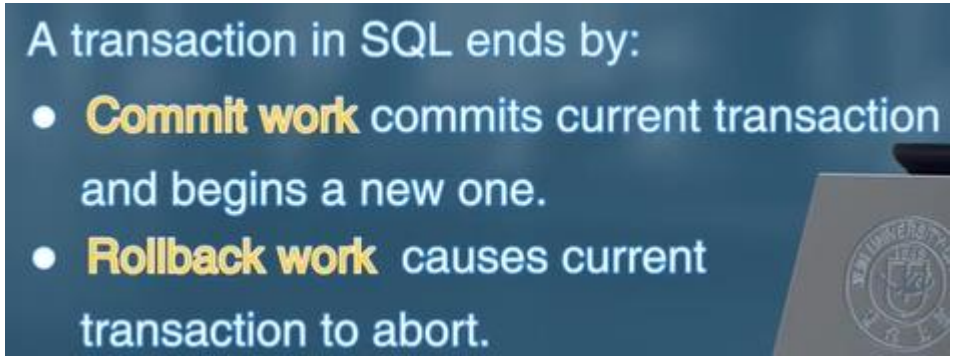
第十四章 事务

一、事务的概念

- **事务**：访问并可能更新各种数据项的一个程序执行单元，用形如***begin transaction*** 和***end transaction***语句界定
- 事务的**性质**：
  - **原子性**：事物是不可分割的，要么执行全部内容，要么根本不执行
  - **一致性**：在没有其他事务并发执行的情况下，保持数据库的一致性
    - Eg.A给B转账，转账之前的A与B钱数的总和=转账之后A与B钱数的总和
  - **隔离性**：每个事务都感觉不到系统中有其他事务在并发地执行，某一个事务在执行的过程当中，不会被其他事务读取到中间结果。
  - **持久性**：一个事务成功完成之后，它对数据库的改变必须是永久的，即使出现系统故障
- 事务执行结束标志：



二、事务的状态

- 事务必须处于以下状态之一：
  - 活动的(active)：初始状态，事务执行时处于这个状态
  - 部分提交的(partially committed)：有一部分语句执行完毕
  - 失败的(failed)：发现正常的执行不能继续后
  - 中止的(aborted)：事务回滚并且数据库已经恢复到事务开始执行前的状态后
  - 提交的(committed)：成功完成后

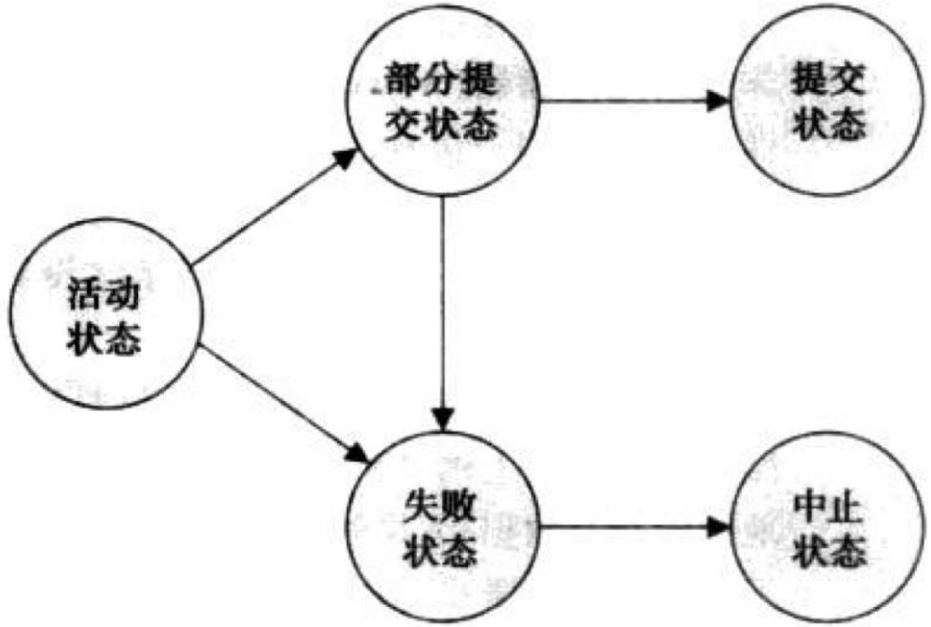
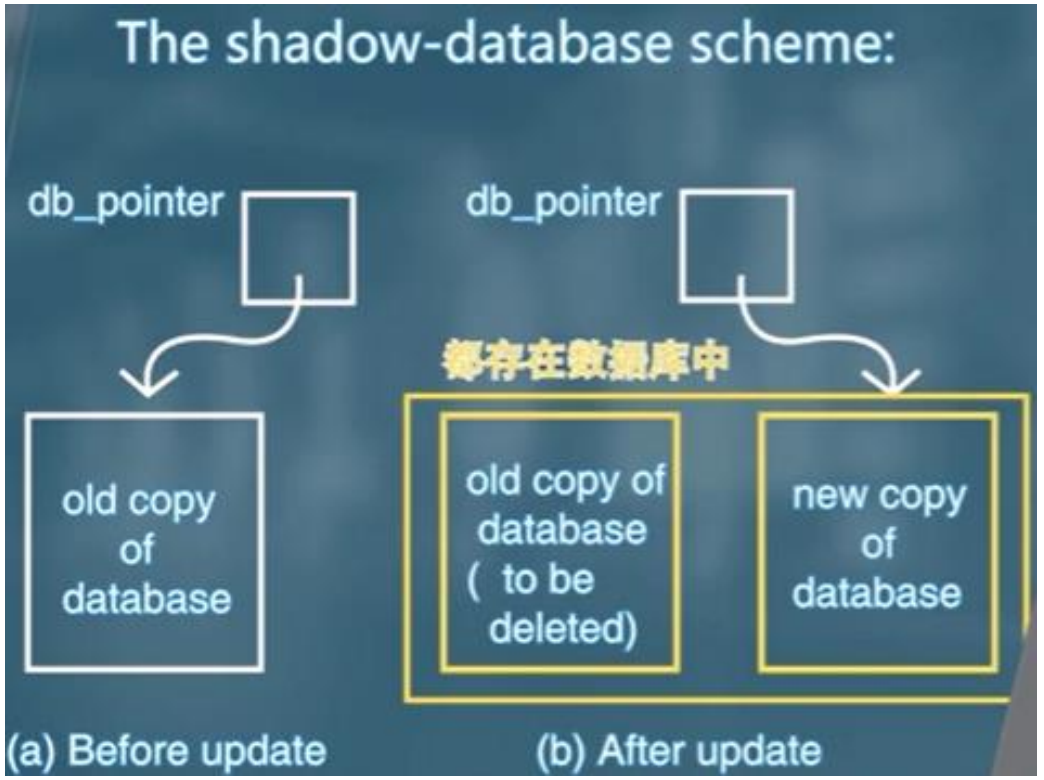


图 14-1　事务状态图

二、事务的状态

- 可恢复管理机制：事务在执行的过程中，可能出现部分提交状态或者失败状态。如果出现失败状态，要将部分提交的数据回滚到原始状态。
- 如何实现可恢复管理机制？
  - 影子数据库->创建一个副本，通过修改指针的指向来访问旧值和新值，不适合并发的事务使用



三、调度

1.调度：多个事务的读写操作按时间排序的执行序列

- 调度中每个事务的读写操作保持原来的顺序

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

调度1：一个串行调度
T_2跟在T_1之后

T_1	T_2
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	

调度2：一个串行调度
T_1跟在T_2之后

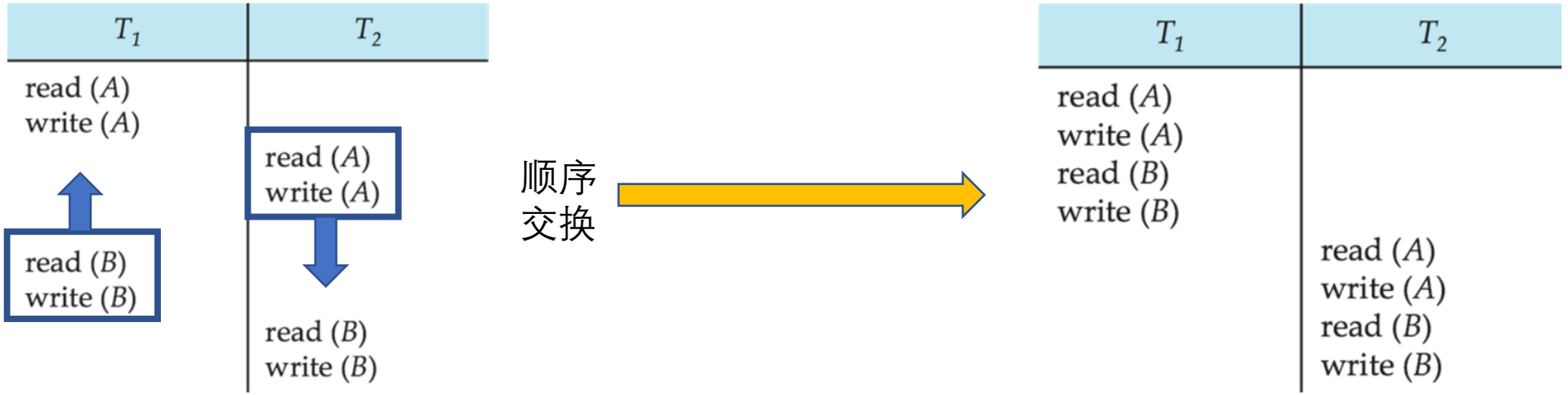
T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

调度3：一个并发调度
结果与调度1等价

三、调度

2.可串行化调度:

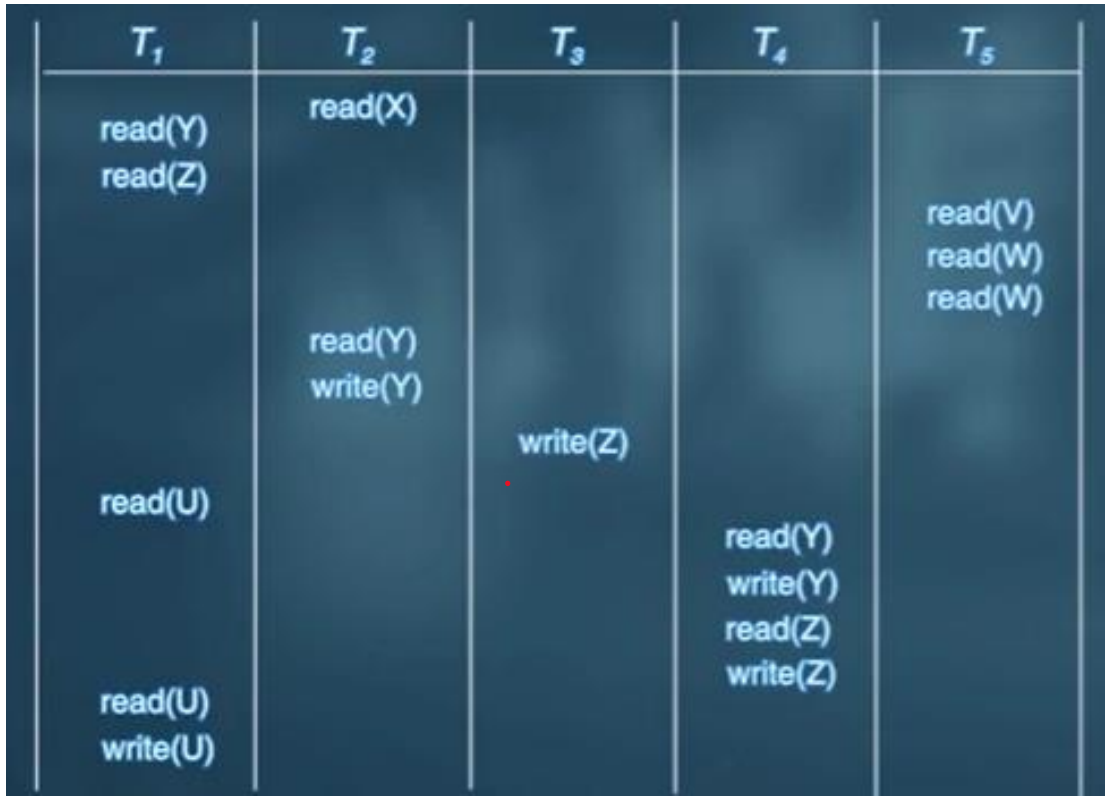
- 分为冲突可串行化和视图可串行化
- 冲突可串行化:
  - **冲突**：当I和J是不同事务在相同的数据项上的操作，并且其中至少有一个是***write***指令时，我们说I与J是冲突的。
    - 如果I和J不冲突，那么就可以将指令I与指令J调换顺序
  - **冲突等价**：如果调度S可以经过一系列非冲突指令交换转换成S'，我们称S和S'是冲突等价的
  - **冲突可串行化**：如果调度S可以经过一系列非冲突指令交换转换成串行调度S'，那么我们说S是冲突可串行化的
    - 举例:



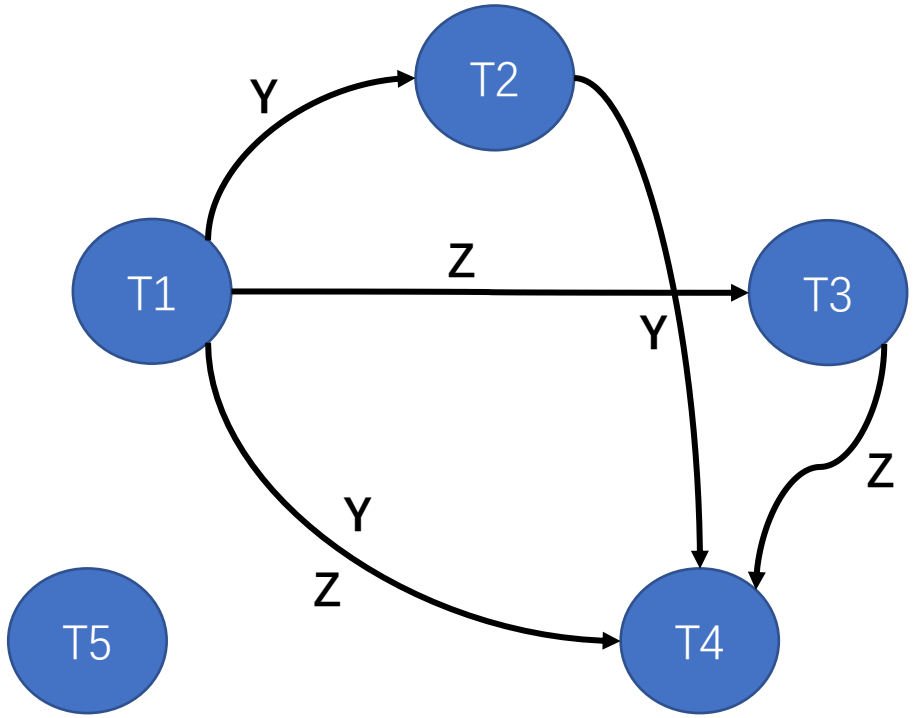
三、调度

2.可串行化调度:

- 冲突可串行化:
  - **冲突可串行化判断:**
    - 画优先图，如果有环路，那就不是冲突可串行化调度
    - 举例:



- 对于U，只有T1对它进行了读写操作，所以没有箭头可画
- 对于V，只有T5对它进行了读操作，所以没有箭头可画
- 对于W，只有T4对它进行了读操作，所以没有箭头可画
- 对于X，只有T2对它进行了读操作，所以没有箭头可画
- 对于Y，存在T1先读，T2再写这样一条关系，所以画一条T1到T2的箭头，并在箭头上标明Y
  存在T2先写，T4再读这样一条关系，所以画一条由T2到T4的箭头，并在箭头上标明Y
  存在T1先读，T4再写这样一条关系，所以画一条T1到T4的箭头，并在箭头上标明Y
- 对于Z，存在T1先读，T3再写这样一条关系，所以画一条由T1到T3的箭头，并在箭头上标明Z
  存在T3先对它写，T4再对它读这一条关系，所以画一条由T3到T4的箭头，并在箭头上标明Z
  存在T1先读，T4再写这样一条关系，所以画一条由T1到T4的箭头，并在箭头上标明Z



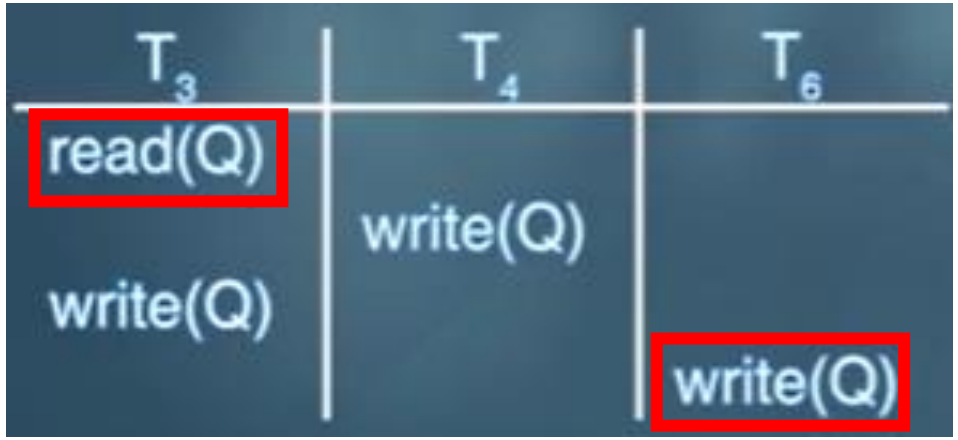
结论：因为没有环路，所以是冲突可串行化调度，只要把环打开，就能得到串行执行的顺序

三、调度

2.可串行化调度:

- 视图可串行化:
  - 只保证读的结果和最后写的结果不变
  - 举例：在本例中，我们只用关注框红框的部分

1



2

为了保证1读取的是原来的数据，应该把调度T_3放在最前面，因为3最后一个对Q进行了写操作，所以应该把T_6放在最后，因此最后得到串行化顺序$T_3 \rightarrow T_4 \rightarrow T_6$

三、调度

3.可恢复调度:

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

T9要读取T8写入的A的数值，如果T9先提交，而T8发生错误重新修改A的值，这时T9提交的结果就是错误的，所以成T9->T8这个调度为不可恢复调度。

四、级联回滚

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

- **级联回滚**：由于T8发生错误回滚，T9也因此回滚，这样的关系称为级联回滚
- 我们不希望级联回滚发生，因为会大大增大工作量
- 如何判断调度是否会发生级联回滚？
  - 如果所有的调度都是可恢复调度，那么便不会发生级联回滚