17–04

17–04

01:00:14 – 01:00:17

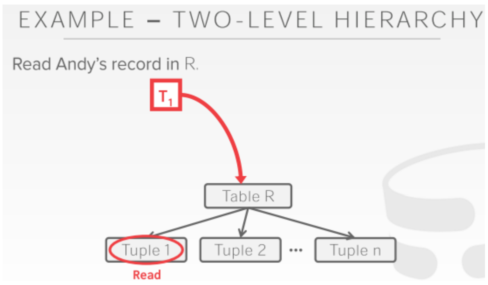
again let’s let’s do an example

我们来看个例子

1.00.17–1.00.19

 because I think that’s going to make things a little bit clearer we have five minutes

因为我觉得这样能让你们更明白这些东西，我们还剩5分钟左右的时间



01:00:21 – 01:00:23

very simple table

这里有一张非常简单的表

1.00.23–1.00.27

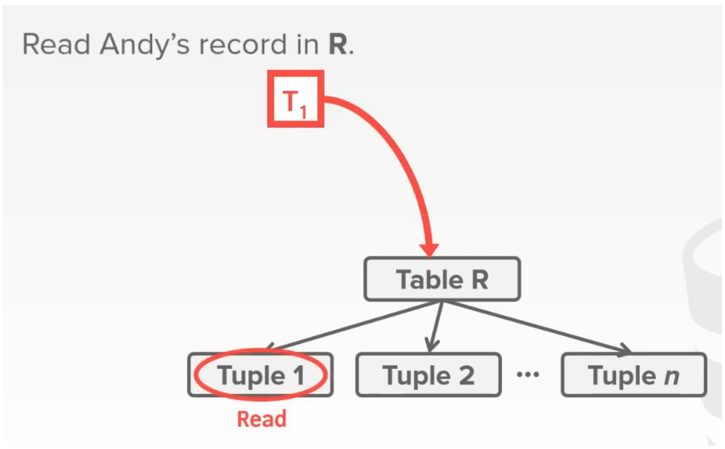
or a very simple example ,two levels, there’s a table, there’s a bunch of tuples

这是一个很简单的例子，它里面有一张表和一些tuple

01:00:30 – 01:00:33

we want to check Andy bank account

我们想去检查Andy的银行账户



01:00:33 – 01:00:35

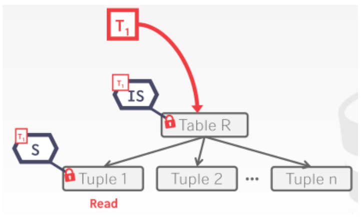
he wants to do a read on tuple one

他想读取Tuple 1

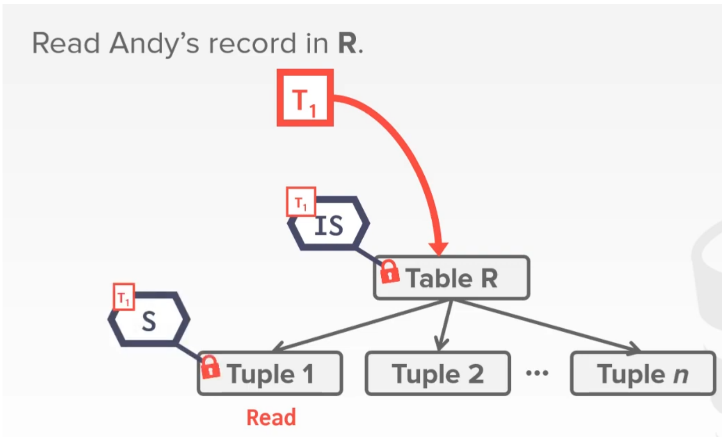
01:00:35 – 01:00:40

~~so he he wants,~~ he’s gonna want just a shared lock on this one to do a read

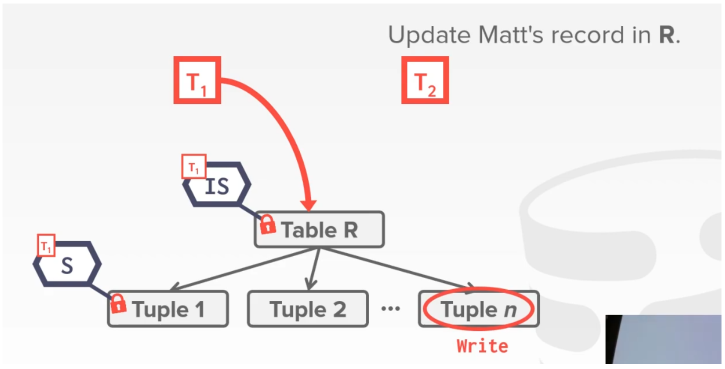
这里他会在Tuple 1上加一个shared lock，以此来读取Tuple 1的内容



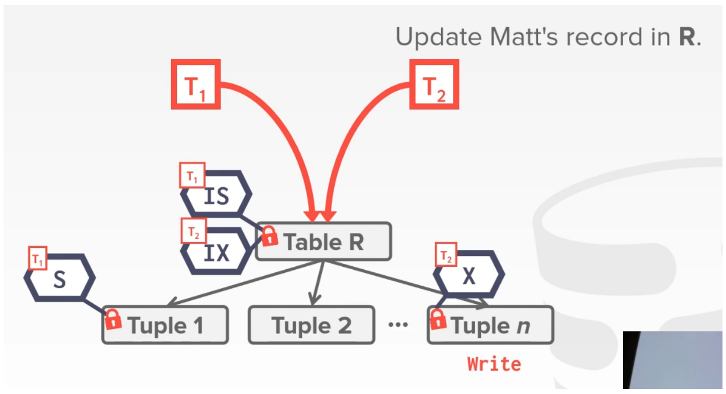
01:00:41 – 01:00:45
but we're gonna have to take an intention shared at the parent node first
但我们会先在父节点上加一个intention-shared lock
1.00.45-1.00.46
basically as a hint to say
这里会作为一个提示，即，
1.00.46-1.00.47
,hey below this node
在该节点下面


Read Andy's record in **R**.

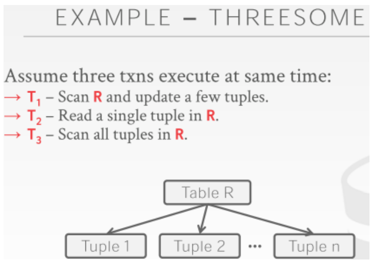
1.00.47-1.00.51
 I'm gonna take an explicit shared lock
我会使用一个显式的shared lock


Update Matt's record in **R**.

01:00:53 – 01:00:55
t2 comes along
接着，T2开始执行
1.00.55-1.00.59
 we want to update my bank balance by one percent
我们想去更新我的银行存款（即存款值乘以1.01）
01:01:00 – 01:01:05
so we want an explicit exclusive lock on this tuple
So，我们想在这个tuple上加上一把显式的exclusive lock
~~01:01:08 – 01:01:11~~
~~so we're gonna try to get~~



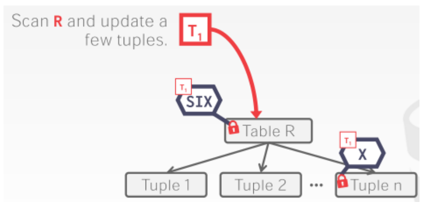
01:01:14 – 01:01:18
yeah we get our intention exclusive lock on the parent node
我们打算在我们的父节点上加一把intention exclusive lock
1.01.18–1.01.20
and we get our exclusive lock on the individual tuple
我们在单个tuple上加上我们的exclusive lock



EXAMPLE – THREESOME

Assume three txns execute at same time:
→ T_1 – Scan **R** and update a few tuples.
→ T_2 – Read a single tuple in **R**.
→ T_3 – Scan all tuples in **R**.

01:01:23 – 01:01:24
now things will get a little bit more interesting
这里所发生的事情就逐渐令我们感兴趣起来
1.01.24–1.01.28
 I think t1 is basically doing the scenario I described before
我觉得T1这里所做的事情就是我之前所描述的那样
1.01.28–1.01.31
where you’re going to do a bunch of reads, and then you’re going to update one tuple
即进行大量读取，接着再去更新某个tuple
01:01:31 – 01:01:32
T2 is gonna read a single tuple
T2会去读取单个tuple
1.01.32–1.01.34
t3 is going to scan all of them
T3会对表中所有数据进行扫描
01:01:35 – 01:01:37
I apologize for going a little fast
抱歉，我讲的速度可能有点快
01:01:37 – 01:01:39
I realize this is probably a little confusing
我讲的可能会有点让你们感到困惑
1.01.39–1.01.42
but we’re getting a long time
我们已经浪费了不少时间



01:01:43 – 01:01:46

so t1 like I said

So，正如我所说的

1.01.46–1.01.48

it wants to read all the tuples and do an update on one

T1想去读取所有的tuple，并对其中一个tuple进行更新

01:01:49 – 01:01:51

so it’s gonna get a shared intention exclusive

So，它会去获取一个shared+intention–exclusive lock

1.01.51–1.01.52

this means

这意味着

1.01.52–1.01.55

I’m taking a shared lock on the entire table

我给整张表加了一个shared lock

01:01:55 – 01:01:58

so I can read all the attributes they’re in all the tuples in this table

So，我可以读取该表中所有tuple中的所有属性

01:01:59 – 01:02:01

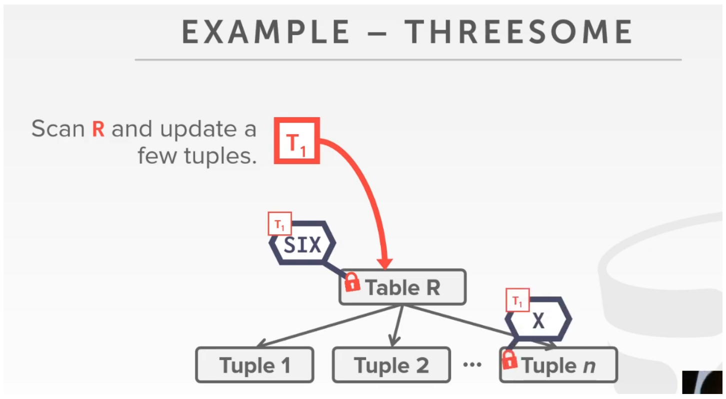
and the intention exclusive part means

intention–exclusive这部分的意思是

1.02.01–1.02.04

I’m gonna update at least one of these tuples down there

我至少会去更新下方其中一个tuple



EXAMPLE – THREESOME

01:02:08 – 01:02:09

in this case it’s tuple n

在这个例子中，我们要更新的是Tuple n

1.02.09–1.12.12

so because this is shared intention exclusive

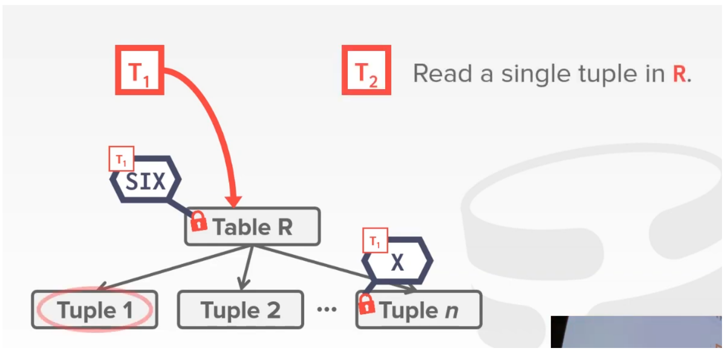
So，因为这里使用的是shared+intention–exclusive lock

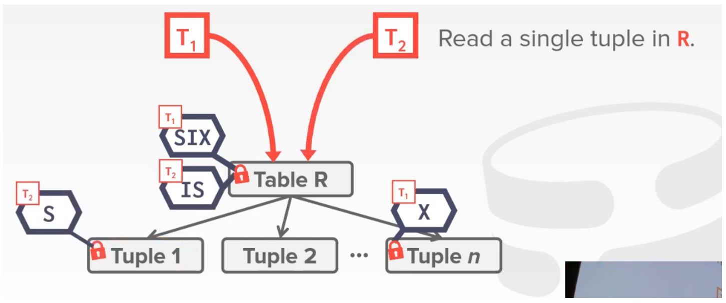
01:02:13 – 01:02:15

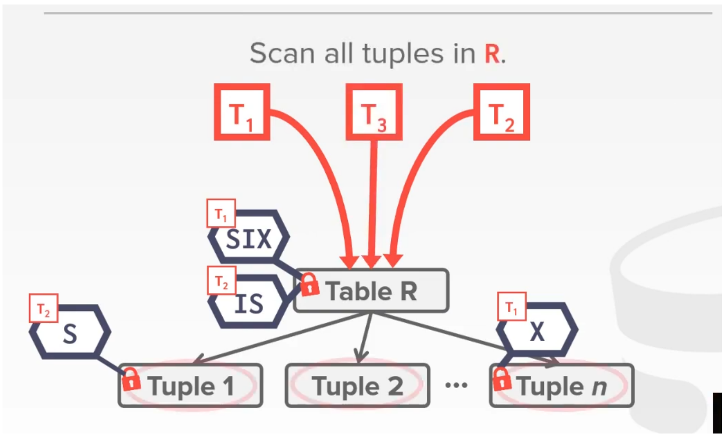
all these tuples are implicitly locked in shared mode

我们会在所有tuple上隐式加上一个shared lock

01:02:16 – 01:02:19

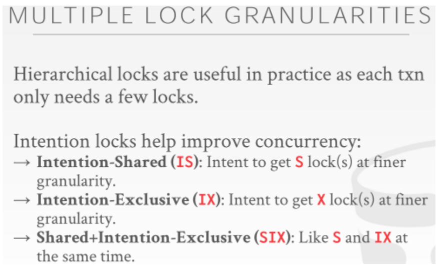
And then this is the only one we actually have to take an explicit exclusive lock on,
此处是我们唯一需要显式加上exclusive lock的地方
1.02.19–1.02.21
 because that’s the only one being updated
因为这是我们唯一要更新的地方


01:02:22 – 01:02:24
t2 wants to read a single tuple
T2想去读取表中某个tuple
1.02.24–1.02.27
 we’re gonna need the shared lock on this guy,
我们需要用到tuple 1对应的那个shared lock
1.02.27–1.02.34
which means we need intention shared at this level
这意味着，我们需要在这一层使用intention-shared lock


01:02:34 – 01:02:35
that’s all good we can do that
没问题，我们可以这么干


01:02:36 – 01:02:39
the last transaction is the one that’s gonna have problems
在执行最后一个事务的时候，我们会遇上问题
01:02:39 – 01:02:41

it wants to do a read on all of them
它想对表中所有数据进行读取
1.02.41-1.02.49
it’s gonna want to ~~shared exclusive shared law ,excuse me~~ explicit shared lock on the table
它想对这张表显示使用shared lock
01:02:49 - 01:02:50
can’t get that
它没法做到
1.02.50-1.02.54
 because it’s not going to be compatible with the shared intention exclusive
因为它无法与shared+intention-exclusive lock相兼容
01:02:54 - 01:02:59
because there’s a write happening on lower in the table ,t3 has to wait
因为这张表中正在执行写操作，T3只能进行等待
01:02:59 - 01:03:05
so basically it wants this explicit shared lock on the table
So，简单来讲，它想去获取整张表的shared lock
1.03.05-1.03.06
can’t have that all can do is wait
除了等待以外，它没有别的办法
01:03:29 - 01:03:30
so I may have gone too quickly
So，我讲得可能太快了
1.03.30-1.03.34
the the example operations changed when I got to this one with three transactions
当我在看这个例子中的三个事务时，里面的操作变了
01:03:35 - 01:03:39
this is no longer doing like the read Andy’s bank balance out bump my account balance by 1%
这不是Andy查他存款余额并给我1%利息那个例子了
01:03:39 - 01:03:43
T1’s reading all of the tuples and then modifying one
T1会读取所有tuple，接着修改其中一个tuple
01:03:44 - 01:03:47
t2 was just doing a read on a single one
T2这里做的是对某个tuple进行读取
1.03.47-1.03.51
,and then t3 is the one that’s trying to read the entire table
接着，T3会试着读取整张表
01:03:51 - 01:03:52
so yeah it’s a different example
So，它是一个不同的案例
1.03.52-1.03.54
 I’m sorry about that if I went kind of quickly
对此我表示抱歉，我讲得太快了


MULTIPLE LOCK GRANULARITIES

Hierarchical locks are useful in practice as each txn only needs a few locks.

Intention locks help improve concurrency:
→ **Intention-Shared (IS)**: Intent to get S lock(s) at finer granularity.
→ **Intention-Exclusive (IX)**: Intent to get X lock(s) at finer granularity.
→ **Shared+Intention-Exclusive (SIX)**: Like S and IX at the same time.

01:04:07 – 01:04:09

so in practice it seemed complicated

So，在实战中，它的情况更为复杂

1.04.09–1.04.10

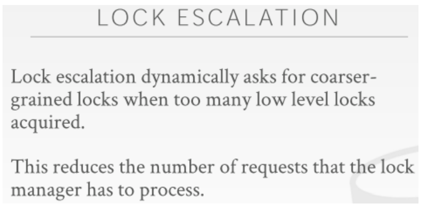
, but it’s actually pretty helpful

但实际上它非常有用

1.04.10–1.04.16

because you can reduce the number of locks that go or the number of trips to the lock manager ,you you reduce the number of lock requests dramatically

因为你可以减少锁的数量，并且显著减少锁请求的数量


LOCK ESCALATION

Lock escalation dynamically asks for coarser-grained locks when too many low level locks acquired.

This reduces the number of requests that the lock manager has to process.

01:04:18 – 01:04:20

and like we mentioned before

就像我们之前提到的

1.04.20–1.04.22

there’s this concept of lock escalation

这里有一个lock escalation（锁升级）的概念

01:04:22 – 01:04:24

so if you already have locks and shared mode

So，如果你的lock是shared模式

1.04.24–1.04.26

you want to bump them to exclusive locks

你想将它们升级为exclusive lock

1.04.26–1.04.28

because you’ve decided you want to do it write on the tuple

因为你决定你想对这个tuple执行写操作

01:04:28 – 01:04:28

You can do that

你可以这么做

1.04.28–1.04.29

once again

再说一遍

1.04.29–1.04.34

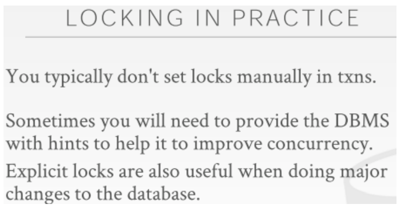
this is designed to reduce the number of trips the lock manager and also doesn’t violate two-phase locking

这是旨在减少lock管理器的工作量以及不违反两阶段锁

01:04:34 – 01:04:37

because you can upgrade your locks you don’t actually release the lock

因为你可以在不释放锁的情况下，对锁进行升级


LOCKING IN PRACTICE

You typically don't set locks manually in txns.

Sometimes you will need to provide the DBMS with hints to help it to improve concurrency.

Explicit locks are also useful when doing major changes to the database.

01:04:41 – 01:04:43

so in practice in real systems

So，在真正的系统中

1.04.03–1.04.46

you're not sitting there telling it which tuples to lock

你不会坐在那里并告诉系统该对哪个tuple上锁

01:04:47 – 01:04:48

you can give hints

你可以给出提示

1.04.48–1.04.48

like I said

正如我说的

1.04.48–1.04.53

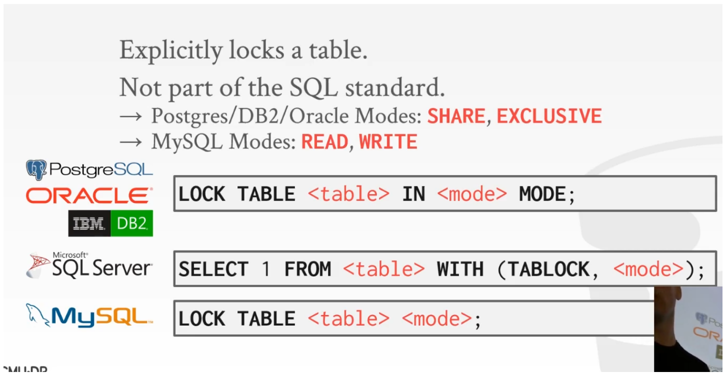
if you know you're gonna do a bunch of operations on a table

如果你知道你要对一张表进行一系列操作

1.04.53–1.04.55

and you want to hold the lock the entire time

在执行的过程中，你想一直拿着锁


Explicitly locks a table.

Not part of the SQL standard.

→ Postgres/DB2/Oracle Modes: **SHARE**, **EXCLUSIVE**

→ MySQL Modes: **READ**, **WRITE**

PostgreSQL
ORACLE
IBM DB2

```
LOCK TABLE <table> IN <mode> MODE;
```

SQL Server

```
SELECT 1 FROM <table> WITH (TABLOCK, <mode>);
```

MySQL

```
LOCK TABLE <table> <mode>;
```

1.04.55–1.04.57

you can't explicitly lock the table

你不能显式锁住这张表

01:04:57 – 01:04.59

so it's not part of the SQL standard

So，这并不是SQL标准中的东西

1.04.59–1.05.05

,but here's examples on how to do it in like Postgres ,Oracle, DB2, SQL Server, MySQL

但这里有个例子，它告诉你该如何在PostgreSQL、Oracle、DB2、SQL Server以及MySQL中做这些事情

01:05:07 – 01:05:10

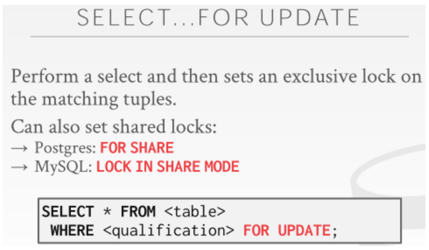
these guys all use the nomenclature that we're learning about now

它们都使用了我们现在所学的术语

1.05.10–1.05.11

which is shared and exclusive

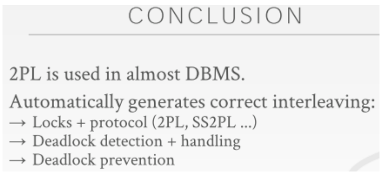
即shared和exclusive

01:05:11 – 01:05:12
And because MySQL loves to be different
因为MySQL就喜欢不走寻常路
1.05.12–1.05.14
 they call them read and write locks
MySQL将它们称为读锁和写锁
1.05.14–1.05.16
because they want to be different
因为他们想要与众不同

SELECT...FOR UPDATE

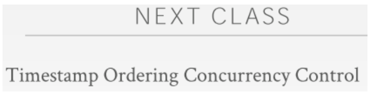
Perform a select and then sets an exclusive lock on the matching tuples.
Can also set shared locks:
→ Postgres: FOR SHARE
→ MySQL: LOCK IN SHARE MODE

```
SELECT * FROM <table>
 WHERE <qualification> FOR UPDATE;
```

01:05:18 – 01:05:20
there’s also this notion of select for updates
这里还有一个Select .... For Update的概念
01:05:20 – 01:05:23
so if you’re doing a read on a tuple that you eventually want to update
So，如果你读取了某个tuple，最终你想对其进行更新
1.05.23–1.05.24
you can give a hint to the database system that says
你可以给数据库系统一个暗示，并表示
1.05.24–1.05.28
 look I know, because you’re doing a read ,you’re gonna request a shared lock
因为我知道你正在执行读操作，你会去请求一个shared lock
01:05:29 – 01:05:30
I’m gonna do a write later on
我之后会执行一次写操作
1.05.30–1.05.33
just take the exclusive lock now and hold it for me
现在，你可以帮我拿一个exclusive lock
01:05:33 – 01:05:39
so you can do select and add this for update
So，你可以在SELECT语句后面加一个FOR UPDATE
1.05.39–1.05.40
that basically tells the system
简单来讲，它会告诉系统
01:05:41 – 01:05:44
~~take the right lock or excuse me~~ take the exclusive lock right now
现在去拿一个exclusive lock
01:05:45 – 01:05:48
and you can also tell it just to take a shared lock
你也可以去告诉它拿一个shared lock
1.05.48–1.05.49
,I don’t know why you would ever do that,
我不清楚为什么有人会想这么做
1.05.49–1.05.52

 because by default it probably should just take a shared lock if you’re doing a read
因为默认情况下，如果你执行的是读操作，它可能拿的就是一个shared lock

CONCLUSION

2PL is used in almost DBMS.
Automatically generates correct interleaving:
→ Locks + protocol (2PL, SS2PL ...)
→ Deadlock detection + handling
→ Deadlock prevention

01:05:57 – 01:05.57
to finish things up
总结一下
1.05.57–1.06.00
like the slide says
正如幻灯片所说的
1.06.00–1.06.02
 it’s used in almost every system out there
几乎所有的DBMS都用到了它
1.06.02–1.06.08
 at least most most widely deployed to commercial systems SQL server ,MySQL, Postgres
至少那些部署最多的商用系统都用到了它，比如SQL server、MySQL和PostgreSQL
01:06:08 – 01:06:11
two phase locking is is big deal
两阶段锁是很重要的一个东西
01:06:11 – 01:06:14
but it’s also not too difficult to implement
但它实现起来也并不困难
1.06.14–1.06.15
and it gives us exactly what we want
它给了我们想要的东西
1.06.15–1.06.17
gives us our serializable schedules
它给了我们Serializable Schedule
01:06:18 – 01:06:22
we just have to be disciplined about whether we’re gonna try to detect our deadlocks and handle them
我们需要明确我们是否该试着去检测并处理死锁
1.06.22–1.06.27
or we’re gonna try to prevent them entirely in the first place
或者，我们会试着将预防它们放在首位

NEXT CLASS

Timestamp Ordering Concurrency Control

01:06:27 – 01:06:34
next class I think it’s going to be Dana talking to you guys about time stamp ordering which is good
下节课的话，我觉得应该是Dana会带你们去看下以时间戳为顺序的并发控制，这个东西很不错