

Read Committed 丢失更新



****丢失更新****问题通常是由以下场景引发的：1. ****事务 A**** 和 ****事务 B**** 都读取了同一数据。2. ****事务 A**** 更新了该数据，但未提交。3. ****事务 B**** 在此时也对相同数据进行了更新（由于没有阻止事务 A 的未提交操作），并提交了更改，导致事务 A 的更改丢失。

在这个例子中，假设事务A先更新数据，称该数据为C，但未提交，由于事务B不能读取未提交的数据，那事务B不是不能读取C吗？



你说得对，在 **Read Committed** 隔离级别下，**事务 B** 不能读取 **事务 A** 尚未提交的数据。因此，事务 B 不会读取到事务 A 更新的数据（C）。然而，**丢失更新** 这个问题发生的原因实际上是事务 A 和事务 B 都修改了同一数据，并且都以各自的修改提交了，但并没有发生显式的冲突或检查。

让我们更精确地分析这个场景：

1. 事务 A 和 事务 B 都对同一数据 C 进行更新，但未提交

- 假设数据 C 初始值为 100。
- 事务 A 读取了 C，并更新了 C（例如，事务 A 将 C 从 100 改为 150），但此时没有提交。
- 此时，事务 B 也读取了 C，并且更新了 C（例如，事务 B 将 C 从 100 改为 200）。此时，事务 B 提交了其修改。
- 当事务 A 提交时，C 会变成 150（事务 A 的更改），但事务 B 的修改（将 C 设置为 200）会覆盖掉事务 A 的修改，导致事务 A 的更新丢失。

2. 为什么这个问题存在

在 **Read Committed** 隔离级别下，事务 B **不能** 读取事务 A 尚未提交的更新。这样，事务 B 在开始时读取到的 C 仍然是最初的值 100。但问题在于，**事务 B 没有看到事务 A 更新的数据，也没有相应的锁机制防止它修改数据**。事务 A 和事务 B 分别更新了 C，但它们相互之间并没有协调或同步，导致事务 A 的更新丢失。

3. 解决方案

Read Committed 隔离级别下没有明确的机制来防止这种丢失更新的情况。为了避免丢失更新，可以考虑以下几种方法：

- **悲观锁 (Pessimistic Locking)**：事务 A 可以在更新数据之前加锁数据，阻止其他事务对数据进行修改。这种方式会导致事务 B 等待事务 A 提交，从而避免冲突。
- **乐观锁 (Optimistic Locking)**：事务 A 在读取数据时记录数据的版本号或时间戳，更新数据时检查版本号或时间戳是否发生变化。如果发生变化，说明其他事务已经修改了数据，事务 A 就可以回滚或重新执行。

- **使用更高的隔离级别（如 Repeatable Read 或 Serializable）**：这些隔离级别通过加锁或其他方式，避免了并发事务对相同数据的修改，确保事务 A 和事务 B 的修改不会冲突。

总结

在 **Read Committed** 隔离级别下，事务 B 不能读取事务 A 未提交的数据，但这并不意味着丢失更新问题就得到了解决。丢失更新的发生是因为事务 A 和事务 B 都能对相同的数据进行修改，而没有进行足够的协调。因此，要防止丢失更新，必须采用 **悲观锁、乐观锁 或 更高的隔离级别**。