

16-01

16 – Concurrency Control Theory

(CMU Databases Systems Fall 2019)

00:16 – 00:22

all right again Thank You DJ drop table for keep things fresh

感谢DJ Drop Table每次为我们做的事情

00:22 – 00:27

all right,so we'd love to talk about real quickly again for the assignments

So, 我们先快速讨论下你们的assignment

ADMINISTRIVIA

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

Homework #4 will be released next week.
It is due Wed Nov 13th @ 11:59pm.

00:27 – 00:31

project three hits been put out and that'll be due on Sunday November 17th

我们已经将Project 3放出了, 你们在11月17号的时候上交

00:31 – 00:33

and I'll talk about that briefly at the end of this class

我会在这节课结束的时候, 简单讨论下它

00:34 – 00:41

~~and then project or sorry~~ homework 4 should be four not three, four will be released next week

这边我写错了, 这里应该是Homework 4, 它会在下周放出

00:42 – 00:45

and that will be then better we due on on the thirteenth, okay

你们要在13号的时候上交

00:46 – 00:49

and as I said, my wife is going we're going to hospital tonight

我之前说过, 我老婆今晚要去医院待产了

00:50 – 00:51

it's gonna happen like nine hours from now

这是9小时以后的事情

00:51 – 00:53

so I'm gonna be gone for two weeks

So, 我有两周不在

0.53-0.55

I won't have office hours next week or the following week

下两周我都不在办公室

00:56 – 0.57

and then next week we will still have classes

下周我们依然有课

0.57-1.01

my PG students will what we take care of those classes

我的研究生会带着你们上课

01:01 – 01:02

Then the following Monday

然后，下周一的时候

1.02-1.07

we'll be getting one more PG student, and then we'll have no class on October 30th

我会让更多的研究生给你们上课，10月30号的时候，我们不上课

01:07 – 01:09

and then the schedule correctly reflects this, okay

这些规划已经反映了这点

你们可以从我的这些日程安排上看出这点

01:10 – 01:12

So any questions about any of this

So，对此，你们有任何疑问吗？

01:13 – 01:19

and that would make arrangements with my admin about having all the midterms in her office

我会与我的课程管理员在她的办公室一起安排下期中相关的事宜

01:19 – 01:23

and then we'll figure out some time you'd go come her office and check out your your midterm

然后，你们可以挑个时间去她办公室查下你们的期中考试成绩

01:23 – 01:26

again just bring your student ID ,so she knows who you are

带好你的学生卡，这样她就知道你是谁了

01:26 – 01:27

Okay

1.27-1.31

and then if you want something regraded, take a photo

如果你想要重新评分，那就给你的卷子拍个照

1.31-1.35

you can't take your midterm with you just take a photo the page you want me regraded and email me

你只需拍下你想让我重新打分的地方，然后以邮件的形式发给我即可

01:35 – 01:37

and we'll take care of it okay

我们会处理它的

COURSE STATUS	
A DBMS's concurrency control and recovery components permeate throughout the design of its entire architecture.	Query Planning
	Operator Execution
	Access Methods
	Buffer Pool Manager
	Disk Manager

01:38 – 01:44

all right, so we're right now in the course is that we've covered the entire stack

就目前来讲，我们已经将整个技术栈都介绍了一遍

01:44 – 01:45

we've covered how to store things on disk

我们已经讨论过如果将数据保存在磁盘

1.45–1.48

we covered how to put things in a buffer pool

我们也介绍了如何将数据放入buffer pool

01:48 – 01:52

the access method scans how to execute operators and how to do query planning

我们也讨论了access method，如何执行这些operator以及该怎么制定查询计划

01:53 – 01:56

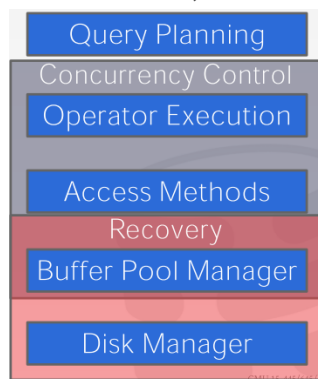
so now we're gonna actually can look look at for the next four weeks

So，在接下来的四周时间里

1.56–1.59

we're gonna go back and look at the entire architecture all over again

我们会回过头去，再看下这整个架构体系



01:59 – 02:05

and now consider two important components ,concurrency control and recovery

然后，我们要去思考两个重要组件，即并发控制和故障恢复

02:05 – 02:09

and these concepts actually permeate all throughout the entire system

实际上，这些概念在整个系统中无处不在

02:10 – 02:11

we kinda need to understand the basics first

我们首先需要去理解这些基础概念

2.11–2.14

and that's why we went through without discussing any of these things

这就是为什么我们先讨论了这些，才开始讨论这两个东西

02:14 – 02:17

and now we're going back ,and seeing ~~how we wouldn't~~

现在我们回过头来再看这些东西

2.17–2.21

you know if you want to enforce concurrency control or make sure that our databases

can be stored on disk safely

你知道的，如果你想使用并发控制，或者确保我们的数据库可以被安全地保存在磁盘上

02:21 – 02:26

~~how do we make sure that~~ how do we modify what we've already talked about to

account for these things and take care of it

我们该如何通过调整我们已经讨论过的东西来解决这些问题呢？

02:27 – 02:34

so again concurrency control and recovery are all you know they're not like should be separate things on the side with a buffer pool manager or an index

So, 你知道的, 并发控制和故障恢复并不是独立于buffer pool管理器或者索引之外的东西

02:35 – 02:41

the entire system needs you aware of how it's gonna be durable, how transactions are gonna run safely

在整个系统中, 你需要去知道该如何将数据持久化, 以及如何让事务安全地执行

02:41 – 02:47

and so that's why we're going we're covering this at the at the second half of the semester

So, 这就是我们要在下半学期所讲的东西

02:47 – 02:50

and I would say also to once we have these two things

我要说的是, 一旦我们有了这两个东西

2.50–2.52

that you can go off the world and build your own database system

那你就拥有了全世界, 你可以去构建你自己的数据库系统

02:52 – 02:56

but these are the last two things we need to actually build an database system

这就是我们构建一个数据库系统时所需的最后两个东西

2.56–2.59

that can run transactions correctly and make sure that everything is safe

它可以让事务正确地执行, 并且确保所有东西都是安全的

03:00 – 03:01

so we're almost there

So, 学会这些, 我们就基本实现了一个数据库系统

MOTIVATION

We both change the same record in a table at the same time.
How to avoid race condition?

← Lost Updates
Concurrency Control

You transfer \$100 between bank accounts but there is a power failure.
What is the correct database state?

← Durability
Recovery

03:02 – 03:06

so to motivate why we want to talk about concurrency control and recovery

So, 我们想讨论并发控制和故障恢复的动机是

3.06–3.08

let's look at two simple scenarios

我们来看两个简单场景

03:08 – 03:10

so let's say that I have an application

So, 假设我有一个应用程序

3.10–3.17

where I want to have two threads try to update the same record in the same table at exactly the same time

我想通过两条线程来试着同时对同一张表上的同一条记录进行更新

03:19 – 03:23

how to make a decision about which one should you succeed, well which one should be our final change

我们该如何决定哪条线程的操作会成功, 哪条线程所做的修改才是我们的最终修改

03:23 – 03:24

and there's a race condition here
这里就会出现条件竞争 (race condition)

03:25 – 03:28

I would have want to come slight before the for another
我想让一条线程略微先于另一条线程执行操作

03:29 – 03:30

the other scenario is that
另一个场景是

3.30–3.33

let's say that I have an application that for my bank
假设我有一个银行应用程序

03:34 – 03:38

and I want to transfer a hundred dollars out of my account into your account
我想从我的账户转100美金到你的账户

03:38 – 03:43

but let's say ~~you know before I you know~~ after I take the money out of my account ,but
before I can put it in your account

但假设，当我从我的账户中取出钱后，在我将钱放入你的账户前

03:44 – 03:48

the the building the data center gets struck by lightning
数据中心所在的大楼被雷劈了

3.48–3.51

the the lose all power our machine crashes or database system crashes
大楼就断电了，我们的机器发生了崩溃，或者数据库系统发生了崩溃

03:52 – 03:53

so when we come back

So，当供电恢复的时候

You transfer \$100 between bank
accounts but there is a power failure.
What is the correct database state?

3.53–3.55

what should be the correct state of the database

那么数据库的正确状态应该是什么呢？

03:55 – 03:57

But what should we actually see

但我们实际应该看到的是什么呢？

MOTIVATION

We both change the same record in a
table at the same time.

How to avoid race condition?



Lost Updates
Concurrency Control

You transfer \$100 between bank
accounts but there is a power failure.

What is the correct database state?



Durability
Recovery

03:58 – 04:02

so the first problem that I'm talking about here at the top

So, 这就是这里我所要讨论的第一个问题

04:03 – 04:05

this is an example of a lost update

这是一个关于丢失更新操作的案例

4.05–4.07

or I have two transactions

假设我有两个事务

4.07.4.09

two threads trying to make an update to the same record at the same time

有两条线程试着在同一时间对同一记录进行更新

04:09 – 04:11

I couldn't have end up missing one

我不能丢失其中任何一个事务所做的操作

04:12 – 04:14

all right how to make sure that doesn't happen

我该如何确保避免这种情况的发生呢?

04:14 – 04:18

and the way we're gonna use ensure that these things happen correctly

我们确保这些东西正确执行的方式是

4.18–4.23

it's through a concurrency control mechanism, a concurrency control protocol

使用一种并发控制机制, 或者一种并发控制协议

04:23 – 04:24

for the second scenario

在第二种情况中

4.24–4.28

where my machine my my my data center catches on fire

假设我的数据中心着火了

04:28 – 04:31

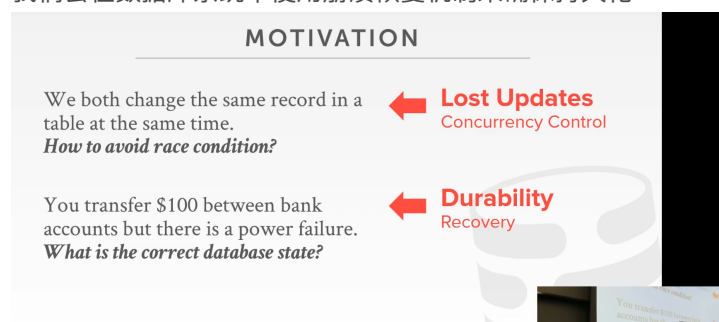
and I you know my I lose power and my machine crashes

并且断了电, 我的机器也发生了崩溃

04:32 – 04:36

we're gonna use the recovery mechanism in the database system to ensure Durability

我们会在数据库系统中使用崩溃恢复机制来确保持久化



04:37 – 04:44

so these concurrency control and durability are one of the main selling points of a database management system

So, 并发控制和持久化是一个数据库管理系统的主要卖点之一

CONCURRENCY CONTROL & RECOVERY

Valuable properties of DBMSs.

Based on concept of transactions with **ACID** properties.

Let's talk about transactions...



04:45 – 04:50

this is why if you're building an application whether it's in the cloud or on a cell phone or on a desktop

如果你构建了一个应用程序，不管它是运行在云端还是手机或者是台式机上的

04:51 – 04:55

You don't want to be in the business of doing these things yourself in your application

你不想在你的应用程序中亲自来做这些事情

04:56 – 4.57

because you're probably gonna get it wrong

因为你可能会搞出问题

4.57–4.59

and you can have losing data or have incorrect data

你可能会丢失数据或者得到错误的数据

05:00 – 05:02

This is why you want to use a database management system

这就是为什么你想要使用DBMS的原因所在了

5.02–5.07

because they have really smart people that have been spending a lot of time to make sure that these things happen correctly

因为它是由很多聪明的人花了大量的时间来开发的，以此来确保这些行为都是正确的

05:08 – 05:10

if you also think about it to it like if you're a startup

假设，如果你是一家初创企业

5.10–5.14

you know the if you're shipping an application

如果你正在交付一个应用程序

05:14 – 05:21

it doesn't matter you know the end of the day what's with what's not gonna sell your product is oh I can I can recover the database after a crash

如果你的数据库系统出现了崩溃，你说你可以恢复其中的数据，这种说法并不会有助于卖出你的产品

05:22 – 05:24

right you need that as a feature you absolutely have to have

因为这是你必须要有有一个功能

05:25 – 05:28

but that's not a differentiating aspect of your application versus your competitors

但这无法让你应用程序和你竞争对手所提供的产品产生差异

05:29 – 05:35

all right, so ~~you don't want to get~~ if you don't want any business of writing a database management system from yourself unless that is your job

So, 你不会想要自己去写一个DBMS, 除非这是你的工作

05:35 – 05:38

for everything else people should rely on

人们应该还有去依靠些其他东西

5.38–5.44

you know high quality software database system software that is vetted to do these things

比如, 那些高质量软件、数据库软件通过审查才能去做这些事情

05:44 – 05:57

so the core concept, that we're going to use through the next four weeks discussing this these, you know running the our system to make sure that things are running in the correct order or that all our changes are so durable

So, 为了确保我们系统中一切东西都是按正确顺序执行, 或者所有修改都被持久化, 我们接下来四周在讨论我们运行系统时使用的核心思想是

05:57 – 05:59

is this idea of transactions

那就是事务的思想

5.59–6.03

that are gonna run with ACID guarantees or ACID properties

它会和ACID这种特性一起使用

06:03 – 06:06

So there's a quick show of hands who here has heard of the acronym ACID before

So, 在座的快速举下手, 你们有谁之前听过ACID吗

06:07 – 06:08

all right ,about half okay

Ok, 半数都听过

06:08 – 06:10

so we'll cover that

So, 我们之后会介绍它

06:10 – 06:12

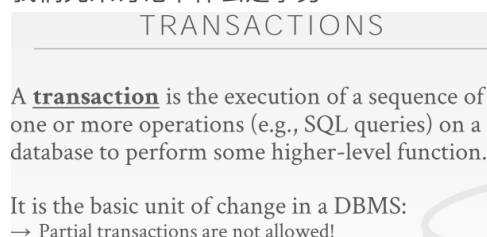
So before we can talk about ACID

So, 在我们讨论ACID之前

6.12–6.13

let's talk about what a transaction is

我们先来讨论下什么是事务



TRANSACTIONS

A **transaction** is the execution of a sequence of one or more operations (e.g., SQL queries) on a database to perform some higher-level function.

It is the basic unit of change in a DBMS:
→ Partial transactions are not allowed!

06:14 – 06:16

so in our world today

So, 在我们如今的世界中

6.16–6.19

if we're talk about in this lecture

用我们课堂上的话讲

6.19–6.27

the transaction is gonna be the execution of a sequence of operations on a database system to perform some higher-level function

事务其实就是，通过在数据库系统中执行一系列操作来执行某种更高级的功能

06:27 – 06:32

and so these operations you can sort of think about as SQL queries or the reads and writes we're doing to the database

So，你可以将这些操作想象成SQL查询或者是我们对数据库所做的读和写操作

06:33 – 06:44

And by higher-level function I mean something like that ~~reputation~~ you know some feature we want our application to perform the steps

我所说的更高级的功能指的是，我们想让我们的应用程序所执行的某种功能

06:45 – 06:47

Like transfer money from my account into your account

比如：从我的账户上转钱给你的账户

6.47–6.49

that would be a high-level function

这就是一个高级功能

06:49 – 06:52

because it's ~~supply~~ program in a transaction in our application

因为这就是我们程序中的一个事务

06:53 – 06:58

No database systems gonna have that did that feature that like that single function can call you know move money

没有任何数据库系统会拥有这种的功能，即通过调用单个函数来实现转账的效果

06:58 – 07:01

this is something you would write in your application up above

这种功能是你写在你的应用程序中的

07:02 – 07:06

so transactions are going to be the basic unit of change in our database management system

So，事务其实是我们数据库管理系统中关于修改操作方面的一个基本单位

7.06–7.07

meaning

这意味着

7.07–7.13

this is well how all changes are going to occur in the in the wrapped inside of a transaction

这意味着所有要做的修改会被包裹在一个事务中

07:13 – 07:16

right whether it's if it's multiple queries or a single query

不管它是多个查询还是一个查询

7.16–7.18

it's always going to be a transaction

它始终都是一个事务

07:19 – 07:22

I suppose you can have a zero query transaction

假设，你有一个零查询事务

7.22–7.23

, but that doesn't really mean anything right

但它其实啥也不是

07:23 – 07:27

but it's assumed that it's one or more operations we want to do

但假设该事务中包含了我们想去执行的一个或多个操作

07:28 – 07:30

and so the key concept though about transactions is that

So, 事务中的核心概念是

It is the basic unit of change in a DBMS:
→ Partial transactions are not allowed!

7.30–7.32

we're not going to allow for partial transactions

对于执行事务来说, 我们不允许只执行该事务中的部分操作的情况出现

7.32–7.35

where transactions are always going to be atomic

事务始终得具备原子性

07:35 – 07:36

but that means that

但这意味着

7.36–7.39

if I have a sequence of five updates I want to do

如果该事务中包含了我想执行的5个更新操作

7.39–7.43

either all five occur or none of them occur

要么这5个更新操作都执行成功, 要么就都不执行

07:43 – 07:47

I can't have you know some like you know maybe the first three out of five succeed and the other two fail

我不能让这种情况发生, 即这5个操作中只有3个成功执行, 剩下2个执行失败了

07:48 – 07:49

right it's either all or nothing

即要么全部执行, 要么全不执行

07:50 – 07:53

and even if you have a single query transaction a single operation transaction

如果你有一个只包含单个操作的查询事务

7.53–7.56

say I have an update query that updates five tuples

假设我有一条更新语句, 它会去更新5个tuple

07:57 – 07:58

all right, still one query

All right, 它依然是一个查询

7.58–8.00

but within that I'm updating five things

但在这个查询中, 我要对5个数据进行更新

08:01 – 08:04

all five had to get updated not not some subset of them

这个5个数据都要被成功更新, 不能只是部分更新成功

TRANSACTION EXAMPLE

Move \$100 from Andy's bank account to his promoter's account.

Transaction:

- Check whether Andy has \$100.
- Deduct \$100 from his account.
- Add \$100 to his promoter account.

08:06 – 08:10

So the transaction example would be that the one I talked about before

So, 这个事务例子其实就是我之前讲过的一个例子

8.10–8.14

I want to move a hundred dollars out of my bank account into my shady promoters account

我想将100美金从我的账户转到一个推广者的账户中

08:14 – 08:17

so the database system doesn't provide this functionality

So, 数据库系统并没有提供这种功能

8.17–8.19

in my application code ,I would write the steps to perform this

我会在我的应用程序代码中写出该功能的执行步骤

08:20 – 08:22

right so in the first step

So, 在第一步中

8.22–8.26

I would say well check to see whether Andy has \$100 he probably doesn't, right

我会说, 先检查下Andy账户中有没有100美金, 他账户中可能并没有这么多钱

08:26 – 08:28

but then if I do

但当我检查完后

8.28–8.30

then you can take the hundred dollars out of my account

那么你就可以从我的账户中取出这100美金

08:30 – 08:32

and then put the hundred dollars into his account

然后, 将这100美金放入他的账户中

08:33 – 08:34

all right

8.34–8.35

again these are separate steps

这些是单独的步骤

8.35–8.44

there's no magic way to just materialize money in a single at the lowest level the hardware to automatically update something ,and another thing at the exact same time 没有任何黑科技可以在最底层（硬件层面）做到自动更新某个值, 然后在同一时间更新另一个值

08:44 – 08:48

All right, there's a bunch of extra stuff we have to do to make sure that this happens atomically

我们还得做一些额外的步骤来保证这些操作的原子性

08:48 – 08:50

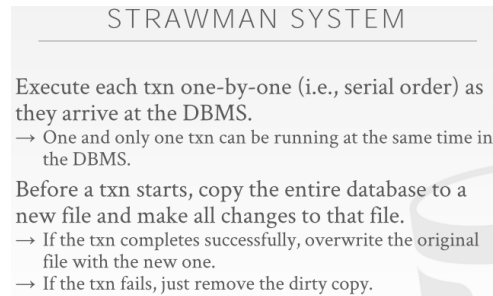
but from the applications perspective

但从应用程序的角度来说

8.50-8.54

you know they invoke this transaction and this will all happen or none of it happens

你知道的，它们执行了这个事务，要么这个事务中的操作被全部执行，要么就都不执行



08:56 – 09:01 ! ! ! !

so let's talk about a really simple database system we could build that could do this for us

So，我们来讨论下我们可以构建出来的一个可以为我们做到这点的很简单的数据库系统

09:02 – 09:06

So let's say we have a database system that has only supports a single thread

So，假设我们有一个只支持单线程的数据库系统

9.06-9.11

meaning only one transaction and only one query can run at a single time

这意味着，一次只能执行一个事务或者一个查询

09:12 – 09:16

And you know if multiple queries are multiple transactions show up in the system

如果该系统要去执行多个事务

09:16 – 09:17

it just puts them in a queue

它会将这些事务放在一个队列中

9.17-9.20

and there's one thread pulling things off that queue and running them one by one

然后通过一条线程从队列中拉取并逐个执行这些事务

09:21 – 09:24

so now before transaction starts Executing

So，在开始执行事务前

9.24-9.25

what they're gonna do is

它们要做的事情是

9.25-9.30

they're gonna copy the entire database file or set of files what however it's architected

它们会去复制整个数据库文件或者一组文件，这取决于它的架构是什么

09:30 – 09:33

it's gonna make a second copy of the database

它会去制作该数据库的第二个副本

9.33-9.36

make all the changes that wants to make to that copy

并将所有的修改应用到该副本上

09:36 – 09:37

and then if it succeeds

如果修改成功

9.37-9.39

and we want to save all our changes

并且我们想保存我们所做的所有修改

9.39-9.45

then we just flip a pointer to say now the new version of the database is the second file I just created

那么我们只需重新设定下我们的指针，将它指向我刚创建的第二个文件（该数据库的新版数据）即可

09:45 - 09:46

all right

9.46-9.50

so this guarantees that atomicity is to be propagated is mentioned

So，这保证了原子性会被传播出去

9.50-9.53

because if I make the copy to the database

因为如果我制作了该数据库的副本

09:53 - 9.55

and then I'm doing five writes

接着，我进行了5次写操作

9.55-9.57

but then the first three happen, and then I crash

但只执行了前三个写操作后，然后系统就发生了崩溃

9.57-9.58

when I come back

当系统恢复后

9.58-10.00

I still have my original copy of the database

我依然留有该数据库的原始副本

10.00-10.01

because I didn't affect that

因为我并没有对它产生影响

10:02 - 10:05

I said that you know so everything is still correct there that's fine

So，所有数据依然是正确的，没啥问题

10:06 - 10:07

Things are being written to disk

这些数据都会被写回磁盘

10.07-10.09

so if I crash

So，如果数据库系统发生了崩溃

10.09-10.11

I could come back and those my disks didn't die

当系统恢复正常后，并且我的磁盘也没有挂掉的话

10.11-10.12

all my data is still there

那么我的所有数据依然保存在磁盘中

10:15 - 10:16

So would this actually work

So, 这种做法实际可行吗?

10:18 – 10:18

he says yes

这位同学表示Yes

10:19 – 10:19

would this be fast

这种做法速度很快吗?

10:19–10:24

says no why,you said no ,so why

你说了No, 那你能说下这是为什么吗?

~~10:34 – 10:35~~

~~if something to the UI please doesn't matter ,right;~~

10:34–10:39

the amount of the matter updates I'm doing in my transaction it doesn't matter

不管我的事务中有多少个更新操作, 这都没关系

10:39 – 10:41

Because I'm copying the file every single time

因为每次我都会复制这个文件

10:41 – 10:42

so I copy entire file

So, 我复制了整个文件

10:42–10:46

and make one change versus a thousand changes like that copy cost is always the same

不管我对该文件进行1次修改还是1000次修改, 复制文件的成本始终是相同的

10:47 – 10:49

But you're right, the copy part is expensive

但你说的的是对的, 复制这一步骤的成本很高

10:49 – 10:54

if it's ~~a one kilobyte or starting~~ a four kilobyte page for my database

如果我对数据库中的一个4kb大小的page进行操作

10:54–10:55

who cares

谁会在意这种成本呢?

10:55–10:57

that's one Hardware read and write, I can I do that pretty quickly

我只需做一次硬件层面的读和写操作, 我可以很快速地完成这些操作

10:57 – 11:00

but if I have one petabyte of data

但如果我的数据大小是1PB

11:00–11:00

now for every single transaction

那么, 在执行每个事务的时候

11:00–11:03

I'm copying one petabyte every single time

那我每次都得复制这1PB数据

11:03–11:04

making my changes

将我所做的修改应用到该数据副本上

11.04–11.06

and then updating the pointer

然后更新指针

11:07 – 11:09

so this is a good example

So, 这是一个很好的例子

11.09–11.10

Well, we design a system

我们设计了一种系统

11.10–11.14

that had the properties that we want and particularly acid properties that we'll talk about

它拥有我们想要的属性，特别是我们之后要谈论的ACID

11:14 – 11:17

But this is going to be super slow to do it this way

但使用这种方式的话，速度就会超级慢

11:18 – 11:19

the other issue is that

另一个问题是

11.19–11.20

we're also running with a single thread

我们系统使用的是单线程运行

11:21 – 11:25

so I didn't say anything about whether the database fits in memory or not

So, 我并没有说这个数据库文件是否放在内存中

11:25 – 11:29

right so now if I'm running with a single thread and tries to touch data that's not in memory, but it's in disk

如果我所运行的系统是单线程的，并且我们试着访问的数据并没有放在内存中，而是放在了磁盘中

11:30 – 11:32

I have to stall my thread until I go fetch it

我就得阻塞我的线程，直到我拿到这些数据

11:33 – 11:34

and I can't run anything else

并且我就不能执行其他东西了

11.34–11.35

because I only have one thread that can do this at a time

因为我一次只能通过一条线程来干活

PROBLEM STATEMENT

A (potentially) better approach is to allow concurrent execution of independent transactions.

Why do we want that?

- Better utilization/throughput
- Increased response times to users.

But we also would like:

- Correctness
- Fairness

11:37 – 11:44

so what we're gonna talk about today and for the next couple weeks is a potentially better approach

So, 我们今天和接下来两周要讨论的是一种可能更好的方案

11.44–11.50

where we're going to allow transactions to run simultaneously at the same time

即我们允许在同一时间同时执行多个事务

11:50 – 11:54

and then we're gonna come up with a way to try to put potentially interleave their operations

我们会试着想出一种方式，以此来交错执行它们的操作

11:55 – 11:56

in such a way that

通过这种方法

11.56–11.57

we maximize our parallelism

我们最大化了我们的并行能力

11.57–12.02

but still get all the safety guarantees that they want ,and correctness guarantees that we want in our database system

但我们依然获得了他们想要的安全性保证，以及我们想在数据库系统中获得的正确性保证

12:03 – 12:06

right and again it's obvious why we want to do this

我们想要这样做的原因显而易见

12.06–12.10

because we talked about two before, when we talked about latching,we talked about query execution

在我们讨论latch和查询执行的时候，我们就已经讨论过这两点了

PROBLEM STATEMENT

A (potentially) better approach is to allow concurrent execution of independent transactions.

Why do we want that?

→ Better utilization/throughput

→ Increased response times to users.

12:11 – 12:14

if we can get we can allow multiple queries that run at the same time

如果我们可以同时执行多个查询

12:15 – 12:17

we're gonna get better utilization of our Hardware

我们就可以更好地去利用我们的硬件

12.17–12.20

better throughput meaning we can do more more work in a same amount of time

更高的吞吐量意味着，在相同的时间内，我们可以做更多的工作

12:21 – 12:24 ! !

and then the system's gonna look more responsive and snappy

然后，系统的响应速度看起来就会更快

12.24–12.31

because now I don't have to wait in that single queue until my quick transaction gets to the front, and then I can run

因为我现在不需要等待我的事务走到队列的前面后，才去执行我的事务

12.31-12.34

I could potentially start running right away

我现在可以直接执行这个事务了

12:34 – 12:37

but now the tricky thing is gonna be is that

但现在棘手的地方在于

12.37-12.38

how do we actually do this interleaving

我们实际该如何交错执行这些操作

12.38-12.43

you know such a way that we don't violate any better correctness guarantees of our system

你知道的，以一种不违反我们系统正确性保证的方式来交错执行这些操作

12:43 – 12:51

and that we don't starve any one transaction you know from taking all the resources and other transactions can't do anything

我们不能让任何一个事务执行的时候占据了全部资源，这会让其他事务什么事情也做不了

12:52 – 12:57

so concurrency control we're talk about today is a is an old concept that goes back to the 1970s

So，我们今天所谈论的并发控制是一个很古老的概念，这得从1970年代说起

12.57-13.02

so when IBM built system R, this is one of the first things they also invented

So，这是IBM在构建System R时，这是他们最先发明的一个东西

13:02 – 13:05

And so in a disk based system

So，在一个基于磁盘的数据库系统中

13.05-13.06

back then

在那个时候

13.06-13.07

of course

13.07-13.09

because memory was limited

因为可使用的内存量有限

13.09-13.12

and any time transaction actually could touch data that's on disk and on a memory

每当事务要去使用存放在磁盘和内存中的数据时

13:12 – 13:16

and therefore it would stall and then now you could let other transaction to run at the same time

因此，这个事务会停下来，那么现在你可以在同一时间去执行其他事务

13:17 – 13:19

in modern systems today

在当下的数据库系统中

13.19-13.21

usually for all it to be applications

通常对于所有的应用程序来说

13.21–13.23

they're not that big the databases aren't that big

它们所使用的数据库数据大小并不大

13:23 – 13:26

so we have enough memory where we could put the entire database in memory

So, 我们有足够的内存, 我们可以将整个数据库放入内存中

13:27 – 13:30

For analytics, you still go to disk ,but that we're not doing transactions there

对于分析型任务来说, 你依然得从磁盘中获取数据, 但我们不会在磁盘中执行事务

13:31 – 13:33

so in a modern system

So, 在一个现代系统中

13.33–13.35

most mostly ultimate databases can fit memory

大多数数据库都能放入内存中

13:36 – 13:38

but now Intel is giving us more and more cores

但现在, Intel赋予我们使用更多CPU核心的能力

13:39 – 13:42

so now we're gonna allow transactions around in different cores at the same time

So, 我们允许多个事务同时不同的CPU核心上执行

PROBLEM STATEMENT

A (potentially) better approach is to allow
concurrent execution of independent transactions

Why do we want that?

- Better utilization/throughput
- Increased response times to users.

But we also would like:

- Correctness
- Fairness

13:42 – 13:44

And then we still need to guarantee all these things

并且我们依然需要去保证这上面所提到的所有东西

13:45 – 13:49

so even though the hardware is different from when how people first invented
concurrency control back in the day

So, 虽然当下的硬件和人们当时发明并发控制时的硬件已经不可同日而语

13.49–13.50

we still have the same problem

但我们依然面临着相同的问题

13:50 – 13:52

So you still want to maximize parallelism

So, 我们依然想去最大化并行能力

TRANSACTIONS

Hard to ensure correctness...

→ What happens if Andy only has \$100 and tries to pay off two promoters at the same time?

Hard to execute quickly...

→ What happens if Andy tries to pay off his gambling debts at the exact same time?

13:53 – 13:55

and as I said a couple times already

我之前已经说过这个好几次了

13.55–13.57

this is gonna be really hard to do

这做起来真的很难

13.57–13.58

and as I said last class

正如我上节课所讲的那样

13.58–14.02

this is probably the second hardest thing to do in database systems to do concurrency control

这可能是在数据库系统中使用并发控制时所遇到的第二大难题

14:03 – 14:08

and this is part of the reason why the NoSQL guys when they first came out ten years ago, they were like we're not doing transactions

这就是为什么十年前搞NoSQL那批人不想使用事务的部分原因了

14:08 – 14:09

okay that's too hard

这个问题太难解决了

14.09–14.10

because they want to run faster

因为他们想要运行地更快

14:11 – 14:16

so it's gonna be super hard for us to guarantee correctness with transactions

对我们来说，要保证事务的正确性真的是太难了

14:17 – 14:17

all right

14.17–14.24

so if I we have \$100 in my bank account ,and I try to give money to people at the exact same time, what should happen

So, 如果我想将我账户中的100美金转到别人的账户，这会发生什么呢？

14:24 – 14:28

~~because I don't want to~~ you know assuming that banks not gonna let me overdraft
假设银行不允许我透支我的账户

14.28–14.31

you know I don't wanna be giving out money I don't actually have

你知道的，我不想将我实际没有的钱给出去

14:32 – 14:34

and then it's also gonna be hard to execute this very efficiently

并且，我们也难以高效地去执行这个操作

14.34–14.38

because again if I do the serial execution case that I talked about in the beginning

如果我使用的是我一开始讨论过的按顺序执行

14.38–14.40

then that's gonna be always correct

那么它的执行结果始终是正确的

14.40–14.42

because only one transaction is running at a time

因为一次只执行一个事务

14:42 – 14:44

And then how many worries about any interleaving

那么我们也不用担心交错执行

14:45 – 14:46

but for now I do want to interleave them

但现在我想要交错执行这些事务

14.46–14.51

I want to bet to be as efficient as possible to be to figure out whether I'm running correct still correctly

我想尽可能高效地去弄清楚我的执行结果是否依然正确

14:52 – 14:54

I just there's me some overhead to figure these things out

我通过一些额外开销来弄清楚这些事情

PROBLEM STATEMENT

Arbitrary interleaving of operations can lead to:

→ Temporary Inconsistency (ok, unavoidable)

→ Permanent Inconsistency (bad!)

We need formal correctness criteria to determine whether an interleaving is valid.

14:56 – 15:01

so what we're essentially trying to do today, and for the next three or four lectures is

So, 本质上来讲, 我们今天以及接下来三四节课所试着做的事情是

15.01–15.06

allow for these interleaving the operations of transactions

我们允许这些事务中的操作交织执行

15:06 – 15:07

and that well as we see is that

正如我们所见到的

15.07–15.09

when we start doing these interleaving

当我们开始交错执行这些操作时

15.09–15.10

we can end up with inconsistent databases

我们最终可能会导致数据库数据不一致

15:12 – 15:14

and sometimes it's okay ,sometimes it's not okay

有时这是Ok的, 有时这并不Ok

15:16 – 15:19

so some inconsistencies will be okay,because they're temporary

So, 有些数据不一致可能是ok的, 因为它们是临时数据

15:19– 15:20

so for example

So, 例如

15.20–15.23

if I'm taking money out of my account putting in your account

如果我将我的钱从我的账户转到你的账户

15:23 – 15:26

again I can't do that atomically at the hardware level

我无法在硬件层面做到原子性

15.26–15.29

I have to do that with you know multiple instructions or multiple operations

我得通过多条指令或者多个操作才能做到这点

15:29 – 15:40

So there will be a brief period in time where I take the \$100 out of my account and then

before I put it in your account ,that hundred dollars it doesn't exist anywhere

在我将这些钱从我的账户转入你的账户前，在一段时间内，这100美金不存在于任何地方

15:40 – 15:41

so that's okay

So, 这是Ok的

15.41–15.43

because that's temporary it's unavoidable

因为它是临时的，这是不可避免的

15:44 – 15:47

the outside world will not see potentially not see that inconsistency

外界可能不会看到这种数据不一致

15.47–15.50

and we'll do some protection mechanisms make sure that they can't see this

我们得做些保护机制，以此来确保外部无法看到这些东西

15:51 – 15:55

and so because we're gonna allow this, this is gonna last actually make this all work

So, 因为我们通过允许这个临时数据，最终会使所有的一切都工作正常

15:56 – 15:58

but the thing we want to avoid permanent inconsistencies

但我们想去避免永久不一致的情况

15.58–16.00

well again if I take the hundred dollars out

Well, 如果我取出了这100美金

16.00+–16.01

and then I crash

然后数据库系统发生了崩溃

16.01–16.03

and I come back that hundred dollars better not be missing

当数据库系统恢复正常后，我希望这100美金最好没有丢失

16:04 – 16:07

all right,a better it better be in the other account or my account ,it can't just disappear

它最好在另一个账户（即我们的转账目标账户）或者是我的账户，它不能消失

16:09 – 16:12

so in order for us to understand whether we're doing the right thing

So, 为了让我们去理解我们所做的事情是否正确

16.12–16.18

whether we're coming out with the interleaving of our transactions that are actually correct

以及我们所交织执行的事务是否正常

16:19 – 16:22

we need a more formal definition of what actually means to be correct

我们需要一种更加正式的定义，即什么才是正确的

16:22 – 16:24

because it's sort of obvious for us right yeah

显而易见

16:24–16:28

if I take a hundred dollars of my account, and before I put it in your account we crash

如果我从我的账户中取了100美金，在我将它放入你的账户前，我们发生了崩溃

16:28– 16:32

like that sort of obvious we know that we don't want to lose a hundred dollars or any amount of money

在遇上这种情况，很明显，我们不想丢掉这100美金或者说任意金额

16:33 – 16:34

but from the database systems perspective

但从数据库系统的角度来看

16:34–16:38

it doesn't know that it's operating no money just sees a bunch of bytes and it's moving them around

它并不知道它是对钱进行操作，它只是看到了一些byte，然后对它们进行移动

16:39 – 16:42

so we need a way for us to reason about whether we're doing the correct thing

So，我们需要通过一种方式来解释我们所做的事情是否正确

DEFINITIONS

A txn may carry out many operations on the data retrieved from the database

However, the DBMS is only concerned about what data is read/written from/to the database.

→ Changes to the "outside world" are beyond the scope of the DBMS.

16:44 – 16:48

so the first thing we need to find what are these operations that we're actually doing

So，首先我们要弄清楚的是我们执行的这些操作实际做了什么

16:48 – 16:49

So as I said already

So，正如我之前说过的那样

16:49–16:52

a transactions **may contain** of one or more operations

一个事务中可能包含了一个或多个操作

16:55 – 16:56

but at a high level

但从一个高级层面来讲

16:56–17:02

~~the database~~ the application could be you know update this insert that make these changes

应用程序可以对数据库中的数据进行更新和插入操作

17:02 – 17:04

but from the database systems perspective

但从数据库系统的角度来说

17.04–17.06

it doesn't know about those high-level queries

它并不明白这些高级查询的意思是什么

17.06–17.09

it just knows that I'm doing low-level reads and writes

它只是知道，我正在做底层的读写操作

17:10 – 17:14

and so the only thing that we can reason about are the things that happened to our database

So, 我们唯一能解释的东西就是在数据库中所发生的事情

17:15 – 17:16

so that means that

So, 这意味着

17.16–17.23

if our transaction involves additional steps or procedures or operations, that aren't reads and writes on the database

如果我们的事务涉及了一些额外的步骤、过程或者操作，这些并不是对数据库进行读和写的操作

17:24 – 17:26

this is outside our purview

这就超出了我们的权限

17.26–17.26

,this is outside our control

超出了我们的控制范围

17.26–17.28

and we can't do anything about it

我们无法对它任何事情

17:30 – 17:31

so to give an example

So, 来看个例子

17.31–17.34

let's say that I'm gonna take a hundred dollars out of my account, I put it in your account

假设我从我的账户中转100美金到你的账户

17:35 – 17:39

and then I send an email to you to say the transfer succeeded

然后，我发了一封邮件给你，以此告诉你转账成功了

17:39 – 17:41

and that we want that to happen in the transaction

我们希望这个操作是在这个事务中发生的

17:42 – 17:47

but then before I can go commit and save my changes,there's a crash

但在我提交这个事务并保存我的修改前，突然发生了崩溃

17:48 – 17:49

so I've sent the email

So, 我已经发送了邮件

17.49–17.51

but then I crashed before I save all the changes

但在我将这些修改保存前，我就崩溃了

17.51–17.53

that email is gone up on the network

邮件已经通过网络发送出去了

17.53–17.55

it's outside the database now

这超出了数据库的控制范围

17:55 – 17:56

it's gone out in the real world

它已经进入了现实世界（即发送出去了）

17.56–17.58

we can't retract that

我们无法收回这封邮件

17:59 – 18:05

So we can only reason about and roll back and persist things that are these low-level reads and writes to our database

So, 我们只能对数据库中那些底层的读和写操作进行解释、回滚以及持久化

18:05 – 18:10

if we make a call to you know to an outside system or whatever like that's beyond us

如果我们调用了外部系统中的东西，那这就超出了我们的控制范围

18:11 – 18:12

no system can handle that

没有系统可以处理这点

18.12–18.16

at least we're talking out here so far

至少我们讨论过的都没法做到

18:16 – 18:16

okay

FORMAL DEFINITIONS

Database: A fixed set of named data objects (e.g., **A, B, C**, ...).

→ We do not need to define what these objects are now.

Transaction: A sequence of read and write operations (**R(A)**, **W(B)**, ...)

→ DBMS's abstract view of a user program

18:18 – 18:27

so the database that were going to be worried about today is going to be defined as a fixed set of arbitrary data objects

So, 我们今天所关心的数据库其实是由任意数据对象所组成的固定集合

18:27 – 18:30

that are each going to have a label or a name

每个数据对象都有一个标签或者名字

18:30 – 18:31

so in this case here

So, 在这个例子中

18.31–18.32

we'll just use ABCD

我们就使用ABCD来命名

18.32–18.34

but we'll just use alphabet characters

我们只需使用这些字母来命名即可

18:35 – 18:37

So the two things to point out here are

So, 这里要指出的两件事情是

18:37–18:40

one I'm not defining what a database object is

第一点是, 我并没有定义database object是什么

18:41 – 18:46

it could be an attribute, it could be a tuple, could be a page ,could be a table, could be a database it doesn't matter

它可以是一个属性, 一个tuple, 一个page, 一张表, 也可以是一个数据库, 这些都可以

18:46 – 18:53

all the same things that we'll talk about today ,and for the next couple couple classes they're all still work on different granularities

我们今天以及下两节课所讨论的所有东西, 它们能用在不同粒度上的

18:54 – 18:55

In practice

在实战中

18:55–18:57

most the time it's going to be based on a tuple

大部分情况下, 它都是基于tuple的

18:58 – 19:03

but we'll see in some cases you can take locks ,you can try to protect databases and tables

但在有些例子中, 你们可以试着用锁来保护数据库和表

19:03 – 19:07

nobody exercise protect single fields that becomes too expensive

没有人会对单个字段进行保护, 这样做的成本太高了

19:08 – 19:09

The other thing to point out too is that

另一件要说的事情是

19:09–19:12

I'm saying the database is a fixed size

假设, 数据库中的数据量大小是固定的

19:12 – 19:13

So that means

So, 这意味着

19:13–19:16

that the only operations we're gonna do are reads and writes

我们唯一要做的操作就是读和写

19:16–19:19

reads or updates of existing things

即对现有的数据进行读取或者更新

19:19 – 19:21

we're not gonna talk about inserts today

今天我们不会去讨论插入操作

19:21–19:22

we're not going to talk about deletes

我们也不会去讨论删除操作

19.22–19.25

the database system always has the same number of things

数据库系统中的数据数量是相同的

19:25 – 19:25

Because that's gonna complicate things

不然，这会让事情变得复杂

19:27 – 19:30

and we'll cover that on Monday next week

我们会在下周一介绍这个

19:30 – 19:33

so for today just assume that we've always have the same number objects

So, 就今天而言，我们会假设我们操作的对象的数量是不变的

19:34 – 19:40

and so now what the database is gonna see is just the sequence of read and write operations on these named Object up above

So, 现在，数据库所看到的的就是对这些named data object所做的一连串读写操作

19:40 – 19:44

so we're to say we use the function R for a read ,and the function of W for a write

So, 这里我们用函数R代表读操作，函数W代表写操作

19:44 – 19:48

so this is the only thing that we can see in our database system

So, 这是在我们数据库系统中我们唯一能看到的東西

19.48–19.53

we can't see anything else, any program logic, that the application may be running for the transaction

我们无法看到其他东西，比如应用程序执行事务时的程序逻辑

19:54 – 19:56

And that's gonna limit the amount of parallels and we're be able to get

这就会限制我们所能达到的并行量

19.56–20.02

because we don't understand some kind of high-level meaning with what the transaction is actually trying to do

因为我们无法理解事务在高级层面所试着做的事情