事务管理（事务基本知识）

单 位：重庆大学计算机学院

- 我卡里面有1000元，在ATM取钱，当钱取出来前一瞬间，我在手机上将钱转走了，我是不是就多出来了1000元呢？

主要学习目标

- 事务的概念
- 事务的特性
- 事务的状态
- 事务的隔离级别

1. 事务概念

- 转账、取钱

1. **read**(A)
2. $A := A - 50$
3. **write**(A)

---

4. **read**(B)
5. $B := B + 50$
6. **write**(B)

- Update 更新数据库多条记录

一 事务

1. 事务概念

事务执行的前后都是合法的数据状态，不会违背任何的数据完整性，这就是“一致”的意思。

事务：构成单一逻辑工作单元的操作集合。

作用：DBMS通过保证要么执行整个事务，要么一个操作也不执行，达到使数据始终处于一个一致状态。(事务是访问并可能更新各种数据项的一个程序执行单元)

事务的ACID特性

C-特性 Consistensy

- **一致性**：在这里，一致性要求事务的执行不改变 A、B 之和。如果没有一致性要求，金额可能会被事务凭空创造或销毁！容易验证，如果数据库在事务执行前是一致的，那么事务执行后数据库仍将保持一致。

A-特性 Atomicity

- **原子性**：假设事务 T_i 执行前账户 A 和账户 B 分别有 \$1000 和 \$2000。现在假设在事务 T_i 执行时系统出现故障，导致 T_i 的执行没有成功完成。我们进一步假设故障发生在 **write**(A)操作执行之后 **write**(B)操作执行之前。在这种情况下，数据库中反映出来的是账户 A 有 \$950，而账户 B 有 \$2000。这次故障导致系统丢失了 \$50。特别地，我们注意到 $A + B$ 的和不再维持原状。这样，由于故障，系统的状态不再反映数据库本应描述的现实世界的真实状态。我们把这种状态称为**不一致状态**(inconsistent state)。我们必须保证这种不一致性在数据库系统中是不可见的。

D-特性 Durability

- **持久性**：一旦事务成功地完成执行，并且发起事务的用户已经被告知资金转账已经发生，系统就必须保证任何系统故障都不会引起与这次转账相关的数据丢失。持久性保证一旦事务成功完成，该事务对数据库所做的所有更新就都是持久的，即使事务执行完成后出现系统故障。

I-特性 Isolation

- **隔离性**：如果几个事务并发地执行，即使每个事务都能确保一致性和原子性，它们的操作会以人们所不希望的某种方式交叉执行，这也会导致不一致的状态。

  正如我们先前看到的，例如，在 A 至 B 转账事务执行过程中，当 A 中总金额已减去转账额并已写回 A，而 B 中总金额加上转账额后还未写回 B 时，数据库暂时是不一致的。如果另一个并发运行的事务在这个中间时刻读取 A 和 B 的值并计算 $A + B$，它将会得到不一致的值。尽管多个事务可能并发执行，但系统保证，每个事务都感觉不到系统中有其他事务在并发地执行。

2. 事务的原子性和持久性

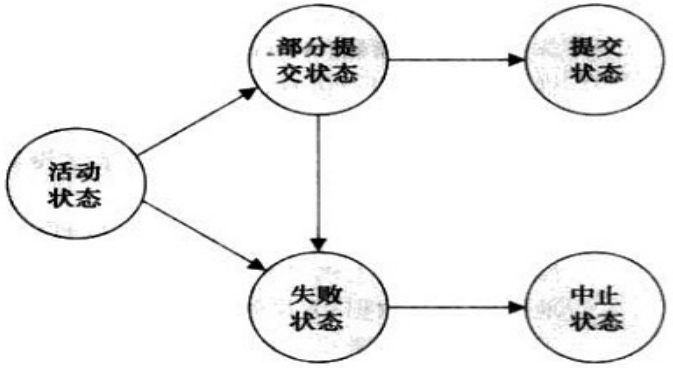


图 14-1 事务状态图

正如我们先前所注意到的，事务并非总能成功地执行完成。这种事务称为**中止**(aborted)了。我们如果要确保原子性，中止事务必须对数据库的状态不造成影响。因此，中止事务对数据库所做过的任何改变必须撤销。一旦中止事务造成的变更被撤销，我们就说事务**已回滚**(rolled back)。恢复机成功完成执行的事务称为**已提交**(committed)。一个对数据库进行过更新的已提交事务使数据库进入一个新的一致状态，即使出现系统故障，这个状态也必须保持。

我们需要更准确地定义一个事务成功完成的含义。为此我们建立了一个简单的抽象事务模型。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)：最后一条语句执行后。
- **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恢复到事务开始执行前的状态后。
- **提交的**(committed)：成功完成后。

事务状态图作用：
便于系统跟踪各事务执行情况
保证事务原子性和持久性特点

事务相应的状态图如图 14-1 所示。只有在事务已进入提交状态后，我们才说事务已提交。类似地，仅当事务已进入中止状态，我们才说事务已中止。如果事务是提交的或中止的，它称为**已经结束的**(terminated)。

二 可串行化调度

1. 调度基本概念

调度：一组指令包括指令在系统中执行的特定时间顺序。
(指令可能来自多个事务)
包括commit,abort指令

串行调度：
调度中凡属于同一个事务的指令都紧挨着一起。
即一个事务的指令都执行完成后在执行下一事务

并行调度：
调度中多个事务的指令在时间上相互交叉地在执行

可串行化调度：
虽然可能是一个并行调度
但在执行的效果上等同于
某一个串行调度执行结果

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

虽保证执行结果正确
但应用并发执行度差

虽应用并发执行度高
但执行结果可能错误

应用并发执行度较高
且执行结果依然正确

图14-2：串行调度示例

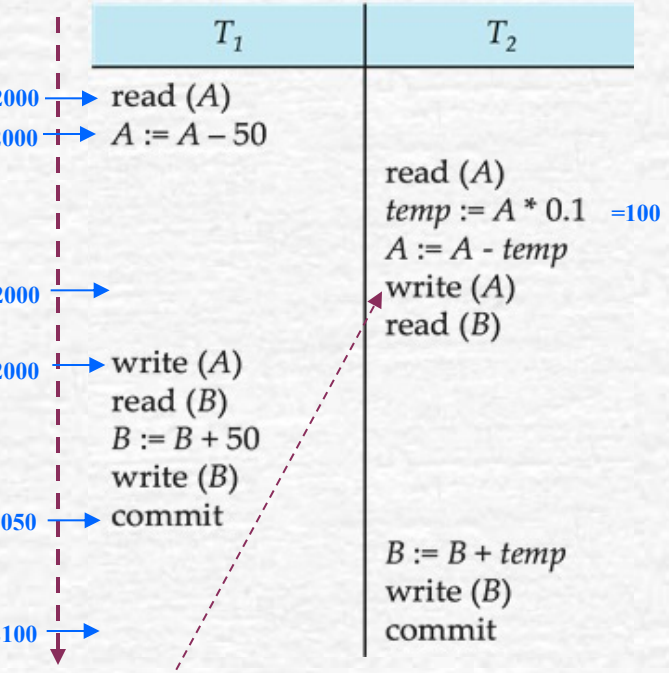


图14-5：并行调度示例

案例2

A+B=3050，异常结果
(T_2写的中间结果被覆盖)

T2→T1 ： A+B=3000
T1→T2 ： A+B=3000

图14-5调度不是一个可串行调度
结果与串行调度T1→T2和T2→T1的结果都不相同

图14-5是可串行化调度？

2. 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 S'，我们称 S 与 S' 是**冲突等价**(conflict equivalent)的。

由冲突等价的概念引出了冲突可串行化的概念：若一个调度 S 与一个串行调度冲突等价，则称调度 S 是冲突可串行化(conflict serializable)的。

注意：针对同一数据对象

T1的指令 T2的指令

1. l_i = read(Q), l_j = read(Q) 非冲突指令
2. l_i = read(Q), l_j = write(Q) 冲突指令
3. l_i = write(Q), l_j = read(Q) 冲突操作指令
4. l_i = write(Q), l_j = write(Q) 冲突操作指令

图14-4是冲突可串行化调度
因① ②两组指令可依次交换

图14-5不是冲突可串行化调度
因如图中标出的两对冲突指令
即不能转换为串行调度T1→T2
也不能转换为串行调度T2→T1

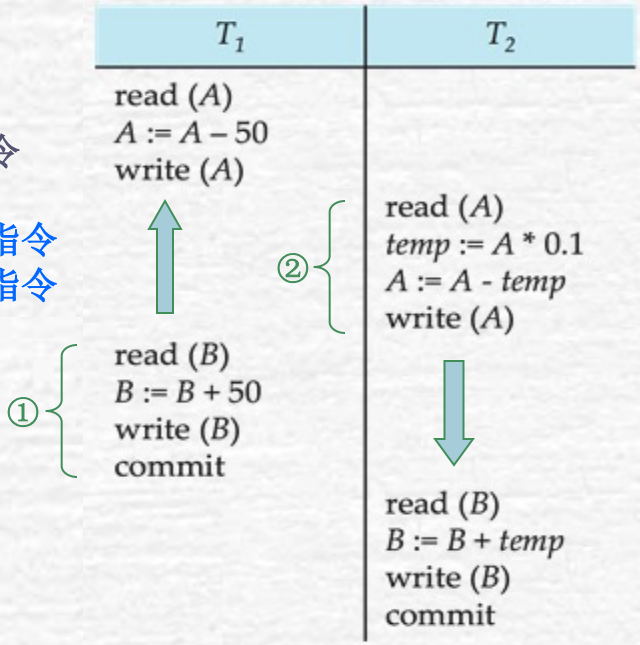


图14-4： 并行调度示例

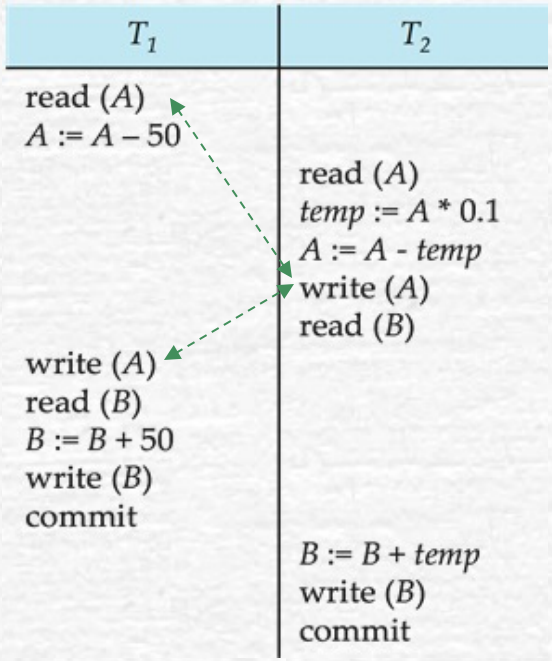


图14-5： 并行调度示例

3. 调度优先图

设 S 是一个调度，我们由 S 构造一个有向图，称为**优先图**（precedence graph）。该图由两部分组成 $G=(V, E)$，其中 V 是顶点集，E 是边集，顶点集由所有参与调度的事务组成，边集由满足下列三个条件之一的边 $Ti \rightarrow Tj$ 组成：

- 在 Tj 执行 **read**(Q)之前，Ti 执行 **write**(Q)。
- 在 Tj 执行 **write**(Q)之前，Ti 执行 **read**(Q)。
- 在 Tj 执行 **write**(Q)之前，Ti 执行 **write**(Q)。

同一数据
即有三种
冲突指令

如果优先图中存在边 $T_i \rightarrow T_j$，则在任何等价于 S 的串行调度 S' 中，T_i 必出现在 T_j 之前。

调度7优先图

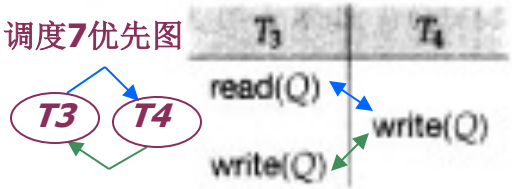
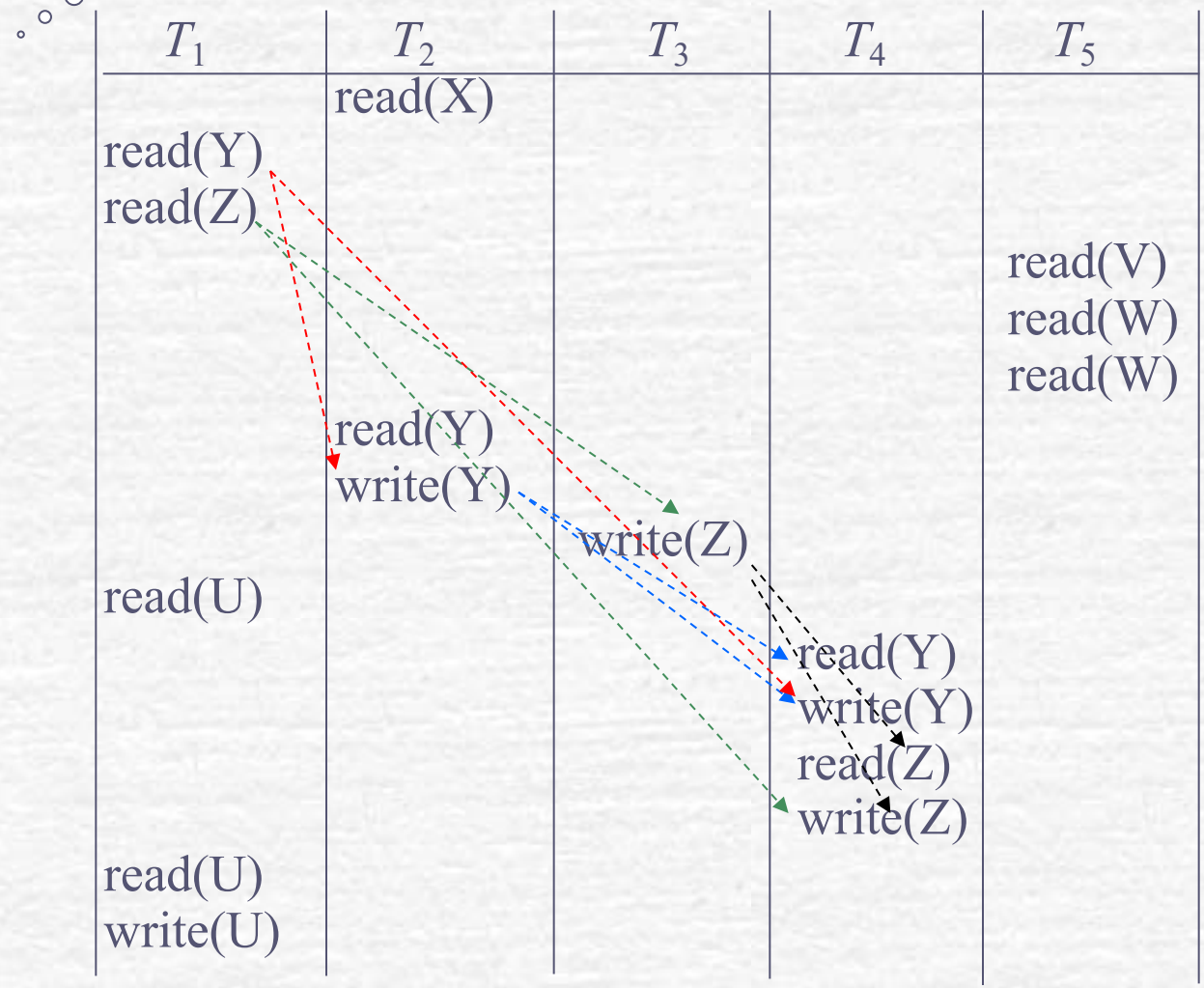


图 14-9 调度 7

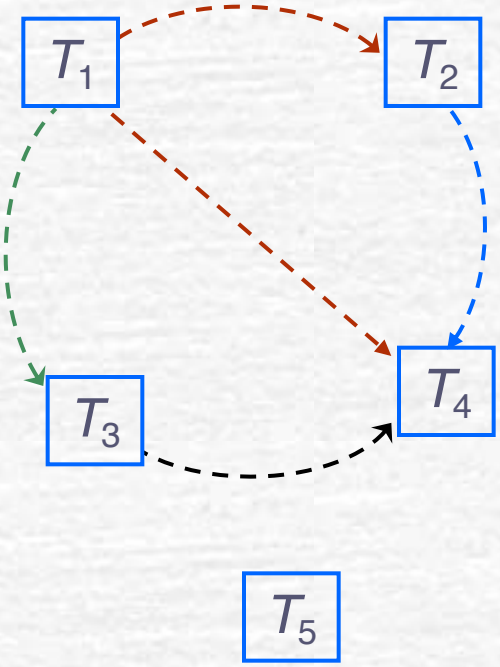
案例4

Example of Precedence Graph

该调度的优先图什么样？



无环，故为冲突可串行化调度！
其等价的串行调度可如下得到：
$T5$ $T1 \rightarrow T3 \rightarrow T2 \rightarrow T4$ 或
$T5 \rightarrow T1 \rightarrow T2 \rightarrow T3 \rightarrow T4$



孤立点（无连线）

4. 冲突可串行化的判定方法

如果调度 S 的优先图有环，则调度 S 是非冲突可串行化的，如果优先图无环，则调度 S 是冲突可串行化的。

调度 4 的优先图如图 14-11 所示。因为 T_1 执行 **read**(A) 先于 T_2 执行 **write**(A)，所以图 14-11 含有边 $T_1 \rightarrow T_2$。又因 T_2 执行 **read**(B) 先于 T_1 执行 **write**(B)，所以图 14-11 还含有边 $T_2 \rightarrow T_1$。

用于判断一个并行调度是否为冲突可串行化调度

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

图14-5 调度4

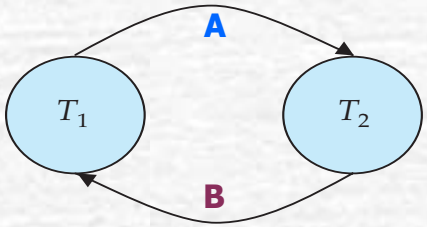


图14-11 调度4的优先图

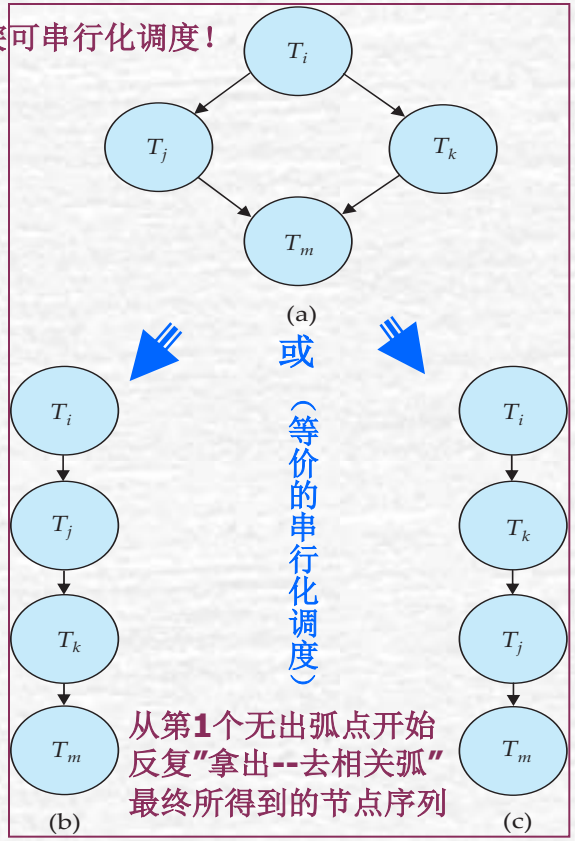
图14-11不是冲突可串行化调度！

这两个调度是否冲突可串行化?

8)冲突可串行化调度等价的串行调度如何得到?

调度4的优先图什么样？

该图是冲突可串行化调度！



案例6

三 事务隔离性和原子性

1. 可恢复调度&无级联调度p. 366

T_6	T_7
read(A)	
write(A)	
	read(A)
	commit
read(B)	

图 14-14 调度 9：一个可恢复的调度

案例7

T_8	T_9	T_{10}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)
abort		

图 14-15 调度 10

案例8

调度 9 是不可恢复调度的一个例子。一个**可恢复调度**(recoverable schedule)应满足：对于每对事务 T_i 和 T_j，如果 T_j 读取了之前由 T_i 所写的数据项，则 T_i 先于 T_j 提交。例如，如果要使调度 9 是可恢复的，则 T_7 应该推迟到 T_6 提交后再提交。

，**无级联调度**(cascadeless schedule)应满足：对于每对事务 T_i 和 T_j，如果 T_j 读取了先前由 T_i 所写的数据项，则 T_i 必须在 T_j 这一读操作前提交。容易验证每一个无级联调度也都是可恢复的调度。

事务的隔离

- 一个数据库可能拥有多个访问客户端，这些客户端并发访问数据库时，如果不采取隔离措施，会不会存在问题呢？
- 数据读问题：脏读、不可重复读和幻读。
- 数据更新问题：第一类丢失更新、第二类丢失更新。

脏 读

- A事务读取B事务尚未提交的更改数据，并在这个数据的基础上进行操作，这时候如果事务B回滚，那么A事务读到的数据是不被承认的。
- 例：常见的取款事务和转账事务：

时　间	转账事务 A	取款事务 B
T1		开始事务
T2	开始事务	
T3		查询账户余额为 1000 元
T4		取出 500 元，把余额改为 500 元
T5	查询账户余额为 500 元（脏读）	
T6		撤销事务，余额恢复为 1000 元
T7	汇入 100 元，把余额改为 600 元	
T8	提交事务	

事务B取款的操作提交前，事务A读取了账户余额，当事务B撤销后，事务A读取余额就是脏读，造成数据不一致。

不可重复读

- 不可重复读是指A事务读取了B事务已经提交的更改数据。假如A在取款事务的过程中，B往该账户转账100，A两次读取的余额发生不一致。

幻 读

- A事务读取B事务提交的新增数据,会引发幻读问题。幻读一般发生在计算统计数据的事务中。
- 例：银行系统在同一个事务中两次统计存款账户的总金额，在两次统计中，刚好新增了一个存款账户，存入了100，这时候两次统计的总金额不一致。

时　间	统计金额事务 A	转账事务 B
T1		开始事务
T2	开始事务	
T3	统计总存款数为 10000 元	
T4		新增一个存款账户，存款为 100 元
T5		提交事务
T6	再次统计总存款数为 10100 元（幻象读）	

不可重复读和幻读的区别

- 不可重复读：指读到了已经提交的事务的更改数据（修改或删除）
- 幻读：指读到了其他已经提交事务的新增数据。
- 两种问题不同的解决办法
  - 防止读到更改数据，只需对操作的数据添加行级锁，防止操作中的数据发生变化；
  - 防止读到新增数据，往往需要添加表级锁，将整张表锁定，防止新增数据（oracle采用多版本数据的方式实现）。

第一类丢失更新

- A事务撤销时，把已经提交的B事务的更新数据覆盖了。
- 例：

时　间	取款事务 A	转账事务 B
T1	开始事务	
T2		开始事务
T3	查询账户余额为 1000 元	
T4		查询账户余额为 1000 元
T5		汇入 100 元，把余额改为 1100 元
T6		提交事务
T7	取出 100 元，把余额改为 900 元	

取款事务A撤销事务，余额恢复为1000，这就丢失了更新。

第二类丢失更新

- A事务覆盖B事务已经提交的数据，造成B事务所做的操作丢失

时　间	转账事务 A	取款事务 B
T1		开始事务
T2	开始事务	
T3		查询账户余额为 1000 元
T4	查询账户余额为 1000 元	
T5		取出 100 元，把余额改为 900 元
T6		提交事务
T7	汇入 100 元	
T8	提交事务	
T9	把余额改为 1100 元（丢失更新）	

2. 事务隔离性级别

SQL 标准规定的隔离性级别如下。

- **可串行化**(serializable)：通常保证可串行化调度。然而，正如我们将要解释的，一些数据库系统对该隔离性级别的实现在某些情况下允许非可串行化执行。
- **可重复读**(repeatable read)：只允许读取已提交数据，而且在一个事务两次读取一个数据项期间，其他事务不得更新该数据。但该事务不要求与其他事务可串行化。例如：当一个事务在查找满足某些条件的数据时，它可能找到一个已提交事务插入的一些数据，但可能找不到该事务插入的其他数据。

  要求独占(封锁)所读数据项
- **已提交读**(read committed)：只允许读取已提交数据，但不要求可重复读。比如，在事务两次读取一个数据项期间，另一个事务更新了该数据并提交。

  不独占(封锁)所读数据项
- **未提交读**(read uncommitted)：允许读取未提交数据。这是 SQL 允许的最低一致性级别。

隔离性降低

通常情况下，要求数据库DBMS始终保持数据的一致性。→ 采用可串行化级别

但特殊情况，可以允许放松要求，如数据字典中统计值；
又如，当一些应用仅仅需要粗略（非精确）统计信息时。
（实例：房地产全国销售数量和金额实时监测/估算值）

} 可选用后三种级别

隔离级别	脏　　读	不可重复读	幻　象　读	第一类丢失更新	第二类丢失更新
READ UNCOMMITED	允许	允许	允许	不允许	允许
READ COMMITTED	不允许	允许	允许	不允许	允许
REPEATABLE READ	不允许	不允许	允许	不允许	不允许
SERIALIZABLE	不允许	不允许	不允许	不允许	不允许

- 不同的隔离级别对事务的处理不同，事务的隔离级别和数据库并发性是成反比的，隔离级别越高，并发性越低。

3. **隔离性级别的实现方式(选讲, p. 368)

X锁-独占

锁: 一个事务可以封锁其访问的数据项，而不用封锁整个数据库。在这种策略下，事务必须在足够长的时间内持有锁来保证可串行化，但是这一周期又要足够短致使不会过度影响性能。对于我们将在14.10节中看到的数据项的访问依赖于一条 **where** 子句的 SQL 语句则情况更为复杂。第15章将介绍一个简单并且广泛用来确保可串行化的两阶段封锁协议。简单地说，两阶段封锁要求一个事务有两个阶段，一个阶段只获得锁但不释放锁，第二个阶段只释放锁但是不获得锁。(实际上，通常只有当事务完成所有操作并且提交或中止时才释放锁。)

S锁 + X锁

如果我们有两种锁，则封锁的结果将进一步得到改进：共享的和排他的。共享锁用于事务读的数据项，而排他锁用于事务写的数据项。许多事务可以同时持有一个数据项上的共享锁，但是只有当其他事务在一个数据项上不持有任何锁(无论共享锁或排他锁)时，一个事务才允许持有该数据项上的排他锁。这两种锁模式以及两阶段封锁协议在保证可串行化的前提下允许数据的并发读。

时间戳: 另一类用来实现隔离性的技术为每个事务分配一个**时间戳**(timestamp)，通常是当它开始的时候。对于每个数据项，系统维护两个时间戳。数据项的读时间戳记录读该数据项的事务的最大(即最近的)时间戳。数据项的写时间戳记录写入该数据项当前值的事务的时间戳。时间戳用来确保在访问冲突情况下，事务按照事务时间戳的顺序来访问数据项。当不可能访问时，违例事务将会中止，并且分配一个新的时间戳重新开始。

快照隔离: 通过维护数据项的多个版本，一个事务允许读取一个旧版本的数据项，而不是被另一个未提交或者在串行化序列中应该排在后面的事务写入的新版本的数据项。有许多多版本并发控制技术。其中一个是实际中广泛应用的称为**快照隔离**(snapshot isolation)的技术。

随堂小测试

- 阐述事务的四个特性
- 在数据库系统中，什么是串行调度和并行调度？分别具有什么特点？如何判断一个并行调度执行的结果是正确的？

课程总结与作业安排

- 基本知识：
  - 事务的概念
  - 事务的特性与状态
  - 调度
  - 串行化
  - 事务的隔离级别
- 扩展学习：
  - 数据库如何保证事务顺利执行？
- 作业

  第14章习题：14.12，14.14，14.15.