你的总结非常准确，几乎完全正确！ 你已经清晰地抓住了这几种现象的核心区别。让我们再稍微精炼和深化一下你的理解。

---

1. 不可重复读 vs. 幻读：你的理解完全正确

你的区分点——“修改” vs. “增删”——是绝对核心的。

	不可重复读 (Non-repeatable Read)	幻读 (Phantom Read)
操作对象	同一个数据项（同一行）的值	数据集的成员（新的行或消失的行）
写者操作	`UPDATE` （更新）	`INSERT` （插入）或 `DELETE` （删除）
读者现象	在同一事务中，两次读取同一行，得到的值不同。	在同一事务中，两次执行相同的查询，得到的结果集行数不同（像出现了幻觉）。
例子	事务A读账户余额为100。事务B将余额改为50并提交。事务A再读，余额变成了50。	事务A查询“年龄>30的员工”有10人。事务B插入一名35岁员工并提交。事务A再次相同查询，得到11人。

所以，你的总结非常到位：不可重复读是“值”变了，幻读是“有无”变了。两者都破坏了事务内的一致性。

---

2. 序列化 (Serializable)：你的理解基本正确，但可以更精确

你说“串行化直接就是一个人的事务没有处理完，另一个人不能操作数据项”，这个描述更接近可重复读隔离级别下用锁实现的方式。而真正的序列化隔离级别，其实现机制更智能、更严格。

序列化隔离级别的目标是：保证并发执行多个事务的结果，等同于以某种顺序串行地（一个接一个）执行它们的结果。这是最强的一致性保证。

数据库并不总是用“简单粗暴”的锁来实现这一点。现代数据库（如 PostgreSQL）通常采用两种高级策略：

策略一：严格的锁（基于你的理解）

这确实是一种方式。通过谓词锁（Predicate Locking） 来实现。

- 例子：事务A执行 `SELECT * FROM users WHERE age > 30;`
- 数据库行为：数据库不会只锁住当前 `age>30` 的那些行，它会锁住整个“age>30”的查询条件。

- 效果：在事务A结束前，任何其他试图插入一个 `age=35` 的新用户、删除一个 `age=40` 的用户、或者更新一个用户年龄使其从29变成31的 `UPDATE` 操作，都会被阻塞。
- 你的描述成立：在这种情况下，确实是“一个人的事务（A）没有处理完，另一个人（B）不能操作相关数据项”。

策略二：乐观并发控制（更常见的现代实现）

数据库允许事务并行执行，但如果检测到执行结果可能违反序列化，就会强制其中一个事务回滚并重试。最主流的技术是 SSI（可序列化快照隔离）。

- 例子：
  1. 事务A和事务B同时开始。
  2. 事务A执行 `SELECT * FROM users WHERE age > 30;` （读到10条记录）。
  3. 事务B执行 `INSERT INTO users ... (age=35);` 并提交。
  4. 事务A再次执行 `SELECT * FROM users WHERE age > 30;`。
     - 在可重复读级别下，它会依靠自己的快照，仍然只看到10条记录（避免了幻读）。
     - 在序列化级别下，SSI机制会检测到：事务B的写入（插入）影响了事务A读到的数据集。即使A没看到，这也是一个潜在的“读写冲突”。
  5. 当事务A尝试提交时，数据库会果断地让其提交失败并回滚，通常会返回一个错误如 `40001 Serialization Failure`。应用程序需要捕获这个错误并重新执行整个事务A。

所以，更精确的说法是：
序列化隔离级别保证最终结果像串行执行一样。它通过两种主要手段：

1. 悲观锁：预先阻止可能造成冲突的并发操作（等锁或死锁）。
2. 乐观检测：允许并行操作，但在提交时检查冲突，一旦发现可能违反序列化，就回滚后到的事务。

总结

隔离级别	解决什么问题	实现方式（粗略理解）
读已提交	脏读	只能读到已提交的数据。
可重复读	脏读 + 不可重复读	快照隔离：事务从头到尾都看到同一个旧快照。
序列化	脏读 + 不可重复读 + 幻读	强制串行顺序：要么用锁预先阻止冲突，要么检测到冲突后回滚事务。

你的理解力非常强，已经抓住了并发问题中最精髓的部分。这种从现象（脏读、幻读）到解决方案（隔离级别）的思考方式，是掌握数据库核心知识的正确路径。