事务

2025年6月9日 星期一 19:19

6、该并行操作是否等价于串行T4T5？为什么？

T4	T5
read(A)	
	read(B)
	B=B-50
	write(B)
read(B)	
	read(A)
Display(A+B)	
	A=A+50
	write(A)
	Display(A+B)

不等价，因为 write(B) 和 read(B) 产生冲突操作

6.是否冲突等价串行调度T1T2，为什么

	T1	T2
Time1	Read(B)	
Time2		Read(B)
Time3		B=B-50
Time4		Write(B)
Time5	Read(A)	
Time6		A=A+50
Time7		Write(A)
Time8	Display(A+B)	
Time9		Display(A+B)

无环，：存冲突等价串行化的

6、给出一个调度(TA,TB,TC)，写出它在时间戳排序协议下的调度结果，为什么？

TA（TS=1）	TB（TS=2）	TC（TS=3）
READ（A）		
WRITE（A）		
	READ（A）	
	WRITE（A）	
		READ（B）
		WRITE（B）
READ（B）		
WRITE（B）		
		READ（A）

R-ts(A) W-ts(A) R-ts(B) W-ts(B)

1

1

2

2

3

3

Rollback

3

6、数据项Q的W-timestamp(Q)=R-timestamp(Q)=20，现有事务Ta和事务Tb，TS(Ta)=30,TS(Tb)=34，Tb先执行read(Q)，Ta再执行Read(Q)，问最后R-timestamp(Q)为多少，为什么。

Ta (TS: 30) Read(Q)

Tb (TS: 34) Read(Q)

R-ts(Q) 34 34

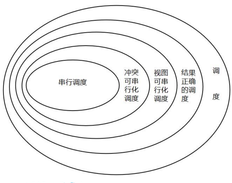
W-ts(A) 20 20

4、 事务的概念及其特点

1.举例说明两阶段封锁协议的加锁过程

1.请给出这五个调度的包含关系

1.正确的调度
2.串行调度
3.可串行化调度
4.满足两阶段封锁协议的调度
5.满足严格两阶段封锁协议的调度



（PPT上的图）

满足严格两阶段封锁协议的调度 C 满足两阶段封锁协议的调度 C 可串行化调度
C 正确的调度 C 所有可能的调度
串行调度 C 可串行化 调度

```
所有可能的调度
|
└── 正确的调度
    |
    └── 可串行化调度
        ├── 串行调度
        └── 满足2PL的调度
            └── 满足严格2PL的调度
```

6.利用银行转账的例子解释事务的原子性

原子性就是“要么全做，要么全不做”。

例子：

你要从A账户转200元到B账户，这需要两步：
从A账户扣200元。
给B账户加200元。
原子性的作用：

如果成功：A的钱扣了，B的钱也到账，转账完成。
如果失败（比如系统崩溃）：
A的钱不会扣，B的钱也不会加，就像什么都没发生过。
不会出现“A的钱扣了，但B没收到”的情况。
为什么重要？

保证钱不会凭空消失或重复！
如果转账中途出错，数据库会自动回滚，恢复成转账前的状态。
一句话总结：
原子性 = 转账必须完整成功，否则完全取消，绝不半途而废！

1. 一个事务执行到一半时，由于外部因素导致系统宕机，那么在重新启动后应当如何做才能保证事务的一致性。

1，恢复机制的核心：日志（WAL - Write-Ahead Logging）

数据库自修改数据前，会先记录**事务日志**（包括操作内容、事务ID、数据修改前后的值等）。
宕机后的恢复流程：

1. **检查日志**：找到所有未完成的事务（未提交或未回滚）。
2. **回滚（UNDO）**：撤销所有未提交事务的修改（恢复到事务前的状态）。
3. **重做（REDO）**：重新执行已提交但未写入磁盘的事务（确保持久性）。

1、为什么使用延迟修改的日志记录不需要记录数据的旧值？

2、说明串行调度和可串行化调度的区别。

延迟修改（Deferred Modification）是一种事务执行策略，其核心特点是：

1. **事务执行期间不直接修改数据库**，而是先记录所有修改操作到日志。
2. **直到事务提交时**，才真正将修改应用到数据库。

不需要记录旧值的原因：

1. **事务未提交时，数据库不会被修改**
    1. 如果事务失败/或系统崩溃，由于数据库被实际修改，直接丢弃日志即可，无需回滚（UNDO）。
    2. 只有提交的事务才会生效，因此无需记录旧值来恢复。
2. **恢复时仅需REDO（重做）**
    1. 崩溃后，只需检查日志中已提交的事务，并重新执行这些操作（REDO）。
    2. 未提交的事务无需处理（因为没有实际修改数据）。

对比立即修改（Immediate Modification）：

1. 立即修改需要记录旧值（UNDO日志），因为事务执行中可能直接修改数据库，崩溃后需要回滚未提交的修改。
2. 延迟修改简化了恢复机制，只需关注已提交的事务。

2. 串行调度 vs. 可串行化调度

（1）串行调度（Serial Schedule）

1. **定义**：多个事务**严格按顺序执行**（一个做一个，无并发）。
2. **特点**：
    1. 无并发冲突，绝对保证一致性。
    2. 性能差（无法利用多核/并行资源）。

（2）可串行化调度（Serializable Schedule）

1. **定义**：**并发执行**的事务调度，但结果**等价于某一串行调度**。
2. **特点**：
    1. 允许并发（提高性能），但通过冲突可串行化（Conflict Serializability）或视图可串行化（View Serializability）保证结果正确。
    2. 数据库通过**锁、时间戳、MVCC**等机制实现。

5、证明为什么两阶段封锁协议能够保证事务集合的可串行化。

✅ 五、2PL 保证冲突图无环 —— 关键证明点

我们来**反证**：

> 假设有一组遵循 2PL 的事务，它们的调度产生了一个**有环的冲突图**。

令环为：T1 → T2 → … → Tn → T1

这意味着：

- T1 与 T2 有冲突操作，且 T1 的冲突操作先发生 → 所以 T1 必须在 T2 **释放锁**之前加锁（因为冲突前先操作）
- 同理，T2 必须在 T3 加锁之前释放某锁，以此类推……
- 最终 Tn 要在 T1 释放锁之前加锁

这就构成一个**逻辑矛盾**：

> T1 加锁 < T2 加锁 < … < Tn 加锁 < T1 加锁 → 环

说明这些事务必须在彼此释放锁之间插入加锁操作，**违反了 2PL 的限制（不能在释放锁之后加锁）**。

因此：

> ❗**如果所有事务遵循 2PL，就不可能产生有向环的冲突图。**