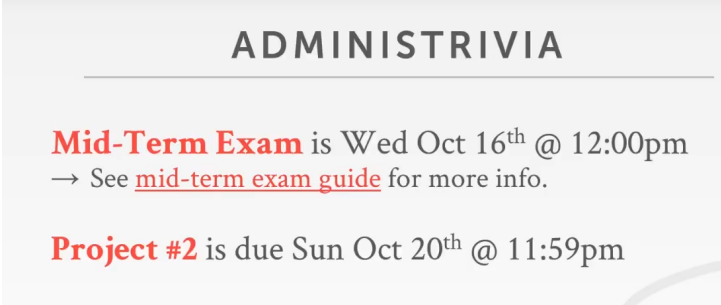
14–01

14 – Query Planning & Optimization I

18:48 之前都没带麦 有的地方听不出来

00:15 – 00:19
okay let`s start it ,right.....into it
Ok，孩儿们，上课吧

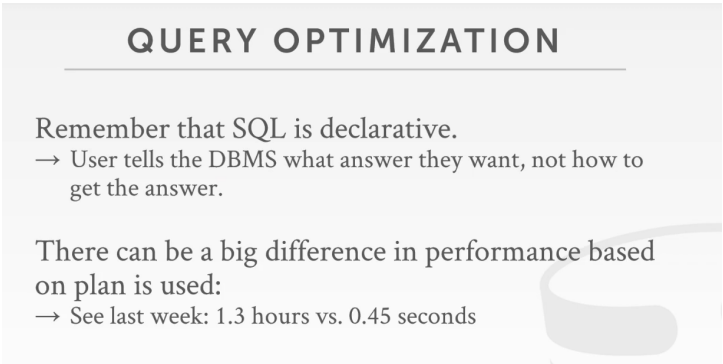


00:21 – 00:31
so the schedule for you guys, the major thing obviously two days from now on Wednesday in this classroom, we will have the mid-term exam
So，我们接下来的安排是，两天后的周三，我们会在这个教室举行期中考试
00:32 – 00:36
study guide who's been online since last week ,but also the practice exam
我上周在网上发布了学习指南，还有样卷
00:37 – 00:43
I've also uploaded all the lectures all the lecture notes and all the slides from everything that's covered in exam over this weekend
上周末我还上传了跟考试相关的所有课程视频，笔记以及幻灯片
00:44 – 00:51
and then there's additional problems in the textbook ... solutions for the odd ones are available online
然后我还传了一些教科书中的额外问题，以及对应的答案
00:52 – 00:54
so any high-level questions about the mid-term
So，对于期中考试，你们还有任何高级点的问题吗
00:56 – 01:01
you need to bring your CMU ID, because it's a class of a hundred people, I don't know everyone
你需要带着你的学生卡，因为这门课一百来号人，我认不全你们每个人

01:01 – 01:04
you should bring a calculator ,if you can't do logs in your head
如果你没法心算log，那么你可以带个计算器
1.04–1.12

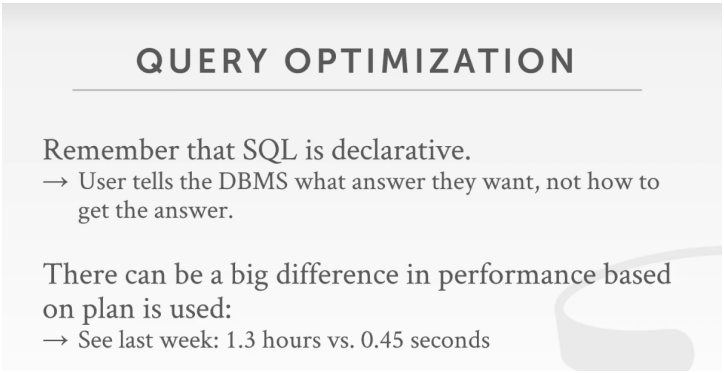
~~and you're not have a one one eight half I~~ let me sheet of paper ,double-sided with handwritten notes
你们可以带一张双面的手写笔记
01:12 – 01:19
No shrunk it down slides hows hunker down, let your notes know slowly shrunken down textbook ,everything I'll be written by hand ,you'll get more out of that way
你带的笔记不能是打印的，只能是手写的
01:20 – 01:22
So any question about any of these things
So，对此你们有任何问题吗
01:24 – 01:32
okay good, and then project two will be due on on Sunday at midnight as well
Ok， Project 2会在周日晚上截止
01:32 – 01:37
I'd like you I posted on Piazza what questions things like that right ,and then the TAs are helping as needed
我已经在Piazza上贴了一些相关的问题，如果你们有需要的话，可以找助教帮你们解答问题
01:38 – 01:38
okay

01:40 – 01:43
all right so we have a lot to discuss today, I don't think I'm gonna get through all of it
So，今天我们讨论的内容有很多，我不觉得今天都能讲完



01:43 – 01:50
so it might spill over to next week after the exam~~- plot there and see what happens~~
So，讲不完的部分我就延后到下周再讲
01:51 – 01:53
so today we're talk about **query optimization** and **query planning**
So，今天我们要讲的内容是查询优化和查询计划
01:54 – 02:00
so the high-level idea what we're trying to do today is that given **that SQL was Declarative**
So，我们今天所要讲的高级思想就是，SQL是声明式的
2.00-2.04
meaning the the application that sent us the query
这意味着，当应用程序向数据库系统发送查询时
02:04 – 02:08
the query says what answer they want us to compute,right
~~该查询中包含了它们想让数据库系统去计算哪些东西~~
该查询会告诉它们想让数据库系统去计算并返回对应的数据

02:08 – 02:13
select this from this table ,produce all the employees that have this particular attribute
即返回该表中满足该属性条件的所有employee信息
02:14 – 02:18
Right, and they and it's not telling us how we actually supposed to compute this
它并没有告诉我们实际该如何计算这些
02:19 – 02:25
right doesn't say do a hash join ,let's look join, although you can do hints like that ,but we can ignore that for now
这里并没有说我们进行的是hash join，这里只是说进行join操作，虽然你可以根据提示来使用hash join，但我们现在将它忽略
02:25 – 02:28
right it's it's just them telling us what answer they want
这里它只是告诉我们它们想要的结果是什么
02:28 – 02:33
and so it's up for us now inside the database system people actually building a database system software
实际上，人们在系统内部构建了数据库系统软件
02:34 – 02:41
it's our responsibility to take that SQL query and figure out the the best way the most efficient way to execute it
我们的任务就是拿到SQL查询语句，并弄清楚执行该语句的最佳方案
02:43 – 02:49
and so if we saw from from last week or the previous weeks when we talked about join algorithms
So，根据我们上节课或者前一周我们所讨论过的join算法来看
02:49 – 02:53
but they can be quite a big difference in performance, depending on what algorithm you choose
根据你选择的算法不同，它们在性能上的差别也不是一点半点
02:53 – 02:58
we had stupid nested loop join would take one point three hours for a table of six megabytes
如果我们要对6MB大小的表使用nested-loop join，那么这就要花1.3小时才能完成
02:58 – 03:03
but if you do it in the hash join worst case scenario was 0.5 seconds
但如果我们使用的是hash join，那么在最糟糕的情况下，我们也只需要0.5秒就能搞定一切
03:04 – 03:08
~~so you know eight spits~~ for this one it's pretty obvious right
So，在这个例子中，这种性能上的差异非常明显
3.08-3.10
we don't wanna do the stupid thing we can figure this out
我们不想通过愚蠢的方式来进行join操作
3.10-3.13
when we start getting more complex queries more complex joining
当我们去做些更为复杂的查询和join操作时
3.13-3.18
these then figuring out how exactly go from this to this it's not always obvious
不同方法间的性能差距可能就不是那么明显了



03:18 – 03:30 ！！！！！

and that's why in this new this is what's going to separate the the high end database systems that are very expensive the Oracle, the DB2 the Teradata is, the SQL servers ,Versus like the source ones are the free ones

这就是高端数据库系统（诸如：Oracle，DB2，Teradata和SQL server）和开源数据库系统的区别所在了，贵有贵的道理

03:30 – 03:31

Postgres is still very good

PostgreSQL依然很优秀

3.31–3.38

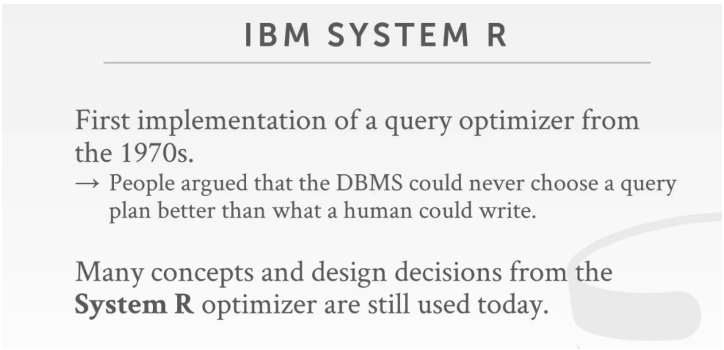
but it's no where the query optimizer is nowhere as sophisticated as SQL servers for example

但它的查询优化器并不如SQL server那样来得成熟

03:40 – 03:44

so the idea about query optimizer goes back to the 1970s

说起查询优化器，这得从1970年代讲起



03:44 – 03:49

so I think I've talked about system R in this class a little bit ~~but I said why~~

So，我之前已经在课上和你们讨论过System R相关的一些东西了

03:49 – 03:54

but back in you know in the 1970s when Ted Cod and wrote the first paper on the relational model

回到1970年代，当时Ted Cod写出了第一篇关于关系模型的paper

03:54 – 03.58

there was basically two people or two groups that picked it up and try to actually implement it

简单来讲，但是只有两个人或者说两组人z选择并尝试对该paper进行实现

3.58–4.01

because getting Ted Cod was a theoretician it was a mathematician

因为Ted Cod是一个理论家，同时也是一个数学家

04:01 – 04:05

So he didn't have to have it in software

So，他无须通过软件的形式对其进行实现
4.05–4.06
,he just proposed an idea relational model
他只是提出了关系模型这个想法
4.05–4.07
said this is the right way to actually build software
并说，这是构建软件的正确方式

04:08 – 04:14
And then there was a group at IBM in San Jose, and then a group at **UC Berkeley**
接着，在当时有两伙人，一伙人是IBM的，另外一伙是UC Berkeley的
04:15 – 04:21
Who ended up taking that payment actually built the you know the first two relational database management system at least most famous ones
他们两伙人构建出了史上前两个关系型数据库系统，并且是最为知名的那种
04:21 – 04:23
no one at Berkeley was called ingress
伯克利那批人开发出来的东西叫做Ingres
4.23*–4.32
that was you know my postgres is called Postgres, the same guy that made ingress also be Postgres ,because it's post ingress
你知道的，Postgres之所以叫Postgres，是因为它是由写Ingres的那个人所写的，作为Ingres的后继产品
04:32 – 04:33
that's what the name comes from
这就是它名字的由来
4.33–4.37
right, think he was my advisor when I was in graduate school
当我在上研究生的时候，开发它的人是我的导师
04:38 – 04:43
and then be the IBM guys, they build this thing called **system R**
然后，IBM那群人开发出的东西叫做System R
04:44 – 04:46
and the project is actually fascinating
这个项目实际上很有趣
4.46–4.50
,because they got like eight or nine people that all had PhDs in math and computer science
开发这个项目的人总共有8或9个人，他们都是数学和计算机科学方面的Phd
04:50 – 04:51
again this is early computer science
这是最早时期的那种计算机科学
4.51–4.54
but there wasn't many people that had graduate degrees in computer science
当时并没有太多人获得计算机科学方面的硕士学位
04:54 – 4.57
they put them out of room says we're gonna build a relational database system
有人将他们拉到一起说，我们去搞一个关系型数据库吧
4.57–5.01

and then every person of the PhDs are carved off the room and part of the problem
每个Phd都各自负责其中一部分问题
05:01 – 05:06
One person who worked on the storage layer,one person worked on the execution engine, one person worked on concurrency control
比如说：一个人负责存储层，另一个人负责执行引擎，还有的人负责并发控制
05:07 – 05:13
and there was one people person pass cylinder she kept query optimization ~~Krita~~ query optimizer
有一个人负责的是查询优化这块
05:14 – 05:23
And so back then the idea that you could have a DBMS take a query a declarative language like SQL
So，在那个时候，DBMS执行查询时所用的是类似于SQL的声明式语言
5.23–5.26
get Ted Cod never actually proposed a language initially with relational model
Ted Codd实际上在一开始提出关系模型的时候，并没有提出相关的语言
05:27 – 05:28
SQL came later on
SQL是之后才出现的
5.28–5.29
because it was embedded by IBM
因为这是IBM所引入的
5.29–5.35
the **Berkeley** guys had this other thing called quel which looks a lot like SQL, but the syntax is different
伯克利那批人使用的是一种叫做Quel的东西，它看起来很像SQL，但语法并不相同
05:35 – 05:42
Michael Stonebraker the guy wouldn't be dressed would he claims Quel was superior to SQL, I disagree
Michael Stonebraker表示Quel要比SQL来得更为优秀，但我不同意这点

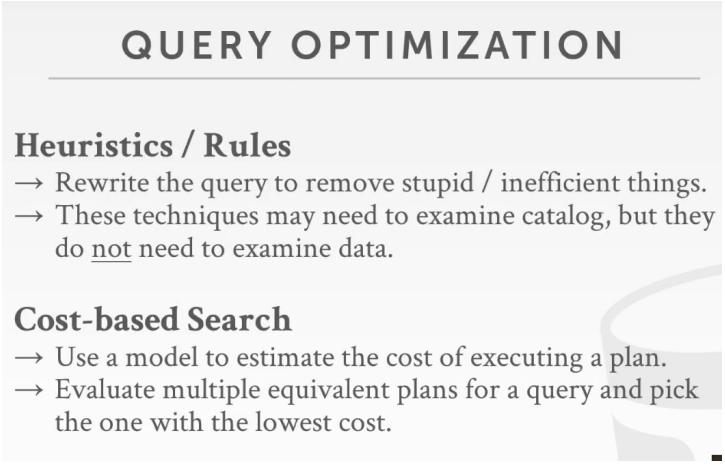
注：Michael Stonebraker因“对现代数据库系统底层的概念与实践所做出的基础性贡献”而获得2014年图灵奖。

05:42 – 05:43
but you know nobody writes it well now
但你知道的，现在并没有人使用Quel
05:43 – 05:56
so but in the back then people argued that there's no way of DBMS gonna take a high–level language like a SQL or quel and generate a query plan as efficient and what a human could do writing by my hand
So，但在那个时候，人们认为DBMS不可能去使用SQL或者Quel之类的高级语言并生成那种如同人手写那么高效的查询计划

05:56 – 5.58
cuz that's what people were doing before the rational model
因为这是人们在关系模型出现前所做的事情
5.58–6.03

if you were writing this these query plans by hand like writing the for loops and do joins and scans by hand
如果你手写这些查询计划，比如手写for循环，join操作，然后手动扫描
06:04 – 06:10
and so in the same way that people will argued when the the C language came out
So，当C语言推出的时候，人们当时也是这样认为的
06:10 – 06:11
people was saying
人们表示
6.11–6.12
oh C's too high–level
Oh，C语言太过高级了
6.12–6.19
it's the compiler is never gonna generate machine code as efficient than what humans can write today or a human humans could write an assembly
编译器没法生成比人手写来得更为高效的机器码，或者汇编代码
06:19 – 06:22
and of course now we know nobody writes for the most part assembly by hand
Of course，我们知道现在大部分的汇编都不需要人去手写了
6.22–6.25
everyone writes in higher–level languages even higher than C
人们使用的是更高级的编程语言去编写代码，这些语言比C还高级
06:25 – 06:27
and compilers do a pretty good job
而且编译器做得很好
6.27–6.30
 in those cases they can do even better than what the average human can do
在这些语言中，一般来说，它们（编译器）所做的工作要比人来得更好
06:30 – 06:35
this is what IBM prove back in the 1970s that you could take a declared language like SQL
于是IBM在1970年代推出了查询优化器，这使得你可以去使用像SQL那样的声明式语言
06:36 – 06:46
and have the query optimizer or the planner generate a query plan that was as good if not better than what a human can actually do or at least the average human
他们通过查询优化器能够生成跟人手写一样好或者差不多好的查询计划
06:47 – 06:50
right so well talk about how the system optimizer works
So，我们会去讨论查询优化器是如何工作的
06:51 – 6.52
but what I'll say
但我要说的是
6.52–6.57
as we go along and talk about different leading up to it actually how we do the cost based search
我们会去讨论进行cost–based search的不同方法
6.57–7.00
that IBM invented back in the 1970s
这其实是IBM在1970年代所发明的东西

07:00 – 07:10
A lot of the designing decisions and assumptions that they made about what the data looks like, and what the query plans look like, to simplify the problem to make it tractable
他们根据数据和查询计划做出了很多设计决策和假设，以此来简化问题，使其能够易于解决
7.10–7.13
are still actually used in practice today
这种方式到了今天我们也依然在使用
07:13 – 07:16
all right and I'll go through what those are as we go along
All right，我会对它们进行逐个讲解



07:17 – 07:20
so with query optimization there's essentially two approaches to doing this
So，在查询优化这块，本质上来讲，有两种策略可以使用
07:21 – 07:27
and again this is also gets into like what distinguishes the good optimizer bad optimizers
这也是好的优化器和差的优化器的区别所在
07:28 – 07:32
so the first approach is use static rules or heuristics
So，第一种方案是使用静态规则或者条件触发
07:33 – 07:34
so the idea here is that
So，这里的思路是
7.34–7.36
we can look at our query plan
当我们查看我们的查询计划时
7.36–7.42
and that matches a certain pattern like a portion of the query time matches a pattern that we know about
如果我们查询中的某些部分满足了我们知道的某种条件
07:43 – 07:44
then we fire off a rule
那么，我们就会触发一条规则
7.44–7.50
that I do some kind of transformation or rewriting of the query plan took to make it more often more
这样我就可以对该查询做一些改造或者重写
07:51 – 07:56
right maybe like if you define stupid things where one equals zero

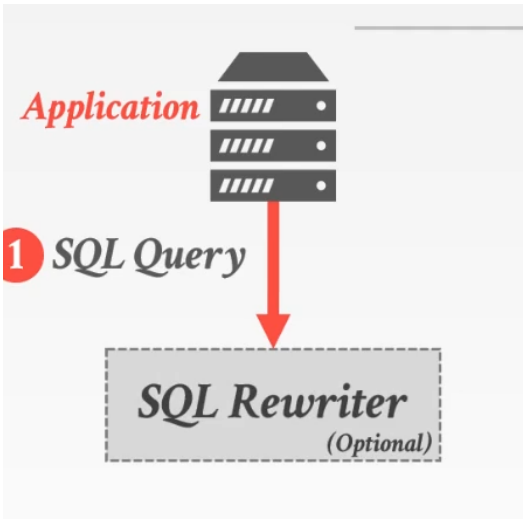
比如，你定义了某些愚蠢的东西，例如：1 = 0
07:56 – 08:00
right then you can have a rule can strip that out very easily
那么你就可以通过一条规则将它去除
08:02 – 08:05
so the important thing I'm saying on these rules that
So，对于这些规则而言，有一点很重要
8.05–8.12
we may need to look at the catalog the system kalam that tells us what our database looks like what our tables looks like
我们可能需要去查看System Catalog，我们得通过它来知道我们的数据库长啥样，我们的表长啥样
8.12–8.18
,that the catalog an is the metadata about the data ,what tables do I have, what columns do they have ,what attributes that they have so forth
catalog中所放的是一堆元数据，它描述了我有哪些表，哪些列，表中的属性有哪些
08:19 – 08:20
so for these rules
So，对于这些规则来说
8.20–8.26
you may have look at a catalog consult enough to understand you know what our underlying tables that should look like
我们得去查看Catalog来理解我们底层的那些表中有哪些东西
08:26 – 08:29
but we never actually have a actual data itself
但实际上我们并不需要去查看这些数据是什么
08:31 – 08:38
right we can fire off these rules without actually going to the table say ,well what did it put to be what am tuple was actually contained
在不需要实际去查看表的情况下，我们通过这些规则就可以知道这里面tuple实际包含的定义信息
08:39 – 08:39
right

08:41 – 08:44
be the alternative is **cost-based search**
另一种方案就是**cost-based search**
8.44–8.48
~~which is instead of~~ you know you don't look at the data
你知道的，你不需要去查看数据
08:48 – 08:51
this one you are gonna have to look at the data in some way
如果使用的是这种方式，那么你就得以某种方式去查看数据
08:51 – 08.53
see the idea what the cost base search is that
这种**cost-based search的思想是**
8.53–9.04
we`ll gonna do a ,we're going to enumerate a bunch of different plans and different ways and ideally intelligent manner around looking at redundant or stupid things

我们会去枚举该SQL所有可能的不同查询方案，并通过某种智能的方式去掉那些多余或者愚蠢的方案
09:04 – 09:09
or with enumerator a bunch of different plans for us that we could choose to execute our SQL query
~~我们从这些枚举出来的这些不同中执行方案选出一个来执行我们的SQL查询语句~~
它会列出一堆不同的查询计划供我们选择执行我们的SQL查询
09:09 – 09:16
and then we're going to use some kind of **cost model** to approximate, which one is the execution cost all these different plans
然后，我们会通过使用某种成本模型来预估所有这些不同方案的执行成本
09:17 – 09:20
of course the idea is that we want to always pick the one that has the lowest cost
Of course，我们总是会选那个成本最低的执行方案
09:22 – 09:28
so obviously getting the bat that cost minor if you have accurate estimations is super hard ,we'll see why as we go along
So，很明显，如果我们要进行准确地估算，这会超级难，原因我们稍后会讲
09:28 – 09:32
and then also in the way you enumerate the plans is difficult too
同样，如果你要去枚举所有的执行方案，那也是很难的
09:32 – 09:32
because as I said
因为正如我说的
9.32–9.37
~~you don't want to look at~~ you have a finite amount of time to look at about different options
你查看这些不同选项的时间是有限的
9.37–9.44
and you don't want to spend you know hours and hours doing a branch of bound search for a query that may take one second to run
~~你不想花太多时间来找一个执行只需要1秒的执行方案~~
你绝对不会为一个执行起来只需要1秒的方案在方案查找上面浪费太多时间
09:45 – 09:48
it's actually quite amazing how how how fast these things can actually be
对于这些东西的实际速度能有多快，我们还是很惊讶的
9.48–9.51
like every time in this class and we'll show some demos later on
就比如我每次在课上展示的那些Demo
09:51 – 09:57
every single time I open up the quickly in the terminal ,I wrote a SQL query, I hit enter it was doing both of these things in milliseconds
每次当我打开terminal时，然后编写SQL查询语句，接着按下回车，数据库系统就会为我们在几毫秒之内做完这些东西
09:58 – 10:04
now again for our queries we look at they're pretty simple, so it's not that mind blowing
对于那些我们所查看的查询来说，它们都非常简单，并没有那么烧脑
10:04 – 10:11

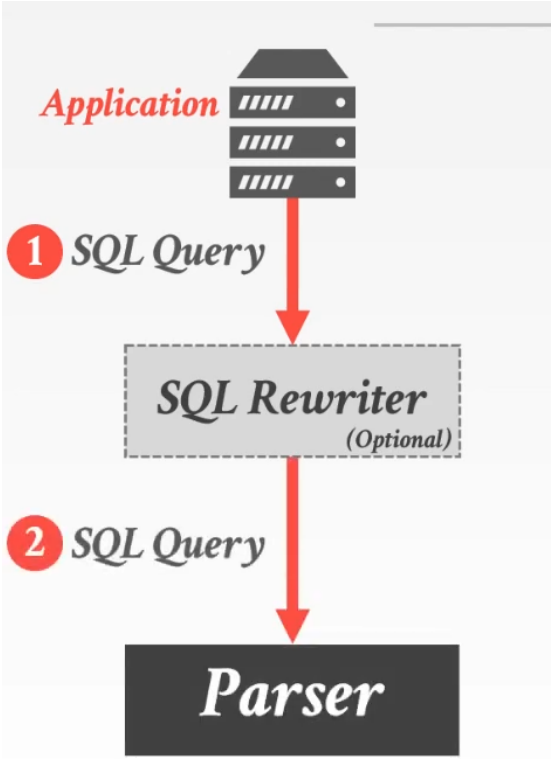
but even still you know you can't run anything forever and find it you know the best possible plan are you try to approximate it
你知道的，你没办法一直像这样为执行找到最佳的查询计划，你得试着去找到跟它类似的查询计划

10:13 – 10:18
so the pipeline for our query optimization path it was looking at the following
So，如图所示，我们查询优化中的pipeline是长这样的
10:18 – 10:20
so the very beginning we have an application
So，在一开始，我们有一个应用程序
10.20–10.22
the application is connected to the database system
该应用程序和数据库系统建立了连接



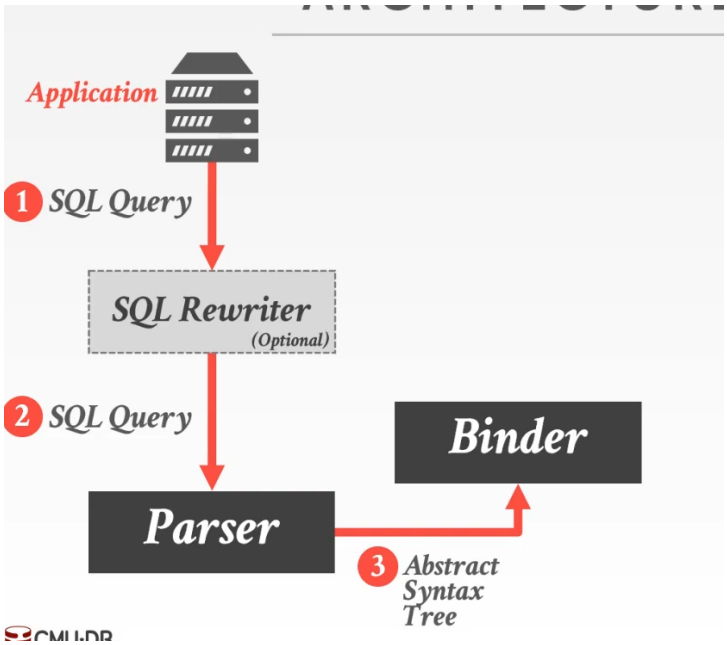
10:23 – 10:25
and then it's going to send us a SQL query
然后，它发送给我们一条SQL查询
10:25 – 10:30
so the first stage would go through our optimization pipeline is called the **SQL rewriter**
So，我们optimization pipeline中的第一个要经历的阶段叫做SQL rewriter
10:31 – 10:33
so the idea here is
So，这里的思路是
10.33–10.34
that we're given SQL
这里我们给出了一条SQL语句
10.34–10.40
we can have some transformation rules that allow us to rewrite the SQL in certain ways
我们可以通过某些转换规则来让我们以某种方式对SQL语句进行重写
10:40 – 10:45
So sometimes this occurs for distributing databases or if you have views
So，有时候分布式数据库中就得做这个，如果你看过的话
10.45–10.50
like ~~The Hobbit ale is~~ for a table name and this thing can say oh I see this table name maybe you write it to be something else

比如：如果这里它看到一个表名，它会说，Oh，我认识这个表名，可能你要为它写点其他东西
10:50 – 10:56 ！！！！！！
let me annotate it with additional information to say this particular table can be found on this node over here or this disk over here
我会使用一些额外的信息对它进行标记，表示你可以在这个服务器节点或者这个磁盘上找到这张表
10:57 – 10.58
so this is optional
So，这个是可选的
10.58–11.04
right this is not that common I don't think most database system don't want to operate directly on SQL
这个东西其实并不常见，我不觉得大多数数据库系统会通过SQL这么直接操作
11:05 – 11:08
but you know this is something you could do to do ,so traditional optimization here
但这是你们可以做的一种传统优化

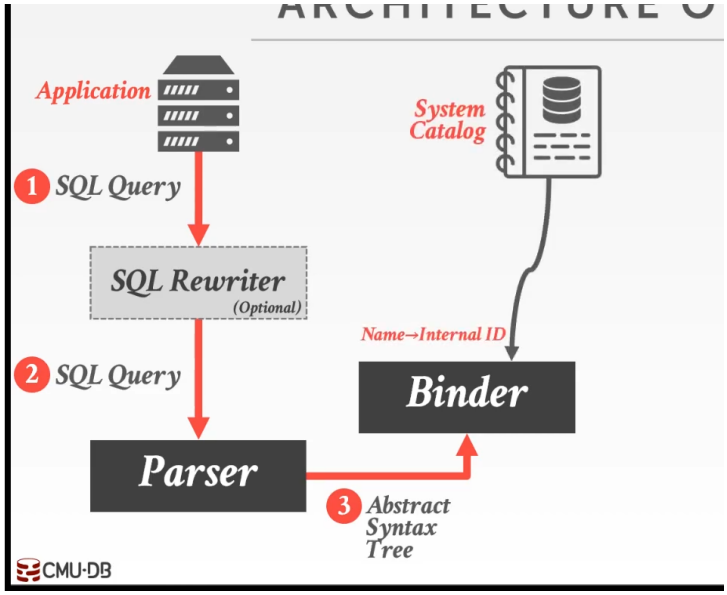


11:10 – 11:14
then we take the SQL query that comes out of this, if we have it, otherwise you go back through the application
然后如果有解析器的话，我们将SQL查询传入解析器中，不然我们就得回到应用程序中了（没法继续往下干了）
11:14 – 11:16
and they passed it through our SQL parser
我们将它传入SQL解析器中
11:16 – 11:20
and this is just getting the lexer token stuff you can you read a compiler class
如果你们学过编译原理的话，那么你就知道，SQL查询会被分解为lexer token之类的东西
11.20–11.23
there's nothing there's no fancy stuff we're doing here
这里我们不需要秀什么操作
11:23 – 11:28

we're just converting a SQL string into the **abstract syntax tree**
这里我们只是将SQL字符串转换为抽象语法树

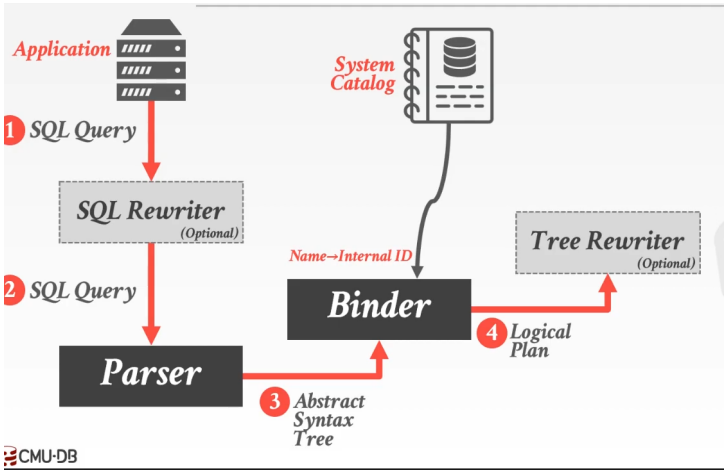


11:29 – 11:32
so then now we see the syntax tree into our **binder**
So，接着我们会看到，这里我们将抽象语法树传入了binder中
11.32–11.41
and a **binder** is responsible for converting the named objects referenced in our SQL query to some kind of internal identifier
binder是负责将我们SQL查询中所引用的那些命名对象转换为某种内部标识符



11:41 – 11:44
So really dude is like consulting the **Catalog**
So，我们通过询问System catalog来做到这点
11:44 – 11:46
so I have a query it select star from table foo
So，假设我有一条SQL查询语句，即select * from foo
11.46–11.50
I don't want to have the rest of my query plan and have to operate on the string foo
我不想让我查询计划中的剩下部分对字符串foo进行处理
11:51 – 11:54
I go to the **catalog** and say hey do you have a table called foo

我得跑到System Catalog处询问有没有一张叫foo的表
11:54 – 11:58
if yes give me some internal identify or to allow me to find it later on
如果存在的话，那么请将它的内部标识符给我，这使得我之后能够找到这张表
11:59 – 12:00
or if it doesn't have it
或者，如果这张表不存在的话
12.00–12.03
like it says you know table foo doesn't exist
它就会说foo表并不存在
12.03–12.04
at this point we could throw an error
此时，我们可能就得抛出一个错误
12.04–12.08
,and say you know you looking up a table, we don't have or the column, we don't have
并说，我们并没有你所要查找的那张表或者那个列
12:09 – 12:13
so now that the binder is gonna emit is a **logical plan**
So，现在binder所要输出的东西叫做逻辑计划（logic plan）
12:14 – 12:15
well explain look where do you tell what this is in a second
Well，我之后会向你们解释这个
12:15 – 12:18
but the high level I think what this is
但从高级层面来讲，我是这样想的
12.18–12.19
there's a **logical plan** and the physical plan
这里有逻辑计划和物理计划
12.19–12.23
a logical plan could say at a high level what the query wants to do
逻辑计划指的是，从一个高级层面来讲，这个查询想干的事情是什么
12:23 – 12:26
I want to scan this table ,I want to read data from this table
比如：我想对这张表进行扫描，从这张表上读取数据
12:26 – 12:28
I'm gonna join these two tables
我想对两张表进行join
12:28 – 12:30
it doesn't say how all you're actually going to do that
逻辑计划中并不会说我们实际该怎么执行该查询
12.30–12.31
that's the physical plan
这是物理计划所要干的事情
12.31–12.35
actually specification with the algorithm you're using comes later on
它会去为我们指定我们之后执行查询计划时具体要使用的算法是什么



12:35 – 12:42

so the binders without a logical plan it's basically a conversion the converted form of the syntax tree now of **internal IDs**

So，在没有逻辑计划的情况下，binder所做的就是将语法树中的名字转换为internal id（内部标识符）

12:42 – 12:45

And we have someone in a relational algebra approximation

这里我们会有一些类似于关系代数的东西

12:46 – 12:49

and then we can feed this now into a **tree rewriter**

然后，我们将它传入一个Tree rewriter

12:49 – 12:50

again this is optional

再说一遍，这一步处理是可选的

12.50–12.53

this actually is more common than the SQL rewriter

实际上，这要比SQL rewriter更为常见

12.53–12.54

this one actually most of database systems provides

实际上，大部分的数据库系统都提供了Tree rewriter

12:54 – 12:56

Because these are the static rules

因为这些都是静态规则

12:57 – 12.59

so to do the tree ~~right~~ rewriting

So，为了对抽象语法树进行重写

12.59–13.04

we have to go to the Catalog potentialy and ask them, hey what does our tables look like what are our attributes look like

我们得去问System Catalog，我们的表长啥样，它里面有哪些属性

13:05 – 13:08

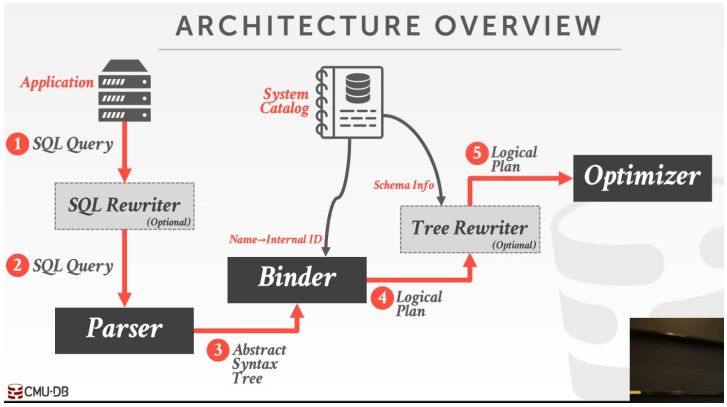
um but yeah we don't have to go to the **cost model**

但这里我们不需要跑到成本模型（**cost model**）那里计算成本

13.08–13.15

we can do this as use your static rule that you want to do for every query ,no matter what the actual data looks like

我们可以通过这些静态规则对每个查询都进行改写，不管实际的数据是什么



13:15 – 13:21

so then the tree rewriter gets picked up still the same logical plan that binder spit out

接着，Tree Rewriter这边会生成和binder处所输出的相同的逻辑计划

13:21 – 13:24

so then now we feed this into our query optimizer,right

So，然后我们将逻辑计划传入我们的查询优化器

13:24 – 13:26

this is like the black magic part

这部分就是个黑科技

13.26–13.31

this is where we're actually gonna do the search to try to use a **cost model** to figure out what the best plan for us is

我们会在这部分通过使用成本模型（**cost model**）来找出适合我们的最佳方案

13:32 – 13:40

so they've been using a combination of schema information that the catalog provides us as well as some of these s these estimates that our **cost model** could provide us

So，查询优化器会用到System catalog所提供给我们的schema信息以及通过提供给我们的成本模型来对这些方案进行成本估算

~~成本模型所提供给我们的一些成本预测来帮我们选出最佳方案~~

13:41 – 13:45

All right this query plan is going to take X amount of times other party plans meant a Y amount of time

比如，使用这个查询计划需要消耗x秒，选另一个查询计划则需要花费y秒

13:45 – 13:47

X is less than Y I want to pick this other one

x要比y小，所以我想选x这种方案

13:49 – 13:59

well we'll see as we go along this this **cost model** is going to be a typically ,it's a synthetic number that the database system computes internally

Well，我们会看到通过成本模型可以得到一个数据库系统内部已经计算好的综合数字

13:59 – 14:01

it has no meaning to be outside world

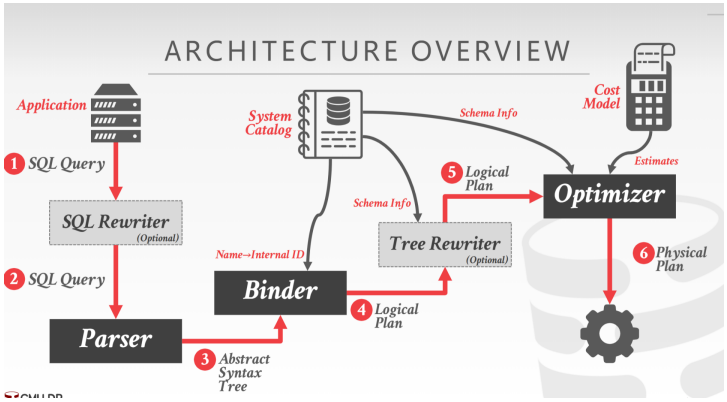
对于外界来说，这些数字没有什么用处

14.01–14.07

you can't take a cost model estimate say oh that's really 20 seconds that's me you know 20 minutes

成本模型所算出来的东西对于外界来说并没有什么用，比如说，它算出来这个查询计划需要20分钟才能执行完

14:07 – 14:11
it’s just internal thing that say this query plan is better than another
对于数据库系统内部来说，这就是一个指标，通过这个指标，它可以判断出这个查询计划要优于另一个查询计划
14.11–14.16
 so just used to compare relatively inside big database system between different plans
So，大型数据库系统内部会通过这个数据来对不同的查询计划进行比较
14:16 – 14:18
and has no bearing to the outside world
这与外界无关，属于内部的工作
14:19 – 14:23
so if you think Postgres this **cost model** it spits out a number, MySQL cost model spit out a number
如果你去看下PostgreSQL，它的成本模型会输出一个数字，MySQL的成本模型也会输出一个数字
14:24 – 14:27
you can’t ... apples and oranges you can’t compare them
你不能在不同的数据库系统间比较这两个数字
14:28 – 14:33
some systems will try to have the class amount of estimates B term to the actual time
有些系统会去试着估计这些执行计划的实际运行时间
14:33 – 14:36
but that that’s tricky and usually not reliable
但这做起来很棘手，通常也并不可靠
14:37 – 14:41
Because hardware changes and the environment changes and so forth
因为硬件会变，环境也会变，还有很多其他因素的存在
14:42 – 14:45
all right – now the optimizer is going to spit out a physical plan
Now，优化器现在就可以生成一个物理计划
14:45 – 14:48
and the physical plan is what our database system can actually execute
物理计划实际上就是数据库系统实际执行查询语句的方式
14:48 – 14.51
so when we talk about query execution I show these plans
So，在我们讨论查询执行的时候，我展示了一些查询计划
14.51–14.53
like it’s the hash join
比如，就拿hash join来说
14.53–14.56
 that you know yeah doing sequential scans, right index scans feeding the hash join
你通过循序扫描或者索引扫描来得到数据，并将这些数据传入hash join
14.56–15.02
 ~~their choppers drawings~~, do my hash aggregation or sorting aggregation that specification happens here
然后我们在此处进行hash聚合或者排序聚合之类的操作



15:02 – 15:08

so once we get outside the optimizer then there's not really anything else we have to do until none of them is executed the query plan

一旦我们结束了优化这一步骤，直到要去执行该查询计划时，我们才会去做些其他事情

15:10 – 15:13

so again and a high level of this is how every single DBMS implements it

So，从高级层面来讲，这就是每个数据库系统实现查询优化的方式

15:13 – 15:18

some things like a SQL rewriter，Tree Rewriter , you don't necessarily have to have

比如SQL rewriter或Tree Rewriter之类的东西，你不一定要有它们（知秋注：因为它们是可选的）

15:19 – 15:22

but the parts of binder and optimizer is this the standard package

但binder和optimizer这两块是必须的

15:25 – 15:27

so just to reiterate between the logic versus physical

So，再重申下逻辑计划和物理计划这块的东西

15:28 – 15:31

again the the the way to think about is

我们要去思考的是

15.31–15.34

that the DBMS generate a logical plan

DBMS会生成一个逻辑计划

15.34–15.40

that is roughly equivalent to the the relational algebra expressions within our query

粗略地讲，这等同于我们查询中的关系代数表达式

15:42 – 15:43

it's **not always a 1:1 mapping**

虽然这并不能一一对应

15.43–15.45

but at a high level you can think of it that way

但从高级层面来看，你可以以这种方式去思考

15:45 – 15:49

I want to do a selection on this table, I'm gonna do a filter, I wanna do a projection ,I would do a join

比如，我想对该表进行select操作，filter操作，projection操作或者join操作

15:50 – 15:52

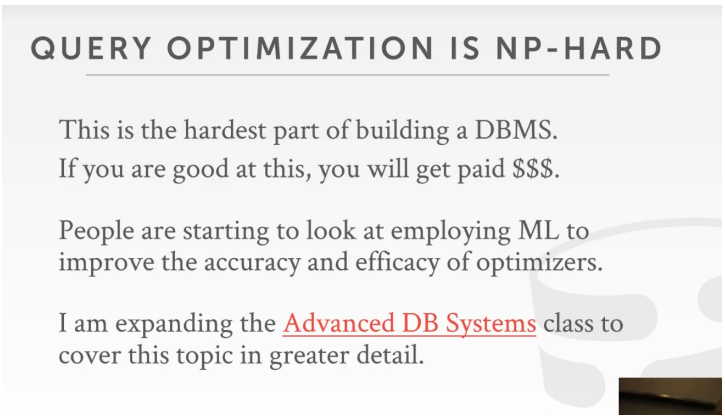
right those are all be contained in a logical plan

这些都包含在逻辑计划中

15.52–15.56

and all those query plan trees that I showed before way I sort of just had the relational algebra symbols
在我之前所展示的那些查询计划树中，它们中都包含了这些关系代数的符号
15:57 – 15.58
it didn't annotate it and say what algorithm I was using
这并不代表我要去使用哪种算法
15.58–16.00
and those have we considered logical plans
我们将这些视为逻辑计划
16:02 – 16:11
the physical operators the physical plan, that's where we actually specifically define what execution strategy we're going to use for those different operators in our query plan
对于物理计划来说，实际上，这才是我们用来定义查询计划中执行方案的地方，即我们要在查询计划中如何使用这些不同的operator
16:12 – 16:14
I'm gonna do an index scan using this index
比如：我使用这个索引来进行索引扫描
16.14–16.19
in this order we're gonna feed my output an index scan into a hash join operator
我们将我们索引扫描所得到的输出结果以一定顺序传给hash join operator
16:20 – 16:24
right it's all the low-level details of how we actually execute it
这些都是我们实际执行时的底层细节
16:25 –16:31
and so that the bunch of different metadata we have maintain attention a physical plan that we don't be care about at this point or this class
So，我们在维护一个物理计划时，会遇上一堆不同的元数据，但在这门课上我们不会去关心它们
16:31 – 16:35
But like if I know I have a order by up above
但如果我知道我的查询计划中有一个Order By操作
16.35–16.40
and I give you a sort-merge join on the same join key the same Order By key
~~我就会在Order By所使用的key上使用sort-merge join~~
我发现sort-merge join使用的key和Order By使用的key是相同的
16:40 – 16:47
then I can annotate my physical operators and my query plan to say this data that I'm spitting out is Order by way is sorted based on this key
~~那么，我可以对我的物理操作符和查询计划进行标记，并说：对于这个我所拆分的数据来说，我想根据这个key来进行Order by排序~~
那么我就可以告诉我的物理操作符和我的查询计划，我的输出数据已经按照这个key排过序了，这个key和Order By操作的key一模一样
16:48 – 16:52
so then you can reason and that you can drop the order by up above
So，那么你就可以丢掉上面的Order By了
16:53 – 17:02
it's probably a 1:1 mapping from the relational algebra to the to the logical plan
我们或许无法将关系代数和逻辑计划一一对应起来
17.02–17.04

not always true, but high level it is
虽然这种说法不一定始终正确，但从高级层面来看是这样的
17:05 – 17:10
but for the physical plan we can't assume it's a 1:1 mapping from the logic which is the physical
但对于物理计划来说，我们没法让它和逻辑计划一一对应起来
17:11 – 17:14
right again if I could have a join plus an order by
如果我有一个join operator以及一个Order By operator
17.14–17.17
but if I do sort–merge join in my physical operator
如果我在我的physical operator中使用了sort–merge join
17.17–17.23
then I can get rid of the order by operator up above
那么我在上面就不需要使用Order By operator了
17:23 – 17:26
so the the way to think about this is
So，我们可以思考一下
17.26–17.31
that the rewriting stuff we'll talk about, and then we get to the cost based search
我们之后会说下重写这块的内容，然后我们会去讲下cost–based search
17:31 – 17:33
those are all operating on logical plans
所有的操作都是基于逻辑计划进行的
17:35 – 17:38
but the final is always e needs to be a physical plan
但最终还是需要变为一个物理计划的
17:39 – 17:39
okay



17:41 – 17:46
So before we get into the nitty–gritty details ,I mean to say that this is super hard
So，在我们深入这些细节之前，我提前声明，这部分超级超级难
17:46 – 17:48
this is the hardest part about database systems
这是数据库系统中最难的地方
17.48–17.50
this is actually the part I know the least amount
实际上，这也是我最不熟悉的地方
17:51 – 17:52

this is why I'm so fascinated
这也就是我为什么对它所着迷的原因
17.52–17.54
and I always want to try to do more
我一直想在这方面做更多努力
17.54–17.56
because like I don't know it ,I don't understand it
因为我对这块知之甚少
17:57 – 18:00
if you can do this, if you if you're good at doing query optimization
如果你对查询优化这块非常了解
18:01 – 18:02
you can get a job immediately
那么你就能立马拿到一份工作
18.02–18.04
move people pay ~~BBBB~~ you some money to do this
人们会很乐意花钱让你来做这方面的事情
18:05 – 18:07
because like as I said
正如我说的那样
18.07–18.10
IBM made stuff in the 1970s
IBM在1970年代就做了这种东西
18:10 – 18:14
there was a lot of query optimization worked under late eighties early early nineties
在上世纪80年代早期或者90年代早期，很多人都在研究查询优化这块
18:15 – 18:17
But now it's all like crusty old people,Right
但现在他们都是群脾气暴躁的老人了
18:18 – 18:28
we're all like retiring or moving on you know what 21 year 21 year old knows about Query optimization other than my students right
除了我的学生以外，有谁能在21岁的时候就对查询优化有所了解呢

18:28 – 18:30
If you can do this
如果你能做这块的话
18.30–18.31
you can got a job immediately
那么你就立马能找到一份工作
18.31–18.33
that's the one email I always get
我天天都能收到这方面的工作邮件
18.33–18.35
I should share that I should show screenshots of this
我应该给你们看下邮件截图
18:35 – 18:41
The email I always get from from people friends at database companies ~~BBBB~~ ,I'm not wearing a mic

我总是收到某数据库大厂的邮件，他们想找我内推人才，我居然忘记带麦克风了
~~18:46 – 18:47~~
~~maybe I pick up most of it~~

18:48 – 18:52
right right so if you can do this
So，如果你可以做这部分工作
18.52–18.54
you'll get paid a lot of money
那么你就能拿到一大笔钱
18.54–18.55
because it's like super hard
因为这块内容实在是太难了
18:56 – 18.57
and it's hard to hire people to do this kind of stuff
而且确实很难招到这方面的人
18.57–19.02
one database system company told me that if they can find people that have a PL background， they can do this
其中一家数据库系统公司告诉我，如果他们能找到有PL背景的人来搞这块，那么他们愿意付一大笔钱
19:02 – 19:09
,another database company told me that they have people that have backgrounds in high energy physics,can can do query optimization
另一家数据库公司告诉我，他们中有员工的背景是高能物理，这名员工负责的就是查询优化
19:10 – 19:15
so this is joke in databases that says like people always say Oh query optimization is as hard as rocket science
So，这其实是数据库系统中的一个玩笑，人们总说，数据库系统中的查询优化难得就跟造火箭似的
19:16 – 19:17
and the joke is that
这里的玩笑是
19.17–19.22
no if you fail at doing query optimization, your backup career plan could do rocket science
如果你不擅长查询优化，那么你的第二职业计划可以是去造火箭
19.22–19.23
, because there's so even harder than that right
因为没有比这玩意更难了
19:24 – 19:30
and this is what's gonna separate the high end guys versus the the open–source guys or the the the smaller systems
这就是高端数据库系统和开源数据库系统以及那些小型数据库系统之间的区别了
19:31 – 19:37
Oracle ,SQL server and IBM and Teradata and all the the the enterprise systems around for a long time
Oracle，SQL server，IBM以及Teradata这些所有企业级数据库系统都在这上面花了很多精力
19:37 – 19:41

they have spent millions and millions of dollars have hired hundreds to hundreds of people to work on these things
他们花了无数的金钱雇佣了上百号人去负责这方面的事情
19:42 – 19:44
and they’re quite sophisticated
查询优化这块实在是太难了
19:45 – 19:52
and so if you know if you can do this kind of stuff you’ll you in demand
So，如果你能做这块的工作，那么你就是他们所需要的人
19:53 – 19.56
So another way you talk we’re gonna talk about this class is that
So，我们要在课上谈论的另一件事情就是
19.56–20.01
you may say, all right well this is like super hard ,can machine learning solve this so can a AI solve it
你们可能会问，Well，这块既然这么难，那么我们能否使用机器学习或者AI来解决呢?
20:02 – 20:02
no but yes right
半对半错
20.02–20.09
so people have tried applying machine learning PR try apply machine learning now more recently
人们最近已经尝试在这方面使用机器学习了
20:10 – 20:11
And seeing some promising results
并且他们也看到了一些有希望的结果
20.11–20.16
but it’s still not not no anywhere near what the you know the commercial systems can do
但这还没有应用到商业数据库系统中
20:17 – 20:21
IBM actually tried something similar back in the early 2000s this thing called leo
实际上，IBM在2000年代初期就已经尝试做出类似的东西了，它叫做Leo
20:21 – 20:22
the learning optimizer
即learning optimizer
20.22–20.24
turns out it was it sucked
事实证明，它很垃圾
20.24–20.27
and everybody you know they shifted in production
他们将这玩意投入了生产
20:27 – 20:32
but every db2 DBA I’ve ever talked to says the first thing they do when they install db2 is turn off that learning crap
但每个DB2的DBA跟我吐槽过，每次安装DB2的时候，他们干的第一件事，就是关掉这玩意
20.32–20.33
cuz it never worked， make things worse
因为这东西没啥用，甚至还让整个系统变得更垃圾
20:34 – 20:39

so just machine learning is a potential way to improve things
So，对查询优化来说，机器学习确实是一种潜在的改善手段
20.39–20.42
but it’s not going to be you know a magic bullet to solve everything
但它并不是那种能够解决一切问题的黑科技
20:43 – 20:44
so as I said
So，正如我所说的
20.44–20.48
this is the one thing about database systems that I’m most excited about and know the least about
这是数据库系统中我最为兴奋的一个地方，同时也是我了解最少的地方
20:49 – 20:52
last semester we covered in the advanced class we did three lectures on this
在上学期的15–721上，我们花了三节课的时间来讲这个
20.52–20.55
I think I’m expanding that to make four or five next semester just ,
我觉得我可能下学期得花4到5节课来讲这块内容
20.55–20.58
because like we build our own query optimizer at CMU
因为我们在CMU的数据库系统中构建了我们自己的查询优化器
20:58 – 21:00
it’s super hard for me to get students come work on it
对我来说，其实很难去带学生一起去做这个