

# 16-02

---

## 16-02

20:02 – 20:03

we'll see one case

我们会看一个例子

20.03-20.05

where if you can get better parallelism

即如果你可以获得更好的并行能力

20.05-20.07

but in practice nobody does this

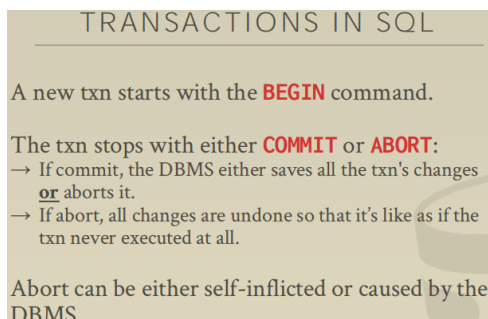
但在实战中，没人能做到这点

20.07-20.09

and we'll get to that later

我们之后会对此进行介绍

=====



20:11 – 20:13

so now from a practical standpoint

So，从实战的角度来说

20.13-20.17

how do you actually implement or use transactions in applications and database systems today

我们实际该如何在应用程序和数据库系统中实现或使用事务呢？

20:17 – 20:18

so in the SQL standard

So，在SQL标准中

20.18-20.22

you have these extra keywords begin commit and abort

我们拥有这些额外的关键字，即BEGIN，COMMIT以及ABORT

20:22 – 20:24

some systems use roll back instead of abort

有些系统使用的是ROLLBACK而不是ABORT

20.24-20.27

I think Postgres MySQL support both

我觉得PostgreSQL和MySQL对这两者都支持

20:27 – 20:32

so we're gonna explicitly start a new transaction with the begin keyword

So, 我们会通过BEGIN关键字来显式声明开始一个新事务

20:33 – 20:37

and then what happens is we make we any queries we then execute or a part of that transaction

接着我们要做的就是写出我们要做的任意查询, 并作为一个事务的一部分来执行

20:37 – 20:41

and then the other call I want to commit or abort

然后我可以通过调用COMMIT或者ABORT来决定是否提交这个事务, 还是中止这个事务

20:42 – 20:45

so if the user says I want to commit, then two things can happen

So, 如果用户说: 我想要提交事务, 那么这里就会发生两件事情

20:46 – 20:48

either the transaction does commit

如果该事务被提交了

20:48–20:50

the database saves all the changes that you made

那么数据库就会保存你所做的所有修改

20:50–20:53

and returns back in acknowledgments is to say they're successful

并返回一个确认消息, 以此表示该事务执行成功

20:54 – 20:58

or the database systems can say, you can't actually commit

或者数据库系统表示, 你不能进行提交

20:58 – 21:01

I'm not gonna let you make those changes

我不会让你做的这些修改生效

21:01–21:03

and I'm gonna go ahead and shoot you and abort you and you have the rollback

我会中止这个事务, 并回滚该事务所做的所有操作

21:03–21:05

and you get a notification that your transaction failed

然后, 你就会收到一个事务执行失败的通知

21:07 – 21:11

right so just because the application calls commit, doesn't mean you're actually gonna commit

So, 因为应用程序调用COMMIT并不意味着你就会提交这个事务

21:12 – 21:14

again, that's a very important concept that we can rely on later on

这是一个非常重要的概念, 我们之后可以用到

21:16 – 21:17

if the transaction gets aborted

如果这个事务被中止

21:17–21:22

then any changes that we made since we called begin will get roll back

任何自BEGIN处开始我们所做的修改都会被回滚

21:22–21:25

and it'll appear as if the transaction never ran at all

这就像是这个事务从未被执行一样

21:25 – 21:29

so that's how we guarantee if I'm moving hundred dollars out of my account to your account

So, 如果我从我的账户转100美金到你的账户

21:29 – 21:32

if the thing fails before we put the money in your account

如果在我将钱打入你账户前, 某个地方出现了问题

21:32–21:34

the transaction gets abort to come back,

那么该事务就会中止执行

21:34–21:37

and if we go back to the state we were before we started our transaction

并且状态会变回我们开始执行事务前的状态

21:38 – 21:40

right, this is how we guarantee that there's no partial transactions

这就是我们保证不存在那种只执行部分操作的事务的方式

21:40–21:40

yes

请问

21:44 – 21:45

His question is

他的问题是

21:45–21:47

why would do we want to tell the DBMS you want to abort

为什么我们想告诉DBMS你想中止这个事务

21:48 – 21:57

so a lot of times there's application code where you say take take take for example ,take the money out of my account, or I'm transferring money

我就拿转账这个例子为例, 我将钱从我的账户中取出, 或者我转钱给其他人

21:58 – 21:59

so I go look at my account first

So, 我首先会去看下我的账户

21:59–22:02

,I read that do I have \$100, yes

我会检查我的账户里面是否有100美金, 查出来确实有

22:02 – 22:04

now go take a hundred as my account

于是就从我的账户中取出100美金

22:04–22:05

but then I'll go read your account

接着, 我会去读取你的账户

22:05–22:08

and your bank's your accounts my flag fraud

我发现你的账户是个欺诈账户

22:08 – 22:11

so now I want to abort and roll that back

So, 现在我想中止这个事务, 并回滚所有的修改

22:11 – 22:13

right ,the simple reason

这是个很简单的原因

22:15 – 22:17

I don't know how often that occurs

我不清楚它的发生频率有多高

22:21 – 22:24

I always say I mean most code I want to commit they want to go to commit ,right

我的意思是，我想进行中止的时候，它们就会中止

我的意思是说，我想要去提交我想提交的东西

22:25 – 22:27

but we have to be able support this

但这个提交的过程得能够支持这一点（知秋注：万一你是个欺诈账户的话，你就得回滚事务）

22:29 – 22:31

So again the main thing to point out here is

So，这里主要指出的一点是

22.31–22.34

this abort could either be self-inflicted

事务的中止可能是自我引发的

22.34–22.38

meaning we tell us if we want abort or the database system tells you you have to abort

这意味着，如果我们想中止这个事务，或者数据库系统告诉你，你得中止这个事务

22:38 – 22:42

and then if you **system** comes back says you have to abort are you got aborted

那么，如果系统回过头来告诉你，你得中止这个事务，那这个事务就会被中止

22:42 – 22:48

then it's up for you in the application code to catch that you get like an exception back,

and says you know your transaction failed

这取决于你是否要在应用程序代码中去捕获这种异常，并说，你的事务执行失败了

22:49 – 22:51

and it'll suggest that you retry it

它会建议你重新执行这个事务

22.51–22.54

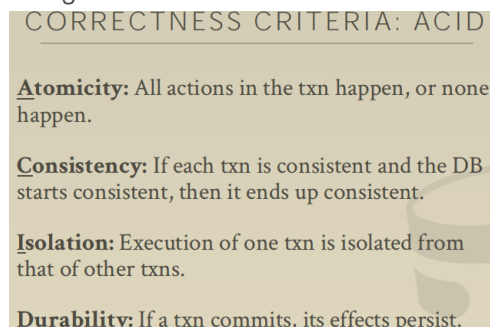
and you have to go back in application code if you actually care about this and retry

again

如果你在意这一点，你就得回到应用程序代码这里，并试着重新执行这个事务

22:55 – 22:55

all right



**CORRECTNESS CRITERIA: ACID**

- Atomicity:** All actions in the txn happen, or none happen.
- Consistency:** If each txn is consistent and the DB starts consistent, then it ends up consistent.
- Isolation:** Execution of one txn is isolated from that of other txns.
- Durability:** If a txn commits, its effects persist.

22:57 – 23:06

so the **correctness criteria** we're going to use now ,for this lecture and going forth through the rest of the semester is going to be defined in terms of this **acid** acronym

So, 我们这节课以及这学期剩下的时间里所要使用的正确性标准就是ACID

23:07 – 23:11

so ACID stands for **atomicity consistency isolation and durability**

So, ACID的意思是原子性、一致性、隔离性以及持久性

23:11 – 23:14

so atomicity is what we already talked about

So, 原子性这块我们之前已经讲过了

23:14–23:17

where we say all that the operations of a transaction have to occur or none of them occur

我们说过, 对于一个事务中的所有操作来说, 要么全部执行, 要么全不执行

23:17 – 23:19

right no partial transactions

不存在这个事务只执行其中部分操作的情况

23:20 – 23:22

consistency is sort of a weird one

一致性是其中比较奇怪的一个东西

23:22–23:24

I'll briefly talk about it

我之后会简单讲下它

23:24–23:28

but it's very handy how it actually means

但它非常便利

23:28–23:29

at least for a single node database system

至少对于一个单节点数据库系统来说是这样的

23:30 – 23:31

so it just says that

So, 它的意思是这样的

23:31–23:37

if the transaction is consistent ,like doorbell ,sorry I forget

如果事务是一致的, 不好意思, 闹铃响了, 我忘记开静音了

23:38 – 23:42

if the transaction is consistent ,and the database system is consistent

如果事务是一致的, 并且数据库系统也是一致的

23:42–23:48

then when the transaction executes ,then the database and state will be consistent

那么当事务执行的时候, 数据库的状态也会是一致的

23:48 – 23:50

so now you're like what is consistent mean

So, 所谓的一致是什么呢?

23:50–23:52

well at a high level it means correctness

Well, 从高级层面来看, 它的意思是正确性

23:53 – 23:54

but then what does that mean

但这意味着什么呢?

23:55 – 23:57

so again we'll cover this in a few more slides

So, 我们会在接下来的几张幻灯片中对此进行介绍

23:59 – 24:03

this one again as it was originally defined by the guy that invented this this acronym  
再说一遍, 它一开始是由发明这个缩略词的人所定义的

24:04 – 24:05

this one was always a really handy one

它一直是一个很方便的东西

24.05–24.12

it's some people feel like he sort of forced this one in here in order to get that the  
acronym to work out

有些会强制做到这点, 以此来具备ACID的特性

24:12 – 24:14

the other thing too is that

对此, 另一件事情是

24.14–24.18

the database **law** is that he made this thing up to make fun of his wife

他所创造的这个数据库法则用来取悦他老婆用的

24:19 – 24:22

Because like his wife didn't like candy or she was like a bitter woman or something

因为他的妻子并不喜欢糖果, 或者说她喜欢苦味的东西

24:22 – 24:24

so he named it after her,

So, 他将这个命名为ACID

24.24–24.26

I don't know whether that was true he's Germans and maybe

我不知道这是不是真的, 他可能是德国人

24:27 – 24:29

but there's another one called base

但这里有一个叫做BASE的东西 (Basically Available, Soft State, Eventual consistency)

24.29–24.31

which is for distributed systems or NoSQL systems

这是对于分布式系统或者NoSQL系统来说的

24.31–34.33

and we'll cover that and a few more lectures

我们会在稍后的课上对它进行介绍

24:33 – 24:36

so there's acid is what we care about here base will cover later

So, 今天我们会讲ACID, BASE的话, 我们之后再讲

24:37 – 24:38

isolation is another important one

另一个重要的东西就是隔离性

24.38–24.39

that means that

这意味着

24.39–24.41

the when our transaction executes

当我们执行我们的事务时

24:42 – 24:43

it should have the isolation

它应当具备隔离性

24.43-24.47

that it's running by itself even though other transactions may be running at the same time

即使在同一时间其他事务也在执行，该事务与它们是隔离开来的

在同一时间有几个事务在同时运行，事务间是彼此隔离开的

24:47 - 24:51

and the database system will provide that that isolation for it

数据库系统会为该事务提供隔离性

24:51 - 24:53

and then durability is

持久性指的是

24.53-24.55

if our transaction commits, all our changes get saved,

如果我们的事务都提交了，并且所有的修改都被保存了

24.55-24.58

and we get back at on acknowledgment that transaction committed

然后，我们会收到该事务被提交的通知

24:59 - 25:05

then no matter what happens to the database whether you know it the Machine crashes, it's not OS crashes ,the machine catches on fire

不管我们的数据库发生什么都没有关系，比如：遇上机器崩溃，OS崩溃，机器起火之类的事情

25:06 - 25:08

then all our changes should be persistent

我们所做的所有修改都应该是持久化的

25.08-25.11

maybe she always could be able to come back and see our changes

当系统恢复正常后，它应该能看到我们所做的修改

25:11 - 25:13

our changes may get overwritten

我们所做的修改可能会被覆盖

25.13-25.14

that's okay

这是Ok的

25:14 - 25:16

but for least for our transaction

但至少对于我们的事务来说

25.16-25.18

you know we know that all has changes got persistent

我们知道，所有的修改都会被持久化

## CORRECTNESS CRITERIA: ACID

Atomicity: “all or nothing”

Consistency: “it looks correct to me”

Isolation: “as if alone”

Durability: “survive failures”



25:20 – 25:23

So another shorthand way of looking at these things you would say

So, 快速记住这些东西的方式是

25.23–25.27

Atomicity just means all or nothing no parts of transactions

Atomicity指的是All or Nothing, 即我们在执行一个事务的时候, 不会只执行其中的部分操作

25:27 – 25:30

because consistency means it looks correct to me and correct will be in quotes

一致性的意思是, 对于我来说, 这个数据看起来是正确的, 这个正确要打个引号

25:31 – 25:32

Isolation means

隔离性的意思是

25.32–25.33

you’re running as if you’re alone

你执行事务的时候, 仿佛只有你一个人在执行事务

25.33–25.36

and then durability means that you’re going to survive all failures

接着, 持久性的意思是, 你的数据会从各种故障下存活下来

### TODAY'S AGENDA

Atomicity

Consistency

Isolation

Durability

25:37 – 25:38

so for today’s class

So, 在今天的课上

25.38–25.39

we’re going to go through each of these one by one

我们会对它们逐个进行讲解

25:40 – 25:42

and describe at a high level

并从一个高级层面来描述它们

25.42–25.49

what it means to determine whether we are achieving the ACID guarantee with a given property of each letter

我们会来看下我们所实现的ACID中这四个字母分别代表的意思

25:49 – 25:51

we’re gonna mostly focus on atomicity and isolation



我们会将重心主要放在原子性和隔离性上面

25.51–25.53

I'll briefly talk about consistency here

我会简单讨论下一致性

25.53–25.55

it doesn't really make that much sense for a single node system

对于单节点系统来说，这没有太多意义

25.55–25.56

it matters more for distributed systems

对于分布式系统来说，它很重要

25:57 – 26:00

and then for durability we're also not really going to talk about it too much

对于持久性这块内容，我们也不打算讨论太多

26:00 – 26:07

because we'll spend that whole two lectures after after I come back on on checkpoints and logging

在我回来讲完checkpoint和logging这两内容后，我们会花两节课来讲这块内容

26.07–26.07

because that's how they're gonna achieve that

因为这是我们如何实现它们的方式

26:09 – 26:10

okay

26.10–26.19

and I'll say also to that acid is what you would get in a if a relational DBMS says they support transactions

如果你们使用的关系型DBMS支持事务的话，那你们就会遇上ACID

26:19 – 26:21

this is typically what I mean

也就是我通常所说的这些东西

26:22 – 26:23

the NoSQL systems that don't do transactions

对于那些不使用事务的NoSQL数据库系统来说

26.23–26.32

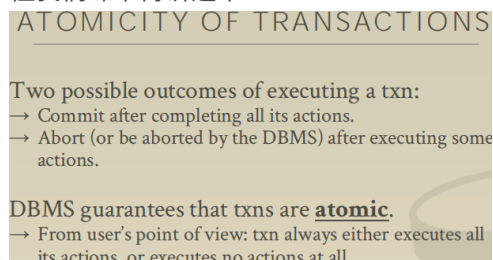
they're typically going to sacrifice often Atomicity, isolation, and consistency actually some of them do get rid of everything

它们通常会牺牲掉原子性、隔离性以及一致性。实际上有些NoSQL数据库直接不使用ACID

26:32 – 26:33

but we'll take that offline

但我们课下再讲这个



ATOMICITY OF TRANSACTIONS

Two possible outcomes of executing a txn:

- Commit after completing all its actions.
- Abort (or be aborted by the DBMS) after executing some actions.

DBMS guarantees that txns are **atomic**.

- From user's point of view: txn always either executes all its actions, or executes no actions at all.

26:34 – 26:36

all right let's talk about atomicity

我们来讨论下原子性

26:36 – 26:37

so as I said already

So, 我之前就已经讲过

26.37–26.40

there are two outcomes of our transaction

我们事务执行的结果有两种

26:40 – 26:45

either it commits and all our changes get get % or get could apply to the database all at once

要么我们在事务中所做的所有修改都被提交并落地到数据库中

26.45–26.47

or it gets aborted

或者, 就是该事务被中止了

26.47–26.52

because of some you know either database says so or application says so

因为这可能是数据库中中止了该事务, 也可能是应用程序中止了该事务

26:52 – 26:56

so again what we're providing the guarantee were providing to our application is

So, 我们为我们的应用程序所提供的保证是

26.56–26.58

that that any transaction that we execute

对于我们所执行的任意事务来说

26:59 – 27:00

all the changes will be atomic

所有的修改都是原子的

27.00–27.01

meaning

这意味着

27.01–27.04

they'll all appears that they happen exactly at the same time

这些事务看起来像是在同一时间发生的

27:05 – 27:06

So again it just means that

So, 这意味着

27.06–27.07

either everything happens or none of it happens

某个事务要么全部执行, 要么全不执行

27:08 – 27:11

so no matter what happens if I say I commit then I know everything got got saved

So, 不管发生什么, 如果我说我这个事务提交了, 那么我就知道所有修改都落地了

## ATOMICITY OF TRANSACTIONS

### Scenario #1:

→ We take \$100 out of Andy's account but then the DBMS aborts the txn before we transfer it.

### Scenario #2:

→ We take \$100 out of Andy's account but then there is a power failure before we transfer it.

***What should be the correct state of Andy's account after both txns abort?***

27:12 – :27:14

so let's look at two scenarios

So, 我们来看两种情况

27:14–27:17

where we could have problems atomicity

我们可能会在原子性方面遇上问题

27:17–27:19

and then we'll see how I actually want to solve it

接着, 我们来看下我们实际该如何解决它

27:20 – 27:22

so again my beloved example

So, 以我喜欢的这个例子为例

27:22–27:24

I'm taking a hundred dollars out of my account and putting it to another account

我从我的账户中取100美金, 并放入另一个账户

27:25 – 27:31

but then we we take the money out of my account ,but then the transaction gets aborted

当从我的账户中将钱取出后, 接着, 该事务被中止了

27:31 – 27:35

the machine doesn't crash, the database system doesn't crash we just get aborted

该机器和数据库系统并没有发生崩溃, 只是我们的这个事务被中止了

27:35– 27:36

the second scenario is

第二种情景是

27:36–27:38

when you can take the hundred hours out,

当你取出这100美金后

27:38–27:40

but now there's a power failure

但现在我们遇上了断电

27:40–27:43

and everything that the database systems running is lost

数据库系统中的所有东西都丢失了

27:44 – 27:46

we come back and what should be the correct state of the database

当系统恢复正常的时候, 它的正确状态是什么呢?

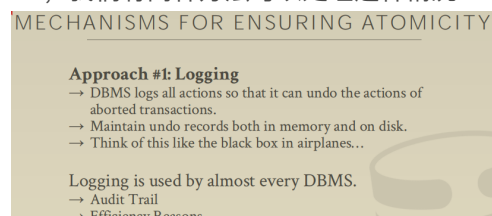
27:47 – 27:47

All right

27:49 – 27:50

so there's two ways we could possibly handle this

So, 我们有两种方法可以处理这种情况



27:52 – 27:54

the most common approach is to do logging

最常见的方案就是做日志 (Logging)

27:54 – 27:55

so when I say logging

So, 当我说logging的时候

27.55-28.00

I don't mean like you know the log debug messages you're using for your projects

我指的并不是你在你项目中用来进行debug的log消息

28:00 - 28:02

right I mean something like write ahead logging

我指的是预写式日志之类的东西 (Write Ahead Logging)

28.02-28.05

but we're actually recording our file on disk

但实际上, 我们会将文件记录在磁盘上

28.05-28.07

here's all just that we're making

这实际就是我们所做的事情

28:07 - 28:08

so what will happen is

So, 这里所发生的事情是

28.08-28.15

~~the database system gonna run, and as it runs a transaction every~~ for every change I

make to the database every update or write I do to the database

对于我对数据库所做的每次修改来说

28:16 - 28:20

I'm gonna make a copy of what the old value was that I'm overwriting

我会对我要覆写掉的旧值制作一份副本

28:21 - 28:24

and then that way if I crash or my transaction gets aborted

如果我遇上了崩溃或者事务中止的情况

28.24-28.27

I had the old value sitting around

我还留有旧值的副本

28.27-28.29

and I can go back and put it back in place

我可以回过头去, 将它恢复原状

28:30 - 28:33

so that when my transaction gets cleaned up after an abort

So, 当我的事务被中止并清理的时候

28.33-28.36

all the original value were still there

所有原来的值都还在那里

28:39 - 28:41

and so the way this is going to work is that

So, 它的工作方式是

28.41-28.45

this is going to be we're gonna maintain these undo records both in memory and on disk

在内存和磁盘中, 我们都要去维护这些Undo Record

28:46 - 28:51

and that way again if we crash while if the transaction is aborted, while we're running

如果当我们执行事务的时候, 事务被中止了

28.51-28.54

then if it's in memory we just go reverse things real quickly

如果这些Undo Record是放在内存中的，那我们就能很快将修改过的部分变回原来的值

28:54 – 28:57

but if Stuff written a disk and then we crash

但如果这些Undo Record是放在磁盘上的，接着我们遇上了崩溃

28:58 – 29:03

then we have our log records on disk that we can then load back in when we turn the database system on

当我们再打开数据库系统的时候，我们可以将这些保存在磁盘上的日志记录加载回来

29:03–29:06

and reconcile put us back in the correct state

并使我们回到正确的状态

29:07 – 29:11

so in a high level you can sort of think of the log as the black box and an airplane

So，从高级层面来讲，你可以将日志视作是飞机上的黑盒子

29:11–29:15

like if there's a major any airplane crash as a major crash

不管怎么说，飞机出事就是大事

29:15 – 29:20

but if an airplane crashes, what the government goes and looks at the black-Box

但如果飞机出了事，政府就会去查看飞机上黑盒子中的内容

29:20 – 29:20

all right

29:20–29:27

because that's gonna record information about what actually is you know what happened in the plane at the moment that it crashed

因为黑盒子里面记录了飞机出事时所有发生的事情

29:27 – 29:30

and then it tries to figure out what was the error what was the malfunction

接着，它会试着弄清楚错误是什么，故障又是什么

29:31 – 29:32

now in the in the airplane case

在这个飞机案例中

29:32–29:33

they can't put the airplane back together

他们没法将飞机恢复原状

29:33–29:34

and the database can

但数据库可以

29:34–29:35

okay so we can put it back together

So，我们可以将数据库恢复原状

29:35–29:37

all right that's what we're gonna use that form

这就是我们使用logging的原因所在了

29:38 – 29:45

so logging at a high level ,right will be used by almost everything we'll make database system that's out there

So, 从一个高级层面来讲, 我们所见到的所有数据库系统几乎都用到了logging

29:46 – 29:52

any database system that says that they're durable to disk ,chances are they're using logging

所有数据库都表示它们会通过使用logging的方式将数据持久化到磁盘

29:52 – 29:58

so in addition to you know having the ability to roll back things and guarantee atomicity

So, 此外, 为了能够做到回滚, 以及保证原子性

29:58 – 30:07

Logging gonna provide us additional benefits in terms of both performance ,and high level concept of high level criteria we may have for application on organization

Logging为我们提供了一些额外好处, 不管是在性能上, 还是在应用程序的组织结构上

30:08 – 30:09

So it's going to turn out that

So, 事实证明

30:09–30.11

when we start talking about logging

当我们开始讨论logging的时候

30.11–30.14

since no disk are expensive write to,

因为磁盘写入的成本很高

30.14–30.18

we can turn random writes into sequential writes through a log

通过日志, 我们可以将随机写入变成循序写入

30:18 – 30:20

all right and that'll make the system run faster

这会使得系统跑得更快

30:20 – 30:21

and then for other applications

对于其他应用程序来说

30.21–30.28

the log is actually essentially going to be, you know a audit trail every single thing your application did

实际上, 日志可以用来跟踪审计你应用程序所做的每一件事

30:29 – 30:31

then you can use that to figure out what was happening,

你可以通过日志来弄清楚之前发生了什么

30.31–30.36

if you ever have an audit or to have questions about know my application did this at this time

如果你对你应用程序在这个时间点所做的事情有疑问

30:36 – 30:41

because and then there was a breach what data I got read or what data I got written

我们可以通过日志来弄清楚我在这个时间点读取了什么数据, 以及写入了什么数据

30:42 – 30:44

so in a lot of financial companies

So, 在很多金融公司中

30.44–30.48

they have to maintain the the log that database system generate for the last seven years

他们必须维护过去七年间数据库系统所生成的日志

30.48–30.50

because of a government regulation

这是因为政府法规的缘故

30:50 – 30:51

So this is a good example

So, 这是一个很好的例子

30.51–30.57

where I can use the log for atomicity, but also get additional benefit from it

我可以通过日志来保证原子性, 并且也能从中获得其他好处

#### **Approach #2: Shadow Paging**

- DBMS makes copies of pages and txns make changes to those copies. Only when the txn commits is the page made visible to others.
- Originally from System R.

Few systems do this:

- CouchDB
- LMDB (OpenLDAP)

30:57 – 31:00

so the other approach to guarantee atomicity that's less common is called **shadow paging**

So, 另一种比较少见用来保证原子性的方案叫做shadow paging

31:01 – 31:04

and this is actually the example that I mentioned the beginning the class

实际上, 我在这节课开始的时候提到过这个例子

31.04–31.05

where I said for every single transaction

我说过, 对于每个事务来说

31.05–31.07

I'm gonna make a copy of the database file on disk

我会在磁盘上制作一份该数据库文件的副本

31:08 – 31:10

all my changes go to that copy

我会将我所有的修改都放在这个副本上执行

31.10–31.11

and then when my transaction commits

当我的事务提交的时候

31.11–31.16

I just swing a pointer and say this is now the master version

我只需要让指针指向这个副本即可, 并表示, 现在这个副本就是该数据的主版本

31:16 – 31:17

so that's essentially what shadow paging is

So, 本质上来讲, 这就是shadow paging所做的事情

31:18 – 31:21

but instead of copying the single file every single time

但我们无须每次都去复制一份文件

31.21–31.26

they'll just copy the individual pages that the transaction modifies when it runs

当事务运行的时候, 它们只需去复制该事务所修改的那些page

31:26 – 31:27

and then when the transaction commits again

接着, 当事务提交的时候

31.27-31.28

you swing a pointer, and say

你只需修改下指针指向的东西，并说

31.28-31.33

all right all of these shadow copy pages are now the master copy pages

所有这些影子副本page现在就是主副本page

31:33 – 31:35

So this is the one of the oldest ideas in database systems

So, 这是数据库系统中一个最为古老的思想

31.35-31.38

this was invented about IBM in the 1970s in system R

这是IBM在1970年代为System R所发明的东西

31:41 – 31:47

this turns out to be super slow ,and problematic for managing data on disk

事实证明，它的速度非常慢，而且在管理磁盘数据方面很有问题

31:47 – 31:49

and when IBM went to go build DB2

当IBM去构建DB2的时候

31.49-31.53

which is the second relational database system they built after system R

DB2是他们自System R之后构建的第二个关系型数据库系统

31:53 – 31:55

they didn't do any of this, they went with the logging approach

他们并没有去使用Shadow Paging这种方案，他们选择去使用Logging这种方案

31:56 – 32:01

because you end up with fragmentation,and it with unordered data sets,

因为你最终会面临磁盘碎片和无序数据集的问题

32.01-32.03

and it gets slower

这会让它的速度越变越慢

32:04 – 32:06

so as far as you know today

So, 就当下我们所知道的

32.06-32.13

the only two database systems that actually do this is shadow paging approach is

CouchDB ,and LMDB

唯二使用这种shadow paging方案的数据库系统就是CouchDB和LMDB

32:16 – 32:17

they say it's for performance reasons

他们表示，这是出于性能方面的原因

32.17-32.18

it's not that common

这并不常见

32.18-32.20

everyone else is gonna do is gonna do logging

其他所有人使用的都是Logging这种方案

32:22 – 32:24

so this is question, yes

有问题吗？请问



# ATOMICITY OF TRANSACTIONS

## Scenario #1:

→ We take \$100 out of Andy's account but then the DBMS aborts the txn before we transfer it.

## Scenario #2:

→ We take \$100 out of Andy's account but then there is a power failure before we transfer it.

*What should be the correct state of Andy's account after both txns abort?*

32:28 – 32:32

oh yes this one keep going this one

你问的是这张幻灯片上的内容吗？

32:37 – 32:40

So for this one is it's the same operation

So, 这个例子中, 它执行的操作其实是相同的

32.40–32.42

take money around account put in your account

即将钱从我的账户放入你的账户

**ATOMICITY OF TRANSACTIONS**

**Scenario #1:**  
→ We take \$100 out of Andy's account but then the DBMS aborts the txn before we transfer it.

**Scenario #2:**  
→ We take \$100 out of Andy's account but then there is a power failure before we transfer it.

*What should be the correct state of Andy's account after both txns abort?*

32:42 – 32:44

this is like we get a **aborted**

在第一个例子中, 我们的事务被中止了

32.44–32.46

like the user says abort my transaction

比如: 用户表示: 请终止我的事务

32:47 – 32:50

everything's still in memory how do I roll that back,  
所有的数据依然存放在内存中, 我该如何将它们回滚呢?

32.50–32.54

this is like a hard crash how do I come back from that  
这是一种hard crash, 我该如何从中恢复正常呢?

32:54– 32.56

and so the point I was trying to make here was

So, 此时我尝试要做的事情是

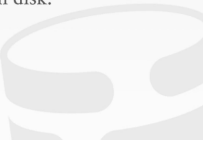
## MECHANISMS FOR ENSURING ATOMICITY

### Approach #1: Logging

- DBMS logs all actions so that it can undo the actions of aborted transactions.
- Maintain undo records both in memory and on disk.
- Think of this like the black box in airplanes...

Logging is used by almost every DBMS.

- Audit Trail
- Efficiency Reasons



32.56–33.02

the log information is gonna reside both in memory ,and eventually also get written out to disk

这些日志信息会放在内存中，并最终也会被写入磁盘

33:03 – 33:04

because if it's in memory

因为如果这些日志信息是在内存中的话

33.04–33.08

then I can quickly go get it and you know flip back the old buttons, right if I abort

如果事务中止了，那么我可以迅速从内存中获取到这些信息，并将数据变回原状

33:08 – 33:10

if I do a hard crash if it's on disk

如果我遇上的是hard crash，并且这些日志信息是在磁盘上

33.10–33.15

then I can reverse things potentially,right when I load the system backup

那么，当我加载系统备份的时候，我也可以将这些数据恢复原状

33:15 –33:16

because again after a hard crash

因为当经历了hard crash之后

33.16–33.18

all the contents of a buffer pool are gone

buffer pool中的所有内容就会消失不见

33:19 – 33:24

and we in you know we need to figure out what was happening at the system at the time crash to put us back in the correct state

你知道的，我们需要弄清楚系统在崩溃的时候发生了什么，以此来让数据回归正轨

33.24–33.25

,yes

请问

33:29 – 33:29

so this question is

So，这个问题是

33.29–33.31

does this require writing to disk reads transaction

这是否需要将读取事务也写入到磁盘

33.32–33.33

yes

没错

33:33 – 33:36

if you care about this okay if you care about not losing data,yes

如果你在意不丢数据的话，那么就将它写入到磁盘吧

33:37 – 33:38

we'll cover that later

我们会之后对此进行介绍

33:46 – 33:49

LMDB, this question is

他的问题是

33:49–33:53

~~is there any why would you ever~~ why you want to do this

为什么我们想去使用shadow paging

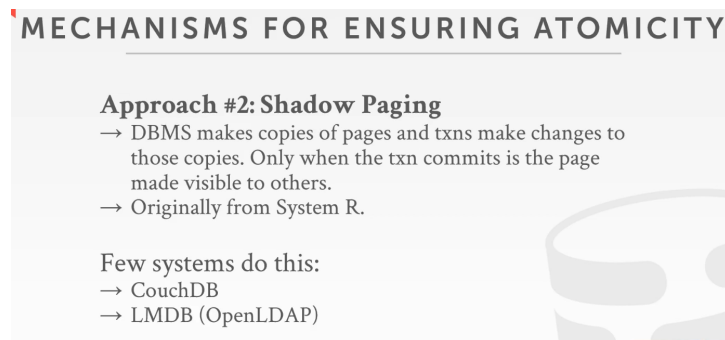
~~33:58 – 33:59~~

~~I'd, if you~~

34:03 – 34:04

it doesn't work

这不可行



**MECHANISMS FOR ENSURING ATOMICITY**

**Approach #2: Shadow Paging**

- DBMS makes copies of pages and txns make changes to those copies. Only when the txn commits is the page made visible to others.
- Originally from System R.

Few systems do this:

- CouchDB
- LMDB (OpenLDAP)

34:04–34:07

right so a few years ago

So, 在几年前的时候

34:07–34:15

my first my first PG student he and I started building a new system using like the new Intel non-volatile memory devices

我的第一个研究生学生和我一起使用了intel的新型非易失性内存设备来构建一个新系统

34:15 – 34:17

and we thought at the time

在那个时候我们思考了一下

34:17–34:23

that with really fast storage like non-volatile memory he's like almost as fast as DRAM  
非易失性内存的速度几乎和DRAM差不多快

34:23–34:26

with really fast storage to do random access

我们使用这种速度很快的存储设备来做随机访问

34:26 – 34:29

that shadow paging would actually turn out to be a better approach

事实证明，shadow paging这种方案其实更好

34:29–34:33

like taking an old idea from the 70s, and running on like today's hardware

这就像是将一个1970年代的想法，放在当下的硬件上进行实验

34:33 – 34:34

it doesn't work

这并不可行

34.34-34.35

write ahead logging always move faster

预写式日志的速度始终更快

34.35-34.37

because you can do these sequential writes

因为你可以进行这些循序写入操作

34.37-34.39

you know you can batch a bunch of things together

你知道的，你可以将一堆东西放在一起批量处理

34:39 - 34:41

and then shove amount all the disk at once

然后，将这些东西一次性写入磁盘

34.41-34.42

with shadow paging

如果使用shadow paging

34.42-34.42

it's all this fragmentation

就会产生磁盘碎片

34.42-34.46

you're copying things every single time,it becomes very expensive

你每次都要去复制些东西，这样成本就会变得非常昂贵

34:48 - 34:52

we'll see multi-version concurrency control which is sort of like this

我们之后会看到多版本并发控制，它有点像这个

34.52-34.58

but instead of copying an entire page before I make a change, I maybe just copy a tuple or a subset of the tuple

在我进行一次修改操作之前，我无须去复制一整个page，我可能只需要复制一个tuple或者一组tuple就可以了

34:58 - 35:03

so shadow paging is sort of how multiple version concurrency control works

So, shadow paging其实就是多版本并发控制的工作方式

35:03 - 35:08

but its shadow paging as defined by IBM is nobody does except for these guys

但IBM所定义的shadow paging其实并没有什么人使用，除了这两个

35.07-35.08

yes

请问

35:15 - 35:16

To their logging stuff

关于logging这方面？

35.16-35.20

yes so I don't spend too much time on running out the disk

So, 我不会在磁盘上面浪费太多时间

35:20 -35:20

but the question is

但这里的问题是

35.20-35.24

is it the case that I if I do much changes

假设我们的场景是这样的，如果我进行了大量的修改操作

35:24–35:28

I create some some undo undo records that are in memory

我在内存中创建了一些Undo Records（撤销记录）

35:28 – 35:31

but then I crash before it's written out the disk is that a problem

但在我们将它写回到磁盘中之前，然后我就发生了崩溃，这会造成问题吗？

35:32 – 35:32

no

并不会

35:32–35:54

because when I come back

因为当我恢复过来的时候

35:54–35:36

all my memories gone

我内存中的数据就全丢失了

35:37 – 35:42

and the therefore I'm going to load the database back up based on how it was on disk

因此，当我加载数据库的时候，我数据库中的内容取决于当前磁盘中它所保存的内容

35:42 – 35:44

and so because those changes never got persisted to disk

So，因为这些修改并没有落地到磁盘中

35:44–35:48

they're there as if they never happened

也就是说，这些修改并没有生效

35:49 – 35:52

so his question which is which is a good point is that

So，他所提的问题中有一个很不错的地方就是

35:52–35:56

~~do I have to do a sync~~ ~~is this mean have to do if I know~~ if I want to say my transaction is committed

如果我想说，我的事务已经被提交了

35:56–36:01

do I have to do a sync ,do I do a flush every single time my transaction commits

每当我事务提交的时候，我是否得进行同步并将它刷到磁盘中

36:01 – 36:02

and I answer is yes

我的答案是Yes

36:02–36:04

but you don't really do it on every single commit

但你不需要对每次提交都进行这种操作

36:04 – 36:06

you batch a bunch together

你可以将它们累积在一起

36:06–36:08

and then do a group commit when you flush them out all together

当你要将它们都刷出去的时候，你可以做一次组提交（Group Commit）

36:08–36:10

that advertises the Fsync costs over time

这就解决了Fsync所带来的开销问题

36:10 – 36:15

but if you dare if you want to guarantee that your data is actually durable

但如果你想确保你的数据已经被持久化了

36:15 – 36:16

You happen you have to write the disk

那你就得将它们写入磁盘

36:17 – 36:21

so but the tricky thing is gonna be in what order you write to disk,

但这里棘手的地方在于你将这些数据写入磁盘的顺序是什么

36:21–36:23

it's gonna matter a lot too

这点非常重要

36:23 – 36:28

so you have to make sure you write the log record that correspond to a change to a data page first, before you write the data page to disk

在你将该data page写入磁盘前，你需要先确保你写入的日志记录对应着该data page上的一次修改操作

36:28 – 36:31

we'll cover what's in a whole day on this as well

我们会花一整天来介绍这个

36:32 – 36:37

and at the point of ~~checking make~~ I made about like oh well the NoSQL guys don't always provide acid

我之前提到过，NoSQL那帮人不想去提供ACID这种特性

36:39 – 36:46

~~some of them would actually not even flush to disk~~ when if they had transactions they would not flush a disk exactly when you say you know complete my transaction

除非当我表示我的事务执行完毕了，那么他们才会将事务刷到磁盘上

36:47 – 36:49

they were sort of every do it every 60 seconds

他们会每隔60秒将数据刷回磁盘

36:49 – 36:50

so that means

So, 这意味着

36:50–36:52

you could crash and lose the last 60 seconds of data

如果你发生了崩溃，你也就只是丢掉最后60秒的数据

36:53 – 36:55

some systems were even worse than this

有些系统甚至比这还糟糕

36:55–36:57

I'll just say it straight up Mongo right

就比如说，MongoDB

36:57 – 37:00

the early version of Mongo is when you do a write

在早期版本的MongoDB中，当你执行一次写操作时

37.00–37.04

it would immediately come back ,and say yeah I got your write,but in actually being do the write

它可能会立即返回，并说：我拿到了你的写操作，但它实际上正在执行这个写操作

37:04 – 37:05

it's a network layer said

网络层表示

37:05–37:06

yeah I got it

我拿到了这个写操作

37:06–37:09

and if you wanted to make sure that your write actually occurred

如果你想确保你的写操作实际已经执行了

37:09–37:11

you have to come back a second time, and say did you actually do that

你必须回过头来去询问DBMS是否执行了这个写操作

37:11 – 37:13

that was the default for them for like four or five years

他们使用这种默认方案大概持续了四五年

37:13–37:15

and their early benchmark moments were amazing

他们早期的评测分数非常令人惊叹

37:15–37:19

because like they would do these writes, and of course it's like yeah I get it no problem

分数高的原因是这样的，比如说：它们会去执行这些写操作，然后Mongo表示我执行了这些写操作

37:19 – 37:22

right but didn't **actually do it**

但实际上它们并没有执行这些写操作

37:22 – 37:26

there's some Mongo fix that company fall anymore

Mongo已经修复了这个问题，现在不会再出现这种情况了

37:27 – 37:29

okay so any questions by atomicity

Ok，关于原子性这块，你们有任何疑问吗？

37:31 – 37:35

again what we'll cover that how we actually guarantee this in a second

我们稍后会介绍我们实际该如何保证原子性

## CONSISTENCY

The "world" represented by the database is logically correct. All questions asked about the data are given logically correct answers.

**Database Consistency**

**Transaction Consistency**

37:37 – 37:43

So consistency as I said before is this nebulous term about correctness of the database

So, 我之前讲过, 一致性是用来描述数据库正确性的一个模糊术语

37:44 – 37:47

so at a high level the way to think about this

So, 从一个高级层面来思考这一点

37:47–37:52

what a database actually is is trying to model some some concept or aspect of the real world

数据库实际上所试着做的事情就是对现实世界中的某些概念或者方面进行建模

37:53 – 37:59

like my my database for my bank is trying to model the old days of a bank where somebody would sit in a ledger

就比如我的银行数据库就试着对银行过去人工所做的事情进行建模

37:59 – 38:01

And record how much money you actually had in your account

并记录你账户中有多少钱

38:01 – 38:04

right it's modeling some some process in the real world

它对现实世界中某些过程进行建模

38:05 – 38:11

so we're gonna say that if we have our database be logically correct

So, 假设我们数据库是逻辑正确的

38:13 – 38:13

meaning

这意味着

38:13–38:15

we don't care has actually actually physically stored

我们无须在意物理存储实际是怎么样的

38:15–38:20

but the data integrity the referential integrity all those things are correct

但只要数据完整性和引用完整性之类的东西都是正确的就行了

38:20 – 38:25

then any questions we asked about that our database will produce correct results

那么, 当我们询问数据库任何问题的时候, 它都会给我们生成正确的结果

38:27 – 38:28

and again that sounds very vague

这听起来非常模糊

38:28–38:30

so let me go into more detail

So, 我们来深入了解下其中的细节

38:30–38:32

there's two types of consistency we could possibly have

这里我们有两种一致性

38:32 – 38:34

now database consistency and transaction consistency

即Database Consistency (数据库一致性) 和Transaction Consistency (事务一致性)

38:35 – 38:38

the spoiler would be database consistency is the one we actually care about

我们实际上关心的是数据库一致性



38.38–38.40

we can actually we can't do the second one,

实际上，我们没法做到第二点

38.40–38.42

and we'll see why in a second

我们稍后会看下为什么

=====

## 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.

38:43 – 38:47

so again the our **correctness** criteria is that

So，我们的正确性标准是

38.47–38.50

our database actually reflects what the real world looks like

我们的数据库实际反映了现实世界的样子

38:50 – 38:53

and so how do we actually enforce that

So，我们实际该如何强制做到这点呢？

38.53–38.55

well we provide the database system with integrity constraints

Well，我们会为数据库系统提供完整性约束（integrity constraint）

38.55–38.59

to say this is what it means to for us to have correct data

以此表示，这就是我们拥有正确数据的意义

这意味着，通过它，我们可以保证拥有正确的数据

38:59 – 38:59

so for example

So，例如

38.59–39.03

if I have a table of people or students

如果我有一张学生表

39.03–39.05

and I'm keeping track of their age

我可以跟踪他们的年龄

39.05–39.09

I can have an integrity constraint that says nobody's age could be less than zero

这里我有一个完整性约束，它表示所有人的年龄都不能小于0

39:09 – 39:11

all right there's no negative ages

这里的年龄都不能为负

39:12 – 39:14

and so the database system could enforce that

So, 数据库系统可以强制做到这点

39:14–39:15

it sometimes insert something with a negative age

有时候它所插入的记录中包含了一个值为负数的年龄

39:15–39:18

you can say that's you can't have that in the real world

你们会说，在现实生活中，不可能有这种情况发生

39:19 – 39:20

I can't let you insert that data

我不会让你插入这个数据

39:22 – 39:24

the other way to think about to also is that the

思考这个的另一个方式是

39:24–39:29

~~so now in addition to these have experience~~

39:29–39:32

now as transactions to start making changes to the database

因为事务现在开始对数据库中的数据进行修改

39:33 – 39:42

that any transaction that execute in the future should be able to see the changes the

correct changes that a transaction in the past made

任何未来执行的事务都应该能看到以前某个事务所做的正确修改

39:44 – 39:45

so what does that mean

So, 这意味着什么呢?

39:45–39:47

~~so if I of transaction say I want run transaction right now~~

So, 如果我想执行事务

39:48 – 39:50

and I make some change in the database

并且我对数据库进行了某种修改

39:50 – 39:54

if you now run a transaction one minute later long as nobody has overwritten my changes

如果你在一分钟后执行了事务1，在此之前，如果没有人覆盖我所做的修改

39:55 – 39:57

you should be able to see my updates

那么，在执行事务一的过程中，你应该能看到我所做的修改

40:00 – 40:01

so in a single node database

So, 在一个单节点数据库中

40:01–40:04

this is not that big you know this is not that big of a deal

这没什么大不了的