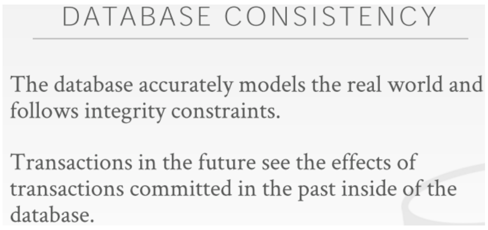
16-03

16-03



DATABASE CONSISTENCY

The database accurately models the real world and follows integrity constraints.

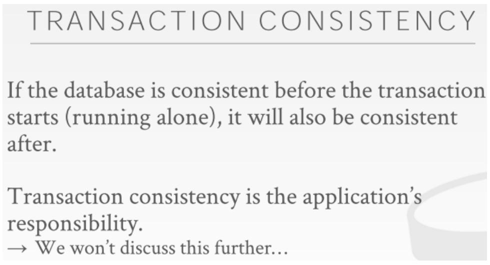
Transactions in the future see the effects of transactions committed in the past inside of the database.

40:04 - 40:05
all right

40.05-40.06
so my transaction commits
So，当我的事务提交了
40.06-40.08
I give back the acknowledgement that I committed
我就会给出我提交了该事务的确认信息
40:08 - 40:11
then you come along and now do another transaction on that same machine
接着，你在同一台机器上执行另一个事务
40.11-40.15
and you read might you should we're gonna read my writes right away
你应该会读取到我写操作所做的修改
40:15 - 40:17
so for a single node database
So，对于一个单节点数据库系统来说
40.17-40.19
this is not that big this is not really an issue
这并不是什么大问题
40.19-40.21
when this matters more is the distributed databases
分布式数据库中会更为关心这个东西
40:22 - 40:25
so now if I'm trying to guarantee strong consistency in my distributed database
So，如果我现在试着保证我分布式数据库中的强一致性
40.25-40.29
if I do a write and I update some account
如果我执行一次写操作，并对某个账户进行更新
40:31 - 40:40
And then you come one millisecond later on another machine for the same logical Database, but on the separate physical machine and you start you now do a read
接着，在1毫秒之后，你在另一台物理机上对同一个逻辑数据库中的数据进行了一次读操作

40:40 – 40:45
you should be able to see my change，if I told the outside world that my transaction committed
如果我已经告诉外界，我的事务已经提交了，那么你应该能够看到我所做的修改

40:45 – 40:47
All right

40.47–40.50
so this will matter more for the distributed databases
So，这在分布式系统中更为重要
40:50 – 40:53
because the NoSQL guys will have this thing called advanced concurrency control
那群使用NoSQL的人表示他们有一个叫做高级并发控制的东西
40.53–40.56
will say I'll propagate changes eventually
他们会说，我最终会传播这些修改
40:56 – 40:59
and not guarantee that everyone sees the exact same state of the DBMS at the exact same time
但并且不保证所有人在同一时刻看到的DBMS状态都是完全相同的
41:00 – 41:01
But for our purposes today
但出于我们今天的目的
41.01–4104
a single ndoe database it doesn't really make sense,
对于单节点数据库来说，这并没有什么意义
41.04–41.05
 it won't be an issue
它不会遇上这种问题

TRANSACTION CONSISTENCY

If the database is consistent before the transaction starts (running alone), it will also be consistent after.

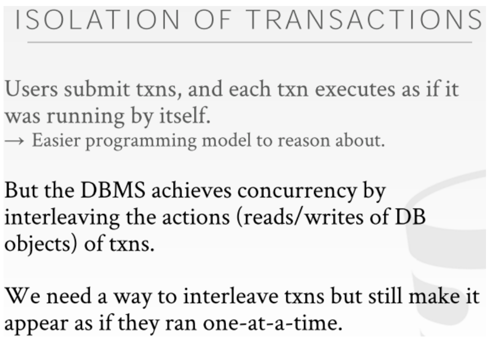
Transaction consistency is the application's responsibility.
→ We won't discuss this further…
41:07 – 41:10
so the other type of **consistency** is **transaction consistency**
So，另一种一致性叫做事务一致性
41:11 – 41:12
and this one again is very hard way
这又是一个很难的东西
41.12–41.14
 but it basically says that
但简单来讲
41.14–41.16
 if the database is consistent before transaction runs
如果数据库在执行事务前，它是一致的

41:17 – 41:19
and our transaction is consistent
并且我们的事务也是一致的
41.19–41.21
 then after we run our transaction
当我们执行完我们的事务后
41.21–41.23
 the end state of the database should be consistent
数据库的最终状态也应该是一致的
41:25 – 41:26
All right

41.26–41.29
so what does it mean to be you know consistent correct
So，一致性正确是什么意思呢
41:29 – 41:33
right that’s a higher level concept that we can’t reason about in our database
这是我们无法在数据库中所解释的一个高级概念
41:35 – 41:38
right we can try to enforce some integrity constraints
我们可以试着强制使用一些完整性约束
41.38–41.43
 ,and we prevent the transaction from doing you know making some changes
我们可以防止事务去做某些修改
41:43 – 41:45
but you know
但你知道的
41.45–41.47
if my application says
如果我的应用程序表示
41.47–41.54
there should be no customer with an account that has you know @cmu.edu
email address
它里面不应存在邮箱后缀为@cmu.edu的客户账号
41:55 – 41:58
and my transaction goes ahead and actually tries to do that
然后，我的事务会尝试去做这件事（知秋注：添加一条这个后缀的客户账号数据）
41:59 – 42:00
I I can’t stop that in my database
我无法在我的数据库中阻止这件事情的发生
42.00–42.02
~~that’s not a bad~~ it’s not a good example
这不是一个好例子
42.02–42.06
~~, because I you know~~ let me rephrase that
让我重新组织下语言
42:06 – 42:08
let’s say there’s the application says
假设，应用程序表示

42.08–42.13
 that nobody taking this class is allowed to have an account on this one system
没有学习这门课的人是不允许拥有该系统的账号
42:13 – 42:17
but my database doesn't have access to whether you're enrolled in this class or not
但我的数据库无权去访问你是否选了这门课
42:17 – 42:19
So the transaction allowed to go ahead and do that
So，我们允许事务去做这件事
42.19–42.22
 ,and the database says okay sure you want to do this in sir, I'm allowed to do that
数据库表示，Ok，我允许你做这件事，你去做吧
42:23 – 42:26
but that's this high level concept, this higher level constraint
但这是一种高级层面的概念，来自更高级层面的限制
42.26–42.28
but the database system doesn't know anything about it
但数据库系统对此一无所知
42:28 – 42:31
So therefore the transaction can consistent
因此，事务可以是一致的
42.31–42.33
, and therefore we can't stop that
因此，我们无法阻止它
42:34 – 42:38
so you know this is something that we can't simply just can't do in our database system
So，你知道的，这种事情我们无法简单地在数据库系统中做到
42:38 – 42:41
we can enforce integrity constraints a referential integrity constraints
我们可以强制使用完整性约束和引用完整性约束
42.41–42.43
we can't afford this high level things
我们无法承受这种高级层面的东西
42.43–42.44
 because we just don't know
因为我们对它一无所知
42:45– 42:47
because it's a human value judgment
因为这是一种人为判断
42.47–42.50
 that we can't codify in our system
我们无法在系统中编写这些东西
42:50 – 42:56
so there's nothing really else to say about this like if you understand the high level what I'm talking about, and that's it
So，关于这个，我们没有什么要讲的，如果你理解我所讲的高级层面的东西是什么的话，那你就懂了
42:56 – 42:58
right that's all that matters okay

这些才是重要的东西



ISOLATION OF TRANSACTIONS

Users submit txns, and each txn executes as if it was running by itself.
→ Easier programming model to reason about.

But the DBMS achieves concurrency by interleaving the actions (reads/writes of DB objects) of txns.

We need a way to interleave txns but still make it appear as if they ran one-at-a-time.

43:00 – 43:02

all right so the moment the other one we care about today is also isolation

So，我们今天所关心的另一个东西就是隔离性

43:03 – 43:04

So isolation again is saying that

So，隔离性指的是

43.04–43.09

~~if our transaction, if we~~ if we have our user submitting users many more transactions

如果用户提交了许多事务

43:09 – 43:13

we want each of them to run assuming that they're running by themself

我们想让每个事务自己做自己的事

43:14 – 43:17

and the reason why we want to provide this this guarantee is that,

我们之所以希望提供这种保障，

43.17–43.23

it makes it way easier to programmer application our logic in our transactions

原因是这能使得我们能更加容易地将我们的逻辑编写到我们的事务中

43:22 – 43:23

if that's the case,

如果是这样的话

43.23–43.25

we assume that we have exclusive access to the database

假设我们拥有对数据库的独占访问权限

43:26 – 43:31

we don't have to worry about any intermediate data we could be reading from other transactions

我们不需要关心我们从其他事务中所读取到的任何中间值

43:32 – 43:35

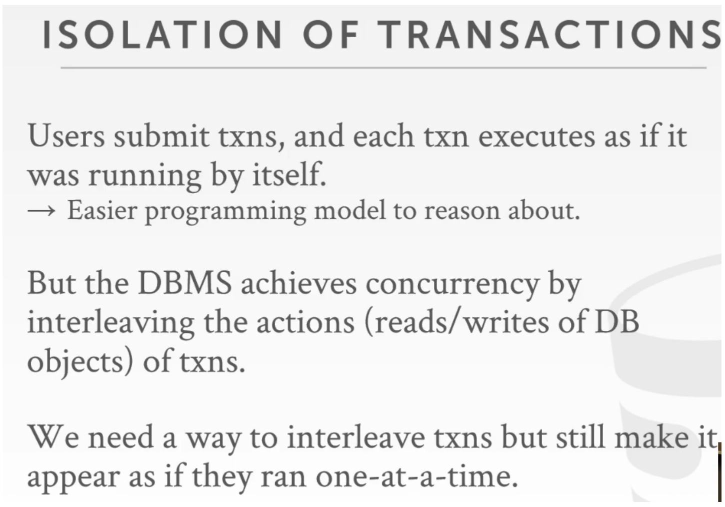
then you know we just write our single-threaded code and that's fine

我们只需去编写我们的单线程代码，这就足够了

43.35–43.36

And it makes life easier

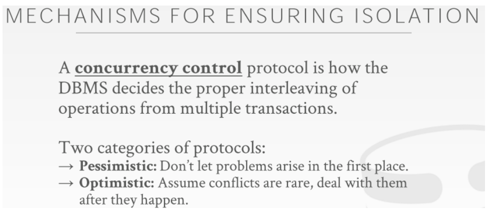
这会让我们处理起来更加容易


ISOLATION OF TRANSACTIONS

Users submit txns, and each txn executes as if it was running by itself.
→ Easier programming model to reason about.

But the DBMS achieves concurrency by interleaving the actions (reads/writes of DB objects) of txns.

We need a way to interleave txns but still make it appear as if they ran one-at-a-time.

43:38 – 43:45
~~so we can achieve that~~ we can achieve this by doing again my straw man approach in the beginning
So，我们可以通过我们在一开始提的strawman方案来做到这点
43.45–43.47
where I just have a single thread actually execute one by one
即我通过单线程来逐个执行这些事务
43:47 – 43:48
but I said that
但我说过
43.48–43.53
we want to be able to interleave transactions to achieve better parallelism at concurrency
我们想能够在并发的时候通过交错执行这些事务以获得更好的并行性
43:56 – 43.58
~~and so we see~~ if you want to be able guarantee this isolation property
So，如果我们想保证这种隔离性
43.58–44.00
, but we still want to interleave
但我们依然想交错执行这些事务
44.00–44.02
 this that becomes difficult
那这就变得困难起来了

MECHANISMS FOR ENSURING ISOLATION

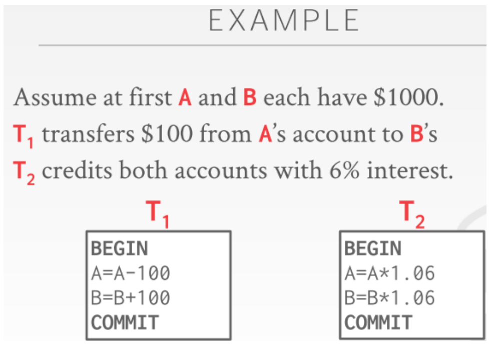
A **concurrency control** protocol is how the DBMS decides the proper interleaving of operations from multiple transactions.

Two categories of protocols:
→ **Pessimistic:** Don't let problems arise in the first place.
→ **Optimistic:** Assume conflicts are rare, deal with them after they happen.

44:03 –44:08
and so the way we're going to provide this way we're going to do this is through a concurrency control protocol
So，我们通过并发控制协议来做到这点
44:09 – 44:13
So we've already talked about concurrency control protocols slightly when we talked about index latching
So，当我们讨论index latch的时候，我们已经稍微讨论了下并发控制协议

44:14 – 44:18
maybe we would have a single data structure and allow multiple threads to access it at the same time
我们可能会有这样一种数据结构，即我们允许多条线程同时访问该数据结构
44:18 – 44:22
and we use our latches to enforce the the correctness of our data structure
我们通过使用latch来强制保证我们数据结构的正确性
44:23 – 44:25
So now we’re gonna do the same thing but for our database objects
So，我们会对我们的数据库对象做相同的处理
44:26 – 44:29
this is why I was making the distinguish between locks and latches
这就是我我为什么要去区分lock和latch的原因
44:29 – 44:31
so latches are protecting the internals of the data structure
So，latch用来保护数据结构中的内部信息
44.31–44.35
 locks are gonna protect these database objects
lock则用来保护这些数据库对象
44:35 – 44:40
so you think I’ll call as like the traffic cop for for the database system
So，你可以将它们想象为数据库系统中的交警
44:40 – 44:44
right it’s sitting saying this we can let this operation go,
它表示，我们可以让这个操作执行
44.44–44.45
this operation has to wait
这个操作需要等待执行
44.45–44.46
or this operation has to abort,
或者，这个操作需要被中止
44.46–44.51
it’s trying to figure out how to interleave things in a way that we end up with a correct state
它会试着去弄清楚我们该以何种方式交错执行这些操作，并最终得到正确的结果
44:52 – 44:55
So there’s two categories of protocols that we’re gonna care about
So，这里我们关心两种并发控制协议
44.55–44.59
, and then this will this we’ll cover on on next week
我们会在下周的时候介绍这个
45:00 – 45:01
right Italy both of these
这两种我们都会讲
45:02 – 45:04
so the first one is a pessimistic protocol
So，第一种是悲观协议
45.04–45.08
,where we’re gonna assume that our transactions are going to conflict to cause problems

我们会假设我们的事务在执行的时候会产生冲突，导致问题的出现
45:09 - 45:14
so we require them to acquire locks before it allowed to do anything
So，在我们允许这些事务执行操作之前，我们会先要求它们去获取lock
45:15 - 45:19
right,you assume that you know you're pessimistic we assume that there's many Problems
假设我们使用的是悲观协议，假设这些事务执行的时候会有各种问题
45:19 - 45:25
you make sure that things go in the correct order by using locks
通过使用lock，你会确保这些事务以正确的顺序执行
45:26 - 45:28
optimistic concurrency control is
乐观并发控制指的是
45.28-45.29
where you **assume that the conflicts are rare,**
我们假设这些冲突问题很少会出现
45.29-45.32
most of the time my transactions aren't going to conflict
大部分情况下，我们的事务间不会产生冲突
45:32 - 45:38
so rather than making them stall and acquire it locks at the very beginning, I just let them run,and do whatever they want
So，我只是让这些事务直接执行，干它们想干的事情，而不是让它们在一开始的时候停下来去获取lock
45:37 - 45:40
and then when they go to commit
接着，当它们提交的时候
45.40-45.44
,go back and figure out whether that was actually the right thing to do, whether there was a conflict
它们会回过头去看看它们做的事情是否正确，执行期间是否存在冲突
45:45 - 45:48
so Monday's class next week will be on two-phase locking
So，下周一的时候，我们会讲两阶段锁
45.48-45.49
,that's a pessimistic protocol
这是一个悲观协议
45:50 - 45:52
Wednesday's class next week will be on timestamp ordering
下周三我们会讲一种基于时间戳顺序的协议
45.52-45.54
, that's considered an optimistic protocol
它被认为是一种乐观协议
45:54 - 45:58
and optimistic concurrency control protocol was actually better here at CMU in the 1980s
实际上，在1980年代，CMU的乐观并发控制协议比其他人的更好



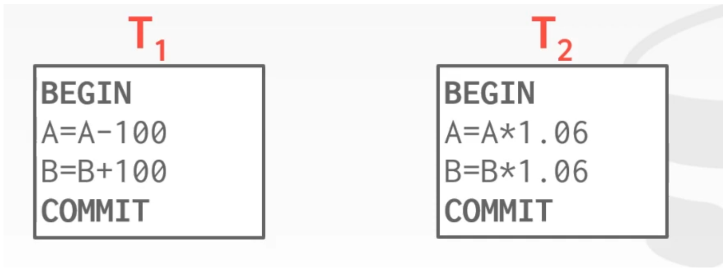
EXAMPLE

Assume at first A and B each have $1000.
T_1 transfers $100 from A's account to B's
T_2 credits both accounts with 6% interest.

```
T1              T2
BEGIN           BEGIN
A=A-100         A=A*1.06
B=B+100         B=B*1.06
COMMIT          COMMIT
```

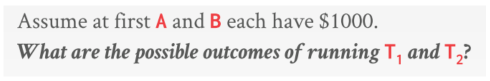
46:00 – 46:01
All right

46.01–46.06
so let's look now at some real examples understanding what it actually mean to have complex
So，我们会通过一些真实案例来理解下它实际为什么复杂
46:07 – 46:10
So again this is my bank account example
So，这是一个关于银行转账的例子
46.10–46.14
where we have two accounts A and B ,it's Andy and its bookie
我们有A和B两个账户，一个是Andy的，另一个是他的博彩账户
46:14 – 46:18
and so we want to transfer 100 dollars out of my account into my bookies account
So，我想从我的一个账户上转100美金到另一个账户上
46:18 – 46:20
but then at the same time
但与此同时
46.20–46.25
 the bank runs transaction where it's going to update the the monthly interest of all the bank accounts
银行的数据库系统试着执行某个事务，该事务会去更新所有银行账户的月利息
46:25 – 46:28
So we're going to update every account with an add 6% interest
So，我们会为所有账户都增加6%的利息



```
T1              T2
BEGIN           BEGIN
A=A-100         A=A*1.06
B=B+100         B=B*1.06
COMMIT          COMMIT
```

46:29 – 46:33
I so transaction one is taking hundred dollars out of A, put a hundred dollars and B
So，T1所做的事情就是从A中取100美金，并转给B
46.33–46.38
 and then transaction two is just computing ,you're incrementing both accounts by adding 6%
T2干的事情就是将所有账户中的钱乘以1.06


Assume at first A and B each have $1000.
What are the possible outcomes of running T_1 *and* T_2*?*

46:40 – 46:44

~~so if we~~ assume that again both bank accounts have a thousand dollars

So，我们假设这两个账户中都有1000美金

46:45 – 46:47

and we want to execute these two transactions

我们想去执行这两个事务

46.47–46.51

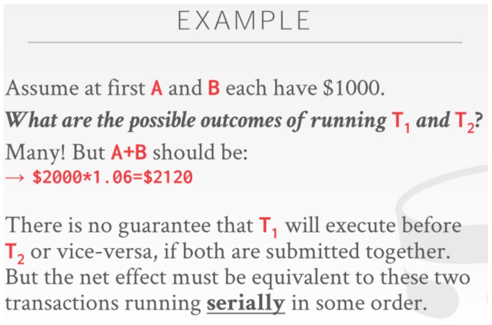
what are the possible outcomes we could have for the state of the database

数据库可能出现的状态是什么?

46:53 – 46:55

all right assuming we have arbitrary interleavings

假设，我们会交错执行这些操作


EXAMPLE

Assume at first A and B each have $1000.
What are the possible outcomes of running T_1 *and* T_2*?*
Many! But A+B should be:
→ $2000*1.06=$2120

There is no guarantee that T_1 will execute before T_2 or vice-versa, if both are submitted together. But the net effect must be equivalent to these two transactions running **serially** in some order.

46:57 – 46.59

well many

Well，我们会得到很多结果

46.59–47.04

right, because we could have t1 maybe go do run run in one query then switch over to t2, then back and forth

因为T1可能是在某个查询中执行的，接着我们切换到了T2，如此反复

47:05 – 47:09

right there’s a bunch of rdifferent ways we can end up with these interleavings

通过这种交错执行，我们最终会得到一堆不同的结果

47:09 – 47:12

but the important thing to point out though is that

但此处重要的地方在于

47.12–47.18

at the end of the day after we execute transaction t1 and t2 in any possible order

到头来，我们以任意可能的顺序执行完T1和T2后

47:18 – 47:21

,to know that our database state is correct

为了知道我们数据库的状态是否正确

47.21–47.26

the final result when we add both the accounts together, should be 2120

当我们将这些账户中的钱加在一起时，最终结果应该是2120

47:26 – 47:28

Because I have a thousand dollars on A

因为A账户中我有1000美金

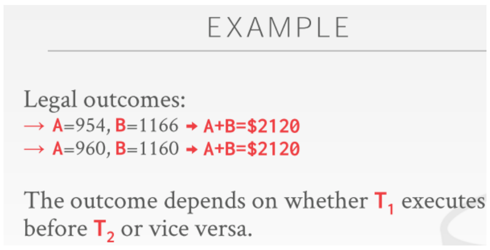
47.28–47.28

thousand dollars to B

B账户中也有1000美金

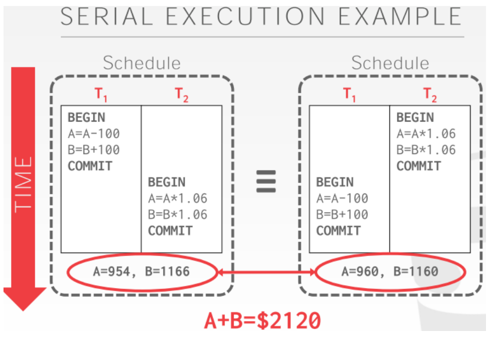
47.28–47.30
 add that together
将它们加在一起
47.30–47.30
that’s two thousand
它们的总和就是2000美金
47.30–47.33
,and then the second transaction wants to add 6% interest
接着，第二个事务想去增加6%的利息
47:34 – 47:39
so we want to guarantee that no matter how we order or interleave our operations
So，我们想去保证，不管我们以什么顺序交错执行我们的操作
47.39–47.42
we always at the end after executing t1 and t2
当我们执行完T1和T2后
47:42 – 47:44
we end up with 2120
我们的最终结果是2120
47:45 – 47:48
so this is a very important property about transactions and database systems,
So，这是事务和数据库系统中一个非常重要的特性
47.48–47.54
 that’s gonna be slightly different than maybe how you you know or have experienced parallel programming before
这可能和你们以前学过的并发编程有所不同
47:55 – 47:59
so in a database system that we’re talk about here
So，在我们这里所讨论的数据库系统中
48:00 – 48:04
even though t1 may be submitted to the database system first followed by t2
即使T1先被递交给数据库系统，紧接着T2被递交给数据库系统
48:05 – 48:08
there’s no guarantee the database system is gonna run t1 first
数据库系统也不会去保证它会先执行T1
48:11 – 48:13
right and the reason why we’re gonna do this is
我们这样做的原因是
48.13–48.17
because we can have any possible interleaving or any any possible ordering
因为我们可能会以任意可能的方式交错执行
48:18 – 48:23
then this is gonna allow us to open up more opportunities to do interleaving to get better parallelism
这就会让我们有更多机会去交错执行这些操作，以获得更好的并行性
48:25 – 48:31
~~if I care~~ my application absolutely had to care say well t1 absolutely execute first, then followed by t2
如果我的应用程序表示，T1必须先执行，接着才去执行T2
48:32 – 48:33

the way you would write that code is
你编写这种代码的方式是
48.33-48.34
 you submit t1
你先递交T1给数据库系统
48.34-48.37
and then only when you get back to your knowledge meant that t1 committed
只有当你收到T1已经被提交的通知时
48.37-48.41
 then you submit T2
然后，你才会去递交T2
48:41 – 48:42
Because the you can’t guarantee that
因为你无法保证它们的执行顺序，所以只能以这种方式保证它们的执行顺序
48.42-48.43
,now in practice
在实战中
48.43-48.44
if you submit t1
如果你向数据库系统递交了T1
48.44-48.46
you know it takes a minute
T1花了1分钟执行完它的工作
48.46-48.47
, then you submit t2
然后，你再递交T2
48.47-48.48
that’s basically the same thing
简单来讲，这是一回事
48:48 – 48:51
but if I submit them at exactly the same time
但如果我在同一时间将它们递交给数据库系统
48.51-48.56
then the database system could interleave them and it anyway it wants
那么，数据库系统就会以它想要的方式来交错执行它们的操作
48:56 – 48.59
but what we’re gonna care about those that
但我们这里所在意的事情是
48.59-49.00
for any arbitrary interleaving
对于任意交错执行的顺序来说
49:00 – 49:08
we want the end state of the database to be equivalent to one where we execute these transactions in Serial order with a single thread
我们希望该数据库的最终状态与我们在单线程中按照顺序执行这些事务所得到的结果相同
49:09 – 49:11
either t1 followed by t2, or t2 followed by t1
不管是先执行T1再执行T2，或者是先执行T2再执行T1
49:12 – 49:14

The end state of the database system has to look like that
数据库系统的最终状态必须看起来像这样

EXAMPLE

Legal outcomes:
→ A=954, B=1166 → A+B=$2120
→ A=960, B=1160 → A+B=$2120

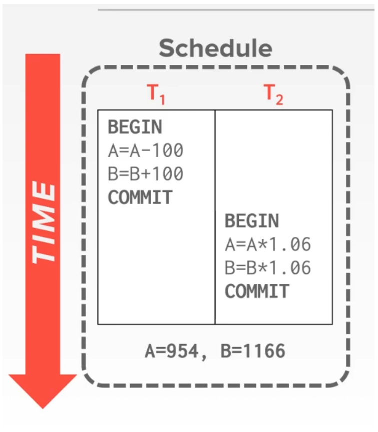
The outcome depends on whether T_1 executes before T_2 or vice versa.

49:16 – 49:17
so now that means that
So，这意味着
49.17–49.23
the number of possible outcomes we could have are for the state A and B could be different
在不同的执行顺序下，我们拥有的A和B的状态可能是不同的
49:23 – 49:26
right so if I have say t1 go first all by t2
So，如果T1先执行，再执行T2
49:27 – 49:31
I’ll have 954 dollars in a ,and and 1166 dollars and B
那么，A账户中就会有954美金，B账户中就会有1166美金
49:31 – 49:33
but I go the other order
但如果我以另一种顺序执行
49.33–49.35
 I’ll have 960 and 1160
那么，A账户中就是960美金，B账户就是1160美金
49:36 – 49:38
but again if I add both of these together
但如果我将两个账户中的钱加起来
49.38–49.39
 I always get 2120
我得到的结果始终是2120
49.39–49.44
,and that’s again that’s equivalent to one where execute in serial order
该结果等同于我们按照顺序执行的结果
49:47– 49:47
so is this clear
So，你们懂了吗


```
BEGIN
A=A-100
B=B+100
COMMIT
BEGIN
A=A*1.06
B=B*1.06
COMMIT
```
49:50 – 49:50
all right

49.50-49.53
so let’s actually look at what the database sees
So，我们来实际看下数据库中是怎么样的
49:53 – 49.56
so for this this is this is called a schedule
So，这种东西叫做Schedule
49.56-50.02
 for our transactions, in a way to read this is that going from the top to bottom for time
我们的事务会以某种方式随着时间自上而下执行这些操作
50:02 – 50:05
and then for each of these columns here we have the transactions
接着，这里的每一列就是一个事务
50.05-50.07
 ,and we have the operations that they’re actually doing
这里面放着这些事务实际干的事情
50:07 – 50:09
so I call began on t1
So，我通过调用BEGIN来开始执行T1
50.09-50.10
 I take a hundred dollars out of A
我先从A账户中取100美金
50.10-50.12
, take put a hundred dollars in B,
将这100美金放入B账户中
50.12-50.13
 and I call commit
然后，我调用COMMIT提交T1
50:13 – 50:16
and then now next time I do a context switch over here
接着，我进行上下文切换到T2
50.16-50.19
 and now I call it t2 and it computes the interest in these guys
我开始执行T2，让它去计算这两个账户的利息
50:19 – 50:19
so for this
So，在这个例子中
50.19-50.23
assume that we only have a single thread that can with a single program counter
我们假设我们只有一条线程，它里面有一个程序计数器
50.23-50.26
,and we can only actually one operation at a time
并且我们一次只能执行一个操作
50:26 – 50:28
like we can interleave them of these different transactions
我们可以交错执行这些不同的事务
50.28-50.32
but at any given time, we going do one thing
但在任何给定的时间中，我们只能做一件事

50:32 – 50:33
So in this case here
So，在这个例子中


Schedule

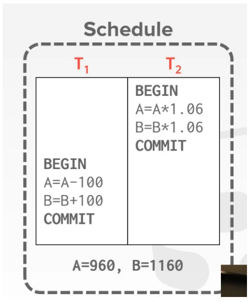
T₁ | T₂
BEGIN
A=A-100
B=B+100
COMMIT

BEGIN
A=A*1.06
B=B*1.06
COMMIT

A=954, B=1166

50.33–50.35
, if we execute t1 followed by t2
如果我们先执行T1再执行T2

50.35–50.38
we end up with this this amounts for A and B
那么，我们最终A账号和B账户中的钱如图所示
50:38 – 50:40
if he asked you t2 first followed by t1
如果我们先执行T2再执行T1


Schedule

T₁ | T₂
BEGIN
A=A*1.06
B=B*1.06
COMMIT

BEGIN
A=A-100
B=B+100
COMMIT

A=960, B=1160

50.40–50.41
, we end with these amounts here
最终它们账户中的金额是这样的
50:42 – 50:44
so again A doesn’t match
So，这两个A的结果并不相同
50.44–50.47
,A`s 954 over here at 960 over here
左边的A账号中的钱是954美金，右边则是960美金
50:47 – 50:52
so they’re technically different from a finite exact amount
So，从技术上来讲，这两个账户中的钱是不一样的
50:52 – 50:54
But from a databases consistency perspective
但从数据库一致性方面来讲
50.54–50.56
for what we’re caring about with transactions,

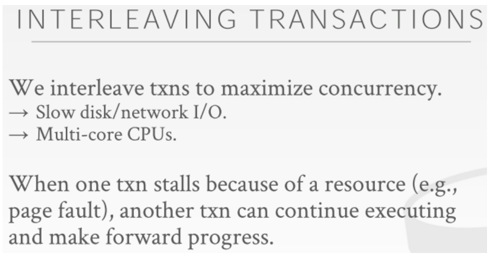
这也是我们在事务中所关心的东西
50.56-50.58
you add them both up and you always get 2120
当我们将A和B的钱相加，我们得到的始终是2120
50:59 – 51:04
so both of these interleavings are both of these orderings are still are equivalent to each other, they're correct
So，这两种交错执行得到的结果是等同的，它们是正确的
51:06 -51:06
yes
请问
51:16 – 51:18
sorry your question is
你的问题是
51.18-51.21 ！！
if you know that you have you 2 transactions exactly the same time
如果我们知道我们同一时间有两个事务在执行
51.21-51.23
could you do it we're like you could you combine them
我们是否能将这两个事务合并为一个事务
51:30 – 51:30
yes
其实是可以的
51.30-51.32
but nobody does it that way
但没有人会这样做
51.32-51.34
and I think that would complicate things right now
我觉得这样会让事情变得复杂
51:35 – 51:37
let's just assume that this is the case
我们就假设这种情况是存在的
51.37-51.39
 also say to what I'm showing here or like
就以我这里所展示的例子来说
51:40 – 51:42
yeah here's why you can't really do that
~~这就是你没法这样做的原因~~
这里为什么不能那样来做
51:42 – 51:46
So when I'm gonna talk about here today are like the the schedule is fixed
So，我今天所讨论的schedule是固定的
51.46-51.50
meaning I know ahead of time exactly what all the transactions actually want to do
这意味着我提前知道了所有事务想要干的事情
51:50 – 51:51
In a real system
在一个真正的DBMS中
51.51-51.52

it’s not like that
它并不是这样的
51.52-51.52
in a real system
在一个真正的系统中
51.52-51.57
you have like you know transactions are showing up they’re calling you know a client opens connection calls begin
当事务出现在DBMS中的时候，某个client打开了与DBMS的连接，并调用BEGIN
51:58 – 51.59
and then it starts executing a bunch of queries
接着，它开始执行一系列查询
51.59-52.01
and you don’t know what the next query is
并且你并不会知道下一个查询是什么
52:02 – 52:04
In this case here
在这个例子中
52.04-52.05
sort of to reason about correctness
我们是为了解释正确性
52:06 – 52:07
you see everything all at once
你一下就能看到所有的信息
52:08 – 52.09
right

52.09-52.10
so on Monday
So，在周一的时候
52.10-52.11
when we talk about two-phase locking,
我们会讨论两阶段锁
52.11-2.14
that’s a dynamic conquerer protocol
它是一个动态分治协议
52.14-52.17
where you don’t know what the queries are gonna be ahead of time
即你不会提前知道要执行哪些查询
52:17 – 52:19
now there’s some cases
这里有一些例子
52.19-52.22
where if you have some introspection but with the applications actually trying to do
我程序内部有实际要尝试去做的一些事
52.22-52.23
then you can actually do what you propose
那么，你实际可以去做你打算做的事情
52:24 – 52:26

but that’s hard nobody actually does that
但这很难做到，实际没人会这样做
52:37 – 52:38
will get that too
我们之后会讲这个
52.38–52.40
so his question is which is correct
So，他提的问题很正确
52.40–52.41
,I said before
我之前说过
52.41–52.43
the database system only sees reads and writes
数据库系统只会看到这些读和写操作
52.43–52.47
like this A equals a minus 100 right
就比如这里的A=A–100
52:47 – 52:49
yes that will get translated to a read followed by write
然后，它会被翻译为一个读操作，后面紧跟着一个写操作
52:49 – 52:51
I’ll see that in a sec yes
我们稍后会看到这个
52:53 – 53:02
yes yeah
请问
53:05 – 53:08
Yeah we’ll get to that yes question is
So，他的问题是
53:09 – 53:12
So this A=A–100,
So，我们来看下这个A=A–100
53.12–53.13
 what is it actually gonna look like
它看起来是怎么样的呢
53.13–53.14
 well in the program logic
Well，在程序逻辑中
53.12–53.16
I would say do a get on A, you’re reading A
我们会去获取A，然后读取A
53.16–53.18
have a copy my local variable
并拥有一份A的本地变量副本
53:18 – 53:19
then I can manipulate it
然后，我就可以对它进行操作
53.19–53.21
and and write it back to the database

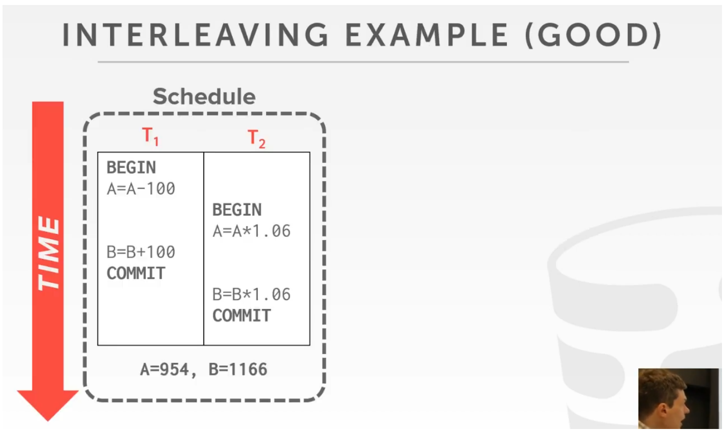
并将它写回数据库
53:21 – 53:25
So each of these transactions would have their own local variables ,that aren’t shared
So，每个事务都会有自己的本地变量，事务彼此之间不会共享这些变量
53:29 – 53:30
his questions
他的问题是
53.30–*53.32
 can you interleave the operation between transactions ,yes we’ll get there exactly yes
我们是否交错执行事务的操作，没错，我们可以做到
53:47 – 53:48
the question is
她的问题是
53.48–53.50
if I have two transactions
如果我有两个事务
53.50–53.53
 that are touching completely different objects not tuples， objects
它们涉及的是完全不同的对象，是对象，不是tuple
53:56 – 53:57
do I need to still serialize this
我是否依然需要按顺序执行?
53.57–53.59
I mean
我的意思是
54:04 – 54:06
so for this one I’m just try toshow equivalency
So，在这个例子中，我们只是向你们展示这种相等性
54:07 – 54:09
If they touch clearly different things
如果这些事务涉及的都是不同的东西
54.09–54.10
 and there’s no conflicts,
它们也就不会产生任何冲突
54.10–54.12
 then you can interleave them any way you want absolutely, yes
那么你就可以以你喜欢的顺序去交错执行这些事务，请问
54:19 – 54:19
her question is
她的问题是
54.19–54.22
 how do I know whether another transaction is touching the same thing, I’m touching
我如何知道另一个事务涉及的对象和我涉及的是否是相同的对象
54:23 – 54:26
again but this is a high-level example
这是一个高级层面的例子
54.26–54.28
the database sees and reads and writes
数据库看到的是这些读操作和写操作

54:28 – 54:30
so I do a read on an object A
So，我对对象A进行读取
54.30-54.32
 you do a read an object A
你对对象A进行读取
54.32-54.35
in order for me to serve your read request ask me to read it for you
为了处理你的读请求，我会为你读取对象A
54:35 – 54:36
so I see everything
So，我看到了所有的东西
54:37 – 54:38
But I don’t see high-level things
但我并没有看到这些高级层面的东西
54.38-54.44
 like I don’t see that your your that you’re going to take the value of a ,and then add 6% to it
我们不会看到这种东西，比如，你要拿到A的值，然后给它增加6%

INTERLEAVING TRANSACTIONS

We interleave txns to maximize concurrency.
→ Slow disk/network I/O.
→ Multi-core CPUs.

When one txn stalls because of a resource (e.g., page fault), another txn can continue executing and make forward progress.
54:47 – 54:53
so again what everyone’s sort of getting up to now is be able to interleave, these these transactions really the operations
So，我们现在要做的就是去交错执行这些事务，其实是交错执行这些操作
54:53 – 54:54
and we’ve already covered this
我们已经介绍过这个了
54.54-54.54
 we want to do this
我们想要做到这点
54.54-54.59
 because this is slow and we have a lot of CPU cores
因为它的速度很慢，我们拥有很多CPU核心
54:59 – 55:00
and so the idea here is again
So，这里的思路是
55.00-55.08
~~ that instead of having the, you know~~if we have to go to disk to get something or wait to acquire a latch on something
如果我们需要跑到磁盘上去获取某个东西，或者等待获取某个对象的latch
55:08 – 55:10
we could have one transaction stall
我们可以让其中一个事务停下来
55.10-55.13

another transaction keep on running and still make forward progress
另一个事务继续执行，并取得了一些进展
55:13 – 55.19
so we’re trying to figure out a schedule or interleaving such that we maximize the performance of the system
So，我们试着找到一种schedule（调度方案），以此来最大化系统的性能
55.19–55.20
and we get the best utilization of our hardware
这样我们就能将我们的硬件性能发挥到极致


INTERLEAVING EXAMPLE (GOOD)

Schedule

T_1	T_2

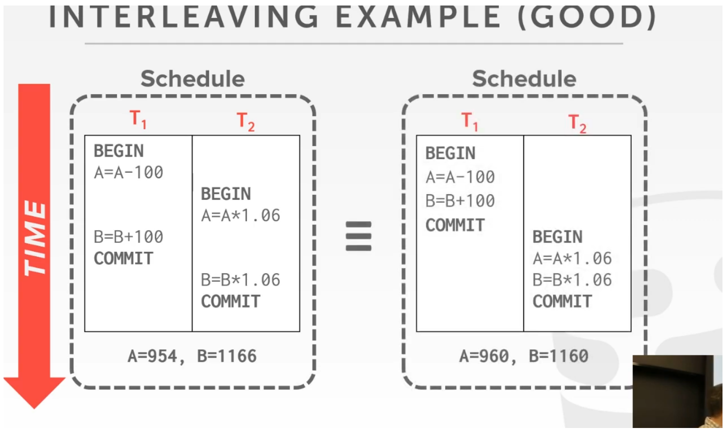
BEGIN
A=A-100	

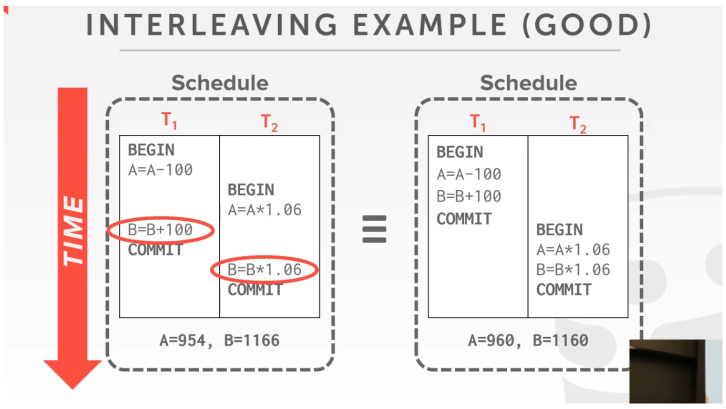
	BEGIN
	A=A*1.06

B=B+100 COMMIT	
	B=B*1.06 COMMIT

A=954, B=1166

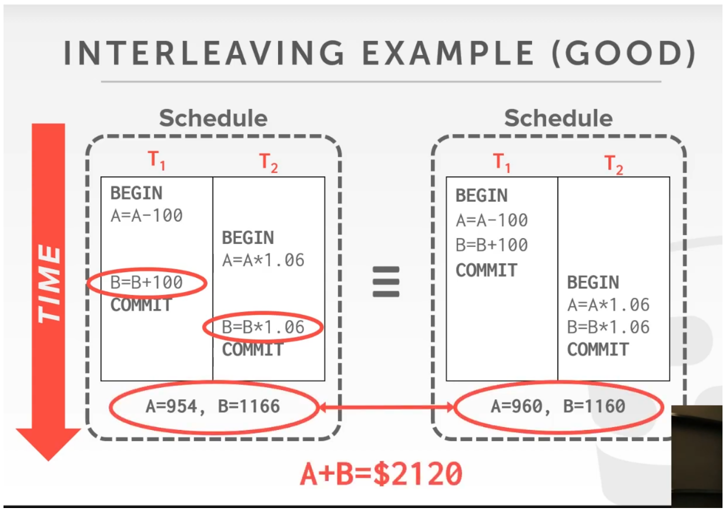
55:21 – 55:23
So if we go back here now to our example
So，我们回过头来看下我们的例子
55:24 – 55:26
So now I‘m interleaving our transactions ,right
So，我交错执行了我们的事务
55:27 – 55:29
going start takes 100 out of A
首先我们从A中取100美金
55.29–55.32
then then does a context switch t2 starts
接着，切换上下文，开始执行T2
55.32–55.34
put compute 6% on A
将A乘以1.06
55.34–55.35
then we go back
接着，我们回过头去
55.35–55.37
and take put the hundred dollars back on B
将这100美金放到B的账户中去
55.37–55.38
,and go back here
然后，回到这里
55.38–55.40
compute the interest on that
去计算B的利息
55.40–55.41

and then we go commit
然后，我们提交事务
55:41 – 55:45
so now again now it's not one transaction running in its entirety at a time
So，现在同一时间有多个事务在运行
55:46 – 55:48
Right we're now able to actually interleave things
实际上，现在我们能够做到交错执行


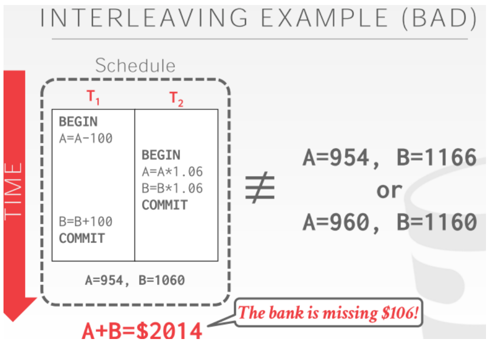
55:48 – 55:49
and this example here
在这个例子中
55.49–55.50
 this is correct
这样是正确的
55.50–55.51
 this is fine
这样做没问题
55.51–55.56
because this is equivalent to a serial ordering of our transactions
因为这和我们按照顺序执行事务的效果是相同的
55:57 –55:59
right the end state of the database is equivalent
数据库的最终状态是相等的


56:01 –56:04
Alright,and so the key thing to point out here is
So，这里要指出的关键地方在于
56.04–56.06

that the reason why this worked out okay
为什么这样做是可行的，它的理由是什么



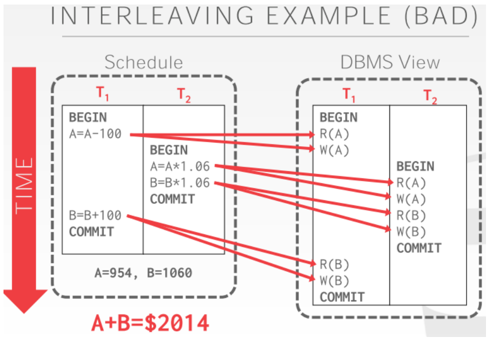
56:06 – 56:07
and then we end up equivalent is that
并且我们最终结果是相等的
56.07–56.18
 we always make sure that we did the operations on t1 first on a given object before we did that operation on on t2 for that same object
我们始终确信在T2对同一个对象执行操作前，T1会先对该对象进行操作
56:19 – 56:20
so I took $100 on a
So，我从A那里取100美金
56.20–56.23
and then I keep you to the interest on a
接着，我去计算A的利息
56.23–56.24
and then I put $100 back and B
接着，我将这100美金放到B的账户中去
56.24–56.26
and then a compute interest on B
然后去计算B的利息

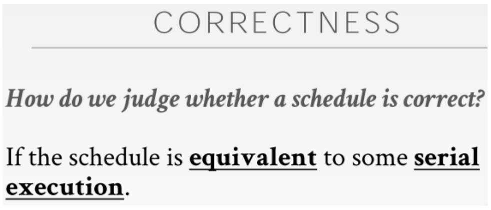


56:28 – 56:32
right so for this interleaving here is
So，对于此处这种交错执行来说
56.32–56.34
we violate that
我们违反了这一点
56.34–56.37

so I take $100 a ,I compute there’s interest on a
So，我从A中先取了100美金，然后我去计算A的利息
56:37 - 56:39
then I compute the interest on B
接着，我又去计算B的利息
56.39-56.41
then I put $100 back on B
然后，我将100美金放到B的账户中
56:41 - 56:43
so now in this case here
So，在这个例子中
56.43-56.45
the when I add up these two values together
当我将这两个值相加在一起
56.45-56.48
I don’t get 2120,I get 2014
我得到的并不是2120，而是2014
56:48 - 56:53
so the bank lost you know $106
So，银行损失了106美金
56:54 - 56:54
all right

56:56 - 56.58
now here’s the hundreds of dollars
虽说这里损失的只是100美金左右
56.58-56.59
but it’s a billion dollars
但如果是10亿美金的话，那就很可怕了
56.59-57.00
all right guys it’s your account hundred dollars is a lot
对于你们的账户来说，100美金算是不少了
57:01 - 57:05
but like you know this is why we want to guarantee that we always have correctness for transactions
这就是我们想保证事务正确性的原因所在了
57:06 - 57:08
especially when you’re doing and ending involves money
特别是，当我们涉及跟钱有关的事情更是如此
57:09 - 57:11
there’s a famous example a few years ago
在几年前有一个非常著名的案例
57.11-57.14
where some Bitcoin exchange I forget where in the world was running on MongoDB
我记得不是特别清楚了，当时某个比特币交易所使用的是MongoDB
57:15 - 57:18
MongoDB did at the time didn’t have support transactions
他们那时候使用的MongoDB并不支持事务
57:18 - 57:26

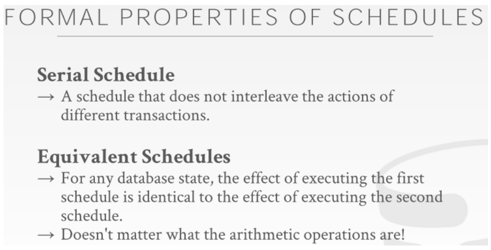
and so some hacker figured out that you can have you can manipulate the the API and have it drain out everyone’s account
某个黑客通过操纵API榨干了所有人账户上的比特币
57:26 – 57:31
so they wiped out the Bitcoin exchange in a single day
So，他们一天之内就毁掉了这个比特币交易所
57:32 – 57:35
Because MongoDB wasn’t doing transactions, that’s part of the story
因为MongoDB不支持事务，这是导致这事情发生的部分原因
57:35 – 57:39
but they didn’t have transactions that’s a bad idea
他们不使用事务，这是个糟糕的想法


57:40 – 57:41
so came back to his point
So，回到这里
57.41–57.43
 well what if the database system actually seen
Well，数据库系统实际看到了什么呢?
57.43–57.45
again,it doesn’t see these higher–level operations
它并没有看到这些更高级的操作
57.45–57.47
it just sees these reads and writes
它所看到的就是这些读写操作
57:48 – 57:49
and so essentially we’re trying to do is
So，本质上来讲，我们所试着做的事情是
57.49–57.55
make sure that for any object that does a read or does a write or read on an object
我们要确保当我们对任意对象进行读或者写操作的时候
57:56 – 57.57
if another transaction is doing the same thing
如果另一个事务也在做相同的操作
57.57–58.03
we’re always going in the right order to determine whether our schedule is correct
我们始终会通过正确的执行顺序来判断我们的调度是否正确

CORRECTNESS

How do we judge whether a schedule is correct?

If the schedule is **equivalent** to some **serial execution**.

58:04 – 58:10
so the way we're going to figure this out, the way we're going to find correctness for what we're talking about here today is
我们今天所讨论的保证正确性的方式是
58.10–58.20
 well say that a schedule of any arbitrary ordering of operations is correct，if it is equivalent to one of serial schedule
如果某个schedule的执行结果等同于按顺序执行的结果，那么我们就会说这种执行顺序的schedule是正确的


FORMAL PROPERTIES OF SCHEDULES

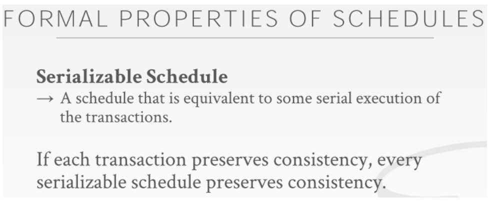
Serial Schedule
→ A schedule that does not interleave the actions of different transactions.

Equivalent Schedules
→ For any database state, the effect of executing the first schedule is identical to the effect of executing the second schedule.
→ Doesn't matter what the arithmetic operations are!

58:21 – 58:23
so the serial schedule we've already talked about
So，我们已经讨论过了什么是Serial Schedule
58.23–58.24
serial schedules just saying that
Serial Schedule的意思是
58.24–58.27
 we actually transactions one after another ,and no interleaving
我们实际会逐个执行事务，而不是交错执行它们
58:27 – 58:29
and then the equivalent policy says that
等效策略指的是
58.29–58.39
if the final state of the database is of the the objects is equivalent
如果数据库中这些对象的最终状态是相等的
58.39–58.44
or it has actually the same values of another database state
或者是，它和另一个数据库状态中的值是相等的
58.44–58.46
then they are equivalent
那么，它们的执行效果就是相同的
58:46 – 59.00
so a a ordering of a schedule well it could be equivalent to at least one exactly one serial ordering I'm not exactly one one or more serial orderings，if the database is still the same state
So，Serial Schedule是包含在具有正确结果的Ordering Schedule之中的，我不确定到底是其中的哪一个，但它们最终都会有一个相同的状态

59:00 – 59:03
but a given schedule could be correct
但对于一个给定的Schedule来说，它可能是正确的
59.03–59.07
it could still be serializable by being equivalent to any possible serial ordering

它可能依然是Serializable的，它等价于任何可能的Serial Ordering Schedule


FORMAL PROPERTIES OF SCHEDULES

Serializable Schedule

→ A schedule that is equivalent to some serial execution of the transactions.

If each transaction preserves consistency, every serializable schedule preserves consistency.

59:09 – 59:12

so this is the formal of property we're going to care about for our schedules serializability

这就是我们在Serializable schedule中所要关心的一种重要属性

==============

59:13 – 59:14

okay it just says that

这里表示

59.14-59.17

a schedule is that is equivalent to some serial execution

如果一个schedule的执行结果等同于某种按顺序执行的结果

59.17-59.20

doesn't matter which one it has to be it has to be one of them

不管它是按照哪种顺序执行

59:20 – 59:22

if that it's equivalent that serial ordering

如果该schedule的执行结果等于按顺序执行的结果

59.22-59.25

then whatever schedule what we're looking at is considered to be serializable

那么，不管我们看的这个schedule是什么，它都是Serializable的

59:26 – 59:29

And this is the gold standard of what you want to get in a DBMS

这就是你想在DBMS中做到的黄金准则

59.29-59.35

this is guaranteeing almost guaranteeing all the protections you could ever want

基于该保证，你可以获得你想要的所有保护手段

59:35 – 59:37

the only one who doesn't guarantee is that

它唯一不保证的东西就是

59.37-59.42

if your transaction is t1 shows up first followed by t2， t1 will commit first

如果T1先执行，后面紧跟着T2，T1会先被提交

59:42 – 59:45

that's called a strict serializeability or external consistency.

这个叫做Strict Serialzability，或者叫外部一致性

59.45-59.47

we don't care about that here

这里我们并不关心这个

59.47-59.48

most systems don't provide that

大部分系统也不提供这种东西

59:48 – 59:51

the only system to provide that that I'm aware of is Google spanner

据我所知唯一提供该功能的系统就是Google的Spanner
59:51 – 59:56
and they need it for some global ads thing, most systems don’t do that
他们在全球广告方面会用到这个东西，但大多数数据库系统不会用到它
59:57 – 01:00:02
most of systems if they say the support Serializability they’re Getting you’re getting what I’m defining here
如果大部分系统表示他们支持有序执行，他们所说的其实就是我这里定义的东西