第十一章　数据库并发控制

- 在多用户和网络环境下，数据库是一个共享资源，**多个用户或应用程序同时对数据库的同一数据对象进行读写操作，这种现象称为对数据库的并发操作。**
- 对并发操作不进行控制会造成一些错误。
  - 飞机定票数据库系统
  - 银行数据库系统

  特点：在同一时刻并发运行的事务数可达数百个；

- **对并发操作进行的控制称为并发控制。**
- 数据库的并发控制以事务为单位，通常使用封锁技术实现并发控制。
- 对数据对象施加封锁，会带来活锁和死锁问题，并发控制机制必须提供适合数据库特点的解决方法。

本次课内容

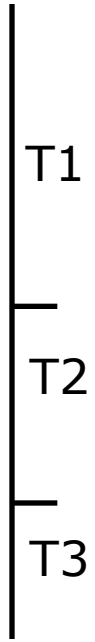
- 并发控制概述
- 封锁机制
- 活锁和死锁

一、并发控制概述

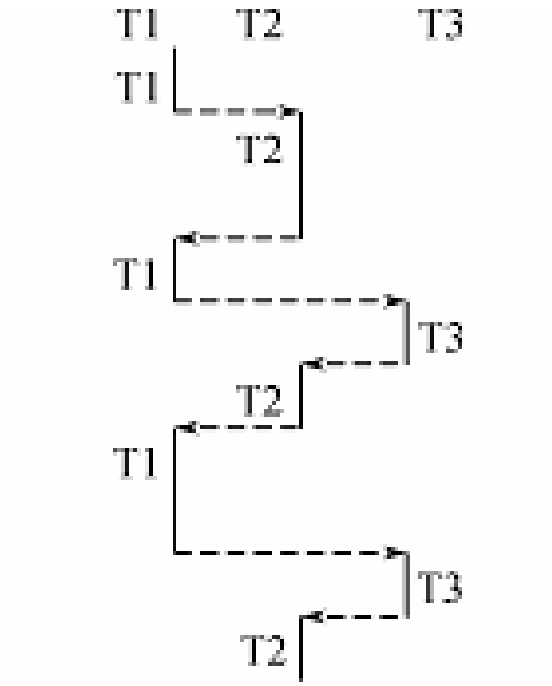
- 事务的ACID特性可能遭到破坏的原因之一是多个事务对数据库的并发操作造成的。为保证事务的隔离性和数据库的一致性，DBMS必须提供并发控制机制；
- 并发控制机制的任务
  - 对并发操作进行正确调度
  - 保证事务的隔离性
  - 保证数据库的一致性

1、不同的多事务执行方式

- **（1）事务串行执行**
  - 每个时刻只有一个事务运行，其他事务必须等到这个事务结束以后方能运行；
  - 不能充分利用系统资源，发挥数据库共享资源的特点；
- **（2）交叉并发方式（Interleaved Concurrency）**
  - 在单处理机系统中，事务的并行执行是这些并行事务的并行操作轮流交叉运行；
  - 单处理机系统中的并行事务并没有真正地并行运行，但能够减少处理机的空闲时间，提高系统的效率；



事务的串行执行方式



事务的交叉并发执行方式

- **（3）同时并发方式**（simultaneous concurrency）
  - 多处理机系统中，每个处理机可以运行一个事务，多个处理机可以同时运行多个事务，实现多个事务真正的并行运行；
  - 最理想的并发方式，但受制于硬件环境；

  本章讨论的并发控制技术以单处理机系统为基础；

- **事务并发执行带来的问题：**
  - 会产生多个事务同时存取同一数据的情况；
  - 若对并发操作不加控制，就可能会存取和存储不正确的数据，破坏事务一致性和数据库的一致性；

2、并发操作可能产生的问题

- 并发操作不加以限制，会产生数据不一致性问题，这种问题共有三类：

（1）丢失修改（Lost Update）

例1：假设某产品库存量为50，现在购入该产品100个，执行入库操作，库存量加100；用掉40个，执行出库操作，库存量减40；分别用T1和T2表示入库和出库操作任务。

- 若同时发生入库（T1）和出库（T2）操作，这就形成并发操作。

- 发生丢失修改的过程

顺序	任务	操作	库存量
1	T1	读库存量	50
2	T2	读库存量	50
3	T1	库存量=50+100	
4	T2	库存量=50-40	
5	T1	写库存量	150
6	T2	写库存量	10

例2：飞机订票系统中的一个活动序列 ；

① 甲售票点(甲事务)读出某航班的机票余额A，设A=16；

② 乙售票点(乙事务)读出同一航班的机票余额A，也为16；

③ 甲售票点卖出一张机票，修改余额A←A-1，所以A为15，把A写回数据库；

④ 乙售票点也卖出一张机票，修改余额A←A-1，所以A为15，把A写回数据库

结果明明卖出两张机票，数据库中机票余额只减少1 ；

（2）读“脏数据” （dirty read）

- 当T1和T2并发执行时，在T1对数据库更新的结果没有提交之前，T2使用了T1的结果，而在T2操作之后T1又回滚，这时引起的错误是T2读取了T1的“脏数据”。

- 发生读“脏数据”的过程

顺序	任务	操作	库存量
1	T1	读库存量	50
2	T1	库存量=50+100	
3	T1	写库存量	150
4	T2	读库存量	150
5	T2	库存量=150-40	
6	T1	ROLLBACK	
7	T2	写库存量	110

（3）不可重复读（Non-repeatable Read）

- 当T1读取数据A后，T2执行了对A的更新，当T1再次读取数据A（希望与第一次是相同的值）时，得到的数据与前一次不同，这时引起的错误称为“不可重复读”。

- 发生“不可重复读”的过程

顺序	任务	操作	库存量A	入库量B
1	T1	读A=50	50	100
2	T1	读B=100		
3	T1	求和=50+100		
4	T2	读B=100	50	
5	T2	执行B=B*4		
6	T2	回写B=400	50	400
7	T1	读A=50	50	
8	T1	读B=400		400
9	T1	求和=450（验算不对）		

- **三类不可重复读：**

  事务1读取某一数据后：

  - 1. **事务2对其做了修改，**当事务1再次读该数据时，得到与前一次不同的值。
  - 2. **事务2删除了其中部分记录，**当事务1再次读取数据时，发现某些记录神密地消失了。
  - 3. **事务2插入了一些记录，**当事务1再次按相同条件读取数据时，发现多了一些记录。

后两种不可重复读有时也称为**幻影现象**（phantom row）

- 并发操作之所以产生错误，是因为任务执行期间相互干扰造成的。当将任务定义成事务，事务具有的特性（特别是隔离性）得以保证时，就会避免上述错误的发生。
- 但是，如果只允许事务串行操作会降低系统的效率。所以，多数DBMS采用事务机制和封锁机制进行并发控制，既保证了数据的一致性，又保障了系统效率。

二、封锁机制

- 封锁机制是并发控制的主要手段；
- 封锁就是事务T在对某个数据对象（例如表、记录等）操作之前，先向系统发出请求，对其加锁；
- 加锁后事务T就对该数据对象有了一定的控制，在事务T释放它的锁之前，其它的事务不能更新此数据对象。

- 封锁具有3个环节：
  - 第一个环节是申请加锁；
  - 第二个环节是获得锁；
  - 第三个环节是释放锁。

  为了达到封锁的目的，在使用时事务应选择合适的锁，并要遵从一定的封锁协议。
- 基本的封锁类型有两种：排它锁(Exclusive Locks，简称X锁)和共享锁(Share Locks，简称S锁)。

- **排它锁（X锁)**

  排它锁也称为独占锁或写锁。一旦事务T对数据对象A加上排它锁(X锁)，则只允许T读取和修改A，其他任何事务既不能读取和修改A，也不能再对A加任何类型的锁，直到T释放A上的锁为止。

- **共享锁（S锁）**

  共享锁又称读锁。如果事务T对数据对象A加上共享锁(S锁)，事务T对数据对象A只能读不能修改，其他事务对A只能再加S锁，不能加X锁，直到事务T释放A上的S锁为止。

- 锁的相容矩阵

T_1 \ T_2	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

Y=Yes，相容的请求

N=No，不相容的请求

使用封锁机制解决丢失修改问题

T_1	T_2
① Xlock A	
② R(A)=16	
	Xlock A
③ A←A-1	等待
W(A)=15	等待
Commit	等待
Unlock A	等待
④	获得Xlock A
	R(A)=15
	A←A-1
⑤	W(A)=14
	Commit
	Unlock A

- 事务T1在读A进行修改之前先对A加X锁；
- 当T2再请求对A加X锁时被拒绝；
- T2只能等待T1释放A上的锁后T2获得对A的X锁
- 这时T2读到的A已经是T1更新过的值15；
- T2按此新的A值进行运算，并将结果值A=14送回到磁盘。避免了丢失T1的更新。

使用封锁机制解决读“脏”数据问题

T_1	T_2
① Xlock C	
R(C)=100	
C←C*2	
W(C)=200	
②	Slock C
	等待
③ ROLLBACK	等待
(C恢复为100)	等待
Unlock C	等待
④	获得Slock C
	R(C)=100
⑤	Commit C
	Unlock C

不读“脏”数据

- 事务T1在对C进行修改之前，先对C加X锁，修改其值后写回磁盘
- T2请求在C上加S锁，因T1已在C上加了X锁，T2只能等待
- T1因某种原因被撤销，C恢复为原值100
- T1释放C上的X锁后T2获得C上的S锁，读C=100。避免了T2读“脏”数据

T_1	T_2
① Slock A	
Slock B	
R(A)=50	
R(B)=100	
求和=150	
②	Xlock B
	等待
	等待
③ R(A)=50	等待
R(B)=100	等待
求和=150	等待
Commit	等待
Unlock A	等待
Unlock B	等待
④	获得XlockB
	R(B)=100
	B←B*2
⑤	W(B)=200
	Commit
	Unlock B

使用封锁机制解决不可重复读问题

- 事务T1在读A，B之前，先对A，B加S锁
- 其他事务只能再对A，B加S锁，而不能加X锁，即其他事务只能读A，B，而不能修改
- 当T2为修改B而申请对B的X锁时被拒绝只能等待T1释放B上的锁
- T1为验算再读A，B，这时读出的B仍是100，求和结果仍为150，即可重复读
- T1结束才释放A，B上的S锁。T2才获得对B的X锁

封锁协议

- 简单地对数据加X锁和S锁并不能保证数据库的一致性。在对数据对象加锁时，还需要约定一些规则。这些规则称为**封锁协议**(Locking Protocol)。对封锁方式规定不同的规则，就形成了各种不同的封锁协议。
- **一级封锁协议**
  - 一级封锁协议是：事务T在修改数据之前必须先对其加X锁，直到事务结束才释放；

顺序	T1	T2	库存A的值
1	Xlock A 获得		50
2	读A=50	Xlock A，等待	50
3	A=A+100 写回A=150 Commit Unlock A	等待	150
4		获得Xlock A 读A=150 A=A-40 写回A=110 Commit Ulock A	110

- 一级封锁协议可有效地防止“丢失更新”，并能够保证事务T的可恢复性。
- 但是，由于一级封锁没有要求对读数据进行加锁，所以不能保证可重复读和不读“脏”数据。

- **遵从一级封锁协议发生的读“脏”数据过程**

顺序	T1	T2	库存A的值
1	Xlock A 获得 读A=50 A=A+100 写回A=150 Unlock A		 50 150
2		读A=150	150
3	ROLLBACK		50

- **二级封锁协议**

  事务T对要修改数据必须先加X锁，直到事务结束才释放X锁；对要读取的数据必须先加S锁，读完后即可释放S锁。

- **遵从二级封锁协议发生的“不可重复读”的过程**

顺序	T1	T2	A的值	B的值
1	Slock A，B 获得 读A=50 读B=100 Ulock A，B	Xlock B 等待 等待 获得	50	100
2	求和=A+B=150 Slock A 得到 Slock B 等待 等待 获得	读 B=100 B←B×4 回写B=400 Commit Ulock B	50	100 400
3	读A=50 读B=400 和=450（验算错误）		50	400

- **三级封锁协议**

三级封锁协议是事务T在读取数据之前必须先对其加S锁，在要修改数据之前必须先对其加X锁，直到事务结束后才释放所有锁。

由于三级封锁协议强调即使事务读完数据A之后也不释放S锁，从而使得别的事务无法更改数据A。三级封锁协议不但防止了丢失修改和不读“脏”数据，而且防止了不可重复读。

	X锁		S锁		一致性保证		
	操作结束释放	事务结束释放	操作结束释放	事务结束释放	不丢失修改	不读"脏"数据	可重复读
1级封锁协议		√			√		
2级封锁协议		√	√		√	√	
3级封锁协议		√		√	√	√	√

三、活锁和死锁

- 1、活 锁
  - 如果事务T1封锁了数据R，T2事务又请求封锁R，于是T2等待。T3也请求封锁R，当T1释放了R上的封锁之后系统首先批准了T3的要求，T2仍然等待。然后T4又请求封锁R，当T3释放了R上的封锁之后系统又批准了T4的请求，……，T2有可能永远等待。这种在多个事务请求对同一数据封锁时，使某一用户总是处于等待的状况称为活锁。
  - **解决活锁问题的方法是采用先来先服务**。即对要求封锁数据的事务排队，使前面的事务先获得数据的封锁权。

2、死锁

- 如果事务T1和T2都需要数据RI和R2，操作时TI封锁了数据R1，T2封锁了数据R2；然后T1又请求封锁R2，T2又请求封锁RI；因T2已封锁了R2，故T1等待T2释放R2上的锁。同理，因T1已封锁了R1，故T2等待T1释放R1上的锁。由于TI和T2都没有获得全部需要的数据，所以它们不会结束，只能继续等待。**这种多事务交错等待的僵持局面称为死锁。**
- 数据库中解决死锁问题主要有两类方法：一类方法是采用一定措施来预防死锁的发生；另一类方法是允许发生死锁，然后采用一定手段定期诊断系统中有无死锁，若有则解除之。

1）预防死锁

- 死锁的预防
  - 产生死锁的原因是两个或多个事务都已封锁了一些数据对象，然后又都请求对已为其他事务封锁的数据对象加锁，从而出现死等待；
  - 预防死锁的发生就是要破坏产生死锁的条件；
- 预防死锁的方法
  - 一次封锁法
  - 顺序封锁法

- **一次封锁法**
  - 要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行；
  - 一次封锁法存在的问题：
    - 降低并发度
      - 扩大封锁范围，将以后要用到的全部数据加锁，势必扩大了封锁的范围，从而降低了系统的并发度
    - 难于事先精确确定封锁对象

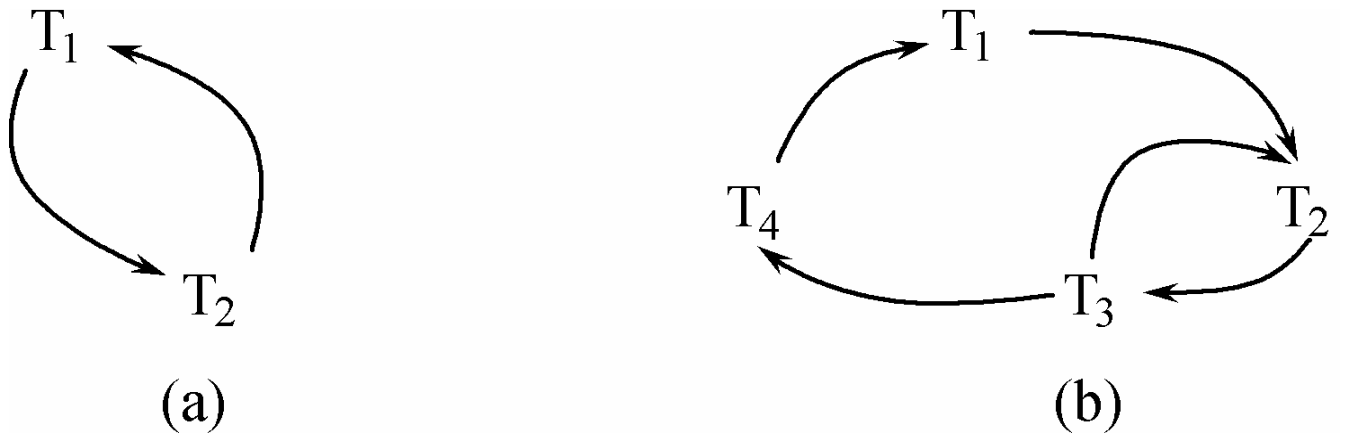
- **顺序封锁法**
  - 顺序封锁法是预先对数据对象规定一个封锁顺序，所有事务都按这个顺序实行封锁。
  - 顺序封锁法存在的问题
    - 维护成本
      - 数据库系统中封锁的数据对象极多，并且在不断地变化。
    - 难以实现：很难事先确定每一个事务要封锁哪些对象

2）死锁的诊断与解除

- 死锁的诊断
  - 超时法
  - 事务等待图法
- **超时法**
  - 如果一个事务的等待时间超过了规定的时限，就认为发生了死锁；
  - 优点：实现简单
  - 缺点：有可能误判死锁

    时限若设置得太长，死锁发生后不能及时发现

- **等待图法**
  - 用事务等待图动态反映所有事务的等待情况
  - 事务等待图是一个有向图G=(T，U)
  - T为结点的集合，每个结点表示正运行的事务
  - U为边的集合，每条边表示事务等待的情况
  - 若T_1等待T_2，则T_1，T_2之间划一条有向边，从T_1指向T_2



事务等待图

- 图(a)中，事务T1等待T2，T2等待T1，产生了死锁；
- 图(b)中，事务T1等待T2，T2等待T3，T3等待T4，T4又等待T1，产生了死锁；
- 图(b)中，事务T3可能还等待T2，在大回路中又有小的回路；

- 并发控制子系统周期性地（比如每隔数秒）生成事务等待图，检测事务。如果发现图中存在回路，则表示系统中出现了死锁。

- **解除死锁**
  - 选择一个处理死锁代价最小的事务，将其撤消；
  - 释放此事务持有的所有的锁，使其它事务能继续运行下去；

- **事务和锁是并发控制的主要机制**，SQL Server通过支持事务机制来管理多个事务，保证数据的一致性，并使用事务日志保证修改的完整性和可恢复性。SQL Server遵从三级封锁协议，从而有效的控制并发操作可能产生的丢失更新、读“脏”数据、不可重复读等错误。