软件工程学院数据库系统及其应用作业

实验课程：数据库系统及其应用　　**年级**：2023 级　　**姓名**：顾翌炜

作业编号：Week-14-15　　**学号**：10235101527　　**作业日期**：2025/06/11

1 冲突可串行化与优先图 (Conflict Serializability)

背景知识：一个调度（Schedule）是冲突可串行化的，当且仅当它的优先图（Precedence Graph）是无环的。优先图中，节点代表事务，如果事务 Ti 的一个操作与 Tj 的一个操作冲突，并且 Ti 的操作先于 Tj 执行，则在图中画一条从 Ti 指向 Tj 的边。

问题：给定以下包含事务 T1 和 T2 的调度 S

时间	T1	T2
1	read(A)	
2	A = A - 100	
3		read(A)
4		A = A * 1.1
5		write(A)
6	write(A)	
7	read(B)	
8		read(B)
9		B = B + 100
10		write(B)
11	B = B - 10	
12	write(B)	
13	commit	
14		commit

要求：

1) 找出调度 S 中所有的冲突操作对。
2) 画出该调度 S 的优先图（Precedence Graph）。
3) 判断该调度 S 是否是冲突可串行化的，并解释你的结论。如果可串行化，请给出一个与之等价的串行调度。

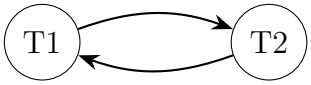
1 解答

冲突操作对

冲突操作的定义：两个操作属于不同事务，作用于同一数据项，且至少有一个是写操作（write）。

冲突操作对	冲突原因
(T1: read(A), T2: write(A))	T1 读 A 后，T2 写 A（时间顺序：T1(1) → T2(5)）
(T1: write(A), T2: read(A))	T2 读 A 后，T1 写 A（时间顺序：T2(3) → T1(6)）
(T1: write(A), T2: write(A))	T1 和 T2 都对 A 写（时间顺序：T2(5) → T1(6)）
(T1: read(B), T2: write(B))	T2 读 B 后，T1 写 B（时间顺序：T2(7) → T1(10)）
(T1: write(B), T2: read(B))	T2 写 B 后，T1 读 B（时间顺序：T2(8) → T1(12)）
(T1: write(B), T2: write(B))	T2 和 T1 都对 B 写（时间顺序：T2(10) → T1(12)）

优先图（Precedence Graph）



是否可串行化

由于优先图里存在环 T1 → T2 → T1，所以不可串行化

2 严格两阶段锁协议 (Strict 2PL)

背景知识: 两阶段锁协议 (2PL) 分为增长阶段（只加锁，不解锁）和缩减阶段（只解锁，不加锁）。严格两阶段锁协议 (Strict 2PL) 要求事务必须持有其所有排他锁 (Exclusive Lock) 直到事务提交或中止，这可以避免级联回滚。大多数数据库实现的都是严格两阶段锁协议。

问题: 假设数据库系统采用严格两阶段锁协议 (Strict 2PL)，并且锁管理器使用以下锁相容性矩阵：

现有两个事务 T1 和 T2：

1) T1: read(A), write(A), commit
2) T2: read(A), read(B), write(B), commit

(T1 持有)	S (共享锁)	X (排他锁)
(T2 请求) S	兼容 (Yes)	不兼容 (No)
(T2 请求) X	不兼容 (No)	不兼容 (No)

要求: 描述当 T1 和 T2 按以下顺序交叉执行时，锁的申请（lock-S/lock-X）、授予（granted）、拒绝/等待（denied/wait）和释放（unlock）的过程。请清晰地列出每一步操作和锁的状态。

时间	操作	T1 的动作	T2 的动作	锁管理器状态
1	T1	read(A)		
2	T2		read(A)	
3	T1	write(A)		
4	T2		read(B)	
5	T2		write(B)	
6	T1	commit		
7	T2		commit	

2 解答

时间	操作	T1 的动作	T2 的动作	锁管理器状态	说明
1	T1	read(A)		A: S(T1)	T1 获取 A 的 S 锁。
2	T2		read(A)	A: S(T1), S(T2)	S 锁兼容，T2 也获取 A 的 S 锁。
3	T1	write(A)		A: S(T2), X(T1) 等待	T1 需升级为 X 锁，但 T2 持有 S 锁，不兼容。T1 阻塞，等待 T2 释放 S 锁。
4	T2		read(B)	A: S(T2), X(T1) 等待；B: S(T2)	T2 获取 B 的 S 锁。
5	T2		write(B)	A: S(T2), X(T1) 等待；B: X(T2)	T2 升级 B 的锁为 X 锁。
6	T1	commit		A: S(T2), X(T1) 释放；B: X(T2)	T1 提交后仅释放自己的 S(A)
7	T2		commit	A: 无锁；B: 无锁 (全释放)	T2 提交后释放所有锁

3 使用日志进行故障恢复 (Recovery with Log)

背景知识: 现代数据库普遍采用预写日志（Write-Ahead Logging, WAL）策略进行故障恢复。当系统崩溃后，恢复管理器通过分析（Analysis）、重做（Redo）和撤销（Undo）三个阶段来恢复数据。

1) Redo 阶段: 重放所有已提交事务以及未完成事务的更新，以确保已提交的修改被持久化。
2) Undo 阶段: 回滚所有在崩溃时仍处于活动状态（未提交或未中止）的事务，以保证事务的原子性

问题: 给定以下一份简化的数据库日志，系统在日志末尾处发生崩溃。

LSN	日志记录	描述
01	<T1, BEGIN>	T1 开始
02	<T2, BEGIN>	T2 开始
03	<T1, A, 10, 20>	T1 将 A 从 10 改为 20
04	<T3, BEGIN>	T3 开始
05	<T2, B, 50, 60>	T2 将 B 从 50 改为 60
06	<T1, COMMIT>	T1 提交
07	<CHECKPOINT>	检查点
08	<T2, C, 80, 90>	T2 将 C 从 80 改为 90
09	<T3, D, 40, 30>	T3 将 D 从 40 改为 30
10	<T2, COMMIT>	T2 提交
11	<T3, A, 20, 10>	T3 将 A 从 20 改为 10
	<– 系统崩溃 –	

要求: 根据上述日志，描述在系统重启恢复期间：

1) 哪些事务需要被 Undo?
2) 哪些日志记录 (根据 LSN) 需要被 Redo?
3) 简要说明为什么 T1 事务不需要任何操作。

3 解答

根据日志记录和崩溃时间点（LSN 11 之后），各事务的状态如下：

1) T1: 已提交（LSN 06）。

2) T2: 已提交（LSN 10）。
3) T3: 未提交（无 COMMIT 记录，最后操作在 LSN 11）。

UNDO 和 REDO

1) **UNDO：**
   所以需要被 undo 的事务是：T3 唯一在崩溃时仍活跃（未提交）的事务，其修改必须回滚
   首先 undo LSN 11（<T3, A, 20, 10>）：将 A 从 10 改回 20。
   再 undo LSN 09（<T3, D, 40, 30>）：将 D 从 30 改回 40。
2) **REDO：**
   T1 已经提交，T3 需要 undo，只有 T2 需要 redo，而检查点之后只有 LSN 08 是 T2 的，所以需要 redo T2 的 LSN 08

T1 不需要操作的原因

1) T1 在检查点之前已提交（LSN 06），且检查点（LSN 07）会确保其修改已持久化到磁盘。
2) 恢复时无需对已提交且检查点确认的事务执行任何操作（既不需要 Redo，也不需要 Undo）

4 死锁检测与处理 (Deadlock Detection)

背景知识: 当多个事务循环等待对方持有的锁时，就会发生死锁。数据库管理系统可以通过维护一个等待图 (Waits-for Graph) 来检测死锁。如果等待图中存在一个环，则表示系统存在死锁。

问题: 考虑以下两个事务 T3 和 T4 的执行片段：

时间	T3	T4
1	lock-X(B)	
2	...	
3	...	lock-S(A)
4	...	...
5	...	lock-S(B)
6	lock-X(A)	...

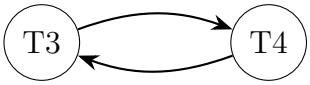
要求:

1) 画出在时间点 6 时，系统的事务等待图 (Waits-for Graph)。
2) 判断此时系统是否处于死锁状态，并解释原因。
3) 如果存在死锁，请提出一种解除该死锁的方法。

4 解答

事务	已持有的锁	正在请求的锁	阻塞原因
T3	X(B)	X(A)	T4 持有 S(A)，不兼容
T4	S(A)	S(B)	T3 持有 X(B)，不兼容

等待图



等待图中存在环（T3 T4），表示事务互相等待对方释放锁，形成循环依赖，也就是死锁。

解决办法：

1) 选择牺牲者（Victim）：
   - 根据优先级、已执行时间或修改量选择一个事务中止（如回滚 T3 或 T4）。
   - 例如：中止 T4（因其可能持有较少的锁或执行时间较短）。
2) 回滚牺牲者：
   - 释放牺牲者持有的所有锁（本例中若中止 T4，则释放 S(A)）。
   - T3 随后可获取 X(A)，继续执行。
3) 重启事务：
   - 牺牲者（T4）稍后重新启动执行。