区

52:47 – 52:50

So again you could be smart about this

So，你们应该可以聪明的处理这个

52.50–52.52

and say alright well I know I’m doing doing distinct

Well，这里我使用了DISTINCT去重

52:52 – 52:53

So within my page

So，在我的page中

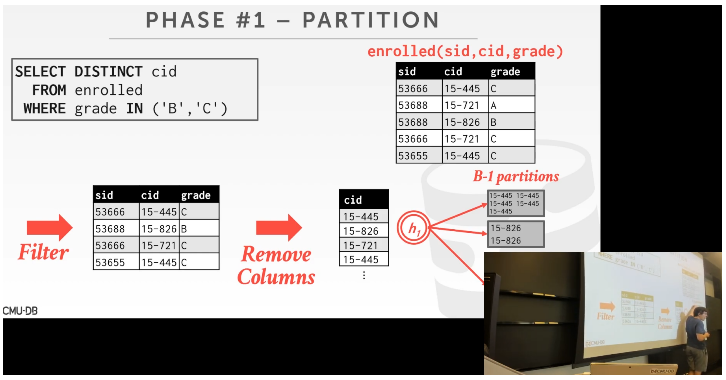
52.53–52.58

if I see the same thing then don't bother putting it into it
如果我看到了相同的东西，那么我就不用费心去将它放进这个分区了
52:58 – 52.59
But for simplicity reasons
但为了方便起见
52.59-53.01
we're blindly just putting it in
我们就把这些东西都塞进去就行了
53.01-53.02
yes
请讲
53:05 – 53:07
Question is what is it partition
她的问题是，分区是什么
53:08 – 53:15
You can think of like a partition is thinking like it's like the the bucket chain and the chain hash table
你可以将一个分区当成一个bucket chain，或者是chain hash table（知秋注：参考Java中hashmap通过Entry这个接口实现的node所做chain）
53.15-53.19
~~you just have~~ within a chain you could have multiple pages
在一个chain中，你可以有多个page
53:19 – 53:22
But I only have one page in memory as I'm populating this
但当我进行填充数据时，在内存中我只有一个page
53:23 – 53:27
Because again for everything every single time I'm gonna cache something and insert it into this
因为每次我进行缓存并往缓存里面插入数据时
53.27-53.29
I'm only inserting into one page
我只会往一个page中插入数据
53.29-53.30
and when this gets full
当这个page满了的时候
53.30-53.32
I guess again written out to disk
我会将这个page写出到磁盘
53:32 – 53:35
And I now allocate another one that I start filling up
现在，我就得再分配另一个page，然后往里面填充数据
53:35 – 53:39
So within memory why I'm doing this first phases, I only need B-1 pages
在内存中，执行第一阶段的时候，为什么我只需要B-1个page呢?
53.39-53.40
because I'd B-1 partitions
因为我的分区数量是B-1
53.40-53.51

what is the number of distinct course id is larger than the number of buffer in the memory?
如果去重后的course id的数量大于内存中buffer的数量会怎么样?
53:51 – 53:55
So this question is what if the number of distinct course IDs
So，他的问题是如果去重后的course id数量.......
53.55–54.00
I dont have enough buffer for the distinct course id. So。。。
我并没有足够的buffer来保存去重后的course id。So。。。
53:59 – 54:01
You do, because you’re hashing it, right
你有啊，因为你对它做了hash处理
54:02 – 54:06
~~You’re dating – mod to~~ take this hash value mod by B–1
因为你对key进行了hash处理，使用B–1对它进行取模
54:07 – 54:10
So in this example here I’m only showing three distinct keys
So，在这个例子中，我只展示了3个去重后的key
54:11 – 54:12
But like I have another class
但比如我有另一门课
54.12–54.17
15–410 back at land in the same bucket as 15–445
15–410这门课的tuple也落在了跟15–445同一个bucket中
54:18 – 54:21
I don’t need to have a partition for every distinct key
我不需要用一个分区来保存每个去重后的key所对应的数据
54:22 – 54:25
The hashing allows them to go into the same thing
hash处理会允许它们都落到一个位置上
54:27 – 54:29
Your face looks like you’d like this confused by this
从你的表情我可能看出你对于这个很困惑
54:31 – 54:36
Right, again so I have 15–410, I’m gonna hash it,
再说一遍，我有15–410，我对它进行hash处理
54.36–54.40
I’m mod on it by B–1 ,it lands in partition 0
我使用B–1对它进行取模，它落在分区0上面
54:41 – 54:44
And so I just append it to this to this page
So，我将它追加到这个page上
54:45 – 54:50
Right, and then the main thing is that 15–410 can’t exist in any other page,
接着，现在的主要问题在于15–410不能存在于其他任何page中
54.50–54.57
because the hash function always guarantee that it’s always gonna point to this one
因为hash函数始终保证15–410指向的始终是分区0这个位置
54:57 – 55:00

If the current page with this partition overflows
如果该分区的当前page溢出了
55.00*-55.02
,I write it out the disk,
我就会将这个page写出到磁盘
55.02-55.04
and I allocate a new page and start filling that up
然后分配一个新的page，并对它开始填充数据
55:10 - 55:13
Yeah, flush the page allocate a new one, yes
没错，将page刷到磁盘，并分配一个新的page
55:15 - 55:19
And again like at this phase all we're doing is this partitioning
再说一遍，在这个阶段，我们所做的就是分区
55:19 - 55:23
So I don't care like I can be smart and say oh I'm doing duplicate elimination,
So，我可以这么说，我正在进行去重操作
55.23-55.25
 I know I already have 15-445 over here,.
我知道我在那里已经有了15-445
55.25-55.26
 I put it in
并且我将它放在了这个分区里
55.26-55.27
 ignore that for now
现在先将它忽略
55:27 - 55:33
Right,it's just I'm blindly putting things into this to the pages and writing them out
现在我只是盲目的将东西放到page中，并将page写出到磁盘
55:38 - 55:38
Yes
没错
55:42 - 55:43
Yes so so this question is
So，他的问题是
55.43-55.50
it's getting written out the disk where am I storing the metadata that says oh partition 0 has these pages
~~当page写出到磁盘的时候，我该将元数据放在哪，该元数据表示分区0中有这些page~~
当page写出到磁盘的时候，我该将元数据放在哪，我会说，oh，分区0中有这些page
55:50 - 55:53
You had that in memory data structure
~~在内存中的数据结构里，你会有这个元数据~~
你可以通过内存中的数据结构来获知
55.53-55.56
you keep track of like partition 0 here's the page 4
~~当你查看这个元数据时，你会看到，比如在分区0中，这里是page 4~~
当你查看后，你会看到，比如在分区0中，这里是page 4
55.56-55.58

 partition 1 here’s the pages for it
然后，这里是属于分区1的page
55:58 – 56:00
But that’s small right that’s like that’s nothing
~~但这个元数据很小，几乎不占什么地方~~
但这个内存数据结构很小，几乎不占什么地方
56:03 – 56:05
This question is are we not considering collisions
他的问题是我们要不要考虑hash碰撞
56.05–57.07
we don’t care at this point
现在我们并不用关心这个
56:09 – 56:09
Right

56:10 – 56:14
~~It’s in actually~~ maybe use another table than distinct ~~maybe that’s following people~~
实际上我们可能会去用另一个表
56:15 – 56:17
But if I’m doing a you know a count
但如果我做的是COUNT
56:19 – 56:21
Again, you can do that more efficiently as well
再说一遍，你可以做的更加高效

PHASE #1 – PARTITION

```
SELECT DISTINCT cid
  FROM enrolled
 WHERE grade IN ('B','C')
```

enrolled(sid,cid,grade)

sid	cid	grade
53666	15-445	C
53688	15-721	A
53688	15-826	B
53666	15-721	C
53655	15-445	C

sid	cid	grade
53666	15-445	C
53688	15-826	B
53666	15-721	C
53655	15-445	C

cid
15-445
15-826
15-721
15-445
⋮

56.21–56.24
but like like I don’t care putting in inside of this,
但我并不在意将东西放在这里面
56.24–56.25
I don’t care this collision scope,
我也不在意碰撞范围
56.25–56.28
 because I’m gonna resolve that in the second phase, when I rehash things
因为我会在第二个阶段的时候解决这个问题，即当我进行重新hash的时候
56:35 – 56:39
Your question is where is this number coming from B–1
你的问题是，B–1这个数字是哪来的
56:40 – 56:43
So that’s the database system telling this query
So，这个数字是由数据库系统告诉这个查询的
56.43–56.45

that's whatever thread or worker that's existing these queries,
不管这些查询中有多少线程或者是worker（知秋注：在线程池中我们会包装worker用于任务调度）
56.45-56.48
 you have this amount of memory to use for query processing
在查询处理这块，你所使用的内存量得有这么多
56:57 – 57:06
Yeah, so like the database system says you're allowed to have B equals 100 pages to do whatever you want to do for execute the query to execute this algorithm
So，数据库系统表示，它允许你使用B个buffer（总共包含100个page）来执行你想在该查询或算法中所做的事情
57:06 – 57:11
~~I'm gonna use B-1 to store my part~~ I'll B-1 partitions
我会使用B-1个分区
57.11-57.12
 because these partition will have one page
因为每个分区都会有一个page
~~57:15 – 57:19~~
Yeah, it sucks yeah your question is if B is really small---,yes
你的问题是，如果B真的很小的话，确实会发生这种情况
57:21 – 57:22
Right, there's nothing you can do
对此，你无能为力
57.22-57.27
~~it's not it's not like you might you know~~ you can't magically just add more memory right justifying that resource
你没法无中生有地去添加更多内存，你只能去调整资源
57:28 – 57:35
The database system you know is the is is doing resource management it's deciding, oh I have a lot of queries that need to execute at the same time
数据库系统会去进行资源管理，比如：它会说，在同一时间它需要去执行大量的查询
57:35 – 57:37
So therefore I can't let them all have a lot of memory
因此，我不能让它们都使用大量的内存
57:38 – 57:41
So this gets into the tuning side of things which is actually very difficult as well
So，这就属于调优方面的内容了，实际上这块内容也非常难
57:42 – 57:42
Yes
请讲
57:51 – 57:53
Ok so he says
So，他表示
57.53-57.55
and I don't have slides with this we'll do it next class
尽管我并没有关于他所提问问题的幻灯片，但这个我们下节课会看到
57:55 – 57.57
He said that
他表示

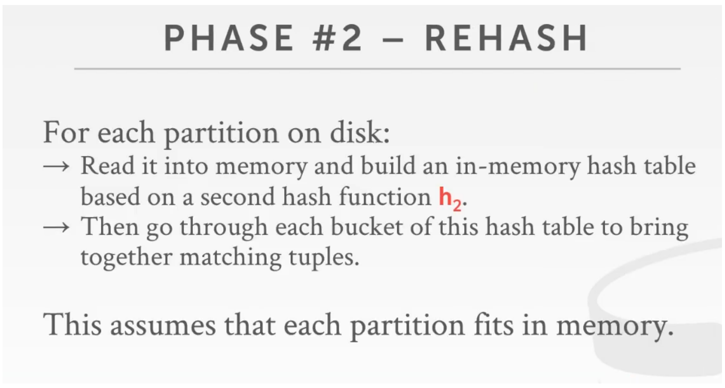
57.57-58.01
you’re screwed let me rephrase what you said
你这说的有点混乱，让我重新组织下你说的东西
58.01-58.06
~~you’re screwed are you better~~ if everything hashes is this bucket
如果所有的东西都hash到了这个bucket中
58:07 - 58:12
So say this is this is most popular course on Campus everyone’s taking 15-445, right
So，比方说， 15-445是校内最热门的课程，每个人都学这门课
58:13 - 58:16
Then as I hash everyone lands there then I’m screwed,right
然后，我将所有数据进行hash，然后它们都落在15-445所对应的bucket中，然后我就凌乱了
58:17 - 58:19
But again this gets into the query planning side of things
但再说一遍，这是查询计划中的内容
58.19-58.27
the database system could look at ,and say oh I know what the distribution of values are for for this column and everyone is taken 15-445
数据库系统会去查看，并表示它知道该列的值的分布情况，所有人都学了15-445
58:27 - 58:29
So therefore if I do this technique
因此，如果我使用了这个技术
58.29-58.33
then I’m it’s not gonna get any benefit
那么我不会得到任何好处
58.33-58.35
because everything’s gonna hash to this and it’s always to work
因为所有东西始终会hash到这个分区中，并且这始终有效
58:35 - 58:37
I might as well just do ask seqential scan
我可能会去进行循序扫描
58:40 - 58:41
This question is
他的问题是
58.41-58.42
you always don’t know about the data
我们永远都不知道这数据是什么
58.42-58.46
 like you know a good database system will know something
一个优秀的数据库系统会知道一些事情
58:46 - 58:49
It won’t be entirely accurate ,but it’ll know something
虽然它所知道的东西并不完全准确，但它确实知道些东西
59:02 - 59:04
So this statement is
So，他想说的是
59.04-59.09
like with the full data set, these can be this be unskilled ,but then this is skilled
像这种全都hash到一个分区的data set，这些好像不好处理，但处理起来没问题的

59:10 – 59:12
Again this is this is next week or two weeks
这是下周或者下下周的内容
59:14 – 59:18
The database system can maintain metadata about every single column
数据库系统可以对每个列的元数据进行维护
59.18–59.24
histogram sketches I do an approximation of what the distribution of value being about looked like
比如，我可以使用直方图，草图之类的来大致看下值的分布情况
59:24 – 59:28
Again for skewed workloads that's harder
如果是不均匀的workload，那就更难了
59.28–59.29
you got a call
你有来电？（知秋注：提问的学生出去接电话了）
59:29 – 59:35
All right, he's got a call yeah all right sorry
他去接电话了

59:35 – 59:37
All right, so for simplicity
方便起见
59.37–59.40
I'm just saying assume uniform distribution
我会假设它们分布的比较均匀
59:40 – 59:41
Okay

59.41–59.43
for skewed work
对于某些不均匀的workload来说
59.43–59.46
again they'll be assert up to a certain point where this technique won't work
在某种程度上，这种技术并不起效
59.46–59.49
and sequential scan will be the better approach
循序扫描则会是更好的方案
59:49 – 59:49
Yes
请问
59:53 – 59:56
This question is what is the overhead of removing columns
他的问题是，移除列的开销是怎么样的
59:57 – 01:00:02
So in in this example here
So，在这个例子中
1.00.02–1.00.08
I'm showing this as like discrete steps like filter and then remove

这里我所展示的步骤并不是一气呵成的，这里我先进行过滤，再进行移除
1.00.08-1.00.11
you can inline a combine these together
你们可以将这两个步骤放在一起处理
01:00:11 - 01:00:13
But again this is another great example there's a trade-off
但再说一遍，这是另一个很棒的例子，它里面存在着某种取舍
01:00:13 - 01:00:16
So if my table is massive
So，如果我的表很大
1.00.16-1.00.22
 and I know that I don't need all the columns up above a tree
我知道这一点，并且我不需要这棵树上的全部的列
1.00.22-1.00.26
, then it's totally worth it to me to pay the penalty to do this projection
对我来说，为了去做这个projection而付出代价是完全值得的
1.00.26-1.00.28
because essentially copping data
因为本质上来讲，就是拷贝数据
01:00:28 - 01:00:29
But if I only have one tuple
但如果我只有一个tuple
1.00.29-1.00.34
then I'll delay that that may be the projection as late as possible
那么我就会尽可能延迟projection操作
1.00.34-1.00.37
because that's gonna be it's just cheaper to do at the very end
因为在最后进行这种操作的话，代价会更低
01:00:38 - 01:00:42
All right,there's a tree how wide and how tall the table is,
这是一颗树，它表示了这张表有多宽和多高
1.00.42-1.00.46
and again the database system can figure this out attempt to
数据库系统会去试着弄清楚这一点
01:00:48 - 01:00:51
Ok, so what we're doing here in the first phase
Ok，So我们在第一阶段所做的事情是
1.00.51-1.00.56
 where we're taking the course ID, we're hashing it, we're putting into these pages for the partitions
我们拿到course id，并对其进行hash处理，然后将它放入分区中的这些page里面



PHASE #2 – REHASH

For each partition on disk:

→ Read it into memory and build an in-memory hash table based on a second hash function h_2.

→ Then go through each bucket of this hash table to bring together matching tuples.

This assumes that each partition fits in memory.

01:00:57 – 01:01:02

so now in the second phase we rehash for every single partition

So，在第二个阶段中，我们会对每个分区进行重新hash

01:01:03 – 01:01:07

now,we're gonna bring them bring the the pages in right

我们将这些page放入内存

01:01:07 – 01:01:14

and then we're gonna build an in–memory hash table, that we can then use to find that the same keys

接着，我们会在内存中构建一个hash table，这样我们就可以用来找这些相同的key

01:01:16 – 01:01:17

so we don't have to do this

So，我们不需要这样做

1.01.17–1.01.20

we could just bring in every single partition and do sequential scan on them

我们可以进入每个分区，并对每个分区进行循序扫描

01:01:20 – 01:01:22

but because we're doing aggregations

但因为我们正在做聚合操作

1.01.22–1.01.30

we know that we don't need to have all of the duplicate keys in memory at the same time

我们知道，我们不需要将所有重复的key都同时存放在内存中