Read Committed 丢失更新

好的，我明白了，你希望我更清晰地解释 **Repeatable Read** 隔离级别是如何解决 **丢失更新** 问题的。让我再尝试用更直观的方式来重新解释。

丢失更新是什么？

在并发事务中，**丢失更新** 指的是两个事务同时更新同一条数据时，**一个事务的更新被另一个事务的更新覆盖掉**。简单来说，就是有一个事务的修改被另一个事务的提交“替代”了，导致这个修改失效。

举个例子：

1. 假设数据 C 的初始值是 100。
2. **事务 A** 读取数据 C，看到它是 100，然后决定将 C 更新为 150，但还没有提交。
3. **事务 B** 也读取了数据 C，看到它也是 100，然后决定将 C 更新为 200，并提交。
4. **事务 A** 提交时，它把 C 更新为 150。结果是 C 最终的值是 150，而事务 B 更新的数据 200 被覆盖掉了。

这里的丢失更新是指：事务 A 的修改（150）在提交后覆盖了事务 B 的修改（200），导致事务 B 的修改丢失。

Repeatable Read 隔离级别是如何工作的？

在 **Repeatable Read** 隔离级别下，事务在执行期间会确保**读取的数据不会被其他事务修改**，也就是说，当一个事务读取数据时，其他事务不能修改这条数据，直到该事务提交。

Repeatable Read的特性：

- 当 **事务 A** 读取数据 C 时，它会“锁定”这条数据，直到事务 A 提交。
- 事务 A 在整个执行过程中都会看到相同的数据（即读取时的值），即使其他事务更新了数据，事务 A 也无法看到这些更新。
- 如果 **事务 B** 试图更新数据 C，它需要等到事务 A 提交之后才能操作数据。

Repeatable Read 如何避免丢失更新？

假设有两个事务 A 和 B，它们都想修改同一条记录 C。我们来看看 **Repeatable Read** 隔离级别是如何解决丢失更新问题的。

场景1：事务 A 和 B 都想更新数据 C

1. **事务 A** 读取数据 C（C = 100），并决定将其更新为 150。
2. **事务 B** 也读取数据 C（C = 100），并决定将其更新为 200。
3. 在 **Repeatable Read** 隔离级别下，事务 A 和事务 B 在读取数据时都不能看到对方的修改。具体来说：
    - 事务 A 读取数据 C 后，在整个事务执行期间，它将始终看到 C 的值为 100。
    - 事务 B 读取数据 C 后，也始终看到 C 的值为 100，直到它提交。

场景2：锁定数据

- 由于 **Repeatable Read** 会为事务 A 锁定数据 C（当事务 A 读取数据 C 时，它就锁住了这条记录），**事务 B** 必须等到事务 A 提交后才能修改数据 C。这样，**事务 B 无法覆盖事务 A 的修改**，因为事务 A 锁住了数据 C。

场景3：事务提交

- 当 **事务 A** 提交时，数据 C 会被更新为 150。此时，事务 B 仍然在等待事务 A 提交。
- 如果 **事务 B** 提交时，它会检测到数据已经被事务 A 修改（如果使用了合适的锁或版本控制），并且不会再对数据 C 进行修改，避免了 **丢失更新**。

简化理解：

在 **Repeatable Read** 隔离级别下：

- 当 **事务 A** 读取了数据并开始修改时，其他事务（例如事务 B）不能在 **事务 A 提交之前**修改数据。这意味着事务 A 和事务 B 无法同时修改同一数据。
- 如果 **事务 A** 和 **事务 B** 都想修改数据，它们会按照一定顺序排队执行，而不是直接覆盖对方的更新。

为什么 Repeatable Read 可以解决丢失更新问题？

- **Repeatable Read** 隔离级别确保了当事务 A 读取数据时，其他事务 B 在事务 A 提交之前无法修改该数据。
- 这避免了一个事务（比如事务 B）的修改覆盖另一个事务（事务 A）的修改，确保了两个事务的修改不会互相丢失。
- 这就是它如何避免丢失更新的原理。

总结：

- **丢失更新** 是指两个并发事务同时更新同一数据，其中一个事务的更新被另一个事务的更新覆盖，导致数据丢失。
- **Repeatable Read** 隔离级别通过锁定数据，使得在一个事务修改数据时，其他事务无法修改相同的数据，避免了事务之间的冲突。
- 这样，事务 A 和事务 B 即使都想修改相同的数据，**Repeatable Read** 会确保其中一个事务提交时，另一个事务必须等到它提交后才可以修改数据，从而防止了丢失更新的发生。

希望这次解释能让你更清楚地理解 **Repeatable Read** 是如何避免 **丢失更新** 的！