

并发控制

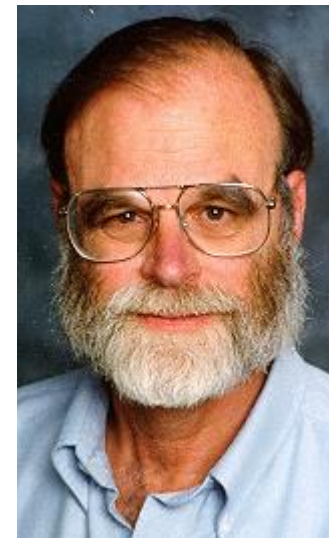


徐辰
华东师范大学
数据科学与工程学院
cxu@dase.ecnu.edu.cn

OLTP面对的问题：数据正确性


- 硬件失效
 - 宕机/停电
 - 硬件损坏
 - 灾难
- 软件错误
 - Bug
 - 恶意攻击
- 并发问题
 - 多个用户同时更新数据出现异常

事务的概念

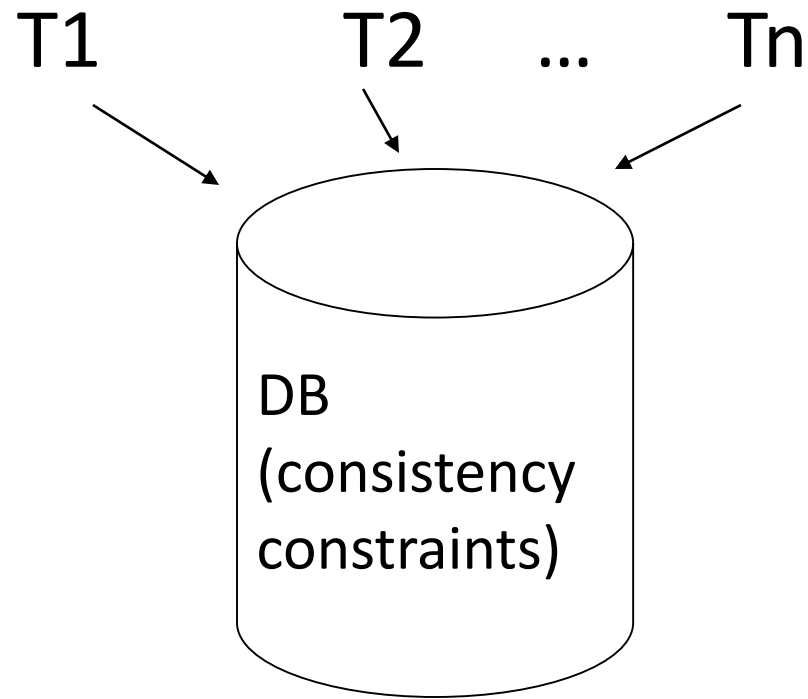


- ACID
 - 原子性 (Atomicity)
 - 一个事务 (transaction) 要么没有开始, 要么全部完成, 不存在中间状态。
 - 一致性 (Consistency)
 - 事务的执行不会破坏数据的正确性, 即符合约束。
 - 隔离性 (Isolation)
 - 多个事务不会相互破坏。
 - 持久性 (Durability)
 - 事务一旦提交成功, 对数据的修改不会丢失。

事务的概念

- ACID
 - 原子性 (Atomicity)
 - 一个事务 (transaction) 要么没有开始, 要么全部完成, 不存在中间状态。
 - 一致性 (Consistency)
 - 事务的执行不会破坏数据的正确性, 即符合约束。
 - 隔离性 (Isolation)  并发控制
 - 多个事务不会相互破坏。
 - 持久性 (Durability)
 - 事务一旦提交成功, 对数据的修改不会丢失。

挑战 – 并发控制



Example:

T1: Read(A)

$A \leftarrow A + 100$

Write(A)

T2: Read(A)

$A \leftarrow A \times 2$

Write(A)

Schedule A

T1	T2	A
Read(A);		25
$A \leftarrow A+100$	Read(A);	
Write(A);		125
	$A \leftarrow A \times 2$;	
	Write(A);	50
		50

并发控制：

DBMS调度器控制并发事务，
以保证数据库一致性的过程

实现技术：

- 封锁 locking
- 时间戳 timestamping
- 验证技术 validation

通过锁实现的调度控制

T1: Lock(A)

Read(A)

$A \leftarrow A + 100$

Write(A)

Unlock(A)

T2: Lock(A)

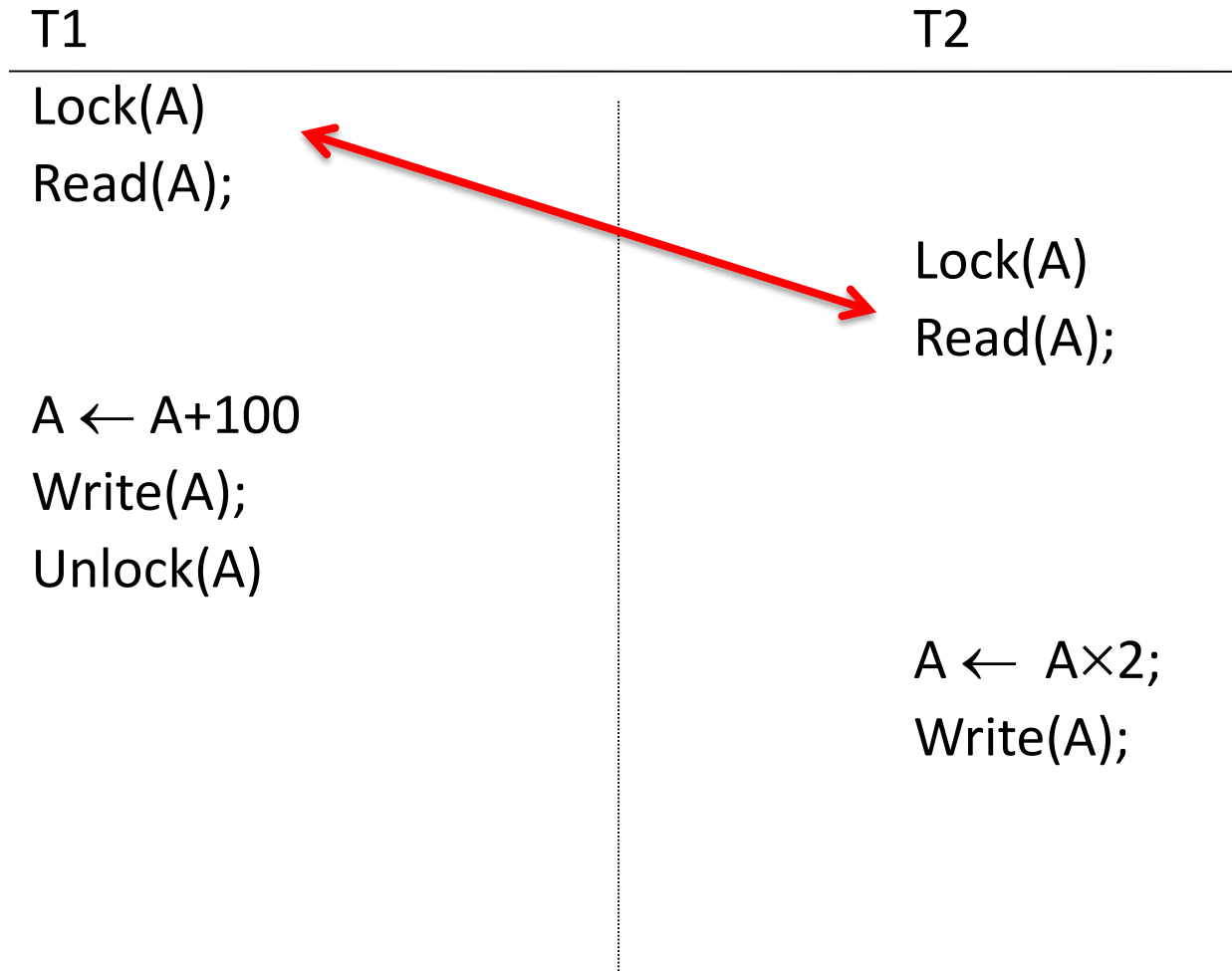
Read(A)

$A \leftarrow A \times 2$

Write(A)

Unlock(A)

加锁后, Schedule A 成为不可能的调度



加锁的正确方式是什么？

Example:

T1: Read(A)
 $A \leftarrow A+100$
 Write(A)
 Read(B)
 $B \leftarrow B+100$
 Write(B)

T2: Read(A)
 $A \leftarrow A \times 2$
 Write(A)
 Read(B)
 $B \leftarrow B \times 2$
 Write(B)

Constraint: $A=B$

Schedule A

T1

T2

Read(A); $A \leftarrow A+100$

Write(A);

Read(B); $B \leftarrow B+100$;

Write(B);

Read(A); $A \leftarrow A \times 2$;

Write(A);

Read(B); $B \leftarrow B \times 2$;

Write(B);

Schedule A

T1

T2

Read(A); $A \leftarrow A+100$
 Write(A);
 Read(B); $B \leftarrow B+100$;
 Write(B);

Read(A); $A \leftarrow A \times 2$;
 Write(A);
 Read(B); $B \leftarrow B \times 2$;
 Write(B);

A	B
25	25
125	125
250	250
250	250

Schedule B

T1

T2

Read(A); $A \leftarrow A+100$
Write(A);
Read(B); $B \leftarrow B+100$;
Write(B);

Read(A); $A \leftarrow A \times 2$;
Write(A);
Read(B); $B \leftarrow B \times 2$;
Write(B);

Schedule B

T1

T2

Read(A); $A \leftarrow A+100$
 Write(A);
 Read(B); $B \leftarrow B+100$;
 Write(B);

Read(A); $A \leftarrow A \times 2$;
 Write(A);
 Read(B); $B \leftarrow B \times 2$;
 Write(B);

A	B
25	25
50	50
150	150
150	150

Schedule C

T1

Read(A); $A \leftarrow A+100$
Write(A);

Read(B); $B \leftarrow B+100$;
Write(B);

T2

Read(A); $A \leftarrow A \times 2$;
Write(A);

Read(B); $B \leftarrow B \times 2$;
Write(B);

Schedule C

T1

T2

Read(A); $A \leftarrow A+100$

Write(A);

Read(B); $B \leftarrow B+100$;

Write(B);

Read(A); $A \leftarrow A \times 2$;

Write(A);

Read(B); $B \leftarrow B \times 2$;

Write(B);

A	B
25	25
125	
250	
	125
	250
250	250

Schedule D

T1

Read(A); $A \leftarrow A+100$

Write(A);

Read(B); $B \leftarrow B+100$;

Write(B);

T2

Read(A); $A \leftarrow A \times 2$;

Write(A);

Read(B); $B \leftarrow B \times 2$;

Write(B);

Schedule D

T1

T2

Read(A); $A \leftarrow A+100$

Write(A);

Read(A); $A \leftarrow A \times 2$;

Write(A);

Read(B); $B \leftarrow B \times 2$;

Write(B);

Read(B); $B \leftarrow B+100$;

Write(B);

A	B
25	25
125	
250	
	50
	150
250	150

可串行化调度 (Serializable schedule)

“正确性原则”：每个事务如果在隔离的情况下执行，将把任何一致的状态转换到另一个一致的状态。

并发事务的正确性原则：调度产生的结果与一次执行一个事务所产生的结果相同。

关键概念：冲突

- 冲突可串行化
 - 一个比可串行化更严格的条件
 - 商用系统中的调度器采用
- 冲突
 - 对于调度中一对连续的动作，如果它们的顺序交换，结果将改变。

关键概念：冲突

- 除以下操作外，其余皆冲突：

- $r_i(X); r_j(Y)$, 只读

- $r_i(X); w_j(Y)$, $X \neq Y$

- $w_i(X); r_j(Y)$, $X \neq Y$

- $w_i(X); w_j(Y)$, $X \neq Y$

- 冲突的条件:
 - 涉及同一个数据库元素
 - 并且至少有一个是写操作的动作

定义：

- 冲突等价的调度：

如果 S_1 能通过一系列的非冲突交换变成 S_2 ，则 S_1 、 S_2 是冲突等价的调度。

- 冲突可串行化：

一个调度是冲突可串行化的，如果它和某些串行调度是冲突等价的。

Example:

$S_c = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$

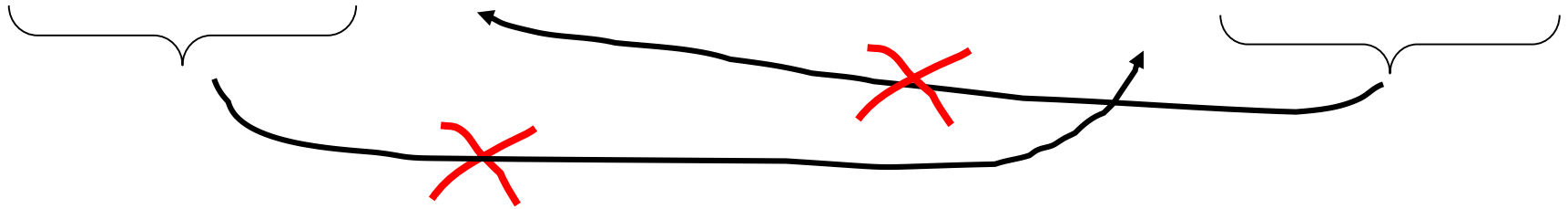
$S_{c'} = r_1(A)w_1(A)r_1(B)w_1(B)r_2(A)w_2(A)r_2(B)w_2(B)$

The diagram illustrates a transformation from a serial schedule S_c to a conflict-equivalent schedule $S_{c'}$. In S_c , the operations are grouped into two transactions: T_1 (operations on A) and T_2 (operations on B). In $S_{c'}$, the operations are reordered to maintain conflict equivalence. The red arrows indicate the movement of $r_2(A)$ and $w_2(A)$ from their original positions in S_c to their new positions in $S_{c'}$, where they are placed after the operations of T_1 .

S_c 是冲突等价的调度

Example:

$S_d = r_1(A)w_1(A)r_2(A)w_2(A) \ r_2(B)w_2(B)r_1(B)w_1(B)$



S_d 不是冲突等价的调度

如何实现事务的可串行化?

T1: Read(A)

$A \leftarrow A + 100$

Write(A)

Read(B)

$B \leftarrow B + 100$

Write(B)

T2: Read(B)

$B \leftarrow B \times 2$

Write(B)

Read(C)

$C \leftarrow C \times 2$

Write(C)

加锁?

T1: **Lock(A)**
 Read(A)
 $A \leftarrow A+100$
 Write(A)
 Unlock(A)
 Lock(B)
 Read(B)
 $B \leftarrow B+100$
 Write(B)
 Unlock(B)

T2: **Lock(A)**
 Read(A)
 $A \leftarrow A \times 2$
 Write(A)
 Unlock(A)
 Lock(B)
 Read(B)
 $B \leftarrow B \times 2$
 Write(B)
 Unlock(B)

加锁方式1:

T1: **Lock(A)**
 Lock(B)
 Read(A)
 $A \leftarrow A+100$
 Write(A)
 Read(B)
 $B \leftarrow B+100$
 Write(B)
 Unlock(B)
 Unlock(A)

T2: **Lock(A)**
 Lock(B)
 Read(A)
 $A \leftarrow A \times 2$
 Write(A)
 Read(B)
 $B \leftarrow B \times 2$
 Write(B)
 Unlock(B)
 Unlock(A)

加锁方式2:

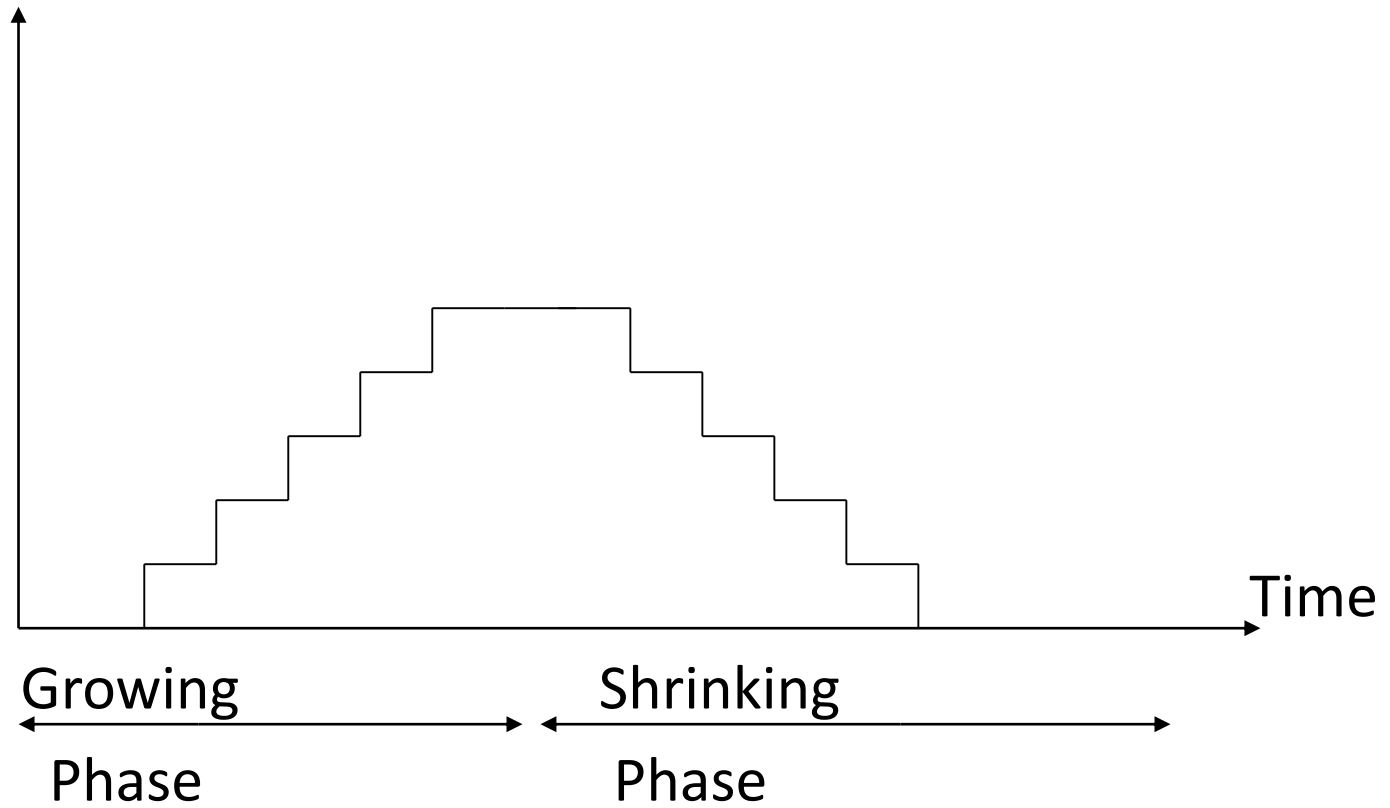
T1: **Lock(A)**
 Read(A)
 $A \leftarrow A+100$
 Write(A)
 Lock(B)
 Unlock(A)
 Read(B)
 $B \leftarrow B+100$
 Write(B)
 Unlock(B)

T2: **Lock(A)**
 Read(A)
 $A \leftarrow A \times 2$
 Write(A)
 Lock(B)
 Unlock(A)
 Read(B)
 $B \leftarrow B \times 2$
 Write(B)
 Unlock(B)

两阶段锁(2-Phase Locking)

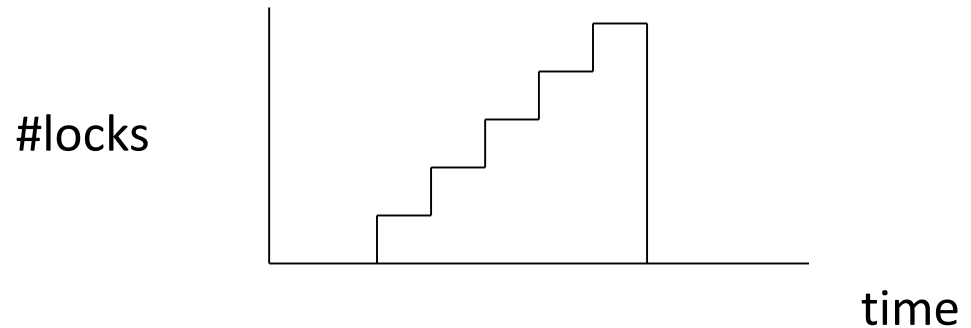
- 2PL条件：在每个事务中，所有封锁请求先于所有解锁请求
- 服从2PL条件的的事务：
 - 封锁扩展阶段
 - 解除封锁阶段

locks
held by
Ti

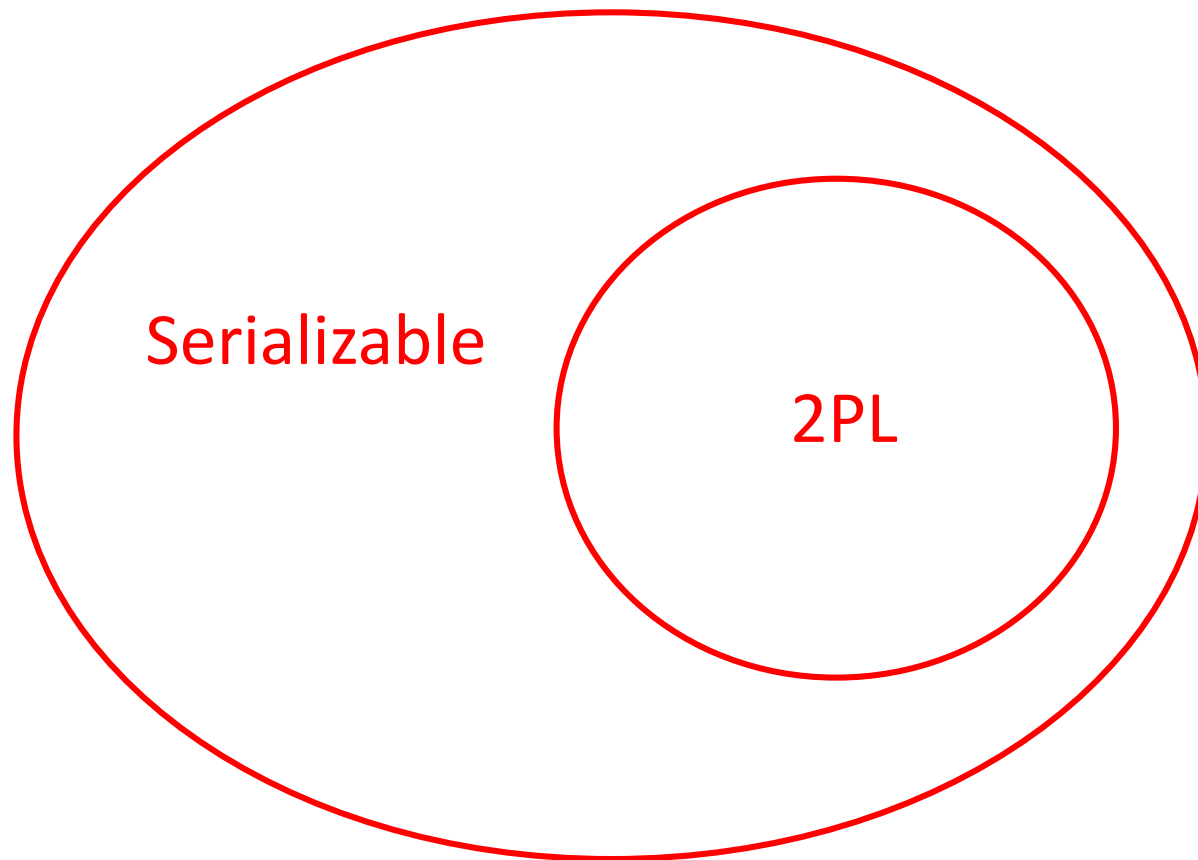


2PL通常的实现方式

- (1) 访问数据前申请锁;
- (2) 事务结束时将锁一齐释放。



两阶段锁协议是冲突可串行性的充分条件
两阶段锁协议不是冲突可串行性的必要条件



并发控制的宗旨

- 保证事务的正确性：
 - 最严格的要求：
可串行化。
- 争取更好的性能：
 - 减少事务间的冲突；
 - 减少死锁的概率。



两者的折中：

- 一方面依靠DBMS的内部实现；
- 另一方面依靠程序员的经验。

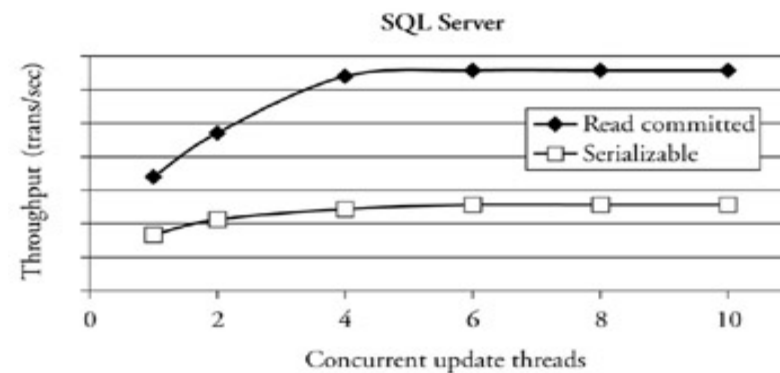
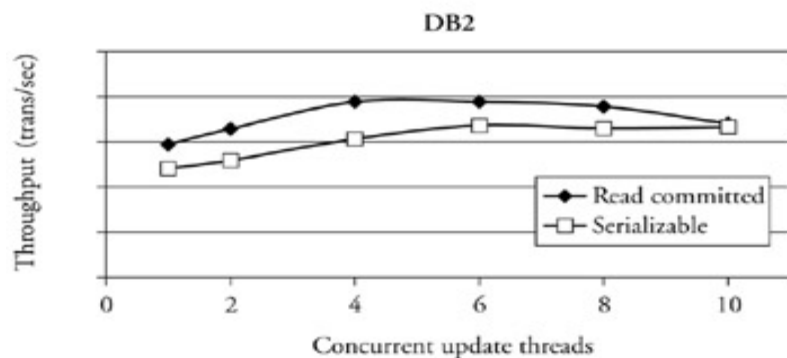
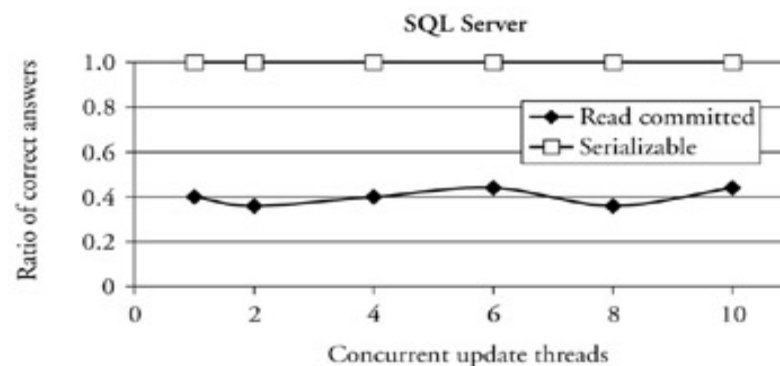
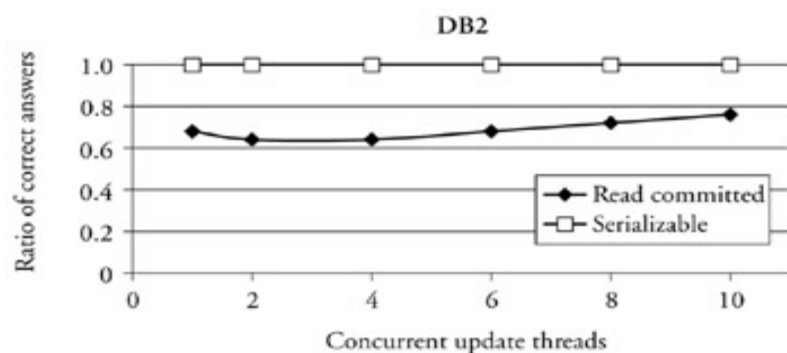
如何减小锁对性能的影响？

- 尽量少加锁。
- 多使用共享锁，少使用排它锁。
- 使用更细粒度的锁。
 - 行级锁 vs 表级锁
- 减少上锁的时间。

事务的隔离级别（从低到高）

- Read Uncommitted (No dirty writes, No lost update)
 - Exclusive locks for write operations are held for the duration of the transactions.
 - 脏读: No locks for read.
- Read Committed (No dirty reads)
 - Shared locks are released as soon as the read operation terminates.
 - 多次发出同一个查询可能得到不同的答案: unrepeatable reads
- Repeatable Read (no unrepeatable reads)
 - Strict two phase locking
 - 同一个查询第二次执行可能读到幻像phantoms元组（数据库插入新元组）
- Serializable (no phantoms)
 - Table locking or index locking to avoid phantoms.

不同隔离级别的性能差异

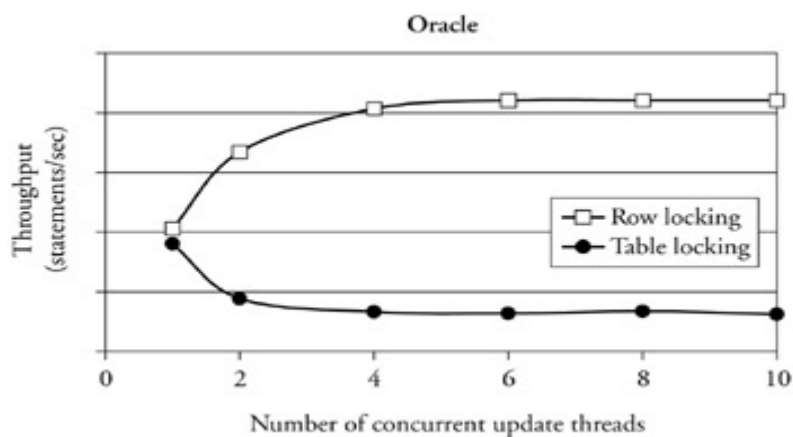
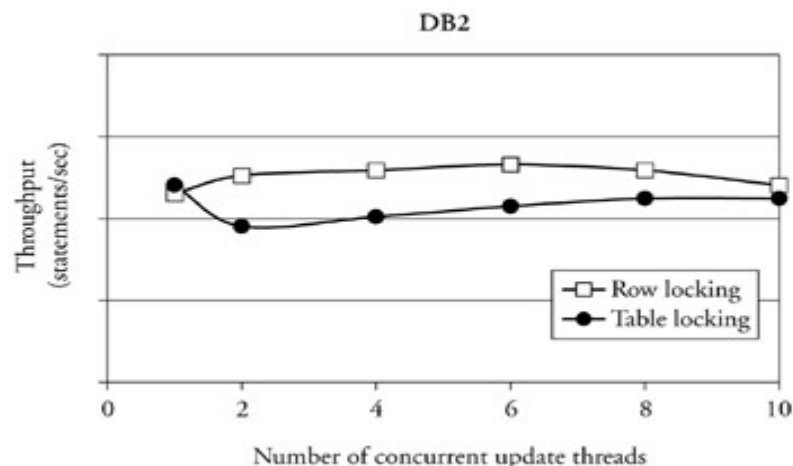
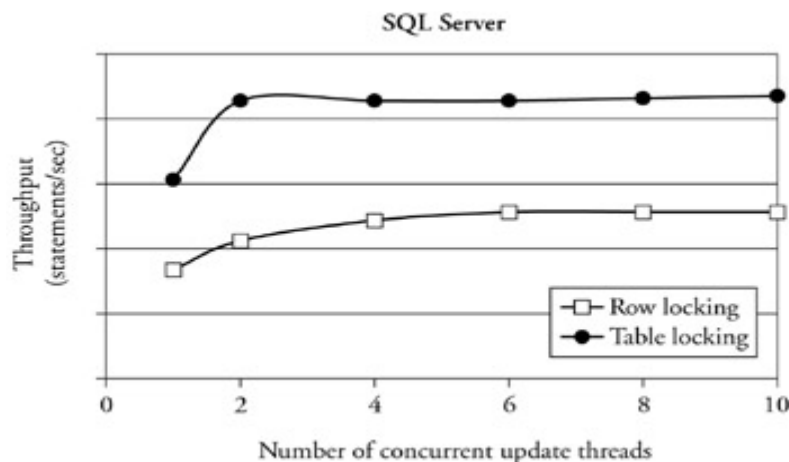


选择适当的隔离级别

- 大部分应用不需要太高的隔离级别。
 - 购物、转账
- 隔离性也可以在应用程序中控制
 - 在数据库中使用较低的隔离级别；
 - 通过应用程序的流程实现更高的隔离级别。

选择加锁的粒度

- 表级锁 vs 行级锁



事务1：全表统计

事务2：简单的插入和更新

越粗粒度的锁，冲突越多。

加锁以外的其它方式？

时间戳

T1	T2	T3	A	B	C
200	150	175	RT=0 WT=0	RT=0 WT=0	RT=0 WT=0
r1(B);				RT=200	
	r2(A);		RT=150		
		r2(C);			RT=175
w1(B);				WT=200	
w1(A);			WT=200		
	w2(C);				
	中止				
		w3(A);			

与RT=175逻辑上违背，
中止

与WT=200逻辑上也违背，
忽略但不中止

Validation 有效性验证

Transactions have 3 phases:

(1) Read

- all DB values read
- writes to temporary storage
- no locking

(2) Validate

- check if schedule so far is serializable

(3) Write

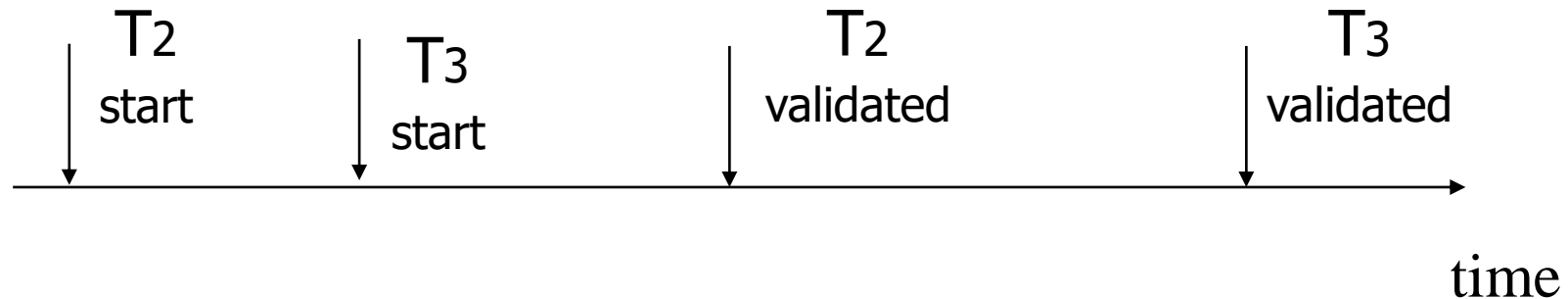
- if validate ok, write to DB

Key idea

- Make validation atomic
- If T_1, T_2, T_3, \dots is validation order, then resulting schedule will be conflict equivalent to $S_s = T_1 T_2 T_3 \dots$

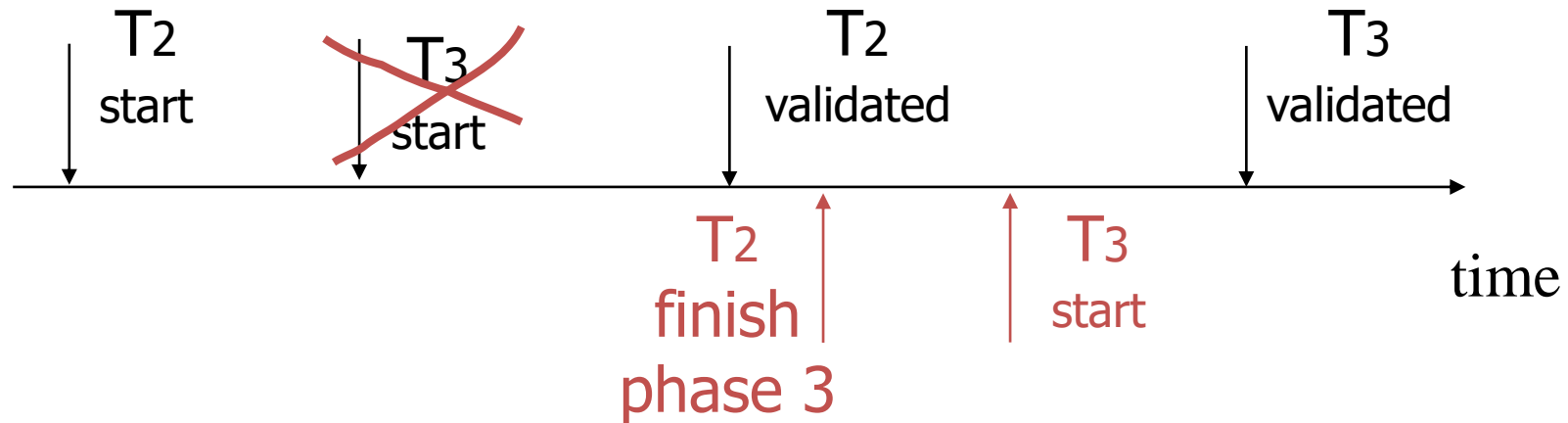
Example of what validation must prevent:

$RS(T_2) = \{B\}$ $RS(T_3) = \{A, B\}$ $\neq \phi$
 $WS(T_2) = \{B, D\}$ $WS(T_3) = \{C\}$



Example of what validation must prevent: ~~allow~~

$RS(T_2) = \{B\}$ $RS(T_3) = \{A, B\} \neq \phi$
 $WS(T_2) = \{B, D\}$ $WS(T_3) = \{C\}$



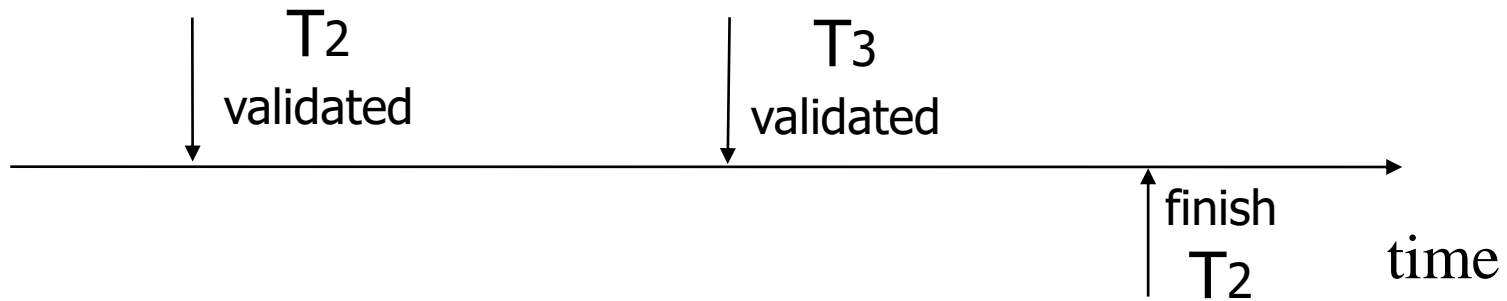
Another thing validation must prevent:

$RS(T_2) = \{A\}$

$RS(T_3) = \{A, B\}$

$WS(T_2) = \{D, E\}$

$WS(T_3) = \{C, D\}$



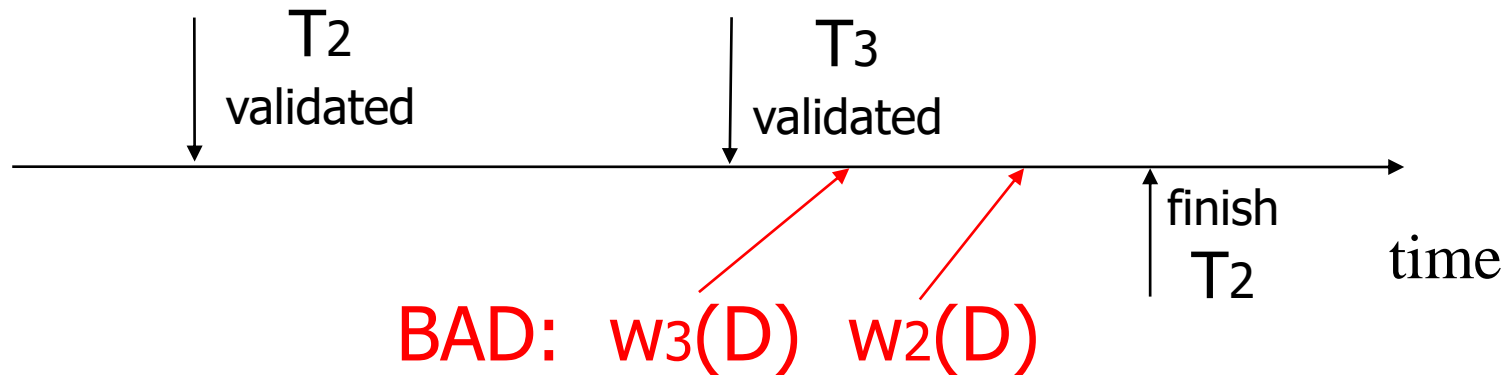
Another thing validation must prevent:

$RS(T_2)=\{A\}$

$RS(T_3)=\{A,B\}$

$WS(T_2)=\{D,E\}$

$WS(T_3)=\{C,D\}$



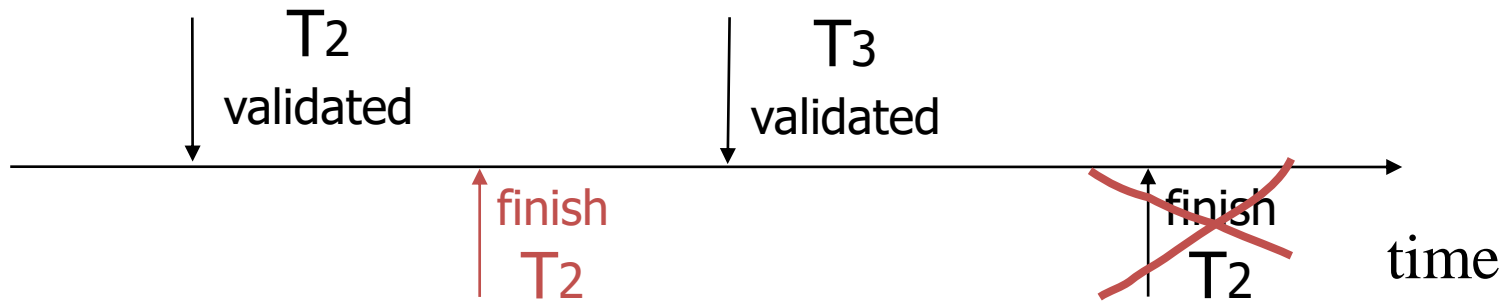
Another thing validation must prevent. ^{allow}

$RS(T_2) = \{A\}$

$RS(T_3) = \{A, B\}$

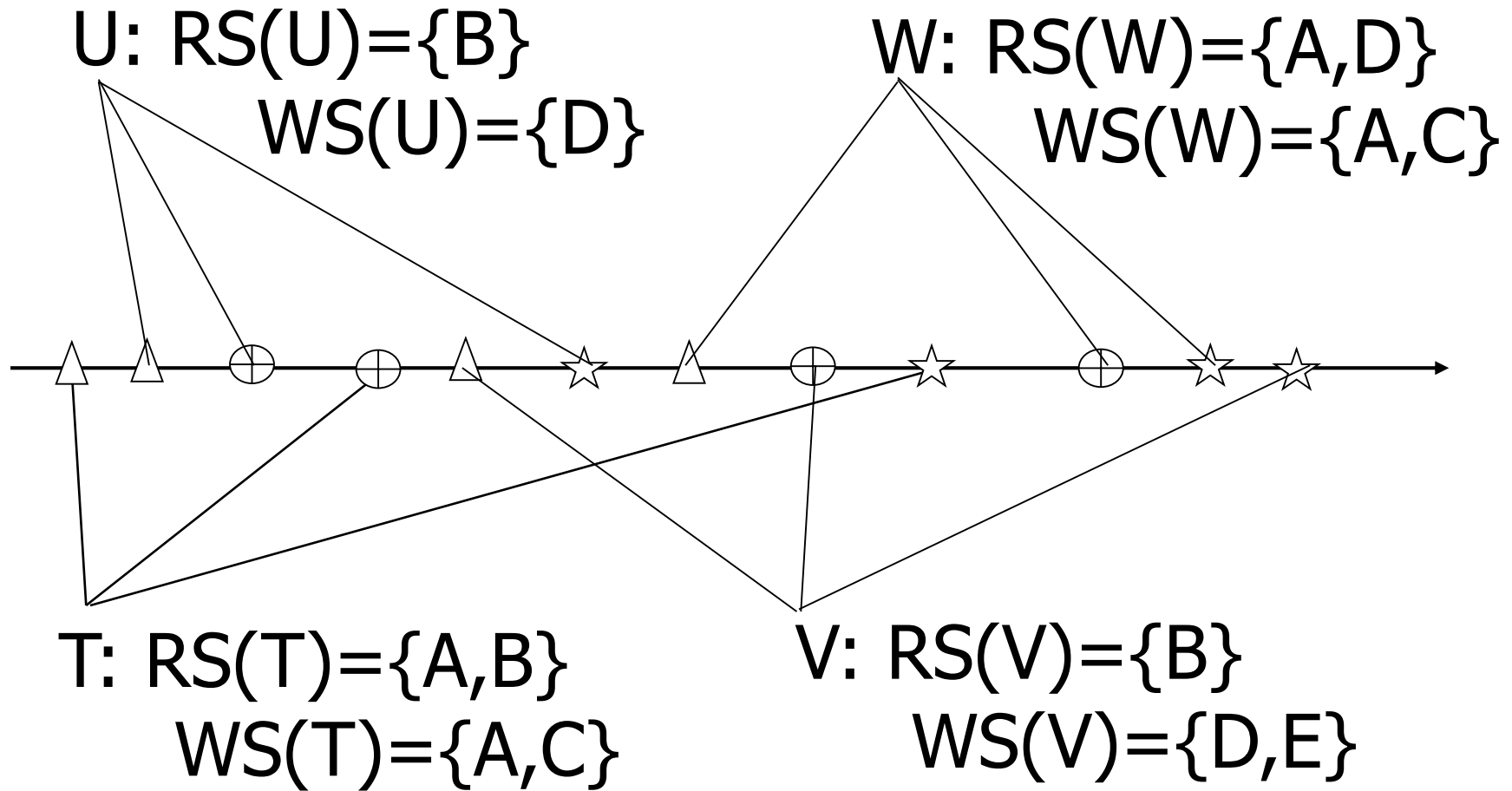
$WS(T_2) = \{D, E\}$

$WS(T_3) = \{C, D\}$

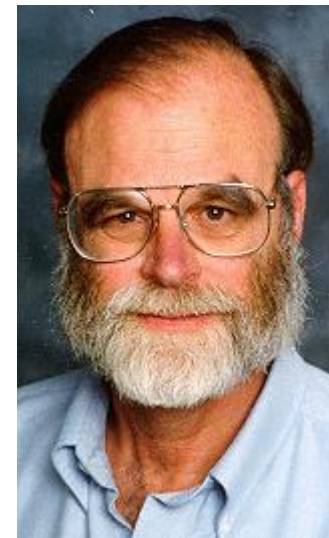


Exercise:

△ start
⊕ validate
☆ finish




回顾：事务的概念



- ACID
 - 原子性 (Atomicity)
 - 一个事务 (transaction) 要么没有开始，要么全部完成，不存在中间状态。
 - 一致性 (Consistency)
 - 事务的执行不会破坏数据的正确性，即符合约束。
 - 隔离性 (Isolation)
 - 多个事务不会相互破坏。
 - 持久性 (Durability)
 - 事务一旦提交成功，对数据的修改不会丢失。

日志

回顾：事务的概念

- ACID
 - 原子性 (Atomicity)
 - 一个事务 (transaction) 要么没有开始, 要么全部完成, 不存在中间状态。
 - 一致性 (Consistency)
 - 事务的执行不会破坏数据的正确性, 即符合约束。
 - 隔离性 (Isolation)  并发控制
 - 多个事务不会相互破坏。
 - 持久性 (Durability)
 - 事务一旦提交成功, 对数据的修改不会丢失。

关系数据库三大贡献

- 关系模型
- 查询优化
- 事务管理

从集中式走向分布式

- NoSQL systems: 打破关系模型
 - BigTable/Dynamo/Cassandra等
- Dataflow systems: 分布式查询处理
 - Hadoop/Spark/Flink等
- NewSQL systems: 分布式事务管理
 - H-store/MegaStore/Spanner等