第十章 事务处理概述

章成源

湖南大学-信息科学与工程学院-计算机科学系

办公室：院楼403

Email：cyzhangcse@hnu.edu.cn

问题背景

- 可编制如下程序:

查询*F005*航班*2021*年*5*月*20*日剩余票数*A;(1)*

if *(A<2)*

拒绝操作，并通知票源不足;

else

更新*A0011*售票点的售票数;

　　更新*F005*航班的剩余票数; } *(2)*

如何用SQL语句分别实现语句(1)和语句(2)?

■语句(1)可用SQL语句表示为:

```
SELECT remainNumber
FROM Flight
WHERE flightNo='F005' AND date='2021-05-20'
```

问题背景

语句(2)是在当F005航班2021年5月20日的剩余票数大于请 求票数时更新Sale和Flight表。该更新包括两个update操作:

```
UPDATE Sale
SET saledNumber=saledNumber+2
WHERE       agentNo='A0011' AND flightNo='F005'
AND date='2021-05-20'
```

```
UPDATE Flight
SET remainNumber=remainNumber–2
WHERE flightNo='F005' AND date='2021-05-20'
```

如果第一个UPDATE语句执行成功，而第二个UPDATE语句执行失败，会发生什么问题呢？？？

问题背景

- 假设F005航班共有200个座位，2021年5月20日机票已售198 张(其中被A0011售出20张)，余票2张。
- 当第1个UPDATE语句执行成功时，即A0011已售票数更新为200. 当系统发生故障重新提供服务时，如果又有售票点请求出售F005航班2021年5月20日机票2张，由于F005的剩余票数未更新(仍为2)，因此满足其要求又出售了2张。
- 结果多卖了2张票！

出现上述问题的原因是什么？

出现故障后，系统重新提供服务时数据库状态与现实世界状态出现了不一致。对 于机票系统来说，一航班的剩余票数加上已售出票数应等于该航班全部座位数。而重新提供服务时，F005的已售票数与剩余票数之和为202(不等于200!)，导致多买了2张。

问题背景

- 为解决上述问题，数据库管理系统引入了**事务概念**，它将这些有内在联系的操作当作一个逻辑单元看待，并采取相应策略保证一个逻辑单元内的全部操作要么都执行成功，要么都不执行。
- 对数据库用户而言，只需将具有完整逻辑意义的一组操作正确地定义在一个事务之内即可。

第10章 事务处理概述

- 10.1 事务基本概念
- 10.2 事务异常与隔离级别
- 10.3 正确的调度
- 10.4 本章小结

10.1 事务基本概念

- 10.1.1 事务的定义
- 10.1.2 事务的ACID特性

10.1.1 事务的定义

- 定义
  - 事务是**用户定义**的一个数据库**操作序列**，这组操作要么全部被执行，要么全部不被执行，是一个不可分割的执行单元。
- 通常情况下，事务由用户**显式**地进行设定：
  - 一个事务的开始由关键词**Begin Transaction（简化成Begin）**来表示。
  - 一个事务的结束由关键词**Commit**或**Abort**来表示：Commit表示全部执行事务的所有操作;Abort表示撤销事务已发生的更新操作。
  - 事务由**Begin与Commit/Abort之间的所有操作**组成。

10.1.1 事务的定义

- [例10.1]账户A余额25元，账户B余额5元，账户A转账10元到账户B中，完整的事务执行语句如下：

  Begin Transaction;

  UPDATE account SET balance = balance - 10 WHERE accountID = A;

  UPDATE account SET balance = balance + 10 WHERE accountID = B;

  Commit;

- 事务提交后，事务包含的这组操作对数据库中数据的修改必须生效。
- 当查询数据库中账户信息时，账户A和账户B的余额分别为15元。

10.1.1 事务的定义

- 注意：如果事务的执行语句如下，即事务的结束标记是回滚，则意味着该事务的所有操作对数据库的修改都会被撤销。

  Begin Transaction;

  UPDATE account SET balance = balance - 10 WHERE accountID = A;

  UPDATE account SET balance = balance + 10 WHERE accountID = B;

  Abort;

- 当再一次查询数据库中账户信息时，账户A和账户B的余额分别被恢复为该事务开始执行时的值，即分别是25和5。

10.1.1 事务的定义

- 事务的**符号化**表示
  - 事务包含的操作分为读操作和写操作，用A, B来表示数据库中的数据项，数据项通常情况下为数据库中的一条记录。
  - **读操作**：事务T1对数据项A的读操作表示为R1(A)，当读取数据项A的值为25时，该读操作也可表示为R1(A,25)；
  - **写操作**：事务T1对数据项B的写操作表示为W1(B)，当写入数据项B的值为15时，该写操作也可表示为W1(B,15)；
  - **Commit操作和Abort操作**：用C1和A1分别表示事务T1的Commit操作和Abort操作。

10.1.1 事务的定义

- 比如在例[10.1]的银行转帐事务**“账户A余额25元，账户B余额5元，账户A转账10元到账户B”**中，可以用A来表示账户A，用B来表示账户B。
  - 第一条更新账户A余额的UPDATE语句中，包括：
    - 1）读取A的账户余额（表达为**R1(A)**）；
    - 2）更新A的账户余额（表达为**W1(A)**）；
  - 第二条更新账户B余额的UPDATE语句中，包括两个操作：
    - 1）读取B的账户余额（表达为**R1(B)**），
    - 2）更新B的账户余额（表达为**W1(B)**）。
  - 因此，第一个转账事务T1的符号化表示如下：

$$R_1(A)W_1(A)R_1(B)W_1(B)C_1 \text{ 或 } R_1(A,25)W_1(A,15)R_1(B,5)W_1(B,15)C_1$$

10.1.1 事务的定义

- SQL语句到读写操作之间的转换
  - 在事务的符号化表示中，实际上是**将事务中包含的SQL语句转换成对相应数据项的读写操作**。
  - 例如，给定以下SQL语句：

    UPDATE account SET balance = balance - 10 WHERE accountID = A;
  - 该SQL语句的含义是对account表中accountID为A的记录的账户余额减10元。该SQL需要转换成**R1(A,25)W1(A,15)**两个操作。

10.1 事务基本概念

- 10.1.1 事务的定义
- 10.1.2 事务的ACID特性

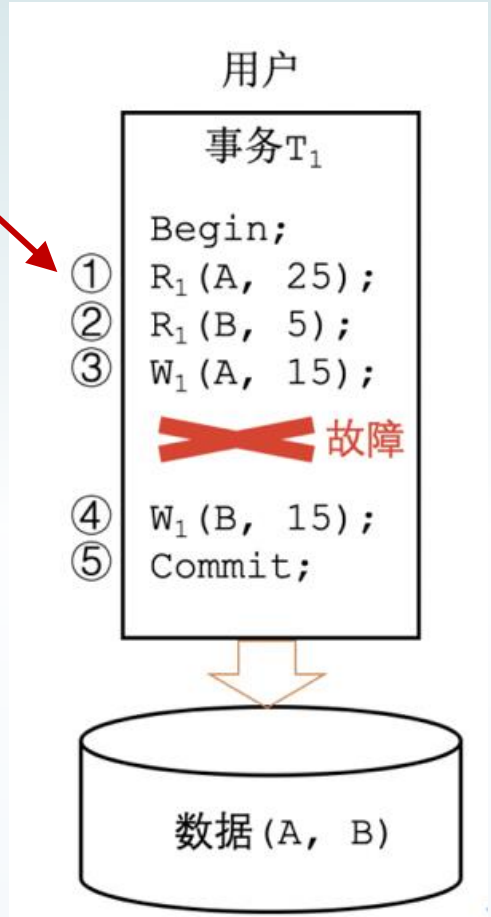
10.1.2 事务的ACID特性

- 事务具有四个特性：
  - **原子性（Atomicity）**
  - **一致性（Consistency）**
  - **隔离性（Isolation）**
  - **持久性（Durability）**
- 这四个特性简称为**ACID特性**。

10.1.2 事务的ACID特性

- **原子性含义：**事务对数据库数据的修改要么全部执行，要么全部不执行。
- [例10.2]：用户从A账户转出10元到B账户中。

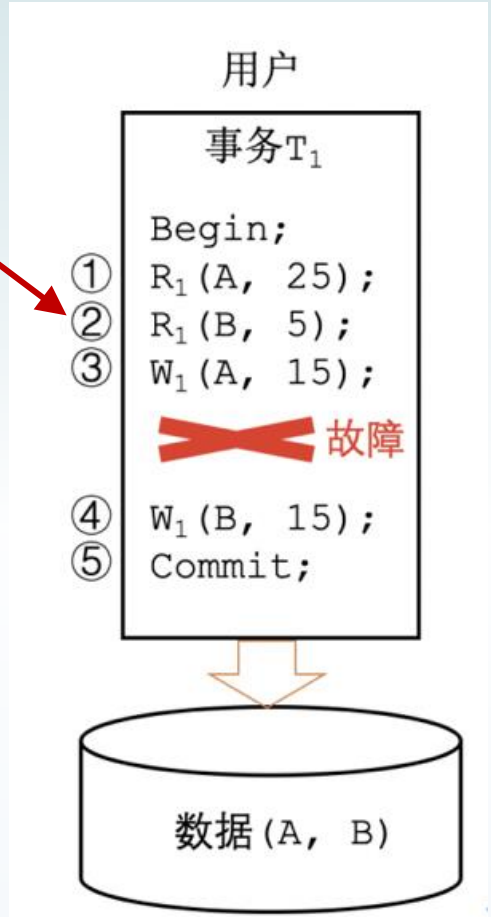
① **R1(A,25)：**读取A的账户余额



10.1.2 事务的ACID特性

- **原子性含义：**事务对数据库数据的修改要么全部执行，要么全部不执行。
- [例10.2]：用户从A账户转出10元到B账户中。

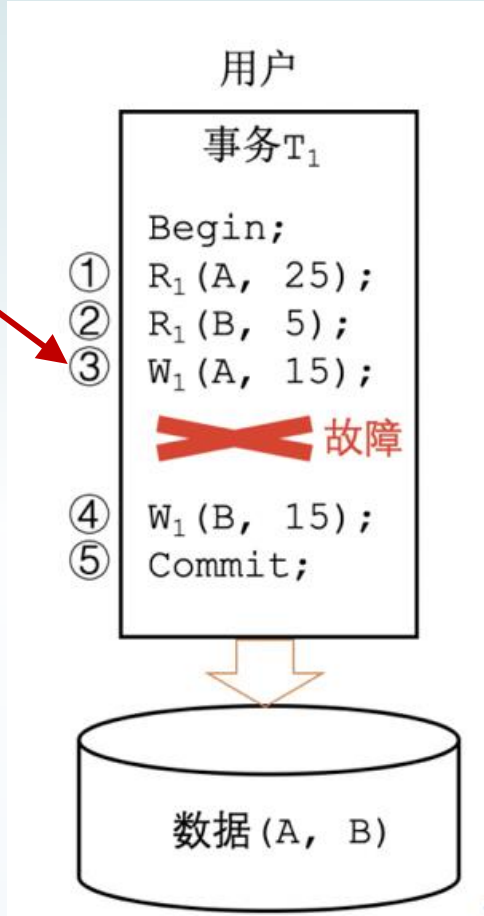
② **R1(B,5)：**读取B的账户余额



10.1.2 事务的ACID特性

- **原子性含义：**事务对数据库数据的修改要么全部执行，要么全部不执行。
- [例10.2]：用户从A账户转出10元到B账户中。

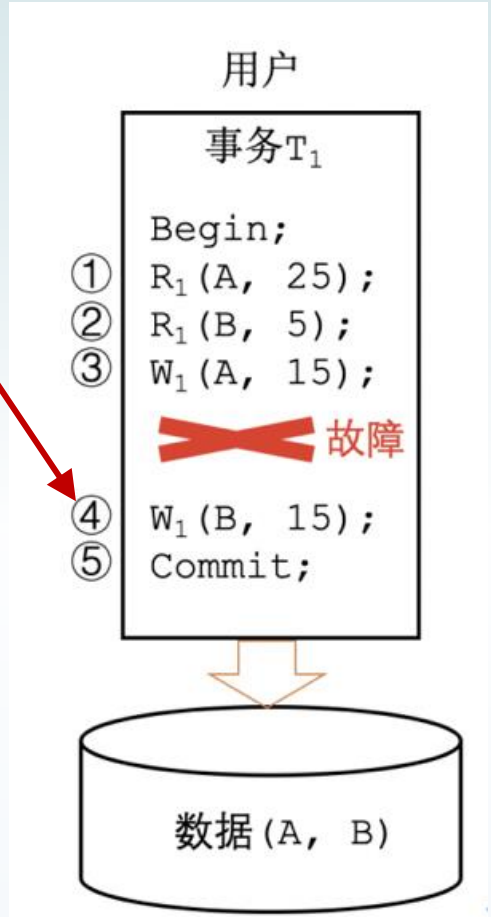
③ W1(A,15)：更新A的账户余额为15



10.1.2 事务的ACID特性

- **原子性含义：**事务对数据库数据的修改要么全部执行，要么全部不执行。
- [例10.2]：用户从A账户转出10元到B账户中。

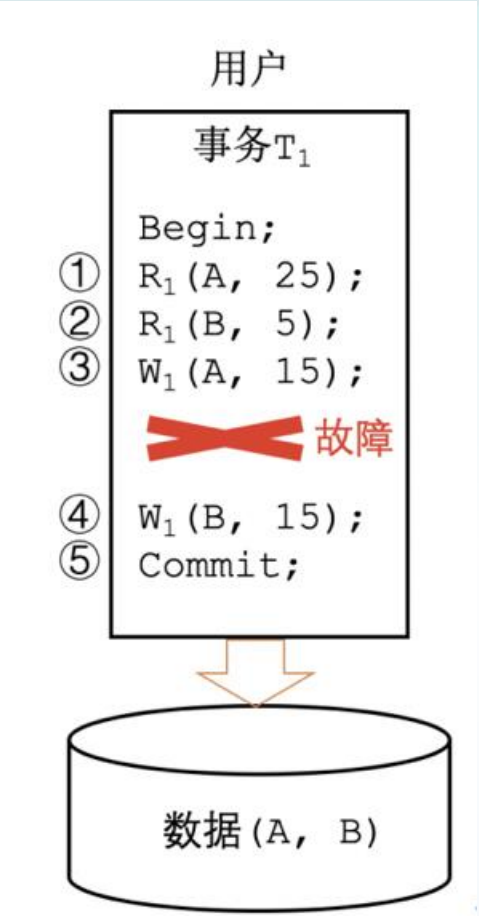
④ W1(B,15)：更新B的账户余额为15



10.1.2 事务的ACID特性

- **原子性含义：**事务对数据库数据的修改要么全部执行，要么全部不执行。
- [例10.2]：用户从A账户转出10元到B账户中。

事务的原子性要求事务对数据库数据的修改要么全部执行，要么全部不执行。在这个例子中，**事务已经执行的操作需要全部回滚，即撤销对数据项的修改**。



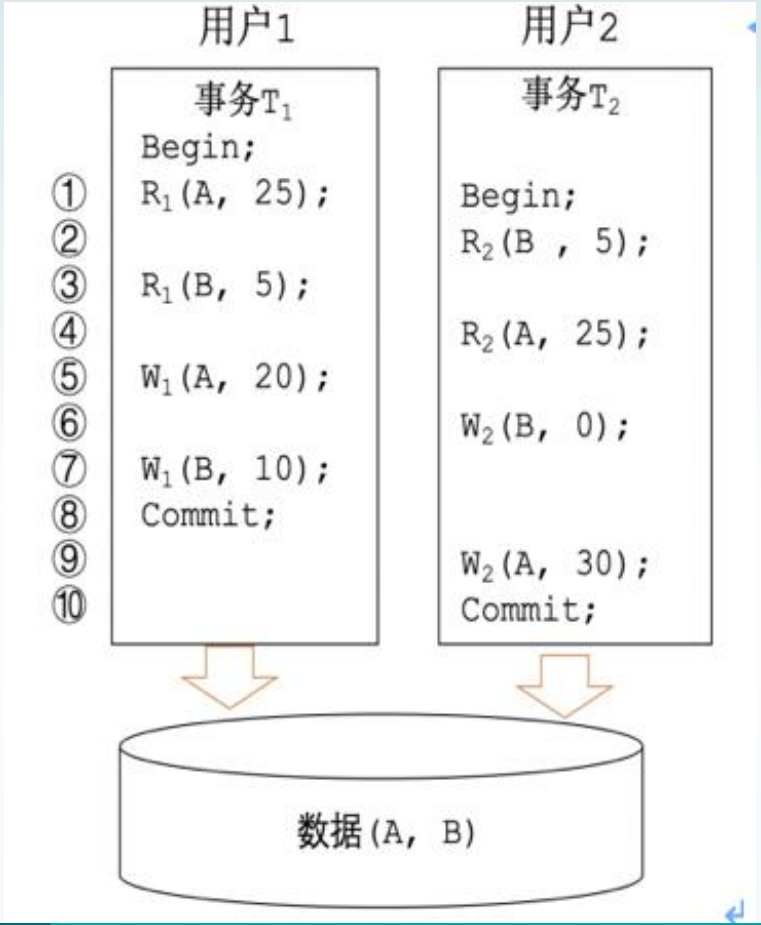
10.1.2 事务的ACID特性

- **一致性含义：**事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态，确保完整性约束不被破坏，也就是说，一旦事务提交，其对数据库状态的改变，不能破坏完整性约束。

10.1.2 事务的ACID特性

- [例10.3]：用户1从账户A转出5元到账户B中，用户2从账户B转出5元到账户A中，即**账户A和账户B同时、相互转5元给对方**。假设A与B的初始账户余额分别为**25元和5元**。

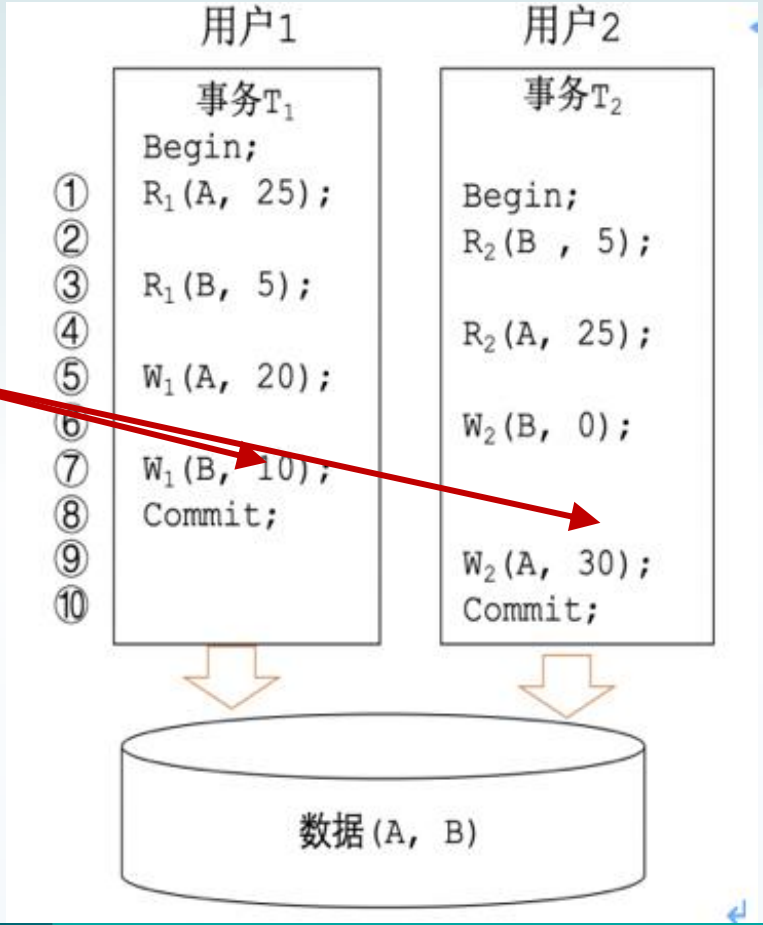
完整性约束条件为：转账前A与B的账户余额总和与转账之后两者的账户余额总和相同。



10.1.2 事务的ACID特性

- [例10.3]：用户1从账户A转出5元到账户B中，用户2从账户B转出5元到账户A中，即**账户A和账户B同时、相互转5元给对方**。假设A与B的初始账户余额分别为**25元和5元**。

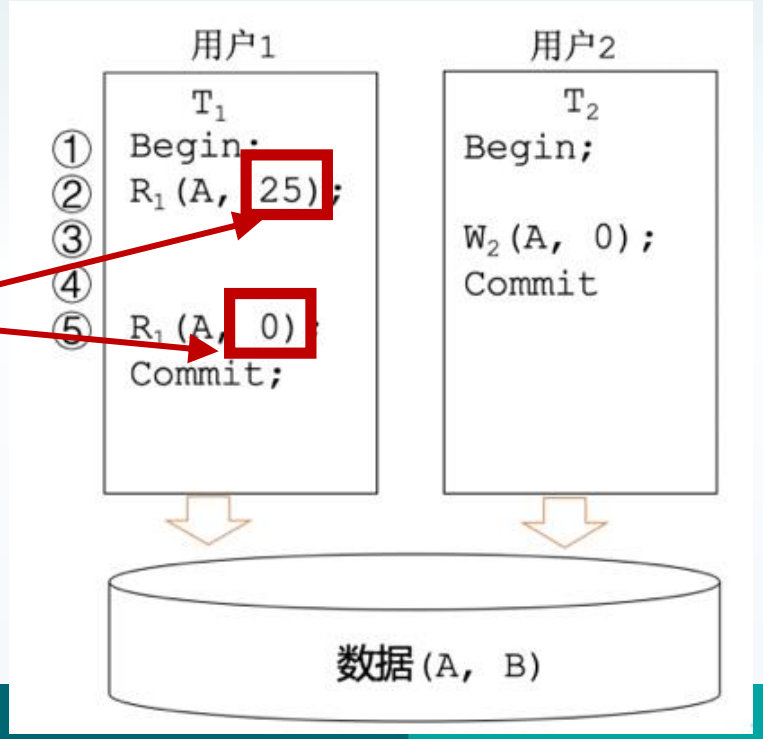
转账结束后A与B的账户余额分别为**10元和30元**，余额和为**40元**，改变了，显然破坏了数据库的一致性状态。



10.1.2 事务的ACID特性

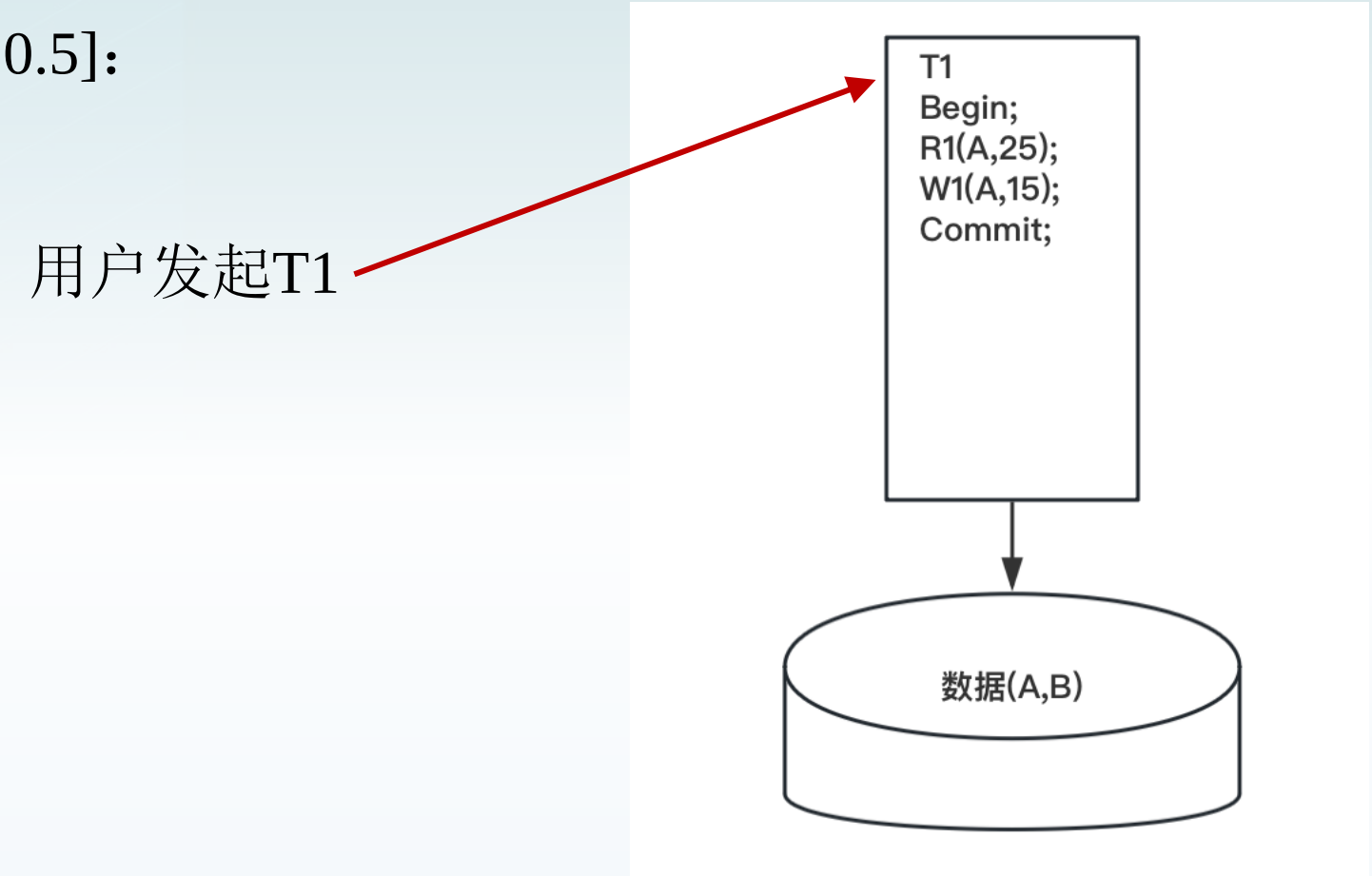
- **隔离性含义：**一个事务的执行不能被其它事务干扰。即一个事务的内部操作及使用的数据对其它并发事务是隔离的，并发执行的各个事务之间不能相互干扰。
- [例10.4]：用户1查询了A账户的余额，用户2取走了同一账户A的所有金额，当用户1再次查询账户余额时，与第一次查询到的账户余额不一致。

事务T1执行过程中，被事务T2**干扰**，使得事务T1分**两次读取同一变量返回的值不一致**，破坏了事务的隔离性。



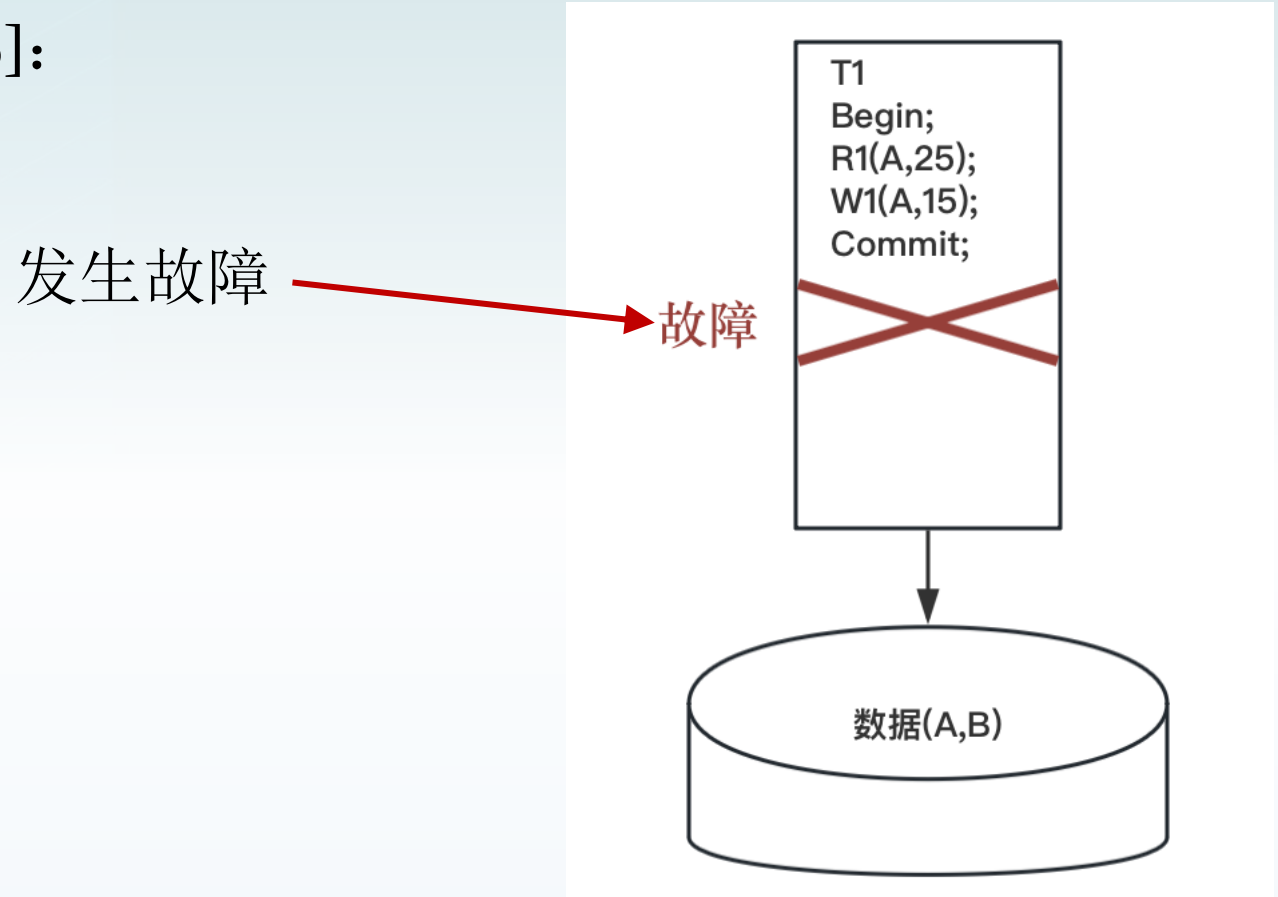
10.1.2 事务的ACID特性

- **持久性含义：**一个事务一旦提交，它对数据库中数据的改变就应该是永久性的，接下来的其它操作或故障不应该对其执行结果有任何影响。
- [例10.5]：



10.1.2 事务的ACID特性

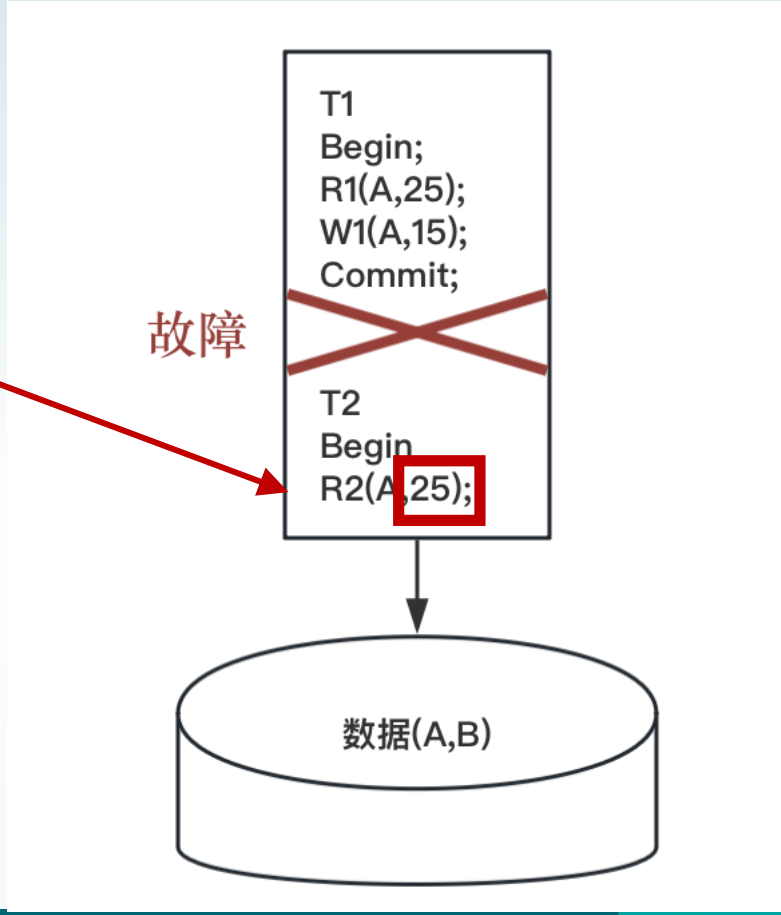
- **持久性含义：**一个事务一旦提交，它对数据库中数据的改变就应该是永久性的，接下来的其它操作或故障不应该对其执行结果有任何影响。
- [例10.5]：



10.1.2 事务的ACID特性

- **持久性含义：**一个事务一旦提交，它对数据库中数据的改变就应该是永久性的，接下来的其它操作或故障不应该对其执行结果有任何影响。
- [例10.5]：

用户发起T2，T1提交的数据没有作用，查询结果是没更改的，违背了**事务的持久性特性**



10.1.2 事务的ACID特性

- 在事务的执行过程中，数据库管理系统确保事务的ACID特性不被破坏。
- 如何保证事务的原子性(A)和持续性(D)？
  - **故障恢复子系统**
- 如何保证事务的一致性(C)和隔离性(I)?
  - **并发控制子系统**

10.1 小结

- 为什么需要事务？
  - 是数据库管理系统保证各类故障、多用户高并发访问下数据不出错的关键技术。
- 什么是事务？
  - 用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位。
  - 具有ACID特性。
- 如何实现事务机制？
  - 故障恢复保证事务的原子性(A)和持续性(D)
  - 并发控制保证事务的一致性(C)和隔离性(I)

第10章 事务处理概述

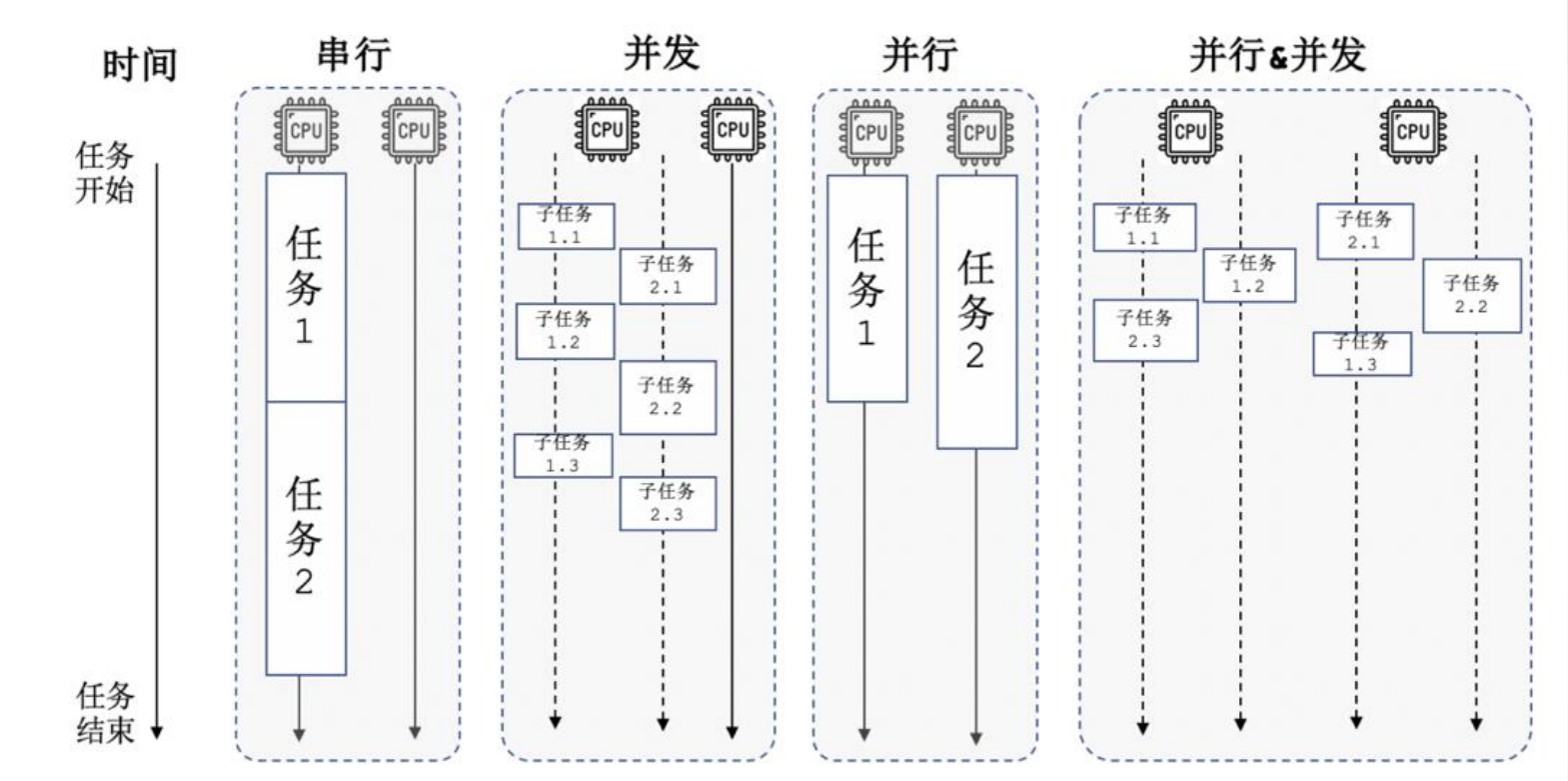
- 10.1 事务基本概念
- 10.2 事务异常与隔离级别
- 10.3 正确的调度
- 10.4 本章小结

10.2 数据异常与隔离级别

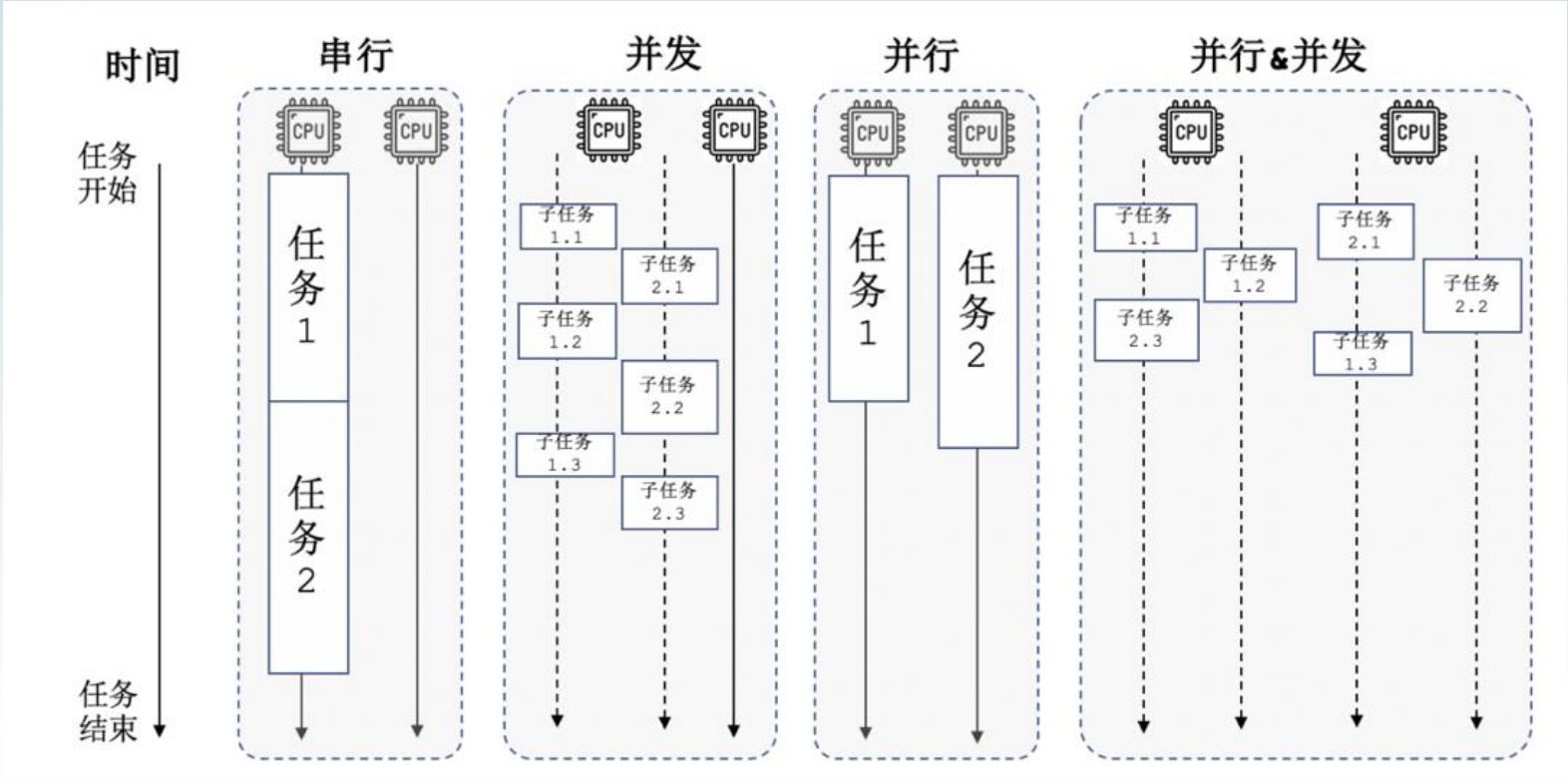
- 10.2.1 事务的执行模型
- 10.2.2 数据异常
- 10.2.3 隔离级别

10.2.1 事务的执行模型

- 事务的执行模型是与操作系统的进程/线程执行模型息息相关的，操作系统的进程/线程执行模型包括**串行执行、并发执行、并行执行、并行与并发混合执行**四种方式。下图给出了计算机系统中线程在2-核CPU的执行模型。

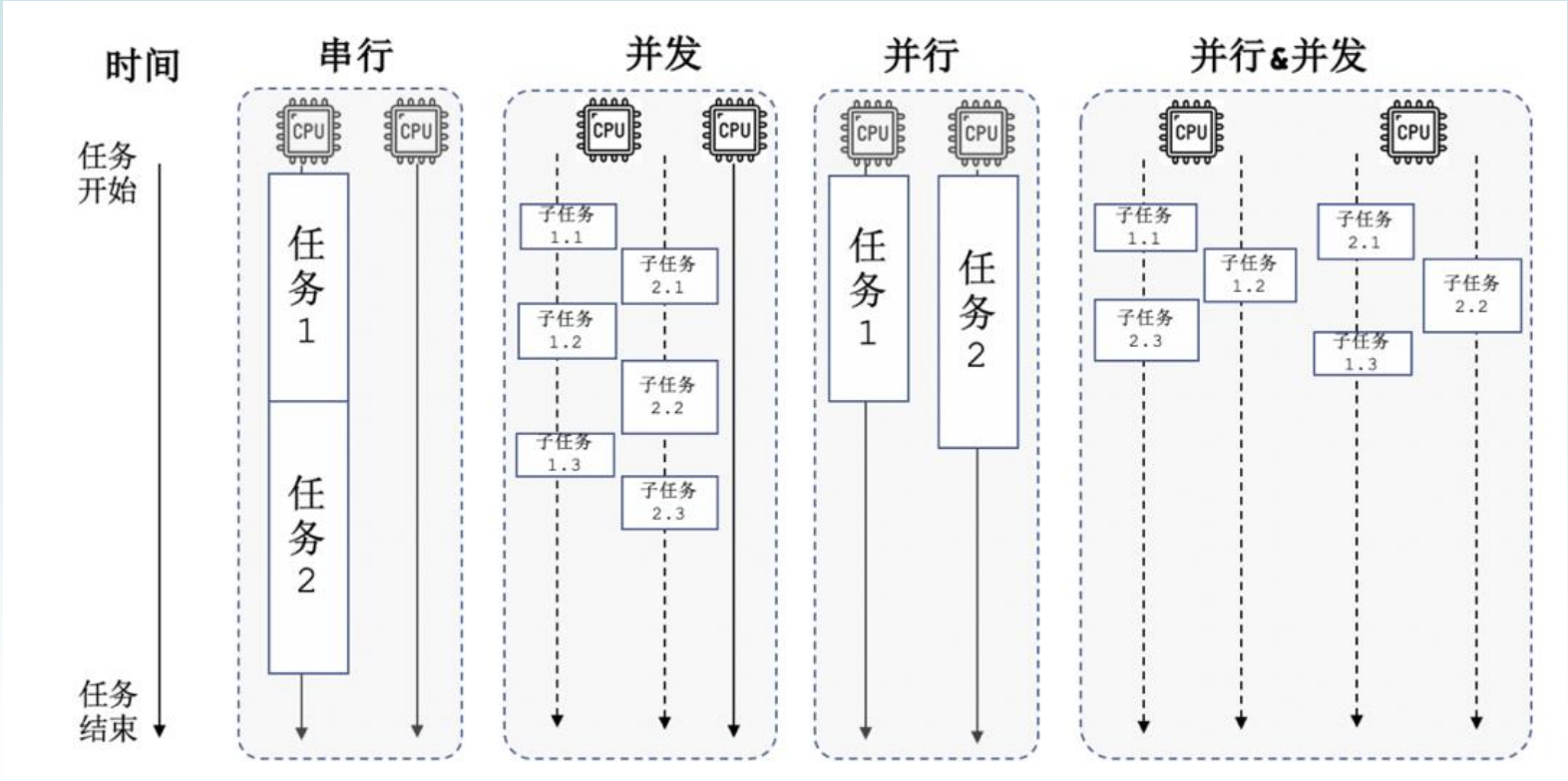


10.2.1 事务的执行模型



- （1）在**串行执行模型**中，单核CPU在执行完一个任务后再开始执行一个新的任务；
- （2）在**并发执行模型**中，每个线程被安排执行各自的任务，各个线程切换CPU的时间片交替运行（也称**分时复用**）；

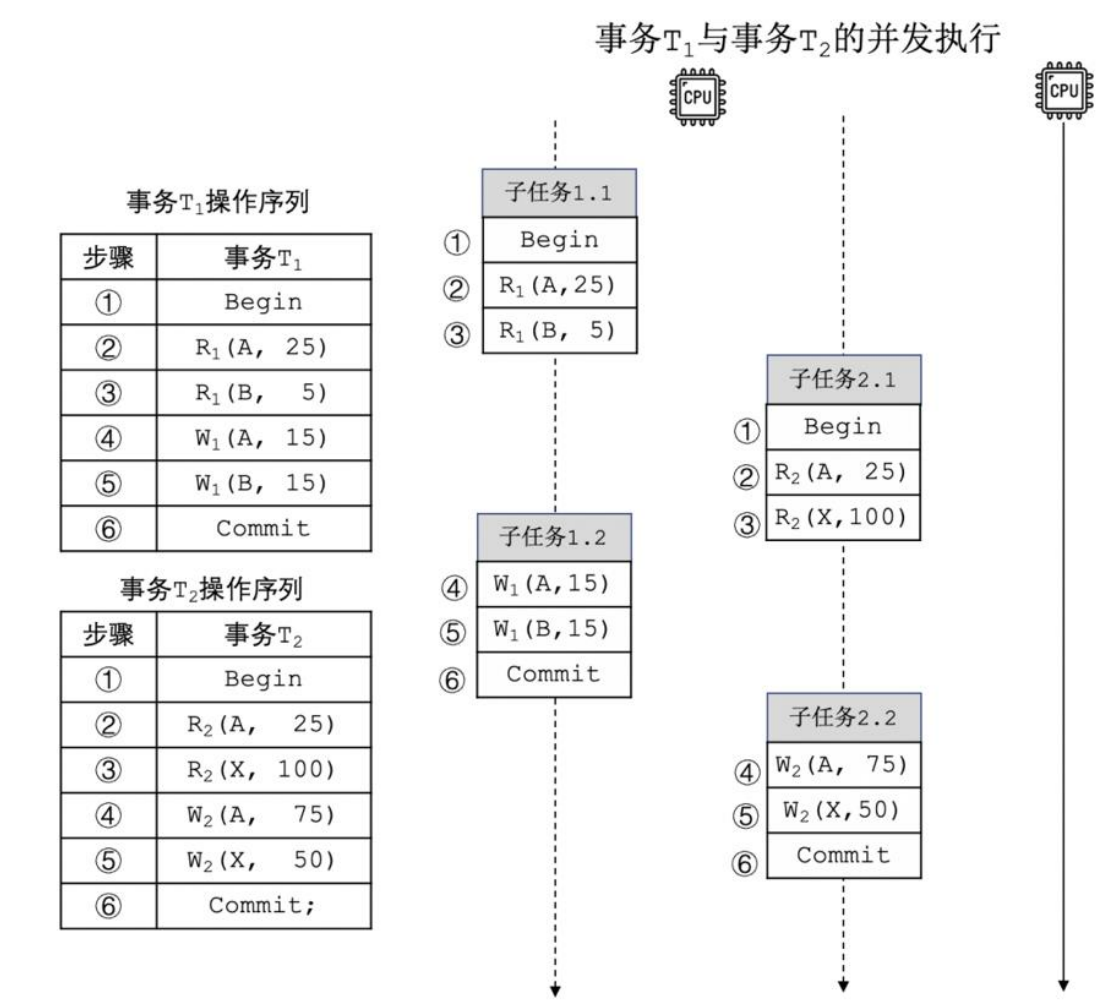
10.2.1 事务的执行模型



- （3）在**并行执行模型**中，每个CPU单独执行任务；
- （4）在**并行与并发混合执行模型**中，每个线程被安排执行各自的任务，同一个CPU的多个线程，切换该CPU的时间片交替运行。

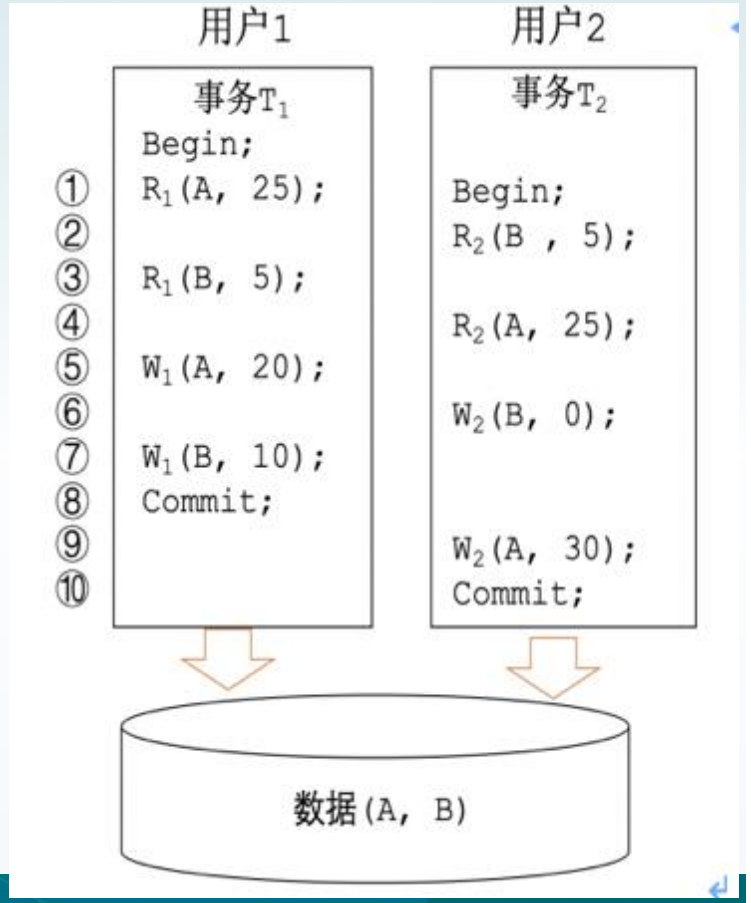
10.2.1 事务的执行模型

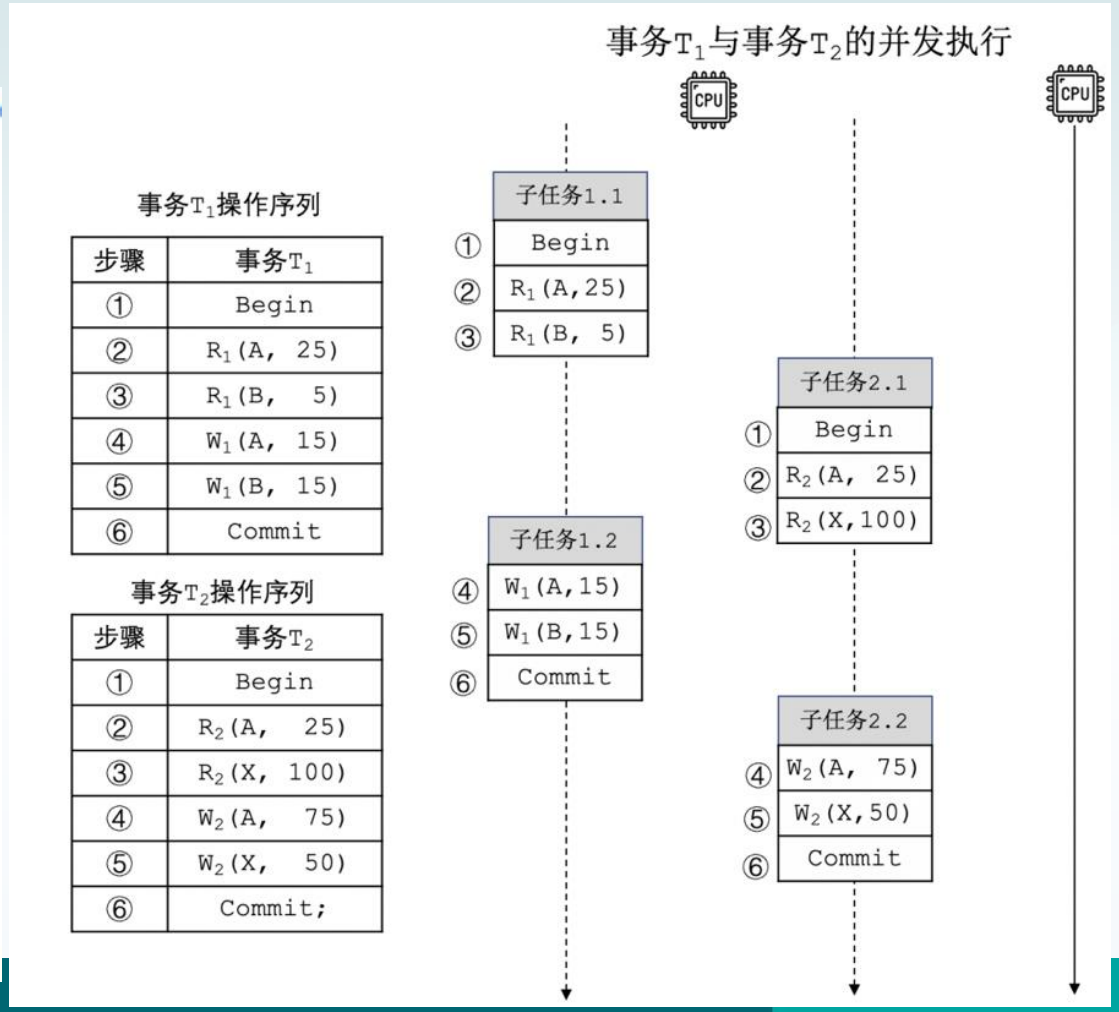
- 一个事务会交由**一个线程或进程**来执行，对应下图所示的任务。子任务对应事务的一个子操作序列。

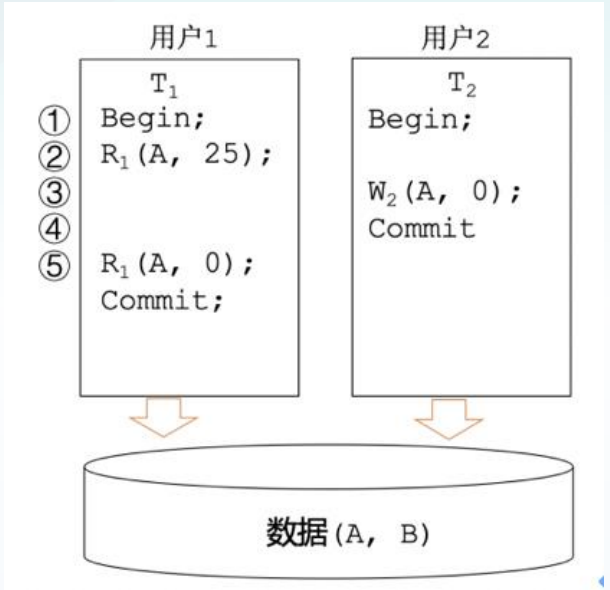


10.2.1 事务的执行模型

- 从上述事务的执行过程来看，**两个事务的执行过程之间存在时间上的重叠**，如果不进行合适的控制，就会出现如图10.2、10.3所示**破坏事务一致性和隔离性**的情况。







10.2 数据异常与隔离级别

- 10.2.1 事务的执行模型
- 10.2.2 数据异常
- 10.2.3 隔离级别

10.2.2 数据异常

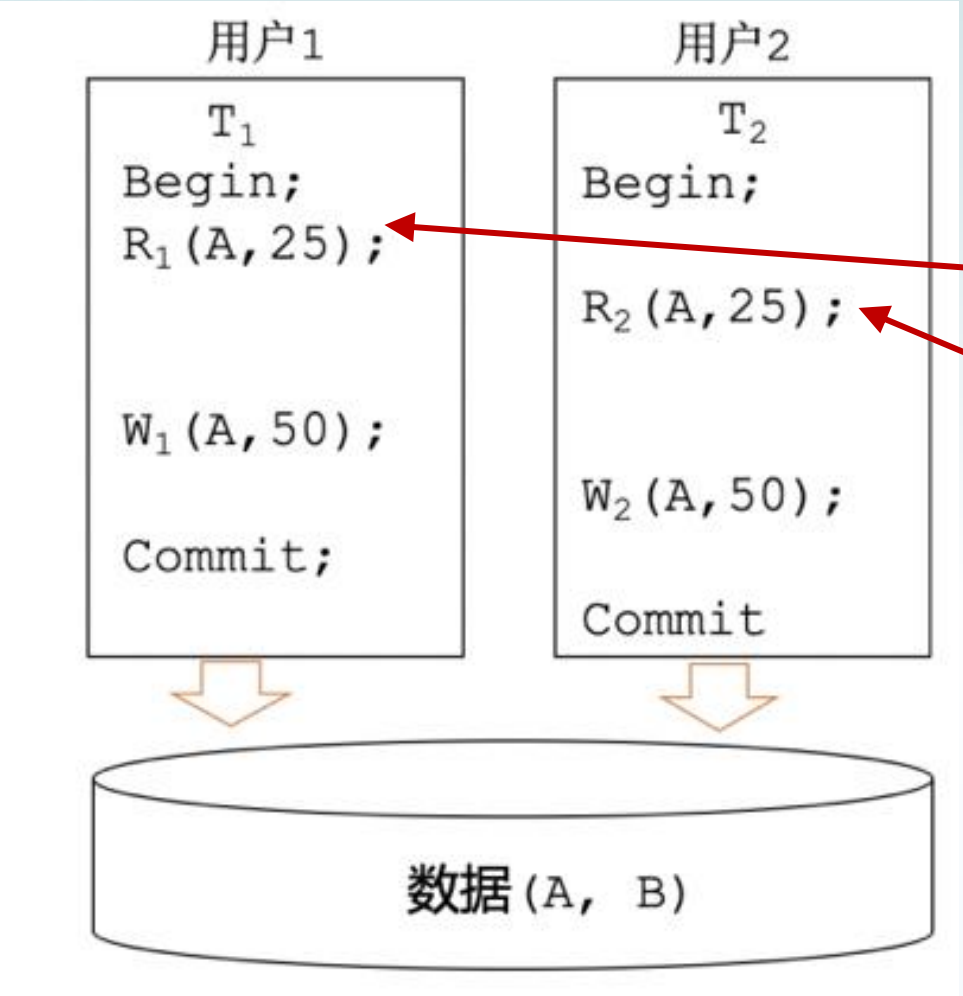
- 数据异常的种类很多。ANSI/ISO SQL-92给出了四类数据异常：
  - **脏写（Dirty Write）**
  - **脏读（Dirty Read）**
  - **不可重复读（Non-Repeatable Read）**
  - **幻读（Phantom Read）**

10.2.2 数据异常

- **脏写：**给定两个事务T1和T2，T1先修改了某一数据项A，在T1提交之前，T2也修改了A的值，之后，T1和T2均提交。**T2的提交导致T1的修改被T2的修改覆盖了**。脏写数据异常可以被符号化表示为…W1(A)…W2(A)…(C1…C2或C2…C1)。

10.2.2 数据异常

- [例10.6]：

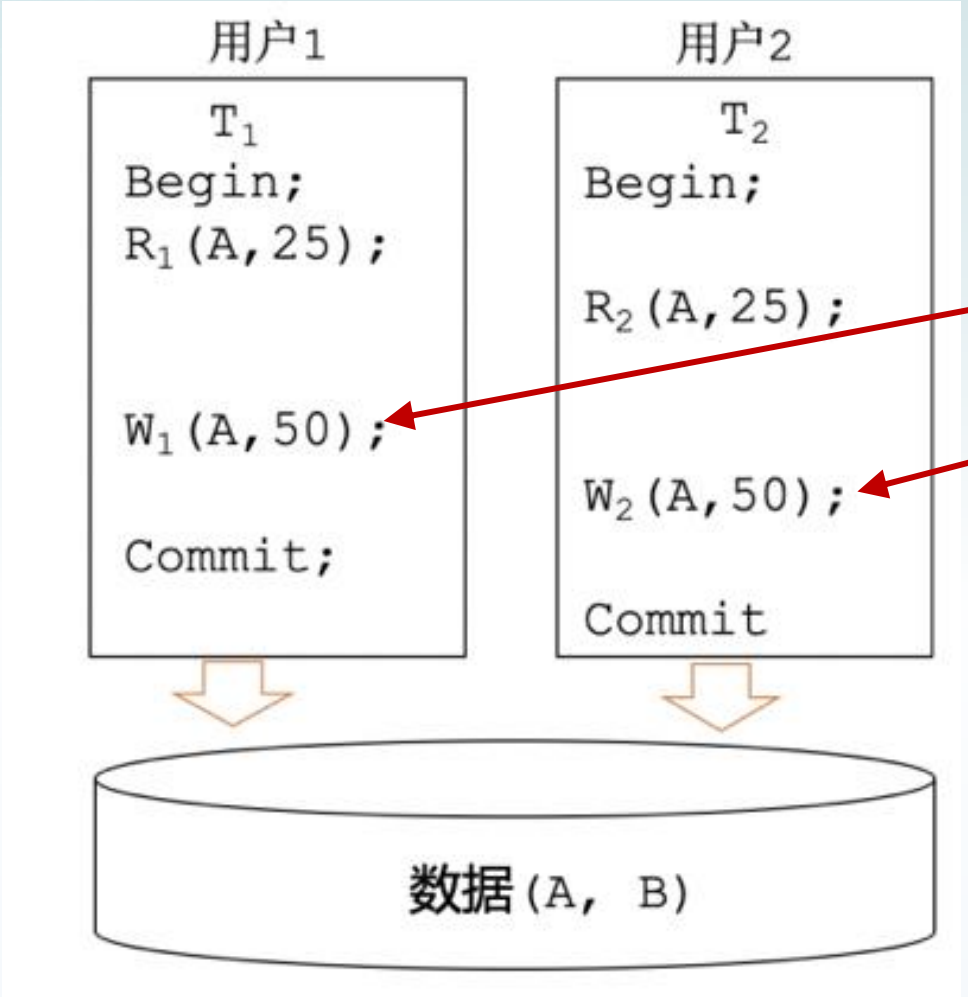


用户1发起事务T1查询了A账户余额，

用户2发起事务T2也查询A账户余额

10.2.2 数据异常

- [例10.6]：

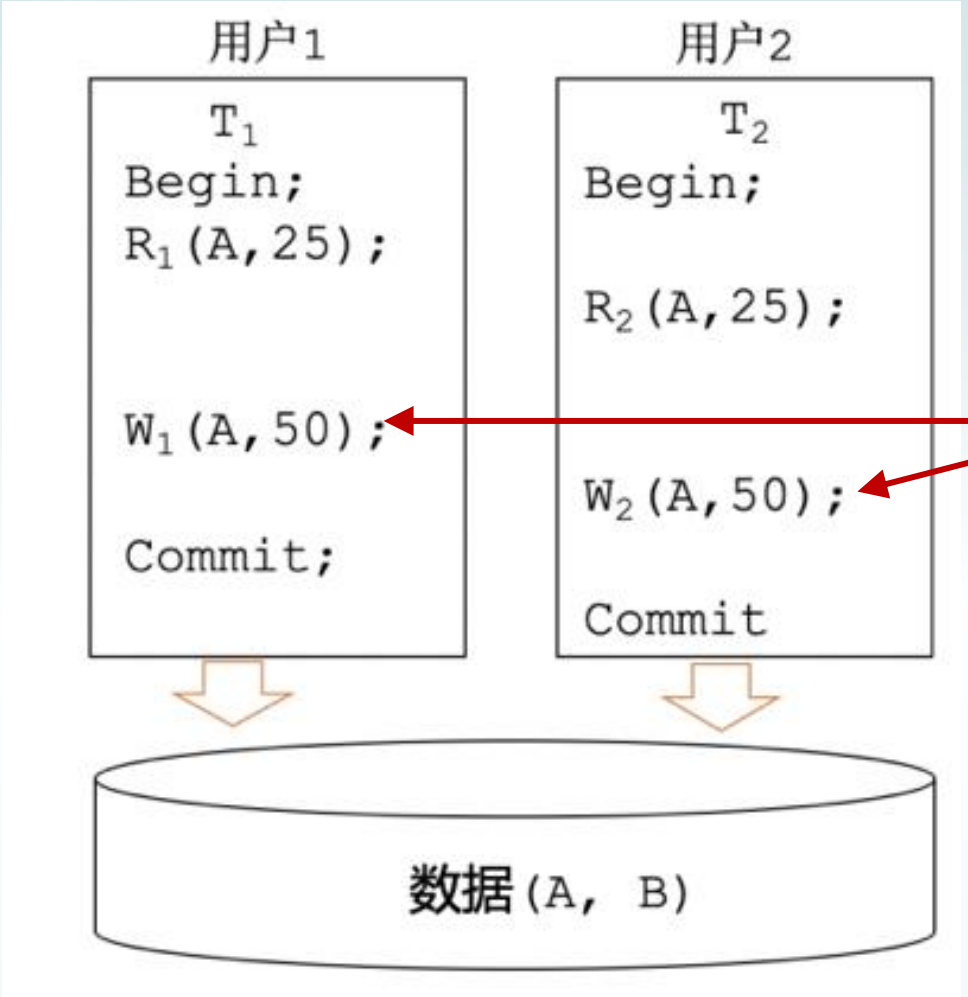


用户1往A账户里存了25元，

用户2也往A账户里存了25元

10.2.2 数据异常

- [例10.6]：



两个用户向A账户里共存了50元，但从最终A账户的余额中可以看到，A的余额仅增加了25元这是因为T1的修改被T2的**修改覆盖**了，从而出现了**脏写**的数据异常。

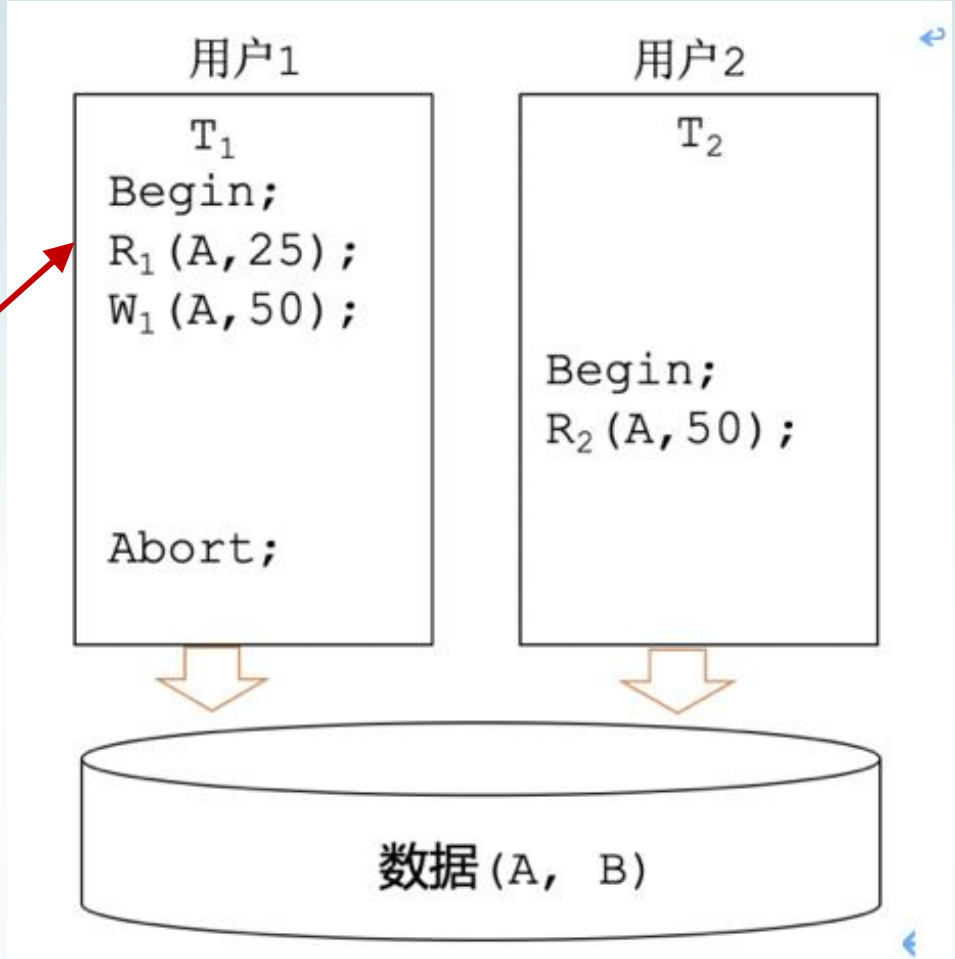
10.2.2 数据异常

- **脏读：**给定两个事务T1和T2，T1先修改了某一数据项A，在T1提交之前，T2读取了A的值，之后，T1回滚，从而导致**T2读取了未提交事务T1的写**。脏读数据异常可以被符号化表示为…W1(A)…R2(A)…A1…。

10.2.2 数据异常

- [例10.7]:

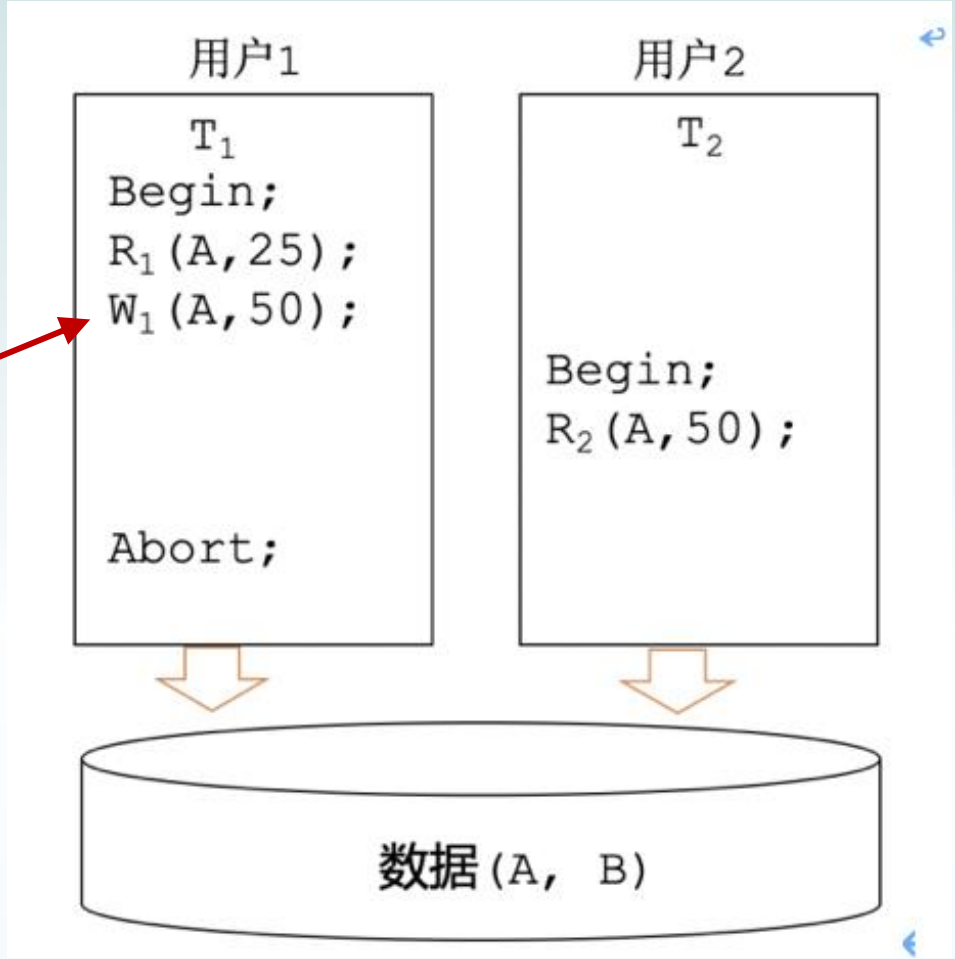
用户1发起事务T1查询了A账户余额（A=25）



10.2.2 数据异常

- [例10.7]:

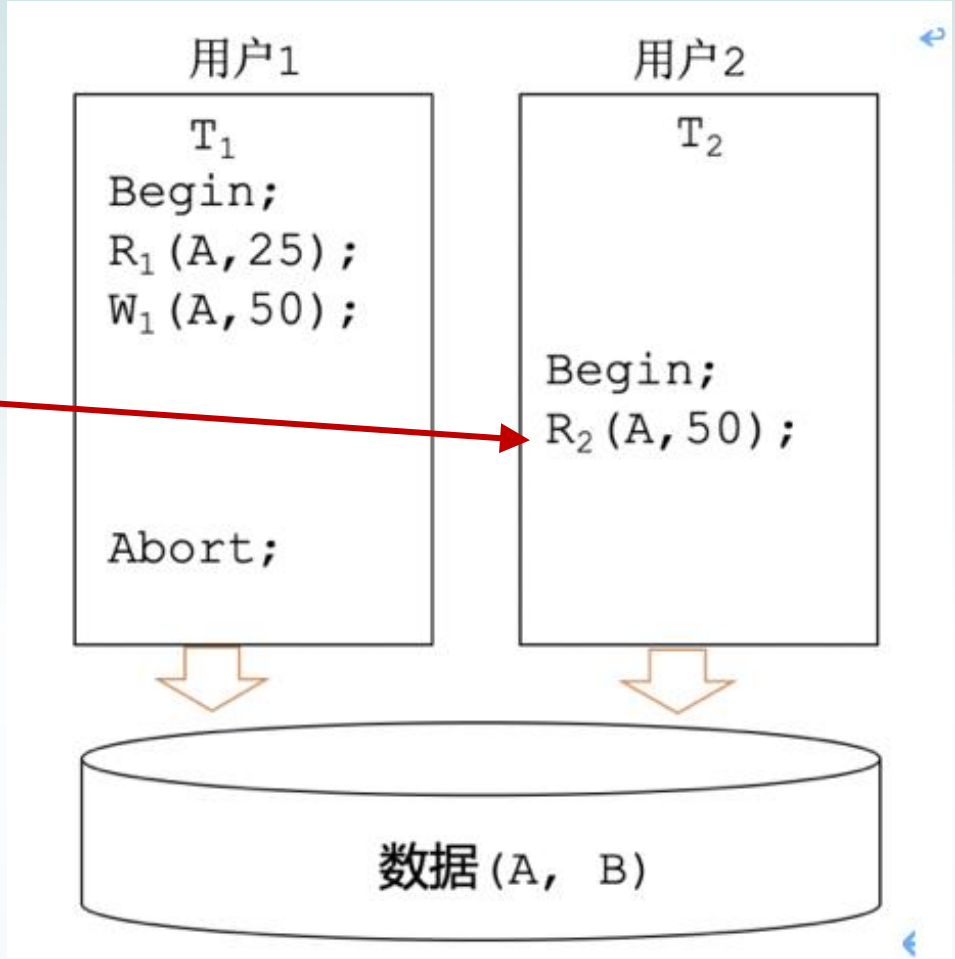
往A账户里存了25元（A更新后的值为50）



10.2.2 数据异常

- [例10.7]：

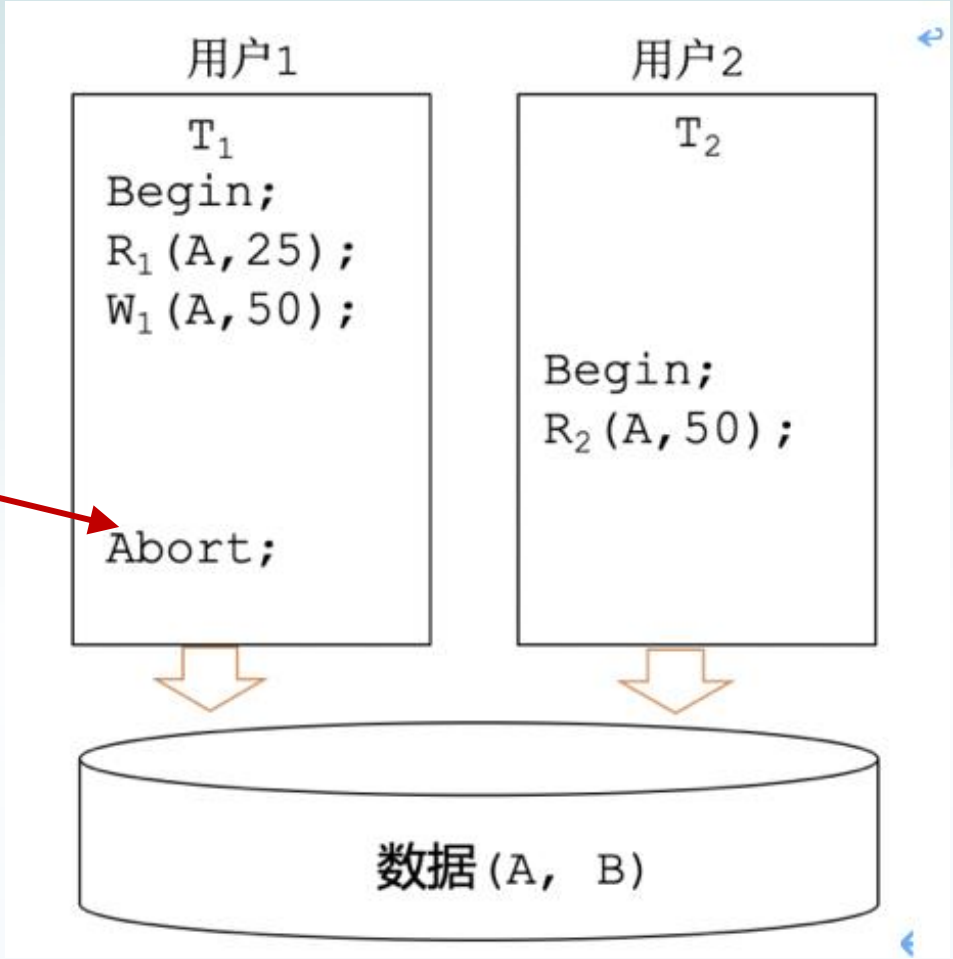
用户2发起了事务T2，并读到了事务T1更新的A值（更新后的A值，即50）



10.2.2 数据异常

- [例10.7]：

事务T1回滚，T1事务撤销了本次的存款，导致T2读到了未提交事务T1的写，从而出现了脏读的数据异常



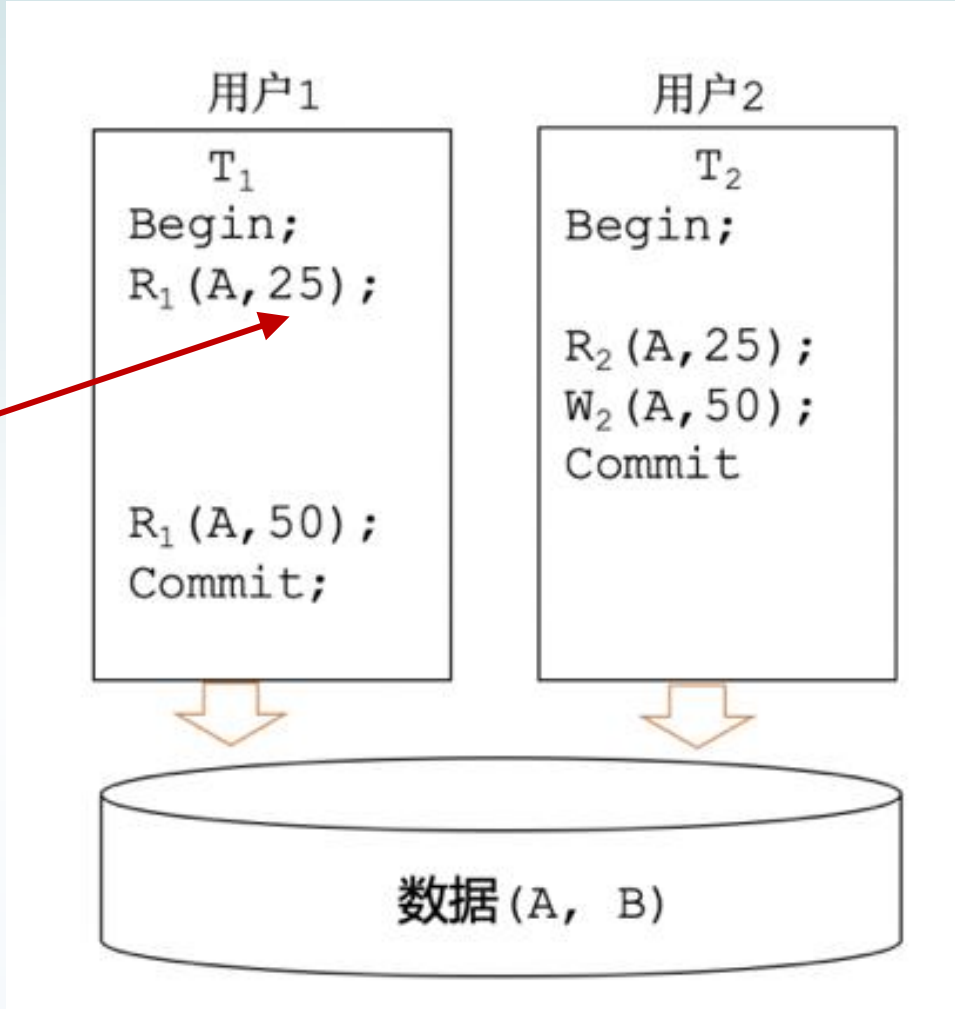
10.2.2 数据异常

- **不可重复读：**给定两个事务T1和T2，T1先读了某一数据项A，随后T2修改了A的值并进行了提交；当T1再次读取数据项A时，读到了T2修改后的A值，此时，**T1发现前后两次读取相同数据项A的值各不相同**。脏读数据异常可以被符号化表示为…R1(A)…W2(A)C2…R1(A)…。

10.2.2 数据异常

- [例10.8]：

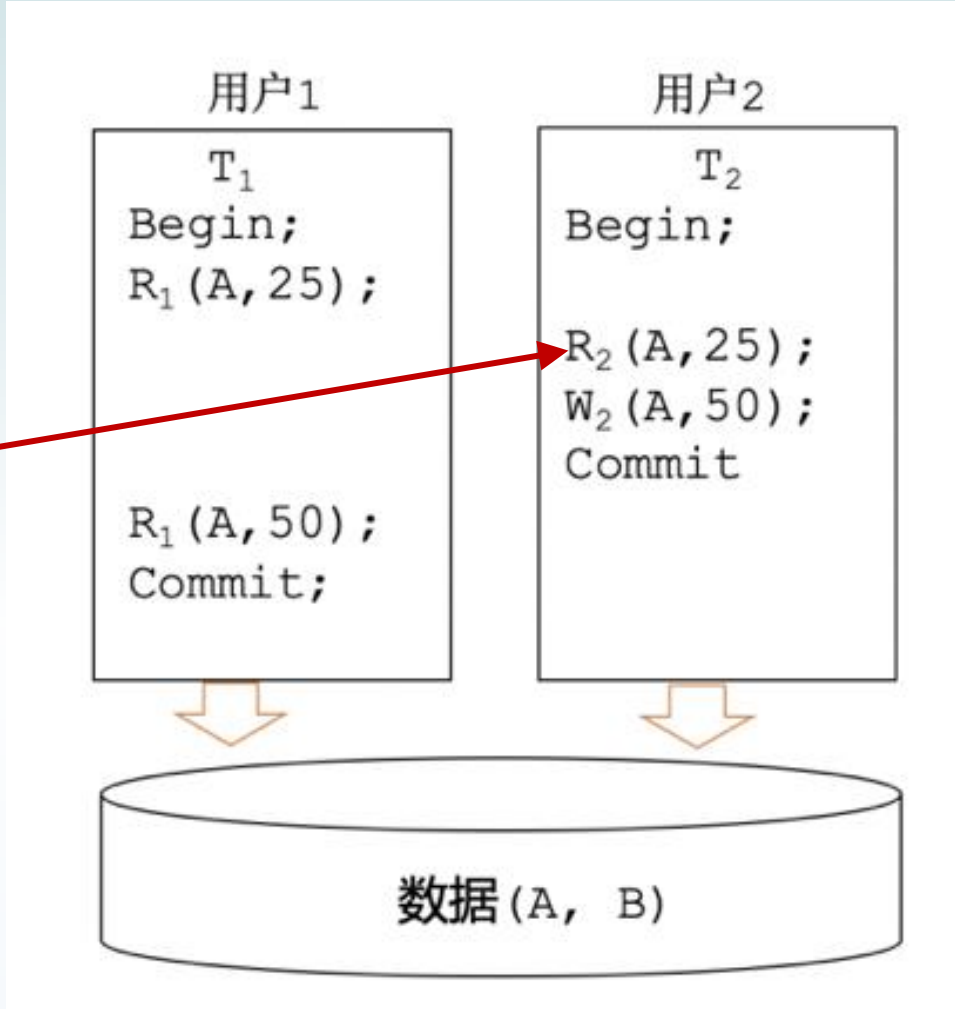
用户1发起事务T1查询了A账户余额（A=25）



10.2.2 数据异常

- [例10.8]：

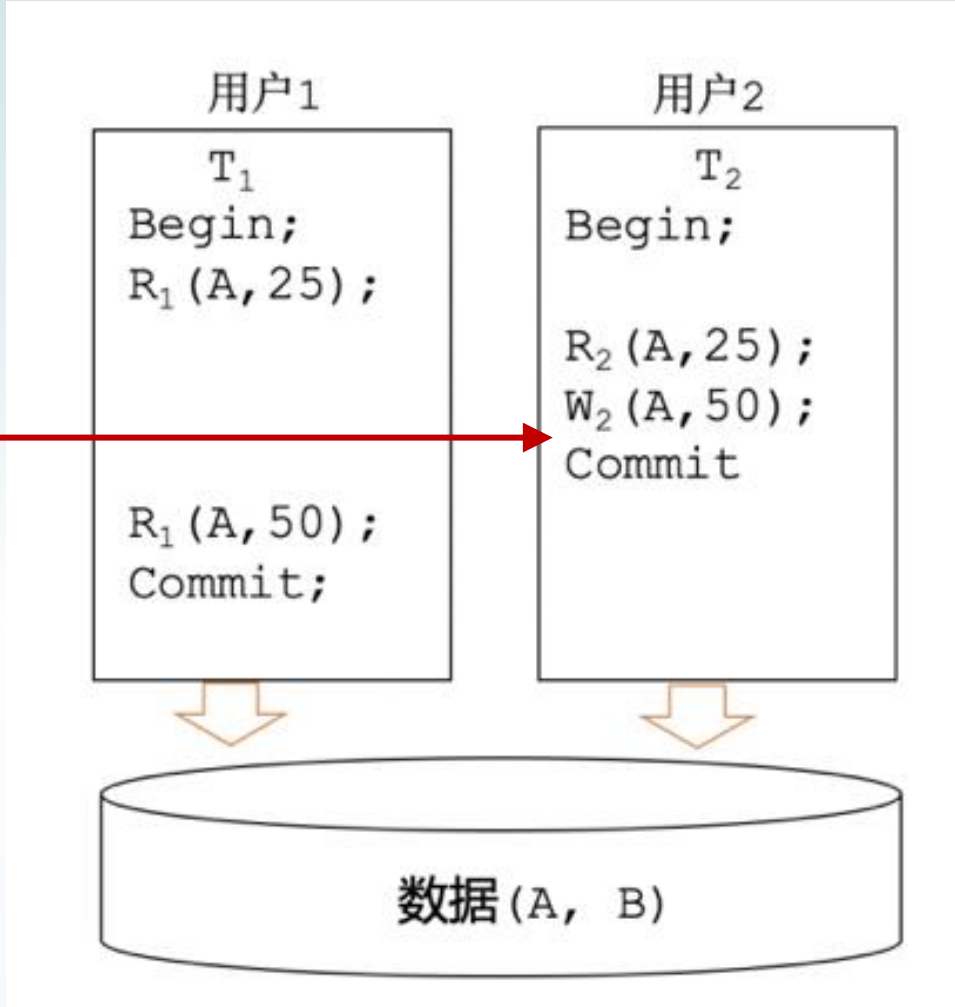
用户2发起事务T2查询了A账户余额（A=25）



10.2.2 数据异常

- [例10.8]：

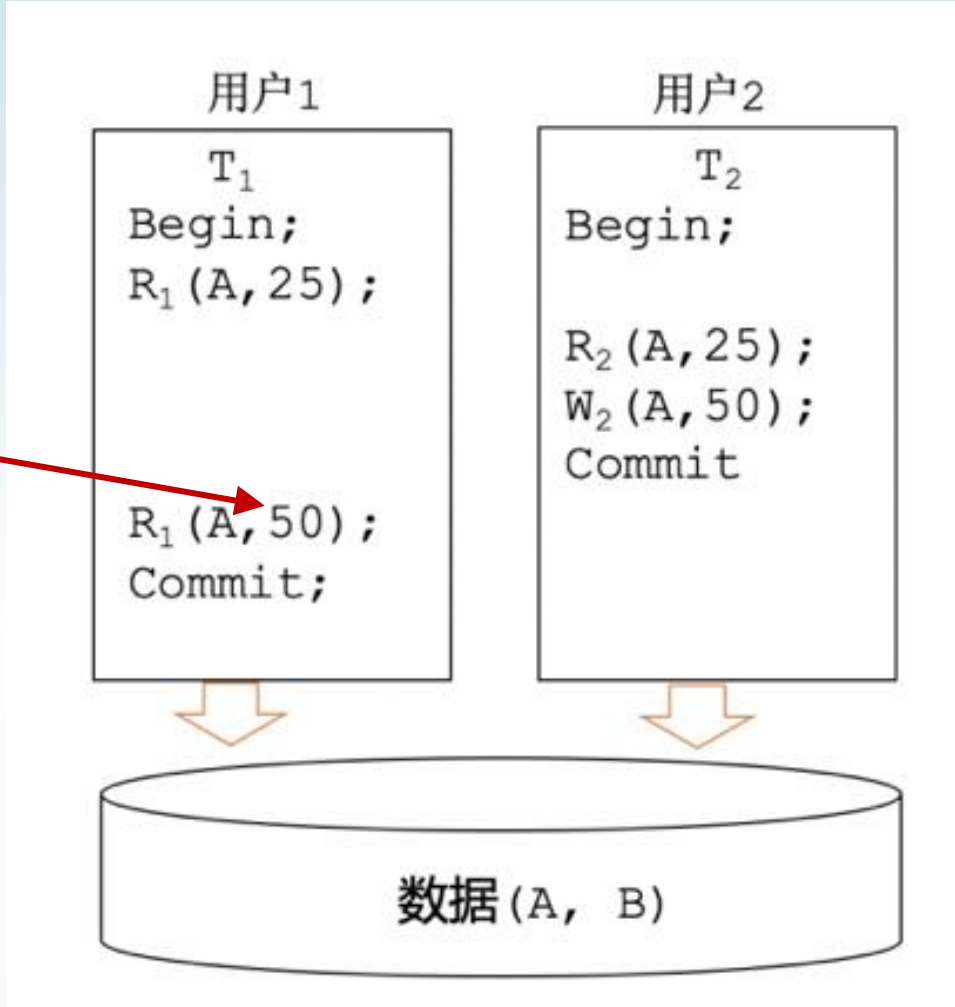
往A账户里存了25元（A更新后的值为50），并提交事务T2



10.2.2 数据异常

- [例10.8]：

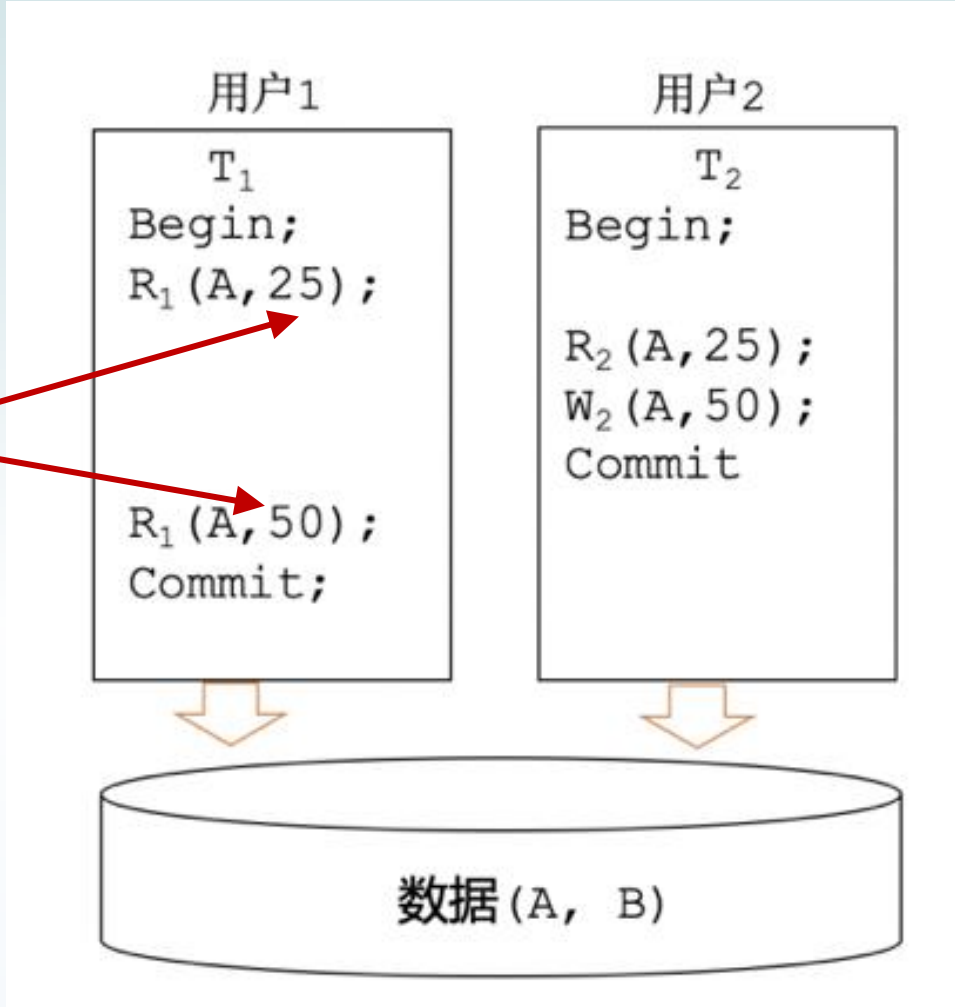
当T1再次读取数据项A时，额（更新后的A值50）



10.2.2 数据异常

- [例10.8]：

T1发现前后两次读取相同数据项A的值各不相同，从而出现了**不可重复读**的数据异常。



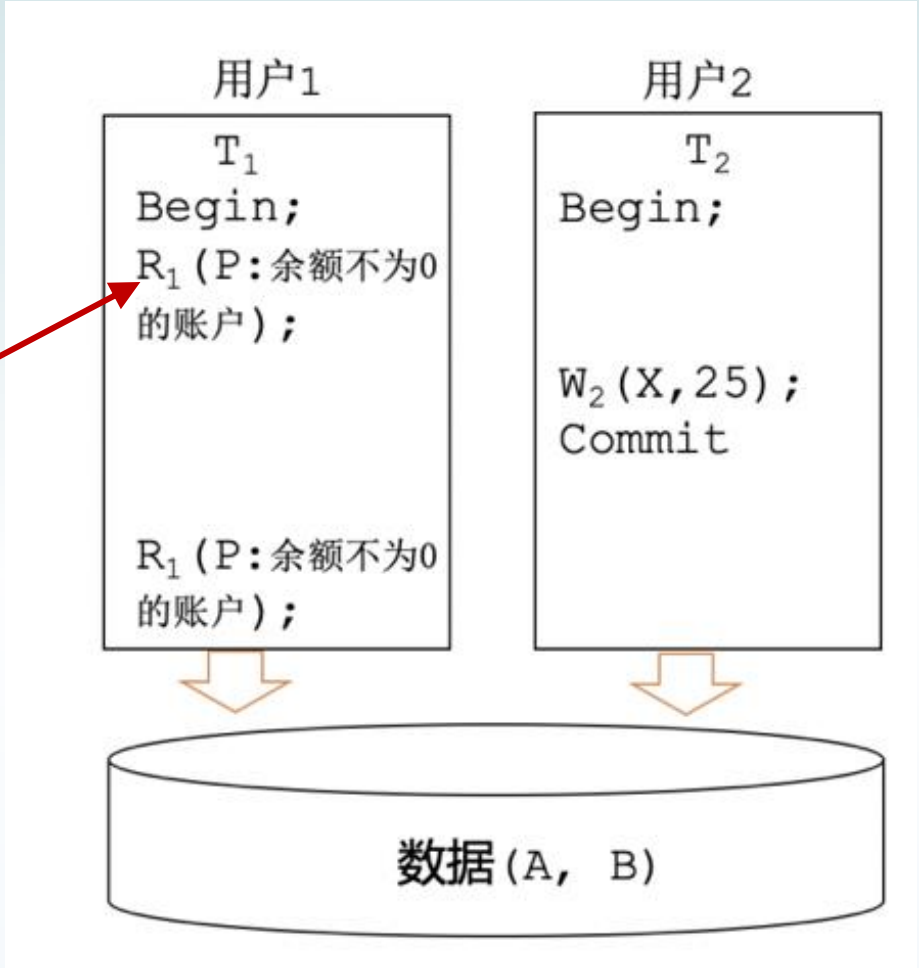
10.2.2 数据异常

- **幻读：**给定两个事务T1和T2，T1先按照某个条件（称为条件谓词）查询数据库中符合条件谓词的数据项，随后T2插入了符合上述条件谓词的数据项，当T1再次按照之前相同的条件谓词查询数据库时，**前后两次相同查询得到的结果不一样**。记R1(P)表示事务T1按照条件谓词P读取数据库中符合查询条件P的数据项集合，W2(A in P)表示事务T2插入了符合条件谓词P的数据项A。则幻读数据异常可以被符号化表示为…R1(P)…W2(A in P)…C2…R1(P)…。

10.2.2 数据异常

- [例10.9]：条件谓词P表示余额不为0的所有账户

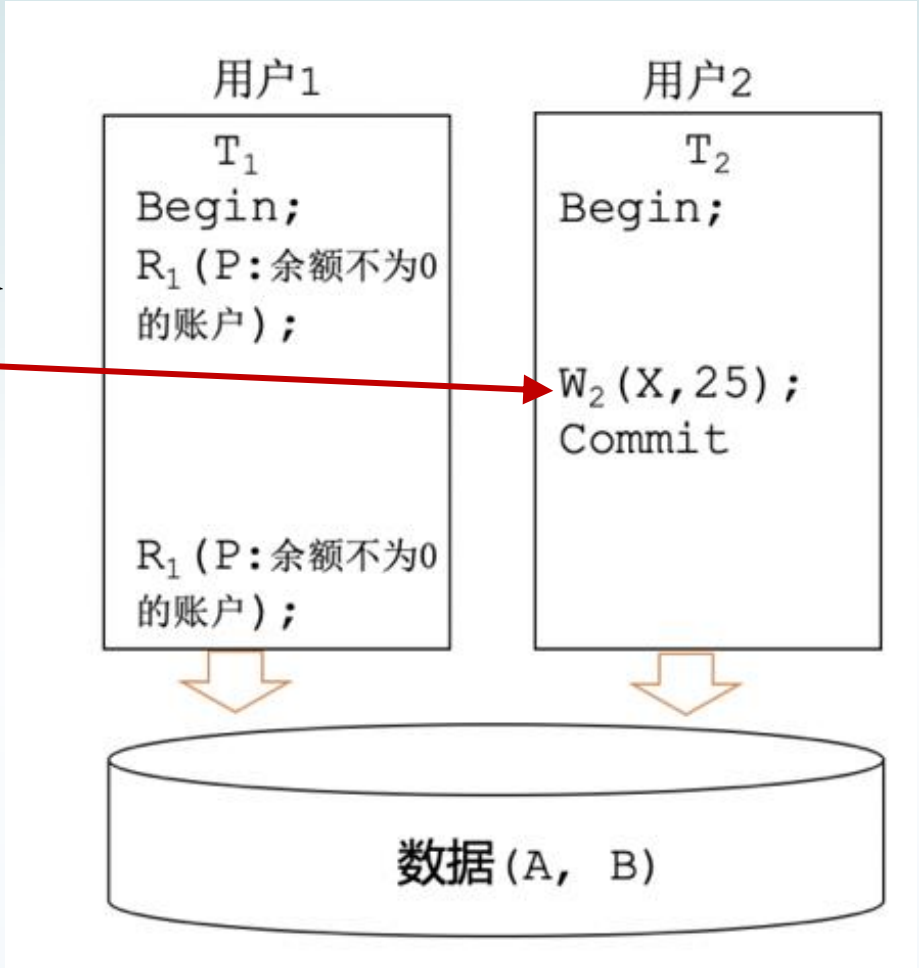
用户1发起事务T1查询了余额不为0的所有账户，返回的集合为**{A, B}**



10.2.2 数据异常

- [例10.9]：条件谓词P表示余额不为0的所有账户

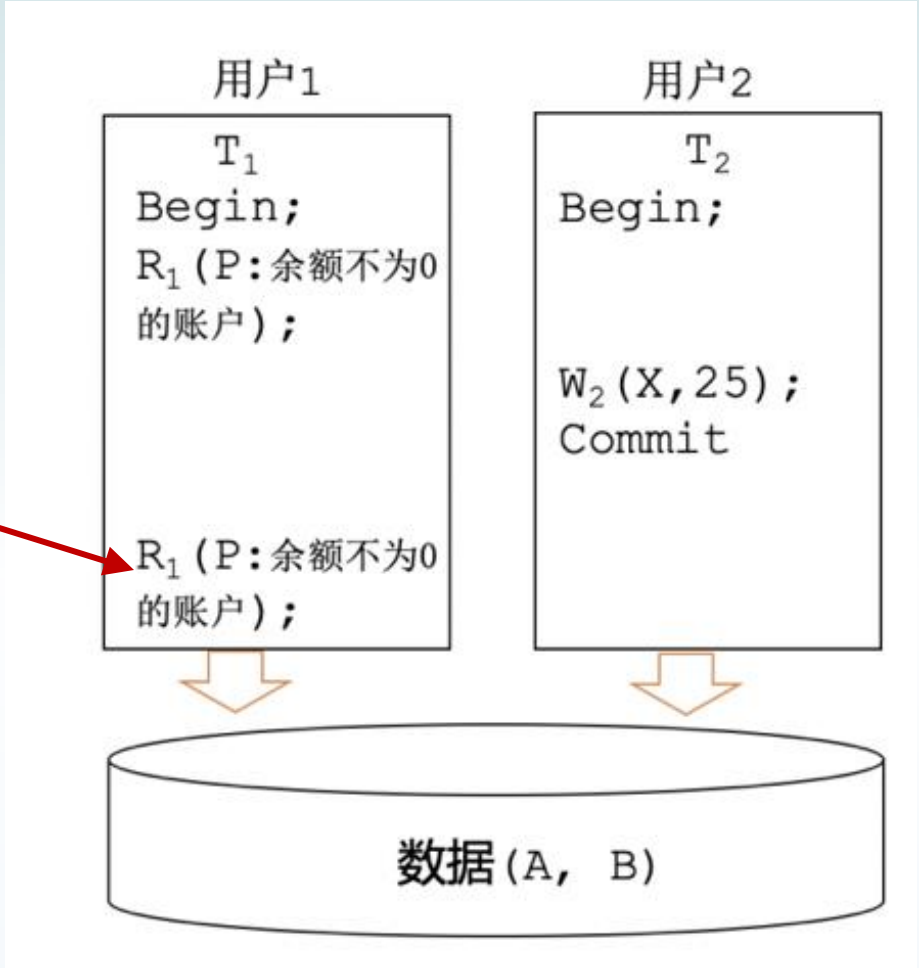
用户2创建了账户X，并存入25元并提交事务T2；



10.2.2 数据异常

- [例10.9]：条件谓词P表示余额不为0的所有账户

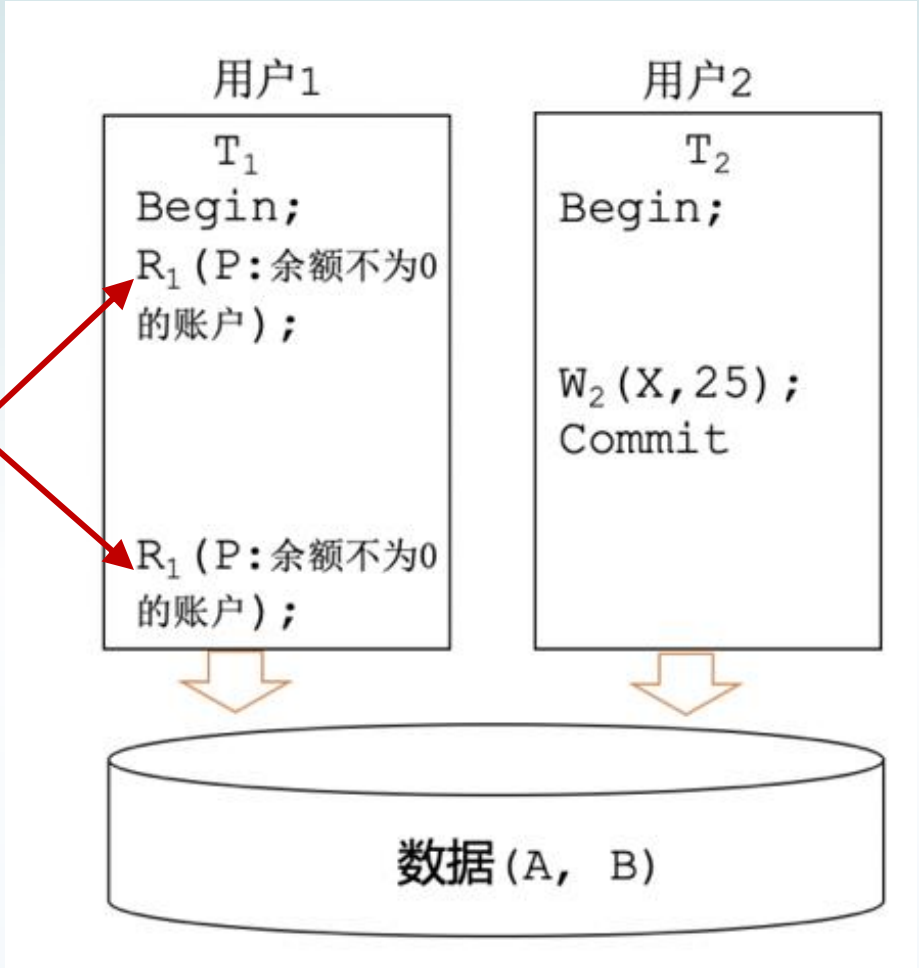
之后，当T1再次读取余额不为0的所有账户时，返回的集合为**{A,B,X}**；



10.2.2 数据异常

- [例10.9]：条件谓词P表示余额不为0的所有账户

此时可以发现，T1前后两次按照相同的条件谓词P查询得到的结果不一样，从而出现了**幻读**的数据异常



10.2.2 数据异常

- 除了ANSI/ISO SQL-92给出了四类数据异常之外，实际现实场景中还存在其它类型的数据异常，例如丢失修改（Lost update）、写偏序（Write skew）、读偏序（Read skew）等。
- 以下给出了丢失修改的**数据异常**描述及其符号化表示。

10.2.2 数据异常

- **丢失修改：**给定两个事务T1和T2，T1先读取了某一数据项A；之后，T2修改了A的值并提交；接着，T1也修改了A的值并进行了提交。T1修改了某一数据项A，在T1提交之前，T2也修改了A的值，之后，T2和T1均提交。**T1的提交导致T2的修改被T1的修改覆盖了**。丢失修改数据异常可以被符号化表示为R1(A)…W2(A)…C2…W1(A)…。

10.2.2 数据异常

- [例10.10]：

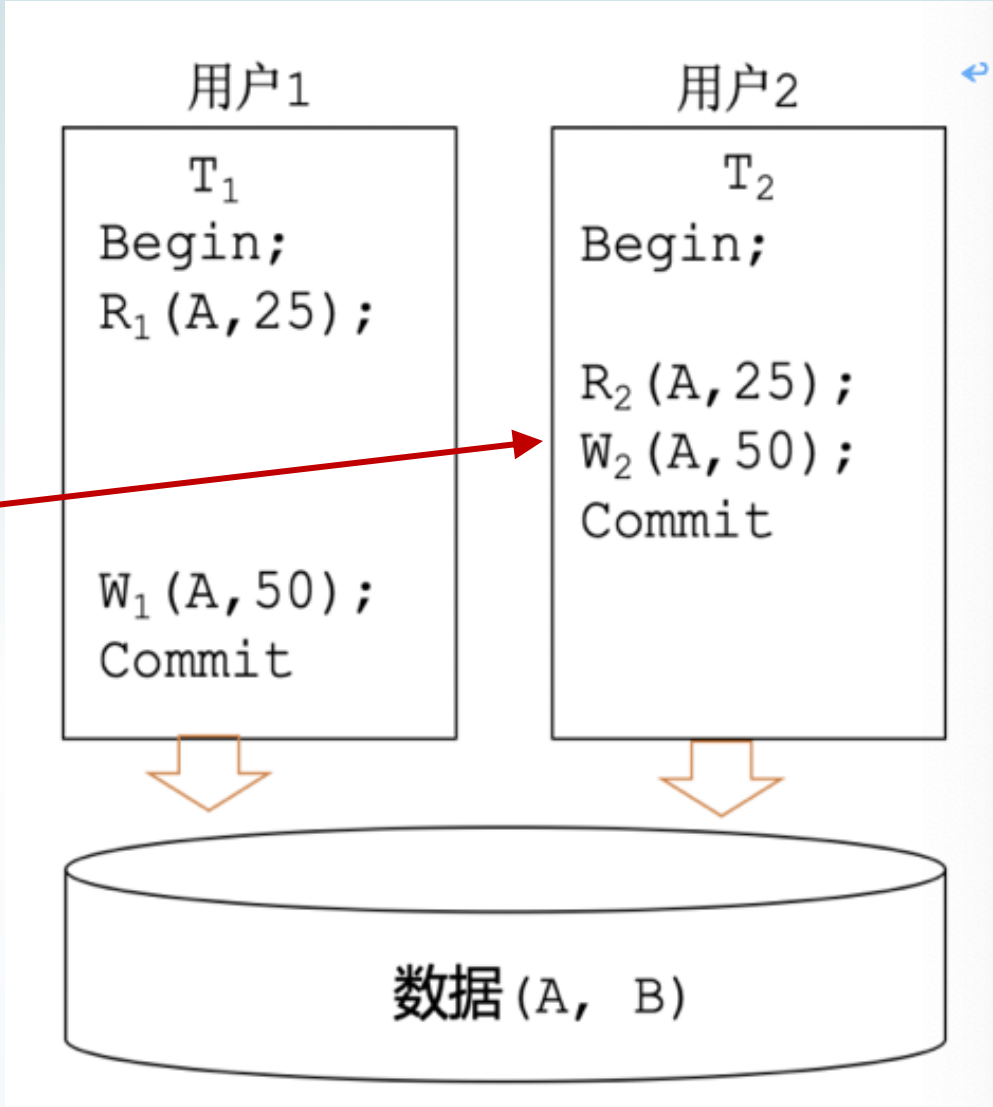
用户1发起事务T1查询了A账户余额，用户2发起事务T2也查询了A账户余额，此时A=25

10.2.2 数据异常

- [例10.10]：

用户2往A账户里存了25元，

用户2提交了事务T2



10.2.2 数据异常

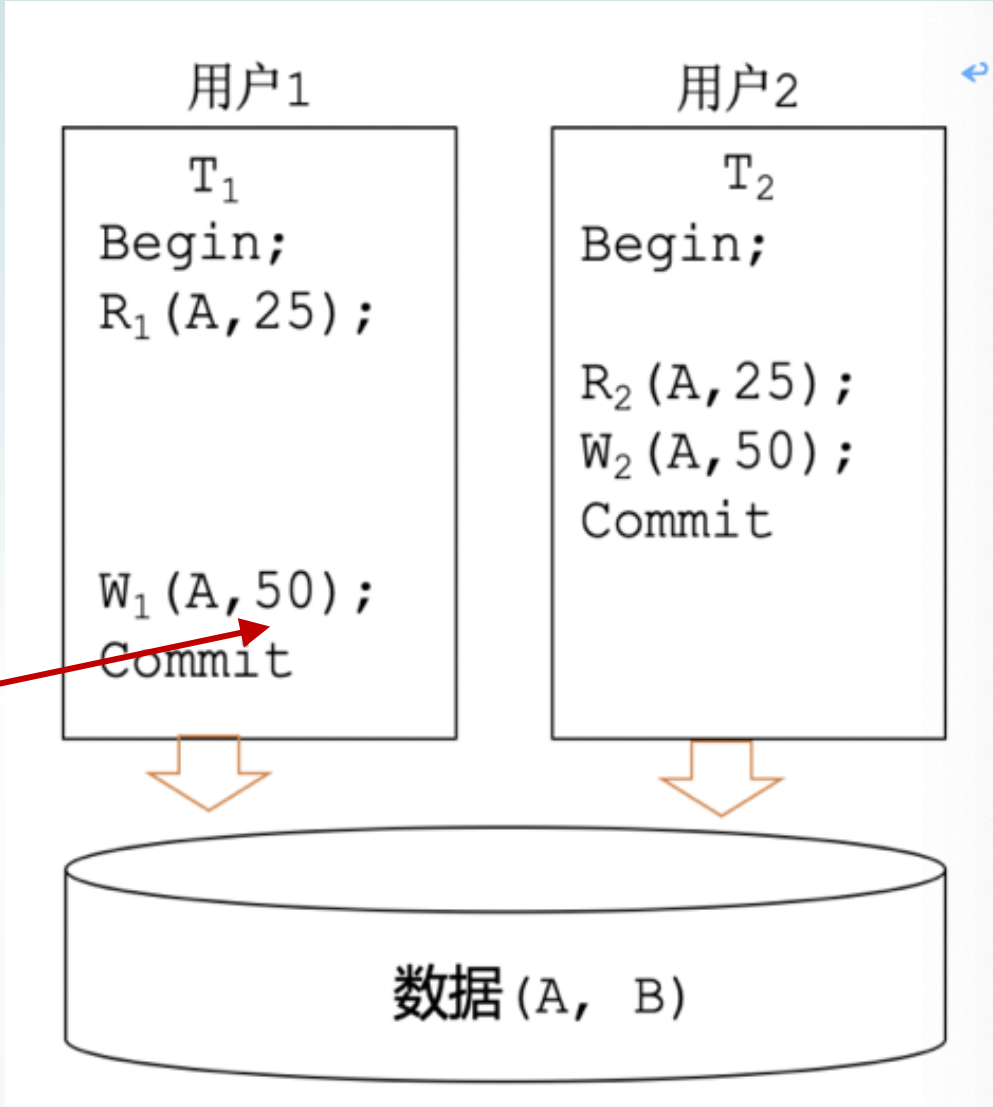
- [例10.10]：

用户1往A账户里存了25元，并提交了事务

10.2.2 数据异常

- [例10.10]：

两个用户向A账户里共存了50元，但从最终A账户的余额中可以看到，A的余额仅增加了25元



10.2.2 数据异常

- [例10.10]：

这是因为T2的修改被T1的**修改覆盖**了，从而出现了**丢失修改**的数据异常。

10.2.2 数据异常

- 讨论：丢失修改和脏写有什么联系和区别？

10.2.2 数据异常

- 讨论：丢失修改和脏写有什么联系和区别？
  - 这两类数据异常都会导致一个事务的写被另外一个事务的写所覆盖，两者的区别在于数据异常的符号化表示。特别要注意的是，在脏写的符号化表示中，两个事务的写之间，不存在前一个事务的提交操作，而丢失修改发生在后一个事务执行写操作之前，前一个事务已经执行了提交操作。

10.2 数据异常与隔离级别

- 10.2.1 事务的执行模型
- 10.2.2 数据异常
- 10.2.3 隔离级别

10.2.3 隔离级别

- 事务的隔离级别是通过如何避免相应的数据异常来定义的。在SQL标准中给出了事务的四类隔离级别，由低到高分别是：
  - 读未提交
  - 读已提交
  - 可重复读
  - 可串行化
- 所有的隔离级别都不允许出现“脏写”数据异常，但对其它数据一致性的保障程度各异。

10.2.3 隔离级别

- 读未提交
  - **“读未提交”（Read uncommitted）不允许“脏写”数据异常，但允许“脏读”、“不可重复读”和“幻读”数据异常。**因此，运行在该隔离级别上的事务被允许读取当前页面上的任何数据，而不管该数据是否是已提交事务写入的，因此，事务可能出现脏读、不可重复读和幻读的情形。

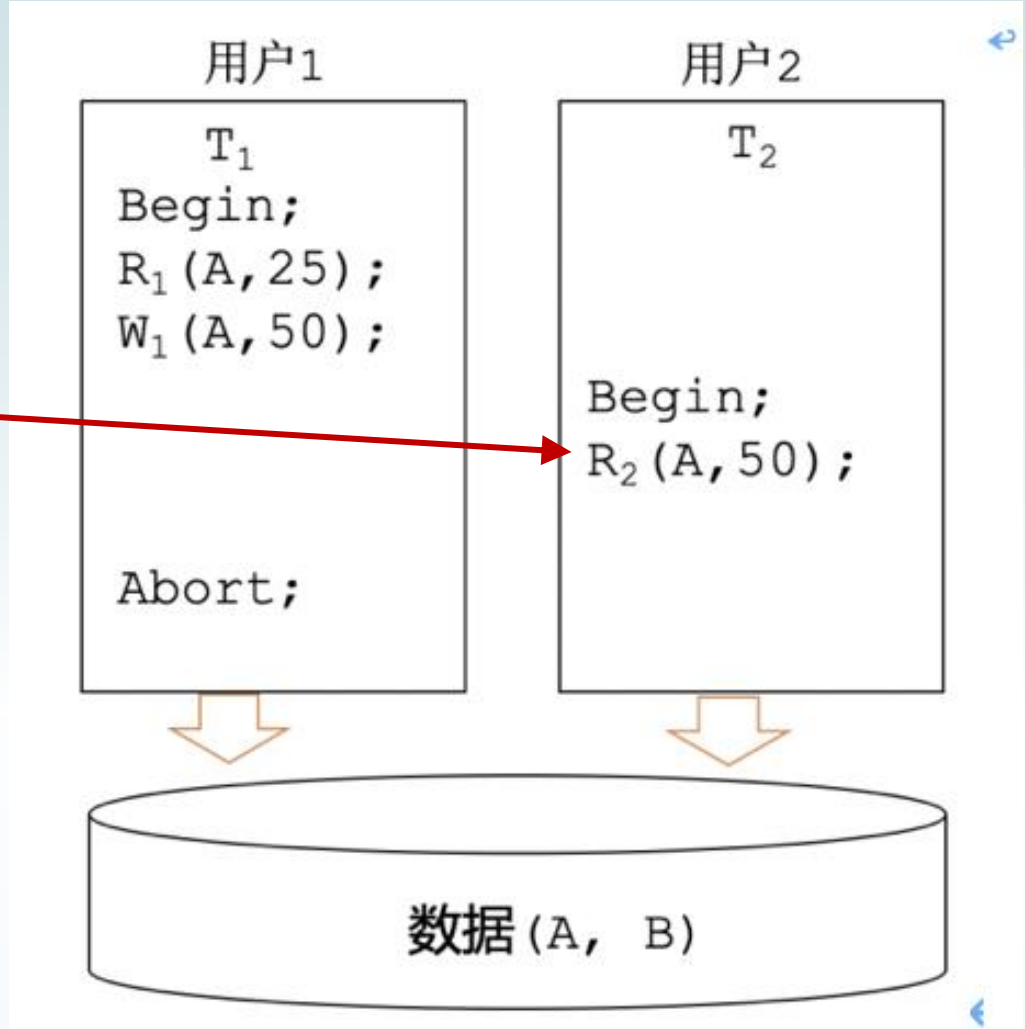
10.2.3 隔离级别

- 读已提交
  - **“读已提交”（Read committed）不允许“脏写”、“脏读”数据异常，但允许“可重复读”和“幻读”数据异常。**具体地，对于一个事务的读操作，其读取的数据项，必须是由一个已提交事务写入的，也就是说，对于未提交事务写入的数据项，其它事务是不能读到的。

10.2.3 隔离级别

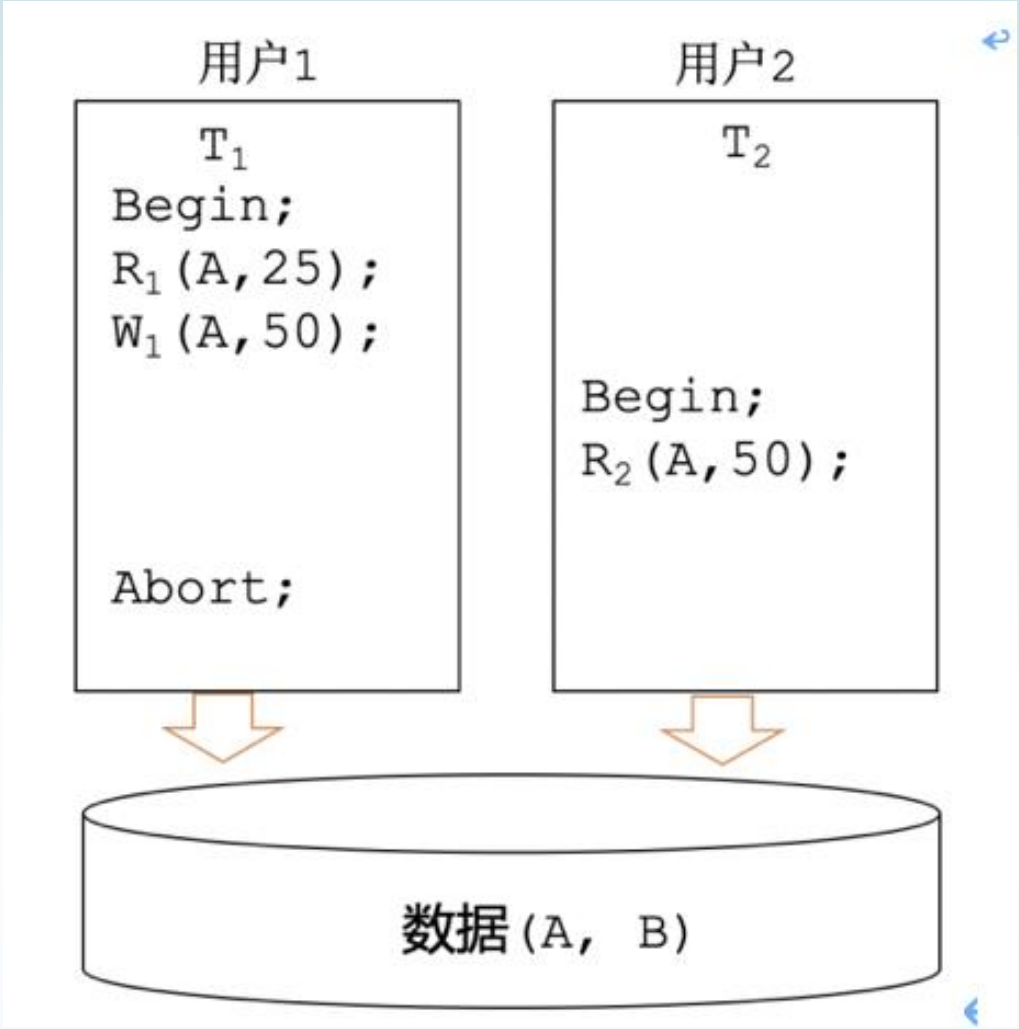
- 例如：

由于T2读取数据项A时，T1尚未提交，因此T1写入的数据项A的值，T2是不能读到的。



10.2.3 隔离级别

- 例如：



在实际的系统实现中，“不能读到”表现为两种形式：

1. 要么T2等待，直到T1提交；
2. 要么T2读到A的旧值。

显然，运行在该隔离级别上的事务**可以有效避免读“脏”数据，但是它不能保证可重复读和不幻读**。

10.2.3 隔离级别

- 可重复读
  - **“可重复读”（Repeatable read）要求避免“脏写”、“脏读”、“不可重复读”数据异常,但允许“幻读”数据异常。**具体地，一个事务如果再次访问同一数据，与此前访问相比，数据不会发生改变。

10.2.3 隔离级别

- 可串性化
  - “可串行化”（Serializable）是最高的事务隔离级别，在该级别下，事务执行顺序是可串行化的，可以避免脏写、脏读、不可重复读和幻读等所有数据异常。

10.2.3 隔离级别

- 下表给出了事务四个隔离级别与数据异常的关系。

事务隔离级别	脏写	脏读	不可重复读	幻读
读未提交	×	√	√	√
读已提交	×	×	√	√
可重复读	×	×	×	√
可串性化	×	×	×	×

10.2.3 隔离级别

- **注意**，“可串行化”隔离级别的定义是给定一组事务的调度，其执行结果等价于某一个**串行**这行这组事务的结果。因此，在该级别下，可以规避脏写、脏读、不可重复读和幻读这些数据异常。但是，给定一组事务的调度，该调度避免出现上述几类数据异常，并不能保证该调度是可串行化的。

第10章 事务处理概述

- 10.1 事务基本概念
- 10.2 事务异常与隔离级别
- 10.3 正确的调度
- 10.4 本章小结

10.3 正确的调度

- 10.3.1 调度与串行调度
- 10.3.2 可串性化调度
- 10.3.3 冲突可串性化调度
- 10.3.4 基于优先图的冲突可串行化验证

10.3.1 调度与串行调度

- 在数据库管理系统中，可能会同时存在多个事务处理请求，这组事务的操作在系统中的执行顺序，称为**事务的调度**。
- **[定义10.1]**：给定n个事务，这n个事务上的一个调度S指的是n个事务的所有操作的一个序列，这个序列表示这些操作的执行顺序，并且满足：对于其中的每个事务T，如果操作Oi在T中先于操作Oj执行，则在调度S中的操作Oi也先于操作Oj执行。
- 从定义10.1可以看出，任何一个合法的调度必须保证两点：
  - 1. 调度必须包含了所有事务的所有操作；
  - 2. 一个事务中所有操作的顺序在该调度中必须保持不变，但是不同事务的操作可以交叉执行。

10.3.1 调度与串行调度

- 例：假设账户A和B的初始值分别是25元和5元，事务T1从账户A转5元给账户B，事务T2从账户B转5元给账户A。以下给出了T1和T2事务的两个调度S1和S2。

调度S1

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
R1(B,5)	
W1(B,10)	
Commit	
	Begin
	R2(B,10)
	W2(B,5)
	R2(A,20)
	W2(A,25)
	Commit

调度S2

T1	T2
	Begin
	R1(B,5)
	W1(B,0)
	R1(A,25)
	W2(A,30)
	Commit
Begin	
R2(A,30)	
W2(A,25)	
R2(B,0)	
W1(B,5)	
Commit	

10.3.1 调度与串行调度

调度S1

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
R1(B,5)	
W1(B,10)	
Commit	
	Begin
	R2(B,10)
	W2(B,5)
	R2(A,20)
	W2(A,25)
	Commit

调度S2

T1	T2
	Begin
	R1(B,5)
	W1(B,0)
	R1(A,25)
	W2(A,30)
	Commit
Begin	
R2(A,30)	
W2(A,25)	
R2(B,0)	
W1(B,5)	
Commit	

调度S1执行完后，账户A和B的余额分别为25和5

10.3.1 调度与串行调度

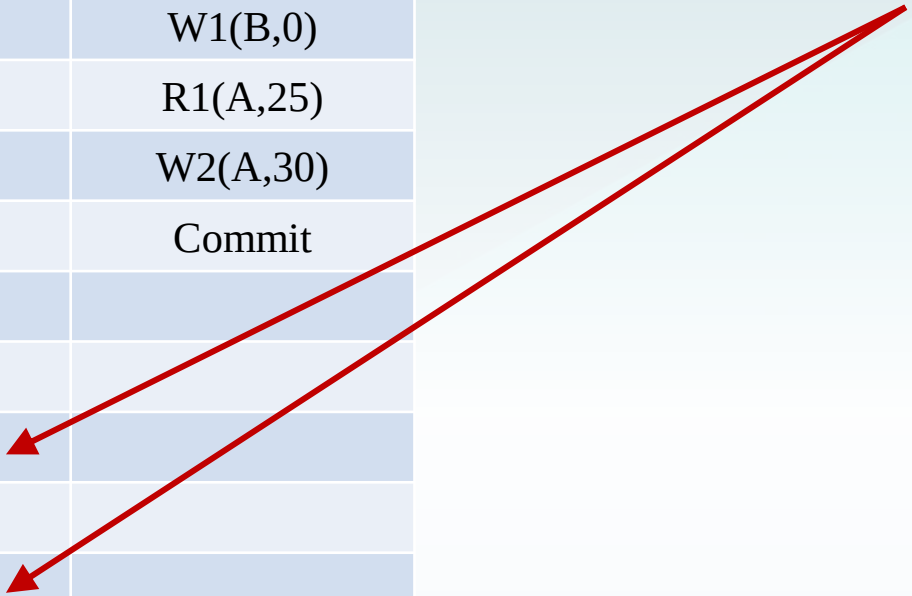
调度S1

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
R1(B,5)	
W1(B,10)	
Commit	
	Begin
	R2(B,10)
	W2(B,5)
	R2(A,20)
	W2(A,25)
	Commit

调度S2

T1	T2
	Begin
	R1(B,5)
	W1(B,0)
	R1(A,25)
	W2(A,30)
	Commit
Begin	
R2(A,30)	
W2(A,25)	
R2(B,0)	
W1(B,5)	
Commit	

调度S2执行完后，账户A和B的余额分别为25和5



10.3.1 调度与串行调度

调度S1

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
R1(B,5)	
W1(B,10)	
Commit	
	Begin
	R2(B,10)
	W2(B,5)
	R2(A,20)
	W2(A,25)
	Commit

调度S2

T1	T2
	Begin
	R1(B,5)
	W1(B,0)
	R1(A,25)
	W2(A,30)
	Commit
Begin	
R2(A,30)	
W2(A,25)	
R2(B,0)	
W1(B,5)	
Commit	

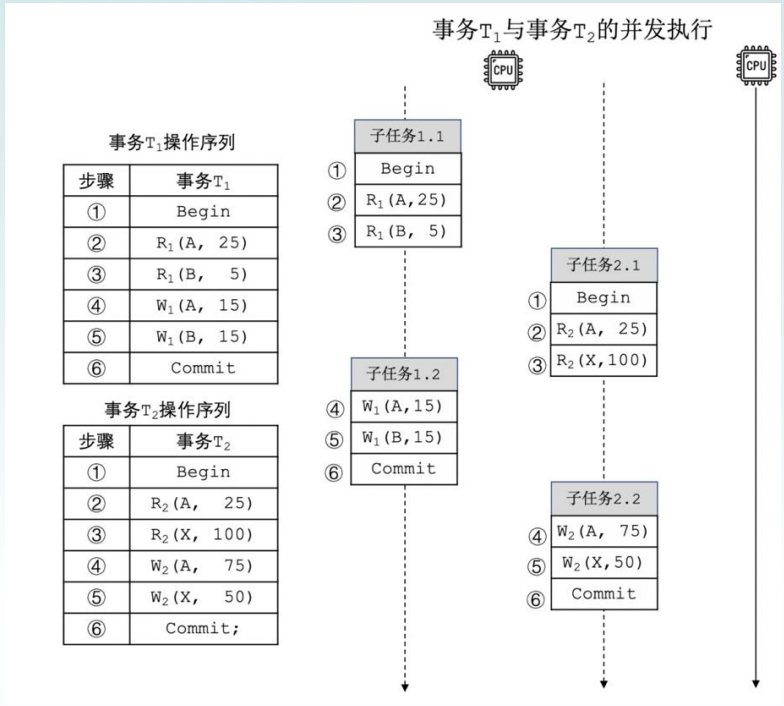
调度S1和S2都是**一个事务的所有操作都执行完才执行另一个事务的所有操作**，这样的调度被称为**串行调度**。

10.3.1 调度与串行调度

- 当多个事务并发执行时，操作系统可能先对一个事务执行一个时间片，然后切换上下文环境，对第二个事务再执行一个时间片，这样来自不同事务的各个操作可能是交叉执行的，这个调度称之为**并发调度**。

10.3.1 调度与串行调度

- 以下给出了T1和T2事务的两个并发调度S3和S4：



调度S3

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
	Begin
	R2(B,5)
R1(B,5)	
	W2(B,0)
W1(B,10)	
Commit	
	R2(A,20)
	W2(A,25)
	Commit

调度S4

T1	T2
Begin	
R1(A,25)	
	Begin
	R2(B,5)
	W2(B,0)
	R2(A,25)
	W2(A,30)
	Commit
W1(A,20)	
R1(B,0)	
W1(B,5)	
Commit	

10.3.1 调度与串行调度

调度S3

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
	Begin
	R2(B,5)
R1(B,5)	
	W2(B,0)
W1(B,10)	
Commit	
	R2(A,20)
	W2(A,25)
	Commit

调度S4

T1	T2
Begin	
R1(A,25)	
	Begin
	R2(B,5)
	W2(B,0)
	R2(A,25)
	W2(A,30)
	Commit
W1(A,20)	
R1(B,0)	
W1(B,5)	
Commit	

并发调度不一定是正确的调度。很可能会造成数据库处于不一致的状态。

10.3.1 调度与串行调度

调度S3

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
	Begin
	R2(B,5)
R1(B,5)	
	W2(B,0)
W1(B,10)	
Commit	
	R2(A,20)
	W2(A,25)
	Commit

调度S4

T1	T2
Begin	
R1(A,25)	
	Begin
	R2(B,5)
	W2(B,0)
	R2(A,25)
	W2(A,30)
	Commit
W1(A,20)	
R1(B,0)	
W1(B,5)	
Commit	

调度S3执行完后，账户A和B的余额分别为25和10 。

10.3.1 调度与串行调度

调度S3

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
	Begin
	R2(B,5)
R1(B,5)	
	W2(B,0)
W1(B,10)	
Commit	
	R2(A,20)
	W2(A,25)
	Commit

调度S4

T1	T2
Begin	
R1(A,25)	
	Begin
	R2(B,5)
	W2(B,0)
	R2(A,25)
	W2(A,30)
	Commit
W1(A,20)	
R1(B,0)	
W1(B,5)	
Commit	

调度S4执行完后，账户A和B的余额分别为20和5。

10.3.1 调度与串行调度

调度S3

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
	Begin
	R2(B,5)
R1(B,5)	
	W2(B,0)
W1(B,10)	
Commit	
	R2(A,20)
	W2(A,25)
	Commit

调度S4

T1	T2
Begin	
R1(A,25)	
	Begin
	R2(B,5)
	W2(B,0)
	R2(A,25)
	W2(A,30)
	Commit
W1(A,20)	
R1(B,0)	
W1(B,5)	
Commit	

按照事务一致性的要求，转账前后账户A和B的**余额总和不变**，均为**30**元。可以看到，执行调度S3和S4，会使数据库中的数据处于**不一致**的状态。

10.3 正确的调度

- 10.3.1 调度与串行调度
- 10.3.2 可串性化调度
- 10.3.3 冲突可串性化调度
- 10.3.4 基于优先图的冲突可串行化验证

10.3.2 可串性化调度

- 数据库管理系统对并发事务不同的调度可能会产生不同的结果，那么什么样的调度是正确的呢？显然，串行调度是正确的。执行结果等价于串行调度的调度也是正确的，这样的调度叫做**可串行化调度**。
- **定义[10.2]：**多个事务的并发执行是正确的，当且仅当其执行的效果与按某一次序串行地执行这些事务时的效果相同，称这种调度策略为**可串行化（Serializable）调度。**
- 可串行性（serializability）是并发事务正确调度的**准则**。按这个准则规定，一个给定的并发调度，当且仅当它是可串行化的，才认为是正确调度。

10.3.2 可串性化调度

- S1和S2的调度可以表示为：

  S1：R1(A,25)W1(A,20)R1(B,5)W1(B,10)C1R2(B,10)**W2(B,5)**R2(A,20)**W2(A,25)**C2

  S2：R2(B,5)W2(B,0)R2(A,25)W2(A,30)C2R1(A,30)**W1(A,25)**R1(B,0)**W1(B,5)**C1
- 在S1中，S1执行了T1事务的所有操作之后才执行T2的所有操作，因此，S1等价的串行顺序为T1T2；
- 在S2中，S2执行了T2事务的所有操作之后才执行T1的所有操作，因此，S2等价的串行顺序为T2T1。
- 标记下划线的操作指的是数据项的最终状态，例如在调度S1中，数据项B和A最终的值分别是由操作**W2(B,5)**和操作**W2(A,25)**写入的。

10.3.2 可串性化调度

- 给定一个并发调度，当且仅当它是可串行化的，才认为是正确调度。那如何评价一个调度是可串行化调度？
  - 理论上，给定一组事务的一个并发调度，**只有将该组事务所有可能的串行顺序的执行结果都枚举出来，然后将该并发调度的执行结果与上述所有枚举的结果逐一比较，如果跟其中某一次串行地执行这些事务时的结果相同，则说明该调度是可串行化调度**，也就是说该调度是正确的。然而，在实际的系统实现中，由于执行事务量巨大，枚举事务所有可能的串行执行结果不具备可操作性，

10.3 正确的调度

- 10.3.1 调度与串行调度
- 10.3.2 可串性化调度
- 10.3.3 冲突可串性化调度
- 10.3.4 基于优先图的冲突可串行化验证

10.3.3 冲突可串性化调度

- 冲突可串行化是可串行化的一个子集，即一组事务的一个并发调度S如果是冲突可串行化，则S一定是可串行化的，因此S是正确的调度。
- 冲突可串行化的核心在于“**冲突**”，在形式化给出其定义之前，先给出“冲突”的定义。给定一个调度S中任意来自不同事务的两个操作Oi，Oj，如果Oi和Oj操作相同的数据项，并且其中至少一个操作是写操作，则称Oi和Oj是冲突操作。可以看出，冲突操作只有三种可能，分别是**读写冲突、写读冲突**和**写写冲突**。

10.3.3 冲突可串性化调度

- 基于冲突的定义，在调度S3中，存在：
  - **读写冲突**：R1(A,25)W2(A,25)、R1(B,5)W2(B,0)；
  - **写读冲突**：W1(A,20)R2(A,20)；
  - **写写冲突**：W1(A,20)W2(A,25)、W2(B,0)W1(B,10) 。

调度S3

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
	Begin
	R2(B,5)
R1(B,5)	
	W2(B,0)
W1(B,10)	
Commit	
	R2(A,20)
	W2(A,25)
	Commit

10.3.3 冲突可串性化调度

- 对于调度S中来自不同事务两个不冲突的连续操作，如果交换这两个操作的顺序产生一个新的调度，则不管初始系统状态如何，这两个调度都产生**相同的最终系统状态**。例如：在如下调度S7：

  S7：R1(A)W1(A)R2(A)W2(A)R1(B)W1(B)R2(B)W2(B)

- 由于交换W2(A)R1(B)，得到新的调度S7’，S7和S7’是等价的：

  S7’: R1(A)W1(A)R2(A)R1(B)W2(A)W1(B)R2(B)W2(B)

10.3.3 冲突可串性化调度

- **定义[10.3]：**如果一个调度S通过一系列交换来自不同事务的两个不冲突的连续操作等到新的调度S’，则称S和S’是**冲突等价**的。
- **[定义10.4]：**如果一个调度S冲突等价于一个串行调度，则称S是**冲突可串行化的**。

R1(A)W1(A)R2(A)W2(A)R1(B)W1(B)R2(B)W2(B)

→ R1(A)W1(A)R2(A)R1(B)W2(A)W1(B)R2(B)W2(B)

→ R1(A)W1(A)R1(B)R2(A)W2(A)W1(B)R2(B)W2(B)

→ R1(A)W1(A)R1(B)R2(A)W1(B)W2(A)R2(B)W2(B)

→ R1(A)W1(A)R1(B)W1(B) R2(A)W2(A)R2(B)W2(B)

- 如上所示，S7经过一系列冲突等价交换，转换成串行调度T1T2,因此，S7是一个可串行化调度。

10.3.3 冲突可串性化调度

- 注意：**冲突可串行化调度是可串行化调度的充分条件，不是必要条件。还有不满足冲突可串行化条件的可串行化调度。**
- 例如，有三个事务T1: W1(B)W1(A)，T2: W2(B)W2(A)，T3：W3(A)。

  调度S8：W1(B)W1(A)W2(B)W2(A)W3(A)是一个串行调度。

  调度S9：W1(B)W2(B)W2(A)W1(A)W3(A)

直观上看，T1和T2写入的A值是无效的，因为T3覆盖了它们的值，因此S9执行的结果与调度S8相同，B的值都是由T2写入的，A的值都是由T3写入的，故调度S9是可串行化的。但是S9调度中既不能交换W1(B)W2(B)，也不能交换W2(A)W1(A)，因此不能通过交换将S9转换成串行调度。也就是说，S9是可串行化的，但不是冲突可串行化的。

10.3 正确的调度

- 10.3.1 调度与串行调度
- 10.3.2 可串性化调度
- 10.3.3 冲突可串性化调度
- 10.3.4 基于优先图的冲突可串行化验证

10.3.4 基于优先图的冲突可串行化验证

- **优先图（Precedence graph）**是冲突可串行化调度的常用检验方法：
  - 给定一组事务的调度S，优先图中的每一个顶点，对应于调度中的每一个事务。不失一般性，记事务Ti对应优先图中的顶点为Vi。对于S中的任意两个事务Ti，Tj，如果存在Ri(A)Wj(A)或Wi(A)Rj(A)或Wi(A)Wj(A)冲突，则优先图中Ti和Tj对应的顶点Vi和Vj，构建一条Vi指向Vj的边，记为Vi → Vj，边的标签为两个事务之间存在的写读、读写、写写冲突。

10.3.4 基于优先图的冲突可串行化验证

- **[定理10.1]：**基于调度S构建的优先图中，如果存在环，则S不是一个冲突可串行化的调度；否则，S是一个冲突可串行化的调度。

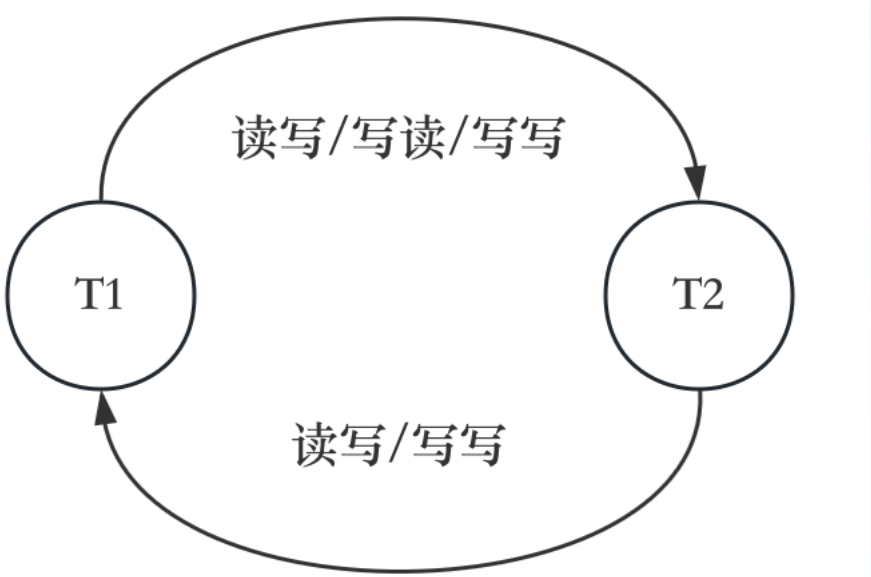
10.3.4 基于优先图的冲突可串行化验证

- [例10.11]：在调度S3中，共包含两个事务T1和T2，可以看到，对应的优先图中存在环（如右图所示），S3不是一个冲突可串行化的调度。

调度S3

T1	T2
Begin	
R1(A,25)	
W1(A,20)	
	Begin
	R2(B,5)
R1(B,5)	
	W2(B,0)
W1(B,10)	
Commit	
	R2(A,20)
	W2(A,25)
	Commit

- **读写冲突**：R1(A,25)W2(A,25)、R1(B,5)W2(B,0)；
- **写读冲突**：W1(A,20)R2(A,20)；
- **写写冲突**：W1(A,20)W2(A,25)、W2(B,0)W1(B,10) 。



第10章 事务处理概述

- 10.1 事务基本概念
- 10.2 事务异常与隔离级别
- 10.3 正确的调度
- 10.4 本章小结

10.4 本章小结

- **事务处理**是数据库管理系统的核心技术之一，是数据库管理系统从实验室的原型系统真正走向市场、支撑关键行业核心业务的基石。
  - 1. 本章首先对事务本身进行了**定义**，系统地讲解了事务的**ACID特性**，即事务的原子性、一致性、隔离性和持续性。通过给出事务的符号化表示；
  - 2. 然后介绍了事务在数据库管理系统中的**执行模型**，介绍了在缺少合理调度情况下系统可能出现的各类**数据异常**，并给出了SQL-92中四类数据异常的符号化表示及其对应的隔离级别。
  - 3. 最后，介绍了什么是**正确的调度**，以及如何**判断**一个调度是冲突可串行化调度。

第十章作业

- 课后习题2-6