

19-01

19 – Multi-Version Concurrency Control

00:18 – 00:19

hi my name is Dana

Hi, 我叫Dana

0.19-0.23

so I'm another one of Andy's PhD students

So, 我是Andy的另一个Phd学生

0.23-0.29

and probably going to be the last of his PhD students to present to you, before he gets back

我也可能是他回来之前, 给你们上课的最后一个Phd学生了

00:30 – 00:35

so today I'm going to be presenting very last lecture on concurrency control

So, 今天我要给你们上的是关于并发控制的最后一节课

ADMINISTRIVIA

Project #3 is due Sun Nov 17th @ 11:59pm.

Homework #4 was released last week.
It is due Wed Nov 13th @ 11:59pm.

00:37– 00:39

so before we start

So, 在我们开始之前

0.39-0.39

a couple of reminders

要提醒你们两件事

0.39-0.45

the first is the project 3 is due on Sun Nov 17th before midnight

首先要讲的是Project 3会在11月17号截止

00:46 – 00:48

we also released homework 4 last week

上周我们也放出了Homework 4的相关内容

0.48-0.52

,and then will be due on Nov 13th before midnight

它的截止日期是在11月13号

00:54 – 00:56

any questions before we begin

在我们开始之前, 你们还有任何问题吗?

MULTI-VERSION CONCURRENCY CONTROL

The DBMS maintains multiple **physical** versions of a single **logical** object in the database:

- When a txn writes to an object, the DBMS creates a new version of that object.
- When a txn reads an object, it reads the newest version that existed when the txn started.

01:03 – 01:03

all right

1.03–1.08

so um today we are going to talk about **Multi-Version Concurrency Control** ,

So, 今天我们要讨论的是多版本并发控制

~~1.08–~~

~~just make sure I just this here~~

01:11 – 01:16

so the first thing I want to point out about multi-version concurrency controls

So, 关于多版本并发控制, 我首先要指出的一点是

1.16–1.20

that its name is a misnomer and this may cause some confusion

它的名字有点用词不当, 并且可能会引发一些误解

01:21 – 01:31

because it's not actually a concurrency control protocol like the ones that you've been learning about in the past two lectures, which are timestamp ordering, OCC and two-phase locking

因为实际上它并不是你们上两节课所学的那种并发控制协议, 它并不是Timestamp Ordering、OCC或者两阶段锁之类的东西

01:33 – 01:42

rather it's a way to architect the system when you have concurrent transactions running by maintaining multiple versions

相反, 它是构建系统的一种方式, 即通过维护多版本数据来做到并发执行事务

01:42 – 01:46

so recall from last week your discussion of optimistic currency control

So, 回想下上节课我们关于乐观并发控制的讨论

1.46–1.49

where transactions maintain a private workspace

我们的事务维护了一个私有空间

01:50 – 01:52

and anytime they read or wrote to an object

每当它们对一个对象进行读或者写操作的时候

1.52–1.56

it will copy that object into that private workspace

它会将该对象复制到它的私有空间中

01:57 – 02:01

well in the multi-version concurrency control is similar to that idea

Well, 多版本并发控制的思路其实和这个思路很类似

02:01 – 02:08

except here instead of having a private workspace for each transactions where we maintain these different versions

不同的地方在于, 我们维护的是这些对象的不同物理版本, 而不是为每个事务创建一个私有工作空间

02:09 – 02:12 ! ! ! !

we're now going to have the version to be part of a global database

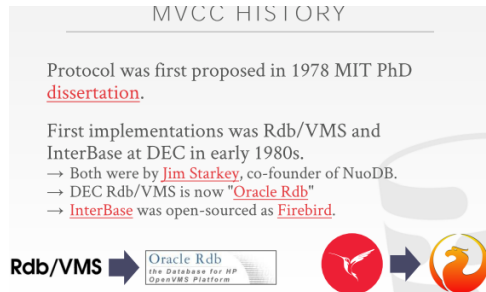
对于这个全局数据库部分数据，我们会有若干版本

02:13 – 02:19

and we're going to determine whether some version of it is visible to a particular transaction

我们会明确该逻辑对象的某个物理版本是否对某个特定事务可见

我们会明确是否将某个版本数据对一个特定事务可见



02:23 – 02:33

so MVCC is used by almost every new database system that's been built in the past ten years or some variant top of it

So，最近十年所出现的新数据库系统基本都用到了MVCC或者是它的变种

02:34 – 02:35

But it's not a new idea

但它并不是什么新的想法

02:35 – 02:39

so it's actually decades old,

So，实际上它已经出现几十年了

2.39–2.49

and the first reference to the idea was in a dissertation by PhD student MIT in 1978
它最先是由MIT的某个Phd学生在1978年的时候所提出的

02:48 – 02:57

so it wasn't until the early 80s at the first ,that the first implementations of it actually came out

So，直到1980年代的时候，它的第一个实现才出现

02:58 – 03:00

and those came out of a company called DEC

这是由一个叫DEC的公司所实现的

3.00–3.03

and they were called Rdb/VMS

它们以前叫做Rdb/VMS

3.03–3.10

which stood for relational database for VMS or **VAX**, which was an old operating system

它的全称是Relation Database for VMS/VAX，VAX是一种很古老的操作系统

03:10 – 03:13

and the other product was called **InterBase**

另一个产品叫做InterBase

03:13 – 03:16

so Dec used to be a major computer company

So，DEC过去曾是一个主流计算机公司

3.16–3.23

it was bought out by a **Compaq** in the late 90s, and then a few years later bought out by HP

它在90年代末期先是被Compaq所收购，接着，过了几年，Compaq又被HP所收购

03:23 – 03:24

so it's no longer around

So, 它已经不复存在了

3.24–3.28

but it did some major pioneering work in database systems

但它在数据库系统方面做了些先驱性工作

03:31 – 03:36

So both RDB/VMS and InterBase were built by a guy named [Jim Starkey](#).

So, RDB/VMS和InterBase都是由一个叫做Jim Starkey的人所构建的

3.36–3.41

who was also credited as being the inventor of Blobs and Triggers

他同样是Blob和Trigger的发明者

03:41 – 03:42

so he's a big deal

So, 他很了不起

03:43 – 03:45

He later went on to co-founder NuoDB

他之后成为了NuoDB的联合创始人

3.45–3.47

which is a new database startup

它是一个新的数据库初创公司

3.47–3.50

and it also happens to use in MVCC

NuoDB也用到了MVCC

03:52 – 03:56

so DEC RDB/VMS was bought up by Oracle

So, DEC的RDB/VMS被Oracle所收购

3.56–3.59

and is now known as Oracle Rdb

它现在被人所熟知的名字是Oracle RDB

04:01 – 04:06

~~and it was an~~ InterBase

再来说说InterBase

4.06–4.10

,was eventually sold by DEC

它最终被DEC卖掉了

4.10–4.12

it went through a few different holding companies

它被多家不同的公司所接手

4.12–4.15

and finally was open sourced

最终, 它被开源了

04:16 – 04:20

and so now it's known under a different name now it's called **Firebird**

So, 现在它换了个名字, 它叫做Firebird

04:20 – 04:23

so it may not be as well-known as MySQL or Postgres,

So, 它可能并不如MySQL或者PostgreSQL那么有名

4.23–4.26

but there's one of the earliest open-source databases out there
但它是最早被开源出来的其中一个数据库

04:28 – 04:31

and Andy had this little fun fact in there from last year

Andy去年在这上面闹了个小笑话

4.31–4.33

or so I'll go ahead and say it

我稍后会讲

04:33 – 04:40

so if you've ever wonder why Firefox web-browser is named Firefox

So, 如果你们想知道火狐浏览器的名字为什么叫火狐

04:41 – 04:44

it's because they were originally called Phoenix

这是因为它一开始被叫做Phoenix (凤凰)

4.44–4.45

,but then they had to change that name,

但之后, 他们必须修改浏览器的名字

4.45–4.49

because it conflicted you know with another system or another product

因为这个名字和其他系统或者其他产品冲突了

04:50 – 04:51

so they changed it to Firebird

So, 他们将浏览器的名字改为Firebird

4.51–4.52

but then they had to change it again

但接着, 他们又得改名

4.52–4.54

because it conflicted with this database system

因为这个名字又和这个数据库系统的名字产生了冲突

04:55 – 04:57

so finally it was called Firefox

So, 最终它的名字被定为Firefox (火狐)

MULTI-VERSION CONCURRENTLY CONTROL

**Writers don't block readers.
Readers don't block writers.**

Read-only txns can read a consistent **snapshot**
without acquiring locks.
→ Use timestamps to determine visibility.

Easily support **time-travel** queries.

05:05 – 05:09

so the main benefit again like what you have to understand about MVCC is

So, MVCC中你需要了解的主要好处在于

5.09–5.14 !!!

that writers don't block the readers ,and the readers don't block the writers

writer不会阻塞reader, reader不会阻塞writer

05:15 – 05:19

so it's only when you have two transactions trying to write to the object at the same time

So, 只有当你有两个事务同时要写对同一个对象试着进行写入操作的时候

05:20 – 05:26

they have to fall back and rely on one of the concurrency control protocols like two-phase locking

它们就得退一步，去使用某种并发控制协议，比如：两阶段锁

05:27 – 05:31

so again you only need to do this when you have a write write conflict

So，当你遇上Write-Write Conflict的时候，你才需要用到它

05:32 – 05:33

so with a high level

So，从高级层面来看

5.33-5.35

,the way this works is

它的工作方式是

5.35-5.40

we're going to assign timestamps to transactions when they arrive in the system

当事务进入系统的时候，我们要给它们分配时间戳

05:41 – 05:49

and then we're going to provide it with a consistent snapshot of the database as it existed at the time that that transaction arrived

接着，当事务到达的时候，我们会为事务提供该数据库的一份一致的snapshot（快照）

05:52 – 5.53

so this means that

So，这意味着

5.53-6.00

they won't see changes from transactions that have not yet been committed in their snapshot

它们不会看到那些还未在它们快照中进行提交的事务所做的那些修改

06:01 – 06:02

and just to clarify

我要澄清一下

6.02-6.06

this is a virtual snapshot

这是一份虚拟快照

06:06 – 06:07

so it shouldn't be confused with

So，它不应该被搞混

6.07-6.13

you know if a physical snapshot or copying the the full database to another location

如果我们将整个数据库复制到另一个位置

6.13-6.15

and then running that transaction on it

接着，我们在这个副本上执行事务

06:16 – 06:18

so again this is um this is just virtual

So，再说一遍，这个副本是虚拟的

06:19 – 06:24

so MVCC is really useful for read-only transactions

So，对于只读事务来说，MVCC真的很有用

6.24-6.30

because the SQL dialect allows you to declare when a transaction is read-only

因为SQL dialect允许你声明该事务是否是只读事务

06:31 – 06:32

and if you do this

如果你这样做的话

6.32–6.38

then the database system does not require you to get any locks or maintain the read write sets

那么，数据库系统就不要求你去获取任何lock或者维护read set或write set

06:39 – 06:40

and this works again

这种方式奏效的原因是

6.40–6.42

because it has a consistent snapshot

因为它有一个一致的快照

6.42–6.47

,and will only see the changes that existed at the moment it started

它只会看到该事务开始时已存在的那些修改

06:47 – 06:52 ...! ! !

and this makes these read-only transactions really efficient and also really fast to do

这使得那些只读事务变得非常高效，且执行速度也很快

06:54 – 06:54

yes

请讲

06:58 – 07:01

I'm even just like I mentioned a minute ago

就像我一分钟前提到的那样

7.01–7.08

like it's essentially just maintaining like a version table or version made it data information

本质上来讲，它就是在维护一个版本信息表之类的东西

07:09 – 07:12

and it's very similar the OCC where you understand the read and write sets

这就像是你所理解的OCC中那些read set和write set

07:12 – 07:16

and we're going to clearly go over this in a lot of detail in the following slides

我们会在接下来的幻灯片中对他们进行深入了解

07:16 – 07:19

this is gonna be the the topic of this lecture ,

这也是这节课的主题所在

7.19–7.20

yeah

请讲

07:34 – 07:36

so it's the we're talking about two reads here

So，假设这里面有两个读操作

7.36–7.40

they will read the the same snapshot the same version

它们会读取同一份快照，或者说同一版本的数据

07:41 – 07:48

So snapshot you know it's more commonly I think referred to as the version, you know the tuple or database object

So, 你知道的, 我更想将snapshot称之为版本, 即tuple或数据库对象的版本

07:55 – 08:02

~~Well can so if you~~ so bear with me for just like you know three minutes probably less

Well, 你先听我讲三分钟再说

8.02–8.04

,and we'll actually that is the first thing we're gonna cover

你问的东西实际上是我们首先要讲的东西

08:06 – 08:09

and I'll answer any other questions you have afterward

稍后我再回答你提出的任何问题

08:20 – 08:21

all right

8.21–8.24

so just to finish up on the slide

So, 为了结束这张幻灯片上的内容, 我要讲的是

8.24–8.26

another advantage of MVCC is that

MVCC的另一个优点在于

8.26–8.32

you're able to support something called time-travel queries

你能够支持一种叫做Time-Travel Query的东西

08:32 – 08:43

so these are queries to actually let you ask the database system for example what was the state of the database you know three days ago, three years ago

So, 实际上, 通过这种查询, 你可以问数据库它三天前的状态是什么, 或者三年前的状态是什么

08:43 – 08:50

and using these and using this versioning, they can actually answer these sort of queries
使用这种多版本, 它们实际可以回答这些查询

08:55–8.56

all right,

8.56–9.03

so the idea of time travel queries was first was an idea of Postgres

So, 这种Time-Travel Query的思想首先是在PostgreSQL中出现的

9.03–9.06

and it originated from Postgres in the 1980s

PostgreSQL在1980年代提出了这个东西

09:06 – 09:16

but PostgreSQL actually removed these time travel queries from their current product .

但实际上, PostgreSQL将这种Tim-Travel Query从他们当下的产品中移除了

9.16–9.22

like, as soon as you know people outside of academia, I started using Postgres more heavily

除了学术界以外的人, 我现在使用PostgreSQL的次数越来越多了

9.22–9.23

can anybody guess why

你们能猜下这是为什么吗？

09:26 – 09:26

well

Well

9.26–9.28

so the reason why is

So, 理由是

9.28–9.36

because essentially what you have to do to actually support time travel queries is you never throw away old versions

因为本质上来讲，如果你要支持Time–Travel Query，那你就永远不能将老版本的数据给丢掉

09:36 – 09:39

so you never garbage collect, right

即永远不做垃圾回收

09:39 – 09:39

so over time

So, 随着时间的流逝

9.39–9.44

you're you know the more and more transactions that commit

事务提交的数量会越来越多

09:44 – 09:47

your your disk base will be filling up very quickly,

你的磁盘空间很快就会满了

9.47–9.50

and eventually it will be full

最终，你的磁盘就会满了

9.50–9.54

and probably very quickly depending on the speed of your transactions

你的磁盘很可能马上就会满，这取决于你执行事务的速度

09:56 – 9.58

and the the other thing is that

另一件事情是

9.58–10.02

time travel queries are not really needed by a lot of applications

并不是很多应用程序都需要使用Time–Travel Query

10.02–10.09

~~like you can't~~, you never really look at you never go to a website, and say like ok I want to know what this webpage looked like three days ago

你从来不会这么干，比如：你跑到某个网站上说，你想知道该网页三天前的样子

10.09–10.11

well I'm not the most use cases

Well, 这并不是一个常见案例

10:12 – 10:20

but Andy mentions that like one common use case for these time travelling queries is in the financial industry

但Andy提过一个关于Time–Travel Query很常见的使用案例，那就是金融领域方面

10:22 – 10:23

so the reason is

So, 理由是

10.23-10.30

~~because you know do to, I know pursuit you know~~ whatever rules and regulations they have to follow

不管他们要遵守什么规则和条例

10:30 – 10:33

they have to actually maintain in the past seven years of transaction history

实际上，他们必须维护过去七年间的交易历史

10:34 – 10:45

so these time-travel queries actually allow them to very easily query the database

So，这些Time-Travel Query就能让他们很容易地查询数据库中的内容

10.45-10.51

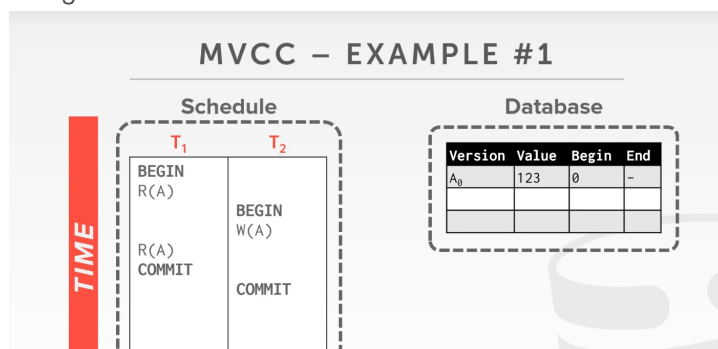
and figure out why ,you know some of money what their what their total revenue was or

whatever they want to look up you know over the past seven years

并弄清楚过去七年间他们的总利润是多少，或者任何他们想查找的

10:55 – 10.56

all right



10.56-11.00

so um and the next few slides we're going to go over two examples

So，在接下来的几张幻灯片中，我们要去查看两个例子

11:01 – 11:04

And what I really want to emphasize here before we start is that

在我们开始之前，我这里真正想强调的东西是

11.04-11.10

MVCC is independent from concurrency control protocols

MVCC不依赖于并发控制协议

11:10 – 11:12

so the purpose of these examples is

So，向你们展示这些案例的目的在于

11.12-11.19

just to basically show you ,how we you know update versions and timestamps in the table

简单来讲，就是向你们展示如何更新表中的这些版本号和时间戳

11:20 – 11:36

~~and also basically like how we figure out which version~~ how we figure out which version is to is visible to the particular transaction ,

还有就是，我们该如何弄清楚哪个版本的数据对特定的事务可见

11:36 – 11:39

right which version of the tuple is visible

即该tuple的哪个版本是可见的

11:40 – 11:43

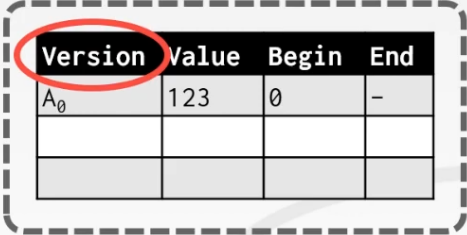
so this first example we'll see how this is going to work

So, 我们会在第一个例子中看下它是如何工作的

11:45 – 11:48

so right now like the first thing to point out is that

So, 我们现在要指出的东西是



Version	Value	Begin	End
A ₀	123	0	-

11.48–11.51

,now we have this version field right

这里我们有一个version字段

11:51 – 11:56

so we can see in this version field ,but it's assigned to A0

So, 我们可以看到, 在这个version字段中, 我们分配了一个A0

11:56 – 12:00

So this means object A version zero, right

So, 这意味着对象A的版本号是0

12:00 – 12:02

so we can assume that

So, 我们可以假设

12.02–12.08

some other transaction has written the value 123 to the database

某个事务将值123写入到了数据库中

12.08–12.10

and whatever transaction wrote it

不管是哪个事务写入了这个值

12:10 – 12:13

timestamp was assigned a timestamp of zero

这里我们所分配的时间戳就是0

12.13–12.14

and we'll go over why in one second

我们稍后会讲这是为什么

12:15 – 12:19

so we also have a begin and end fields

So, 我们还有begin和end字段

12:19 – 12:21

and so these are just timestamps

So, 这里面放着的都是时间戳

12.21–12.25

it doesn't matter if they're logical physical hybrid

你不需要在意它们是逻辑的还是物理的, 或者两者混合

12:25 – 12:28

as long as they you know they're always increasing

你知道的，它们的值始终是增加的

12:28-12:34

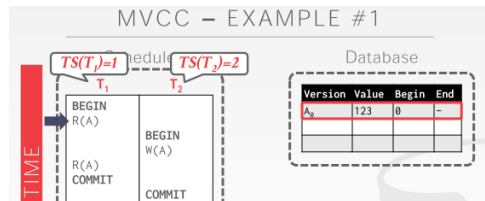
and follow the other you know and I guess our our valid timestamps

并遵循我们的其他时间戳

12:34 - 12:35

right like you learned in the past few lectures

就比如你前几节课中学到的那样



12:41-12:42

alright

12:42-12:45

so let's begin

So, 我们开始吧

12:45-12:48

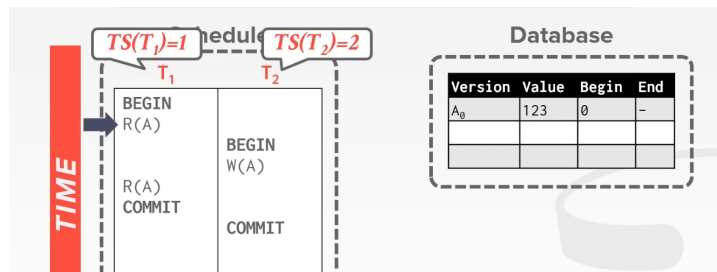
so when a new transaction arrives

So, 当一个新事务到达的时候

12:48-12:52

we're going to be looking at transactions t1 and t2

我们要去查看T1和T2



12:52 - 12:57

so here T t1 arrives and it's assigned a timestamp of 1

So, 当T1到达的时候，我们会将时间戳1分配给它

12:58 - 13:01

right so now we're going to begin, okay

So, 我们开始吧

13:01 - 13:05

so for the first thing we want to do is what I do a R(A)

So, 首先我们想做的事情是，我想执行R(A)

13:05 - 13:07

so what we're going to do is

So, 我们接下来要做的事情是

13:07-13:13

~~we're going to consider~~ you know time transaction 1's timestamp which is 1

你知道的T1的时间戳是1

13:13 - 13:14

and we're going to take a look at our table

我们要去查看下我们的表

13:14–13:27

and figure out which tuple is visible to it by finding you know where its current timestamp is between beginning and End

通过弄清楚当前时间戳处于开始时间和结束时间中的哪个位置，我们以此来决定哪个tuple对它是可见的

13:27 – 13:28

so in this example

So, 在这个例子中

13:28–13:31

the beginning is 0

开始时间是0

13:31–13:36

and the timestamp of 1 is between 0 and the end which is infinity, right

时间戳1是在0和无穷大之间

13:36 – 13:39

so it's going to go ahead and breed version a 0

So, 它就会将A的版本号设置为A0

13:40 – 13:40

all right

13:42 – 13:42

all right

13:42–13:45

so now we have transaction t2

So, 现在我们有一个事务T2

13:45–13:48

and we're going to assign the timestamp 2

我们给它分配的时间戳是2

13:48 – 13:50

so the first thing we want to do here is

So, 这里我们首先想做的事情是

13:50–13:52

we want to write a

我们想对A执行写操作

13:52 – 13:54

so at this point

So, 此时

13:54–13:54

what we're going to do is

我们要做的事情是

13:54–14:00

we're going to create a completely new version of a,

我们要创建A的一个全新版本

14:00–14:01

which will be a 1

即A1

14:02 – 14:04

right because we're just incrementing the version counter
因为这里我们做的只是去增加版本号计数器

14:06 – 14:10

and right and so what we're going to do here is

So, 我们这里要做的事情是

Database			
Version	Value	Begin	End
A_0	123	0	-
A_1	456	2	-

14.10–14.15

the beginning timestamp is going to be set to the timestamp of t2

我们要将A1的begin timestamp设置为T2的时间戳

14:15 – 14:18

the end timestamp again be set to infinity

它的end timestamp被设置为无穷大

14:18 – 14:20

and then the last thing we're going to do is

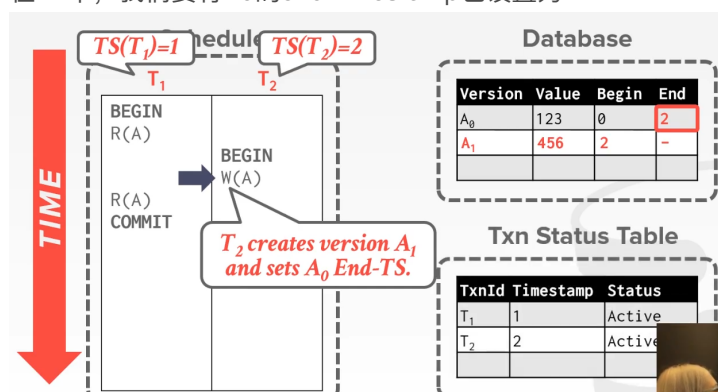
接着, 我们要做的最后一件事情就是

14.20–14.30

we're going to update the end timestamp of version a 0 to also be a timestamp of 2

right for transaction 2

在T2中, 我们要将A0的end timestamp也设置为2



14:33 – 14:33

all right

14.33–14.36

so one thing you might have noticed that

So, 你们可能已经注意到一件事情了

14.36–14.46

we're missing so far is like with with just either the information that we had so far,

before this transaction status table popped up

在填充事务状态表前, 我们目前已经有了这么多信息

14:46 – 14:47

the one thing that we're missing is that

这里我们忘说的一个东西就是

14.47–14.51

we don't really know the current state of the transactions in the database

我们并不清楚数据库中这些事务的当前状态

14:51 – 14:52

so for example

So, 例如

14.52–14.56

you know the transactions here are currently active

你知道的, 这些事务当前处于活跃状态

14:56 – 14:58

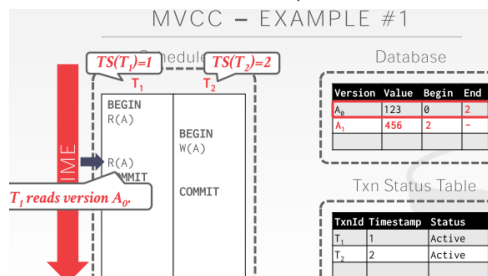
but what if they abort

但如果它们被中止的话

14.58–15.05

you know then you would have to go back and reverse the timestamps accordingly if it was aborted

如果这些事务被中止的话, 那么你就需要回过头去, 将这些时间戳恢复原样



15:12 – 15:14

right so as you can see

So, 正如你们所看到的

15.14–15.19

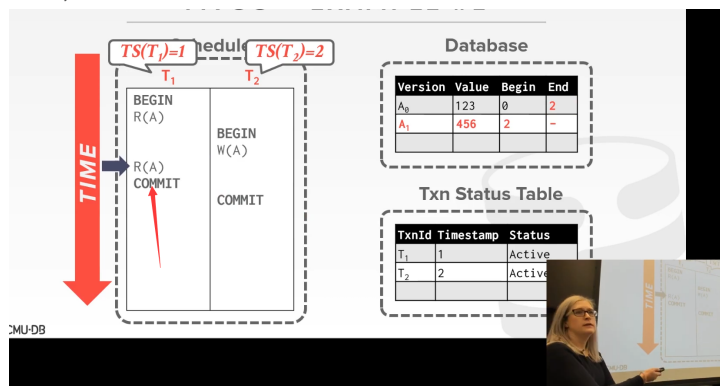
here we're just going to start filling out the transaction status table

此时我们开始填充事务状态表中的信息

15.19–15.23

at this point both transactions are active

此时, 这两个事务的状态都是active (活跃)



15:23 – 15:25

then finally we're going to do this $R(A)$

接着, 我们要去执行这个 $R(A)$

15:25 – 15:28

so what version is it going to read
So, T1所要读取到的A版本是什么呢?

15:29 – 15:32

anyone right

有人知道答案吗?

15:33 – 15:34

A sub-zero

A0, 对吧

15:34–15:38

because again it's timestamp still lies between the beginning and end here
因为它的时间戳依然是在begin timestamp和end timestamp之间

15:39 – 15:41

So it's gonna go ahead and read version a0,

So, 它会读取到的版本是A0

15:41–15:46

oh and finally its gonna commit

最后, 它会被提交

15:46 – 15:47

so at the very end

So, 在最后的时候

15:47–15:48

after this commits

当T1提交了之后

15:48–15:51

then the then transaction t2 will commit

接着, T2会进行提交

15:51–15:52

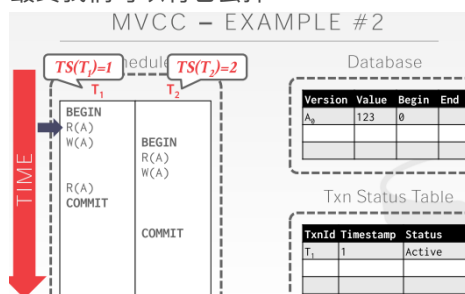
will update the status table

它会去更新状态表

15:52–15:55

and we can blow it away eventually

最终我们可以将它丢掉



15:57 – 16:00

so for the second example

So, 在第二个例子中

16:00–16:03

we're gonna start with sort of the same setup right

我们这里的设置和第一个例子相同

16:04 – 16:08

so we have transaction t1 with a timestamp of 1

So, T1的时间戳是1

16:08 – 16:11

and transaction t2 we're assigning a timestamp of 2

我们分配给T2的时间戳是2

16:11 – 16:16

and it's the same state in the in the database table

数据库中的状态和前一个例子中是相同的

16.16–16.20

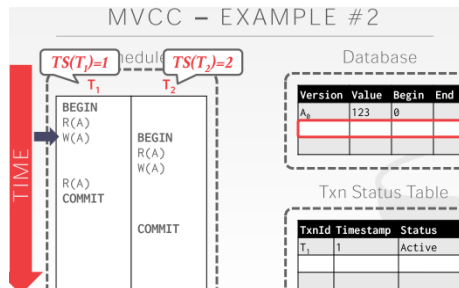
and so far we're just starting transaction t1

目前为止，我们只开始执行T1

16:20 – 16:23

we're saying it's timestamp to 1 and it's status is active

我们这里表示T1的时间戳是1，它的状态是active



16:25 – 16:27

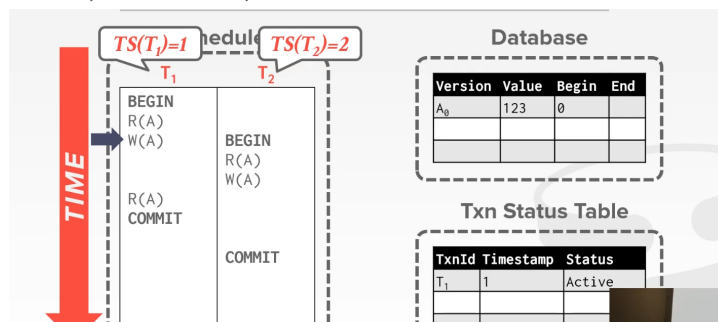
so first we're going to do a R(A)

So，首先，我们要执行R(A)

16.27–16.34

I think at this point it's pretty clear that we're going to read version a0

我觉得，此时很明显，我们要读取的版本是A0



16:35 – 16:37

and next we're gonna do a W(A)

接着，我们要执行W(A)

16:37 – 16:38

so again just like in the last slide

就和上一张幻灯片一样

16.38–16.41

we're going to create a completely new version

我们要创建一个全新版本的A

16:41 – 16:43

I'm inserting in our database table

我将它插入我们的数据库表中

16.43–16.47

where it's gonna be version a 1 with value 4 5 6

它的版本是A1，它的值是456

16.47–16.52

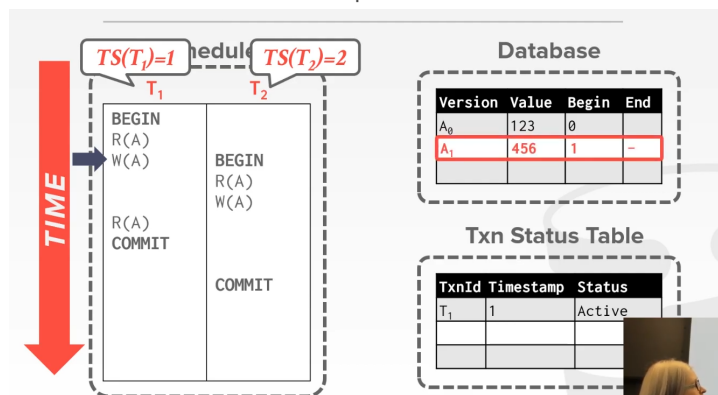
,and the beginning timestamp will be 1 ,right it will be whatever this timestamp is

它的begin timestamp是1或者是其他值

16:52– 16:54

and the end we will assign to infinity again

这里我们将它的end timestamp设置为无穷大



16:55 – 16:58

and the last thing to not forget is that

我们不要忘记做最后一件事

16:58–17:00

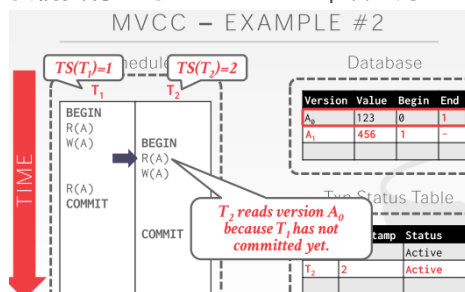
we need to go up to a₀ ,

我们需要跑到A₀这里

17:00–17:05

and assign the end timestamp to be the current timestamp of transaction t1 which is 1

我们要将它的end timestamp设置为T1的当前时间戳，即1



17:07 – 17:10

alright so now we're going to begin transaction 2

So, 现在我们开始执行T2

17:10 – 17:13

so the first thing we're going to do is a R(A)

So, 我们首先要做的事情是执行R(A)

17:13 – 17:15

so in this case

So, 在这个例子中

17:15–17:25

which transaction is it going to read or sorry which version is it gonna read excuse me

T2要读取的是A的哪个版本呢?

17:25–17:25

a 1

A1

17:25–17:26

and why is that

为什么是这样呢?

17:32 – 17:33

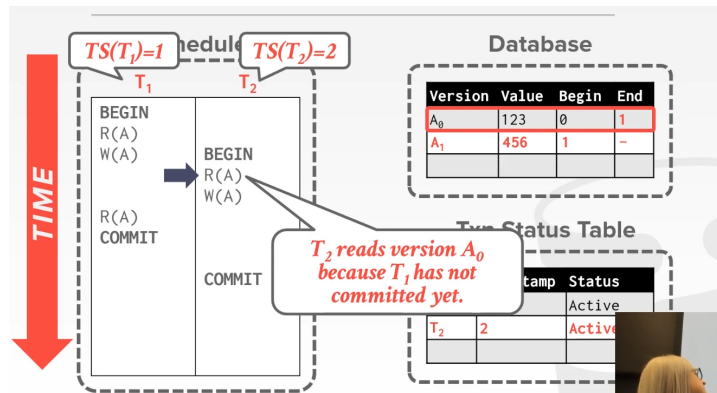
right yeah somebody I guess yeah

有人猜是这样的

17.33–17.34

so in this case

So, 在这个例子中



17.34–17.39

um it's gonna be oh sorry for a yes

抱歉，我说错了

17:40 – 17:42

so one thing that we have to pay attention to

So, 我们需要注意的一件事情是

17.42–17.44

and this is a little tricky right now

现在，它有点棘手

17:44 – 17:46

one thing I forgot to mention that the start is

我一开始忘记提的一个东西是

17.46–17.49

I understand you guys didn't have time to go over isolation levels

我理解你们没时间去看隔离级别方面的东西

17:51 – 17:55

so Andy wanted you guys to just review the slides and also the lecture from last year

So, Andy想要你们去回顾下去年对应的幻灯片和课程

17:56 – 18:03

so I'm just gonna provide some high-level hints for isolation levels for when you go over those slides and the homeworks, right

So, 当你们去看这些幻灯片和做Homework的时候，我会为你们提供一些隔离级别方面的提示

18:03 – 18:06

but it might not make full sense at this point

但此时讲的话，可能并没有太多意义

18:06 – 18:08

but basically like at a very high level

但从一个高级层面来讲

18.08–18.10

depending on the isolation level you have

取决于你所使用的隔离级别

18:12 – 18:15

it may choose either version a 0 a 1

它选择的可能是A0或者A1

18:15 – 18:23

but let's assume it's sort of it's the strict serializable or ,excuse me serializable isolation

但假设它的隔离级别是Serializable

18.23–18.26

which is sort of what you guys have been using up until this point

你们应该都已经使用过这个了

18:26 – 18:27

and this point

此时

18.27–18.30

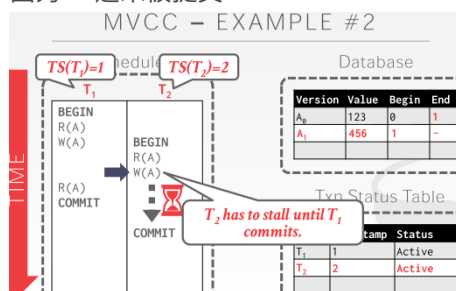
it will it has to read a 0 ,

它需要读取A0

18.30–18.31

because a 1 has not yet committed

因为A1还未被提交



18:33 – 18:34

alright

18.34–18.36

so now we're gonna do a W(A)

So, 现在我们要执行W(A)

18.36–18.38

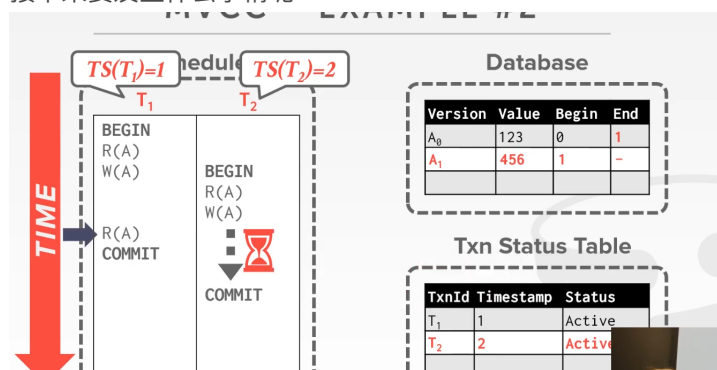
,and so in this case

So, 在这个例子中

18.38–18.39

what's gonna happen next

接下来要发生什么事情呢?



18:40 – 18:43

well again here we have a write write conflict right

Well, 这里我们遇上了Write-Write Conflict

18:43 – 18:48

so assuming we're using 2PL

So, 假设我们使用的是2PL (两阶段锁)

18.48–18.51

t2 is gonna have to stall until t1 commits

T2必须等到T1提交后，才会继续执行

18:51 – 18:51

all right

18.51– 18.55

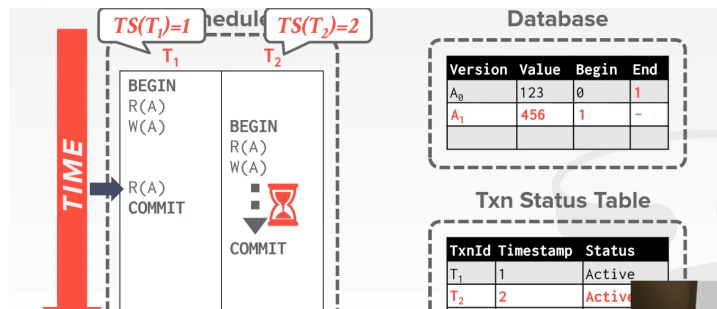
so let's keep this going

So, 我们继续

18.55–18.56

so now we're back to t1

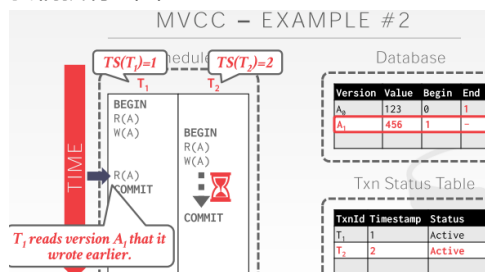
So, 我们切换回T1



18.56–18.57

we're going to do a R(A)

我们执行R(A)



18:58 – 18:59

and in this case

在这个例子中

18.59–19.03

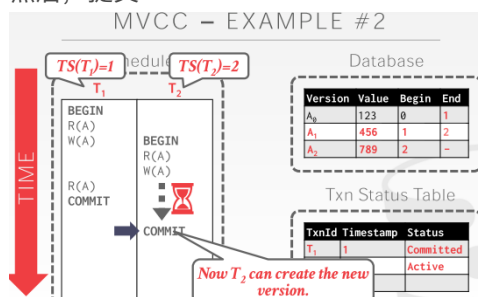
it's gonna just read the same version that it wrote couple minutes ago right

T1会去读取它前几分钟刚修改过的版本，即A1

19:04 – 19:06

and it's gonna go ahead and commit

然后，提交T1



19:07 – 19:08

alright

19.08–19.09

so now we can go back here

So, 现在我们可以回到T2

19.09–19.16

and we can go ahead and now we're going to create the new version a 2 with value 789

现在, 我们要去创建A的新版本, 即A2, 它的值是789

19:16 – 19:20

we're going to assign it the timestamp of 2 with an end time step of infinity

我们将A2的begin timestamp设置为2, end timestamp设置为无穷大

19:20 – 19:25

and we're going to update the end timestamp of A1 to 2 as well right

接着, 我们要将A1的end timestamp设置为2

19:25 – 19:26

so at this point

So, 此时

19.26–19.37

you know whether t2 actually commits or not ,is really dependent on the concurrency

control protocol as well as the isolation level

实际上, T2是否被提交, 这取决于并发控制协议和隔离级别

19:37 – 19:38

so that's something to keep in mind

So, 这是你们要记在脑子里的东西

19:39 – 19:48

but really this this example ,the purpose of this example is just to show you how we

update the object versions, maintain the transaction status table

但向你们展示这个例子的目的在于, 我们该如何更新这些对象的版本号, 以及维护事务的状态表

19:48 – 19:51

and also figure out which tuples are visible ,all right

并弄清楚哪些tuple是可见的

19:52 – 19:53

Any questions on this

对此有任何问题吗?