

数据库原理

第7章 数据库管理

辽东学院 鲁 琴

本节要点

数据库基础概念

数据库原理

关系数据库

关系数据模型

关系数据语言

数据库设计

数据库管理

数据库新技术

安全性 ⊕

完整性 ⊕

并发控制 ⊖

故障和恢复

复制

并发控制概述 ⊖

并发操作的调度 ⊖

封锁 ⊖

死锁和活锁 ⊖

事务

数据不一致问题

可串行化调度

封锁类型

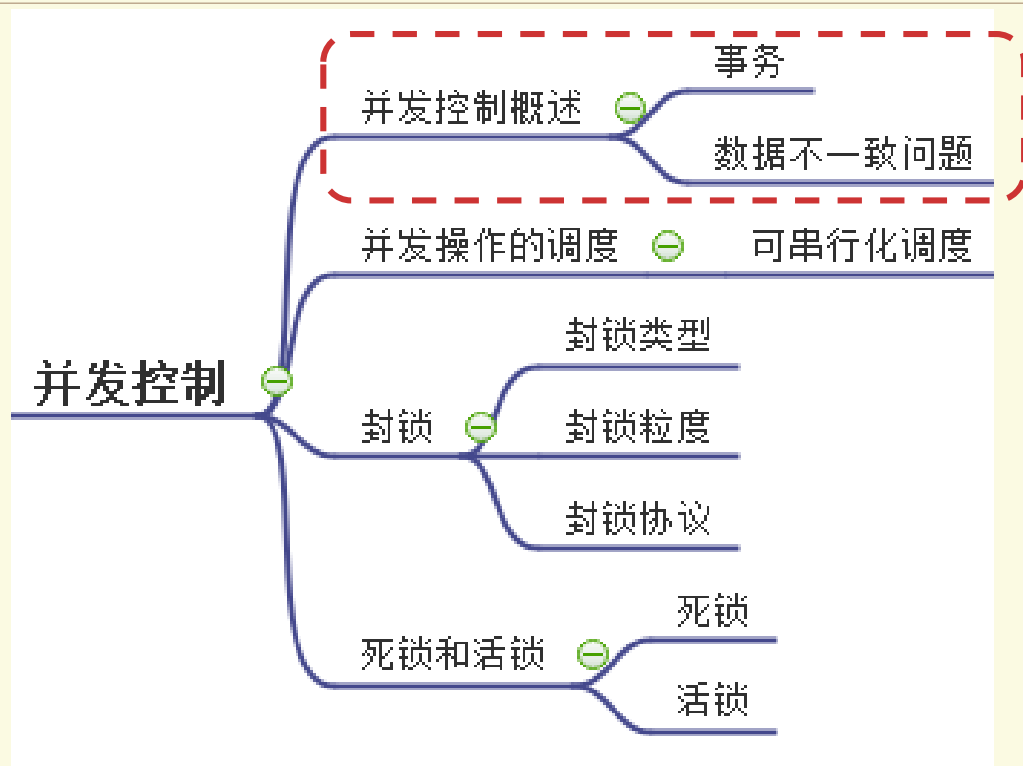
封锁粒度

封锁协议

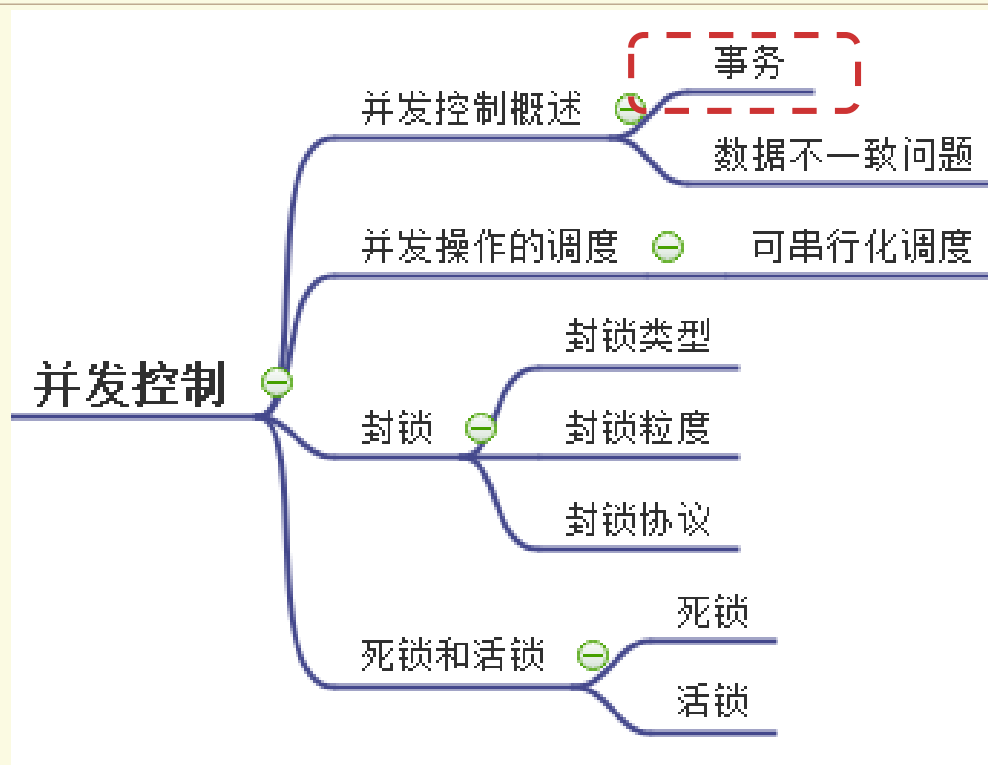
死锁

活锁

3 并发控制

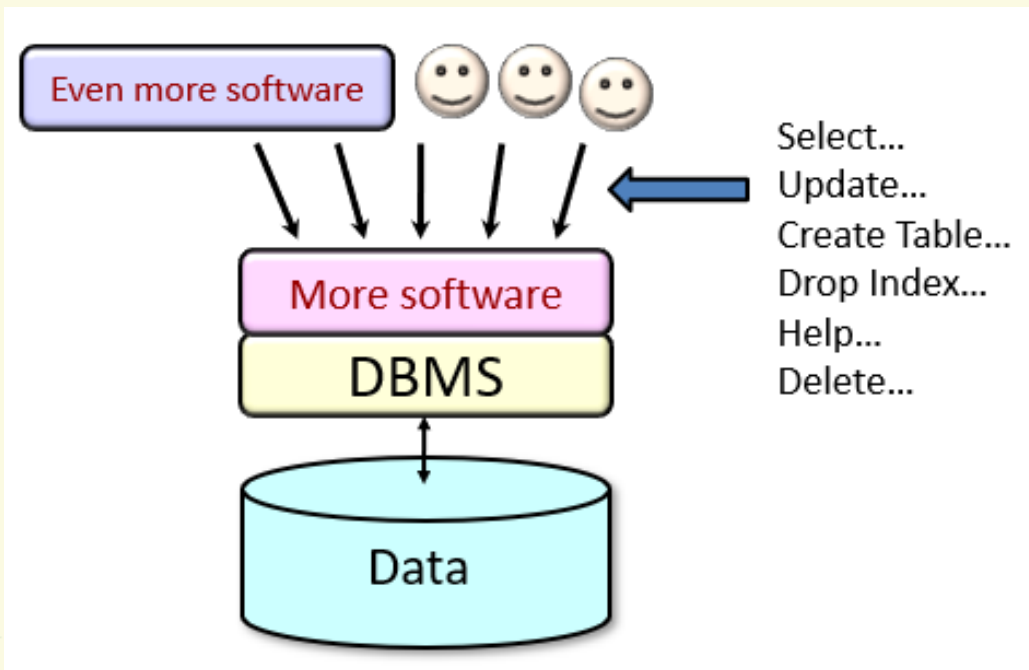


3.1 并发控制概述



数据库并发访问

事务是并发控制和恢复的基本单位



- 对多用户并发存取同一数据的操作不加控制可能会存取和存储不正确的数据
- **DBMS**必须提供并发控制机制

一、事务

- 从用户的观点看，对数据库的某些操作应是一个**整体**，也就是一个独立的工作单元，**不能分割**。
- 例：电子资金转账（从账号A转一笔款子到账号B）客户可能认为是一个独立的操作，而在DBS中是由几个操作组成的

```
read(A);  
A=A-50;  
write(A);  
read(B);  
B=B + 50;  
write(B).
```

这些操作要么
全都发生
要么由于出错
(可能账号A已
透支)
而全不发生.



事务

一、事务

- **事务**(Transaction)是用户定义的一个**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
- **事务**和**程序**是两个概念
 - 在关系数据库中，一个事务可以是一条SQL语句，一组SQL语句或整个程序
 - 一个应用**程序**通常包含多个**事务**
- **事务**是恢复和并发控制的基本单位

定义事务的两种方式

— 显式方式

- 事务的**开始**由用户显式控制或DBMS自动隐含
- 事务**结束**由用户显式控制

— 隐式方式

- 当用户没有显式地定义事务时，由DBMS按缺省规定**自动划分事务**

显式定义事务

事务开始

BEGIN TRANSACTION

SQL 语句1

SQL 语句2

.....

COMMIT

事务结束

~~BEGIN TRANSACTION~~

SQL 语句1

SQL 语句2

.....

~~ROLLBACK~~

事务结束语句

● COMMIT

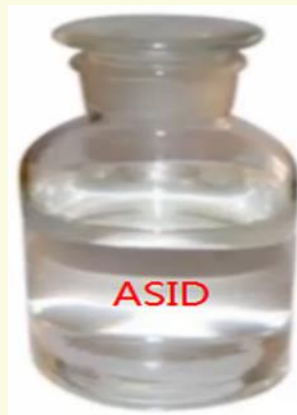
- 事务正常结束
- 提交事务的所有操作（读+更新）
- 事务中所有对数据库的更新永久生效

● ROLLBACK

- 事务异常终止
- 事务运行的过程中发生了故障，不能继续执行，回滚事务的所有更新操作
- 事务回滚到开始时的状态

事务的ACID特性：

- 原子性（**A**tomicity）
- 一致性（**C**onsistency）
- 隔离性（**I**solation）
- 持续性（**D**urability）



(1) 原子性

- 事务是数据库的逻辑工作单位
 - 事务中包括的诸操作要么都做，要么都不做



(2) 一致性

- 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。
 - 一致性状态：数据库中只包含成功事务提交的结果
 - 不一致状态：数据库中包含失败事务的结果

一致性与原子性是密切相关的

例：银行转帐：从帐号A中取出一万元，存入帐号B。

— 定义一个事务，该事务包括两个操作

- 第一个操作是从帐号A中减去一万元
- 第二个操作是向帐号B中加入一万元

— 这两个操作要么全做，要么全不做

- 全做或者全不做，数据库都处于一致性状态。
- 如果只做一个操作则用户逻辑上就会发生错误，少了一万元，这时数据库就处于不一致性状态。

(3) 隔离性

- 一个事务的执行不能被其他事务干扰
 - 一个事务内部的操作及使用的数据对其他并发事务是隔离的
 - 并发执行的各个事务之间不能互相干扰

(4) 持续性

- 持续性也称永久性（**Permanence**）
 - 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
 - 接下来的其他操作或故障不应该对其执行结果有任何影响

事务的特性(续)

- 保证事务ACID特性是事务处理的重要任务
- 破坏事务ACID特性的因素
 - 多个事务并行运行时，不同事务的操作交叉执行
 - DBMS必须保证多个事务的交叉运行不影响这些事务ACID特性，特别是原子性和隔离性
 - 事务在运行过程中被强行停止
 - DBMS必须保证被强行终止的事务对数据库和其他事务没有任何影响
- 这些就是DBMS中恢复机制和并发控制机制的责任

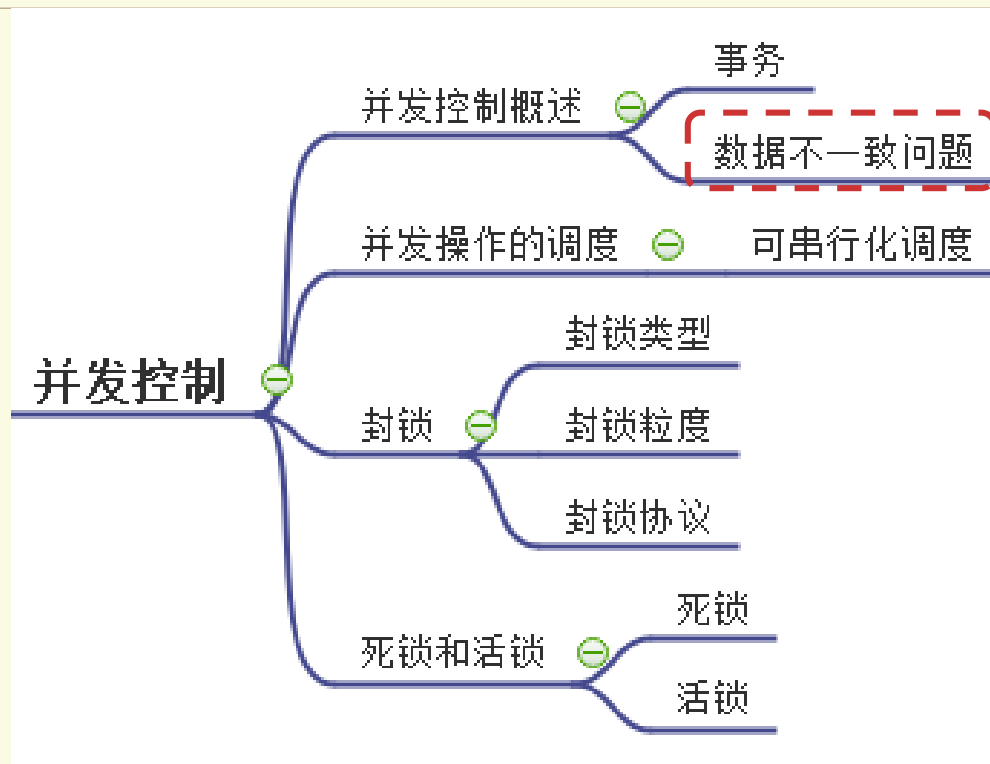
多事务执行方式

- 事务**串行**执行 (serial)
 - 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
- **交叉并发**方式 (interleaved concurrency)
 - 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
 - 是**单处理机**系统中的并发方式，能够减少处理机的空闲时间，提高系统的效率
- **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

事务并发执行带来的问题

- 对多用户并发存取同一数据的操作不加控制可能会存取和存储不正确的数据，破坏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
- 并发控制机制是衡量一个**DBMS**性能的重要标志之一

3.1 并发控制概述



二、并发操作与数据的不一致性

T_1	T_2
① 读A=16	读A=16
②	
③ $A \leftarrow A-1$ 写回A=15	
④	$A \leftarrow A-1$ 写回A=15

✓ 产生原因

- 由两个事务**并发操作**引起
- 在并发操作情况下，对 T_1 、 T_2 两个事务的操作序列的调度是随机的
- 若按上面的调度序列执行， T_1 事务的**修改**就被**丢失**。
- 因为第4步中 T_2 事务修改A并写回后覆盖了 T_1 事务的修改

并发控制概述（续）

- 并发操作带来的数据不一致性
 - 丢失修改（lost update）
 - 不可重复读（non-repeatable read）
 - 读“脏”数据（dirty read）

(1) 丢失修改

- 事务1与事务2从数据库中读入同一数据并修改
- 事务2的提交结果破坏了事务1提交的结果



- 导致事务1的修改被丢失

T ₁	T ₂
① 读A=16	读A=16
②	
③ A←A-1 写回A=15	
④	A←A-1 写回A=15

(2) 不可重复读

- 事务1读取数据
- 之后，事务2执行更新操作



- 从而使事务1无法再现前一次读取结果

T_1	T_2
① 读A=50 读B=100 求和=150	
②	读B=100 $B \leftarrow B * 2$ 写回B=200
③ 读A=50 读B=200 求和=250 (验算不对)	

三类不可重复读

1) 读-更新

- 事务1读取某一数据
- 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

后两种也称为**幻行现象**

2) 读-删除

- 事务1按一定条件从数据库中读取某些数据记录
- 事务2删除了其中部分记录
- 当事务1再次按相同条件读取数据时，发现某些记录神秘地消失了

3) 读-插入

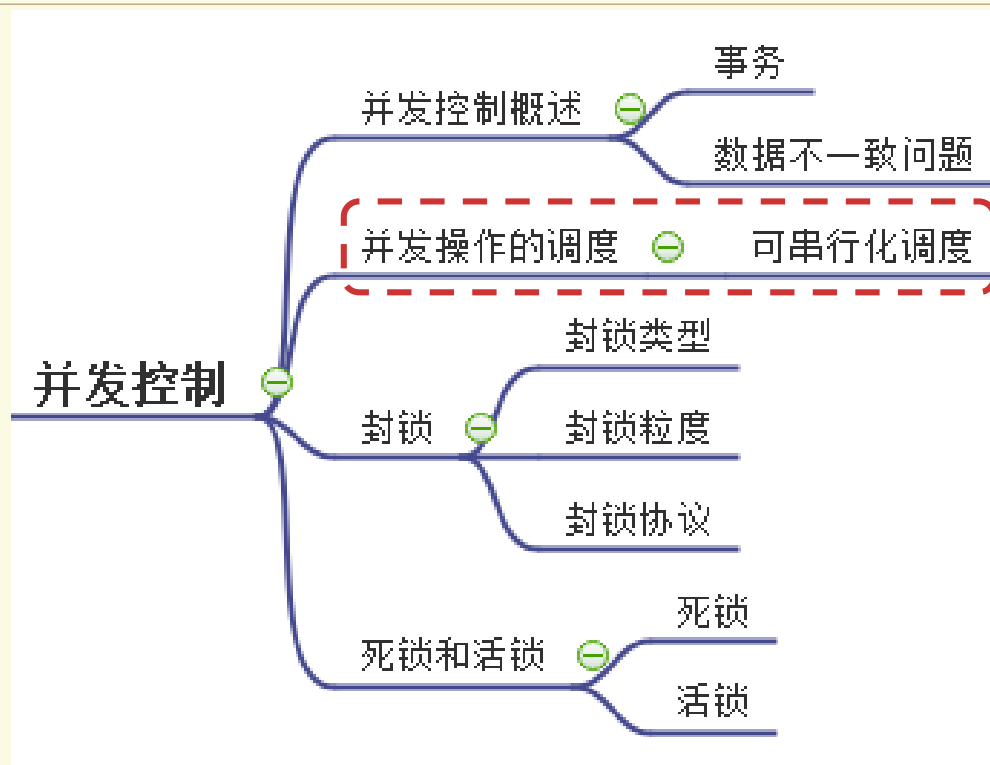
- 事务1按一定条件从数据库中读取某些数据记录
- 事务2插入了一些记录
- 当事务1再次按相同条件读取数据时，发现多了一些记录

(3) 读“脏”数据

- 事务1修改某一数据，并将其写回磁盘
- 事务2读取同一数据
- 之后，事务1由于某种原因被撤销，这时事务1已修改过的数据恢复原值
- 事务2读到的数据就与数据库中的数据不一致，是不正确的数据，又称为“脏”数据。

T ₁	T ₂
① 读C=100 C←C*2 写回C	读C=200
②	
③ ROLLBACK C恢复为100	

3.1 并发控制概述



3.2 并发操作的调度

- 计算机系统对并行事务中并行操作的调度是随机的，而不同的调度可能会产生不同的结果
- 将所有事务串行起来的调度策略一定是正确的调度策略
 - 如果一个事务运行过程中没有其他事务在同时运行
 - 也就是说它没有受到其他事务的干扰
 - 那么就可以认为该事务的运行结果是正常的或者预想的

可串行化的调度

- 以不同的顺序**串行**执行事务也有可能产生不同的结果
 - 但由于不会将数据库置于不一致状态，所以都可以认为是**正确的**
- 几个事务的并行执行是正确的
 - 当且仅当其结果与按某一次序串行地执行它们时的结果相同。这种并行调度策略称为**可串行化（Serializable）的调度**

并发操作的调度（续）

- 可串行性是并行事务正确性的唯一准则

- 例：现在有两个事务，分别包含下列操作：
- 事务1：读B； $A=B+1$ ；写回A；
- 事务2：读A； $B=A+1$ ；写回B；
- 假设A的初值为2，B的初值为2。

4种调度方式

(a) 串行调度策略，正确的调度

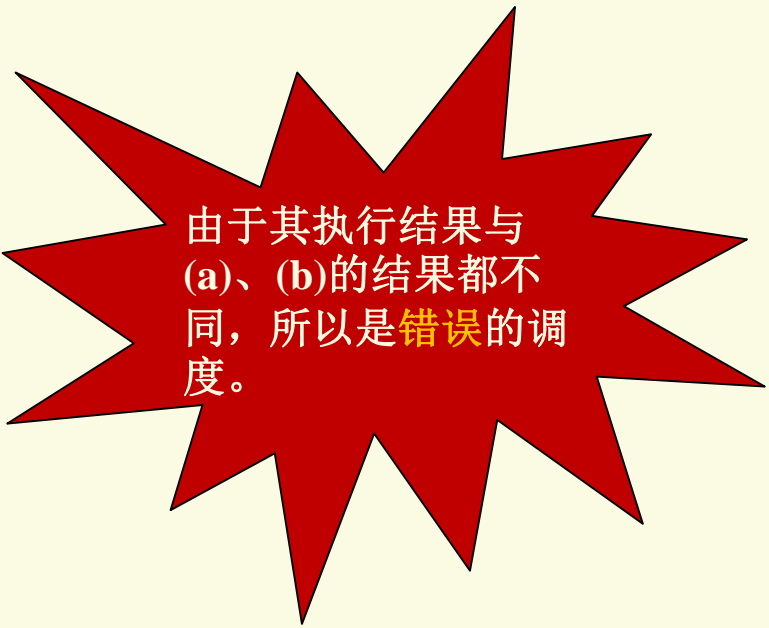
T_1	T_2
读 $B=2$ $A \leftarrow B+1$ 写回 $A=3$	读 $A=3$ $B \leftarrow A+1$ 写回 $B=4$

(b) 串行调度策略，正确的调度

T_1	T_2
读 $B=3$ $A \leftarrow B+1$ 写回 $A=4$	读 $A=2$ $B \leftarrow A+1$ 写回 $B=3$

(c) 不可串行化的调度

T_1	T_2
读B=2 $A \leftarrow B+1$ 写回A=3	读A=2 $B \leftarrow A+1$ 写回B=3



由于其执行结果与
(a)、(b)的结果都不
同，所以是**错误**的调
度。

(d) 可串行化的调度

T_1	T_2
读B=2	等待
$A \leftarrow B + 1$	等待
写回A=3	等待
	读A=3
	$B \leftarrow A + 1$
	写回B=4

由于其执行结果与串行调度 (a) 的执行结果相同，所以是**正确**的调度。

不同调度策略

- 串行执行
 - a. 串行调度策略
 - b. 串行调度策略
- 交错执行
 - c. 不可串行化的调度
 - d. 可串行化的调度

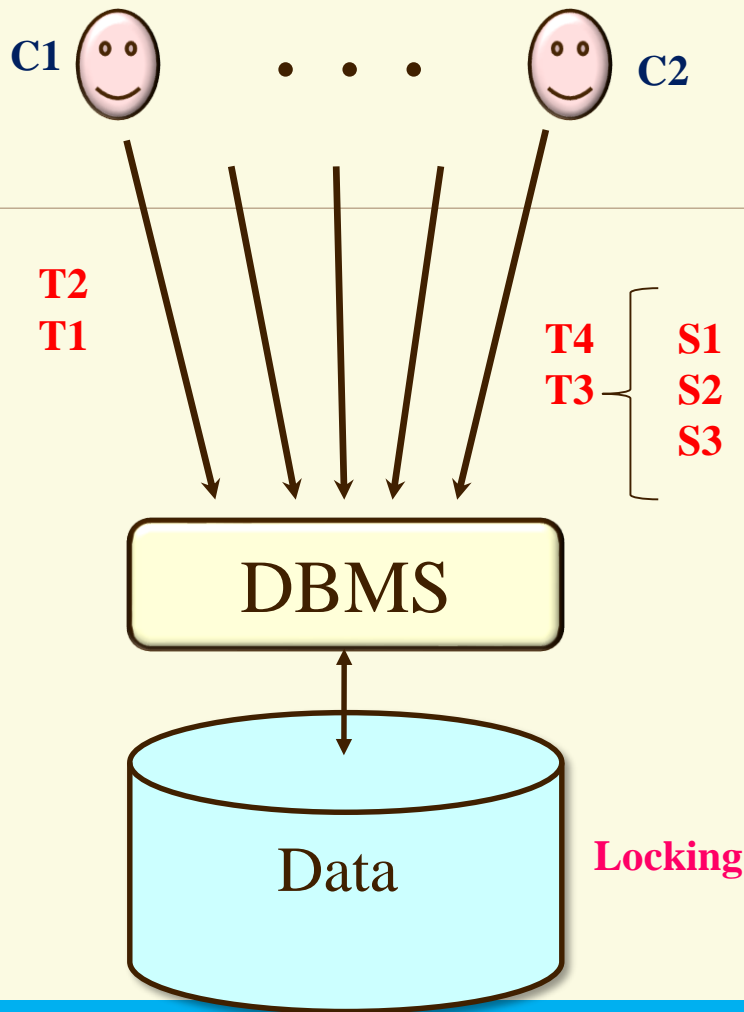
可串行化的调度策略

- 为了保证并行操作的正确性，DBMS的并行控制机制必须提供一定的手段来保证调度是**可串行化**的
- 从理论上讲，在某一事务执行时禁止其他事务执行的调度策略一定是可串行化的调度，这也是**最简单的调度策略**
- 但这种方法实际上是**不可行**的，因为它使用户不能充分共享数据库资源

Example

假设客户端 C1 发布事务 T1;T2 ,
客户端 C2 发布事务 T3;T4.
对于这4个事务有多少种可
串行化的调度顺序?

- A. 2
- B. 4
- C. 6
- D. 24



Example

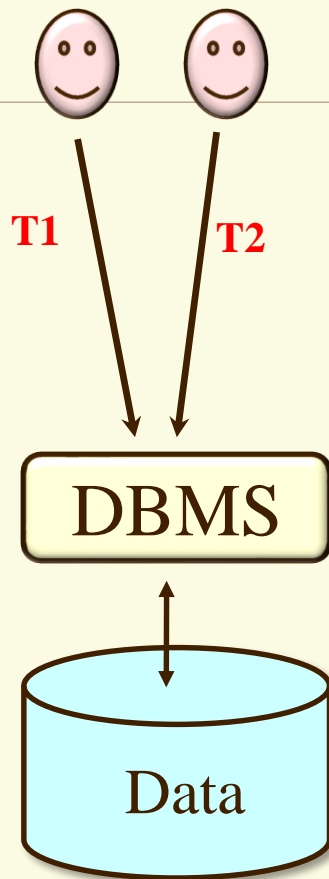
有一个关系 $R(A)$ 包含2个元组 $\{(5),(6)\}$ ，现有2个事务：

T1: Update R set $A = A + 1$;

T2: Update R set $A = 2 * A$.

假设每个事务都在保证事务隔离性和原子性的前提下并发执行，下面哪个选项不可能是 R 的正常最终状态？

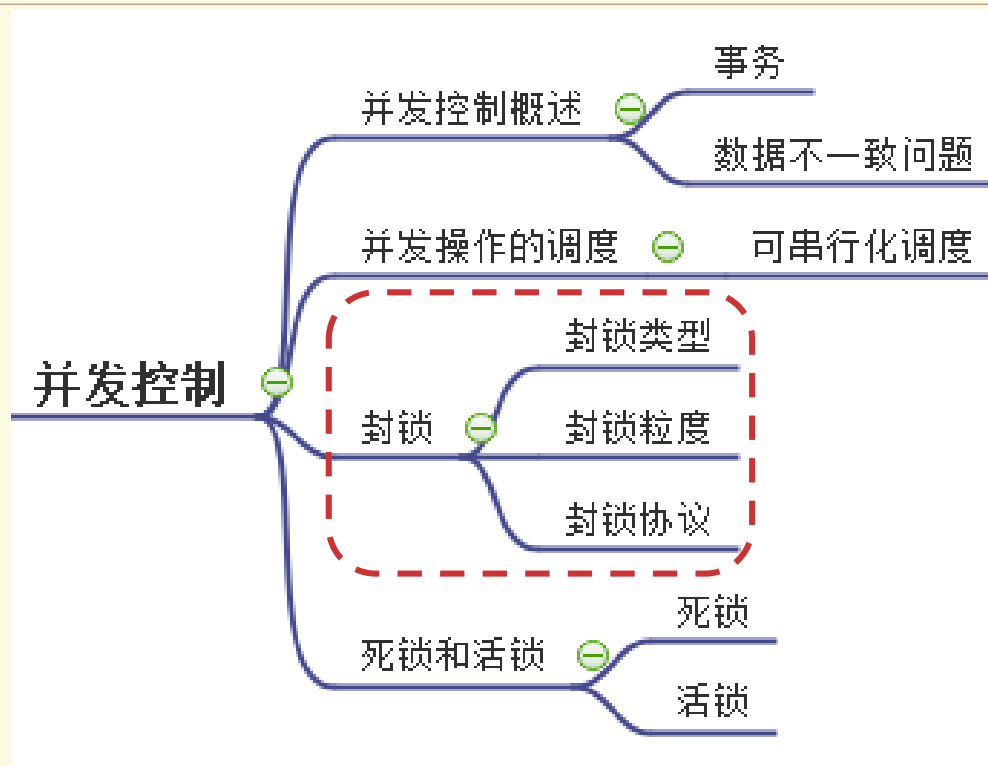
- A. $\{(10),(12)\}$
- B. $\{(11),(13)\}$
- C. $\{(11),(12)\}$
- D. $\{(12),(14)\}$



保证并发操作调度正确性的方法

- 封锁方法
 - 两段锁（2PL）协议
- 时标方法
- 乐观方法

3.1 并发控制概述



3.3 封锁

- 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
- 封锁是实现并发控制的一个非常重要的技术

3.3 封锁

3.3.1 封锁类型

3.3.2 封锁粒度

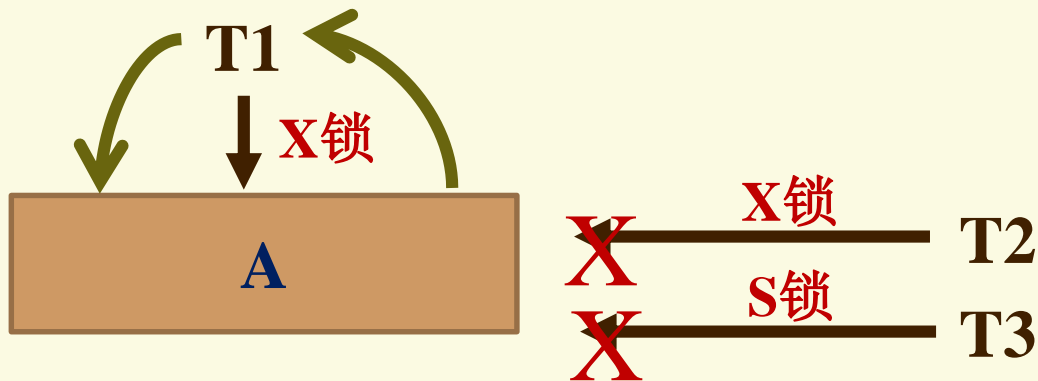
3.3.3 封锁协议

3.3.1 封锁类型

- 基本封锁类型
 - 排它锁（**eXclusive lock**，简记为**X锁**）
 - 共享锁（**Share lock**，简记为**S锁**）

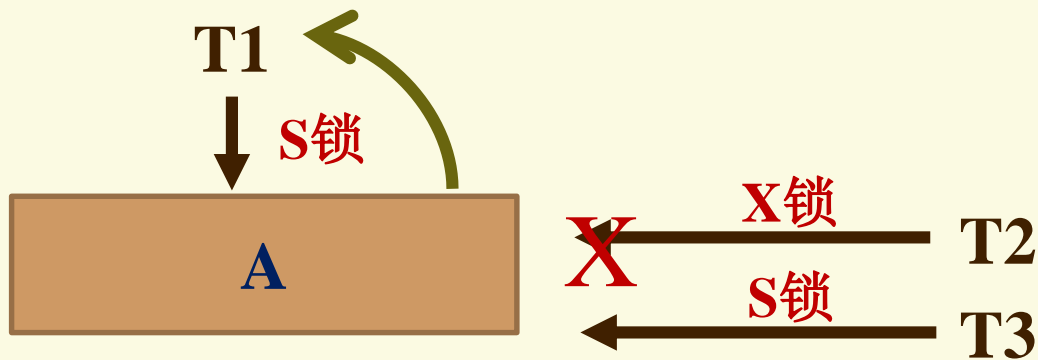
排它锁

- 排它锁又称为**写锁**、**X锁**。
 - 若事务T1对**数据对象A**加上X锁，则只允许T1读取和修改A，其它任何事务都不能再对A加任何类型的锁，直到T1释放A上的锁



共享锁

- 共享锁又称为读锁、S锁。
 - 若事务T1对数据对象A加上S锁，则其它事务只能再对A加S锁，而不能加X锁，直到T1释放A上的S锁。



封锁类型的相容性

$T_1 \backslash T_2$	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

3.3 封锁

3.3.1 封锁类型

3.3.2 封锁粒度

3.3.3 封锁协议

3.3.2 封锁粒度

- **X锁**和**S锁**都是加在某一个**数据对象**上的
- 封锁的数据对象可以是**逻辑单元**
 - 属性值、属性值集合、元组、关系、索引项、整个索引、整个数据库等
- 封锁的数据对象也可以是**物理单元**
 - 页（数据页或索引页）、块等

封锁粒度

- 封锁对象可以很大也可以很小
 - 对整个数据库加锁
 - 对某个属性值加锁
- 封锁对象的大小称为封锁的粒度

封锁粒度（续）

- 封锁粒度与系统的并发度和并发控制的开销密切相关

封锁的粒度	被封锁的对象	并发度	系统开销
大	少	小	小
小	多	高	大

- 选择封锁粒度时必须同时考虑开销和并发度两个因素，进行权衡，以求得最优的效果

封锁粒度一般原则

- 需要处理大量元组的用户事务
 - 以关系为封锁单元；
- 需要处理多个关系的大量元组的用户事务
 - 以数据库为封锁单位；
- 只处理少量元组的用户事务
 - 以元组为封锁单位

3.3 封锁

3.3.1 封锁类型

3.3.2 封锁粒度

3.3.3 封锁协议

3.3.3 封锁协议

在运用X锁和S锁对数据对象加锁时，需要约定一些规则，这些规则为**封锁协议**（Locking Protocol）

- 何时申请X锁或S锁
- 持锁时间
- 何时释放

两种封锁协议

- 三级封锁协议---保证数据一致性
- 两段锁协议---保证并行调度可串行性

一、三级封锁协议

1级封锁协议

2级封锁协议

3级封锁协议

1级封锁协议

- 事务T在修改数据R之前必须先对其加X锁，直到事务结束才释放
- 正常结束（COMMIT）
- 非正常结束（ROLLBACK）
- 1级封锁协议可防止丢失修改，并保证事务T是可恢复的
- 在1级封锁协议中，如果仅仅是读数据不对其进行修改，是不需要加锁的，所以它不能保证可重复读和不读“脏”数据。

用封锁机制解决三种数据不一致性示例

T ₁	T ₂
① Xlock A 获得	
② 读A=16	
③ A←A-1 写回A=15 Commit Unlock A	Xlock A 等待 等待 等待 等待
④	获得Xlock A 读A=15 A←A-1
⑤	写回A=14 Commit Unlock A

(a) 没有丢失修改

用封锁机制解决三种数据不一致性示例

T_1	T_2
<p>① Xlock A 获得</p> <p>② 读A=16 $A \leftarrow A-1$ 写回A=15</p> <p>③</p> <p>④ Rollback Unlock A</p>	<p>读A=15</p>

(a.1) 读“脏”数据

用封锁机制解决三种数据不一致性示例

T ₁	T ₂
①读A=50 读B=100 求和=150 ②	Xlock B 获得 读B=100 B←B*2 写回 B=200 Commit Unlock B
③读A=50 读B=200 求和=250 (验算不对)	

(a.2) 不可重复读

2级封锁协议

- 1级封锁协议加上事务T在读取数据R之前必须先对其加S锁，读完后即释放S锁
- 2级封锁协议可以防止丢失修改和读“脏”数据
- 在2级封锁协议中，由于读完数据后即可释放S锁，所以它不能保证可重复读

用封锁机制解决三种数据不一致性示例

T ₁	T ₂
① Xlock C 读C= 100 C←C*2 写回C=200	
②	
③ ROLLBACK (C恢复为100) Unlock C	Slock C 等待 等待 等待 等待
④	获得Slock C
⑤	读C=100 Commit C Unlock C

(c) 不读“脏”数据

用封锁机制解决三种数据不一致性示例

T ₁	T ₂
<p>① Slock A 获得 读A=50 Unlock A</p> <p>② Slock B 获得 读B=100 Unlock B</p> <p>③ 求和=150</p>	<p>Xlock B 等待 等待 获得Xlock B 读B=100 B←B*2 写回B=200 Commit Unlock B</p>

T ₁ (续)	T ₂
<p>④ Slock A 获得 读A=50 Unlock A</p> <p>Slock B 获得 读B=200 Unlock B</p> <p>求和=250 (验算不对)</p>	

(c.1) 不可重复读

3级封锁协议

- 1级封锁协议加上事务T在读取数据R之前必须先对其加S锁，直到事务结束才释放
- 3级封锁协议可防止丢失修改、读脏数据和不可重复读

用封锁机制解决三种数据不一致性示例

T ₁	T ₂
① Slock A 读A=50 Slock B	
读B=100 求和=150	
②	Xlock B 等待 等待 等待 等待 等待 等待 等待 等待
③ 读A=50 读B=100 求和=150 Commit Unlock A Unlock B	
④	获得Xlock B 读B=100 $B \leftarrow B * 2$ 写回B=200 Commit Unlock B
⑤	

(b) 可重复读

三级协议的主要区别

— 什么操作需要申请封锁以及何时释放锁（即持锁时间）

	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		√		√	√	√	√

二、两段锁协议

- 可串行性是并行调度正确性的唯一准则
- 两段锁（2PL）协议就是为保证并行调度可串行性而提供的封锁协议

“两段”锁的含义

- 两段锁协议的内容：

- 1. 在对任何数据进行读、写操作之前，事务首先要获得对该数据的封锁
- 2. 在释放一个封锁之后，事务不再获得任何其他封锁

- 事务分为两个阶段

- 第一阶段是获得封锁，也称为**扩展阶段**；
- 第二阶段是释放封锁，也称为**收缩阶段**。

两段锁协议（例）

- 事务1的封锁序列:

遵守2PL

Slock A ... Slock B ... Xlock C ... Unlock B ... Unlock A ... Unlock C;

- 事务2的封锁序列:

不遵守2PL

Slock A ... Unlock A ... Slock B ... Xlock C ... Unlock C ... Unlock B;

两段锁协议（续）

- 并行执行的所有事务均遵守两段锁协议，则对这些事务的所有并行调度策略都是可串行化的



- 所有遵守两段锁协议的事务，并行执行的结果一定是正确的
- 事务遵守两段锁协议是可串行化调度的**充分条件**，而不是必要条件
- 即可串行化的调度中，不一定所有事务都必须符合两段锁协议

3 并发控制

3.1 并发控制概述

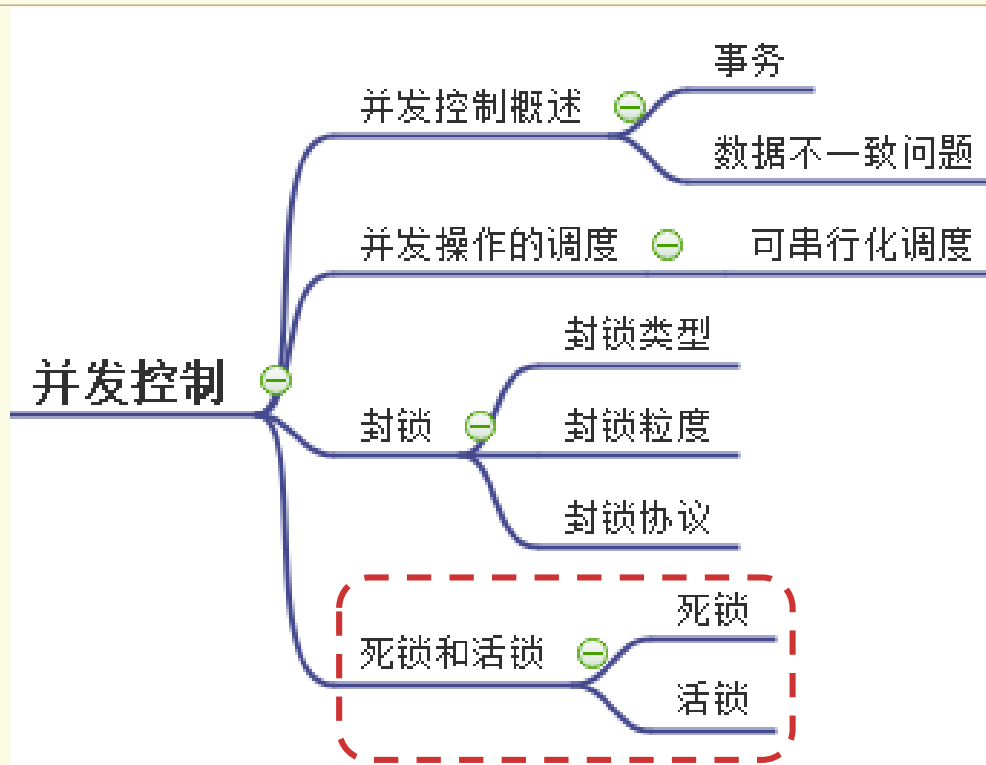
3.2 并发操作的调度

3.3 封锁

3.4 死锁和活锁

3.4 死锁和活锁

- 封锁技术可以有效地解决并行操作的一致性问题
- 但也带来一些新的问题
 - 死锁
 - 活锁



一、活锁

● 什么是活锁

T2有可能永远等待

T ₁	T ₂	T ₃	T ₄
lock R	.	.	.
.	lock R	.	.
.	等待	Lock R	.
Unlock	等待	.	Lock R
.	等待	Lock R	等待
.	等待	.	.

系统可能使某个事务永远处于等待状态，得不到封锁的机会，这种现象称为“活锁”（Live Lock）

如何避免活锁

采用**先来先服务**的策略

- 当多个事务请求封锁同一数据对象时，封锁子系统按请求封锁的先后次序对这些事务排队
- 该数据对象上的锁一旦释放，首先批准申请队列中第一个事务获得锁

二、死锁

什么是死锁

T1	T2
Xlock R ₁	·
·	·
·	Xlock R ₂
·	·
Xlock R ₂ 等待	·
	Xlock R ₁

系统中有两个或两个以上的事务都处于等待状态，并且每个事务都在等待其中另一个事务解除封锁，它才能继续执行下去，结果造成任何一个事务都无法继续执行，这种现象称系统进入了“**死锁**”（Dead Lock）状态。

解决死锁的两类方法

1. 死锁的预防
2. 死锁的诊断与解除

预防死锁为何能解决死锁

- 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

(1) 一次封锁法

- 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
 - 一次就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
 - 数据库中数据是不断变化的，原来不要求封锁的数据，在执行过程中可能会变成封锁对象，所以很难事先精确地确定每个事务所要封锁的数据对象

(2) 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
 - 维护成本高
 - 数据库系统中可封锁的数据对象极其众多，并且随数据的插入、删除等操作而不断地变化，要维护这样极多而且变化的资源的封锁顺序非常困难，成本很高。

顺序封锁法存在的问题

- 难于实现

- 事务的封锁请求可以随着事务的执行而动态地决定，很难事先确定每一个事务要封锁哪些对象，因此也就很难按规定的顺序去施加封锁。

- 例：规定数据对象的封锁顺序为A,B,C,D,E。事务T3起初要求封锁数据对象B,C,E，但当它封锁了B,C后，才发现还需要封锁A，这样就破坏了封锁顺序。

死锁的预防

- 结论

- 在操作系统中广为采用的预防死锁的策略并不很适合数据库的特点
- DBMS在解决死锁的问题上更普遍采用的是**诊断并解除死锁**的方法

2. 死锁的诊断与解除

- 方法

- 由DBMS的并发控制子系统定期检测系统中是否存在死锁
- 一旦检测到死锁，就要设法解除

超时法

等待图法

检测死锁的方法（1）

- 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

检测死锁的方法（2）

● 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
- 并发控制子系统周期性地（比如每隔**1 min**）检测事务等待图，如果发现图中存在回路，则表示系统中**出现了死锁**

死锁的解除

- 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

本章小结

数据库基础概念

数据库原理

关系数据库

关系数据模型

关系数据语言

数据库设计

数据库管理

数据库新技术

安全性 ⊕

完整性 ⊕

并发控制 ⊖

故障和恢复

复制

并发控制概述 ⊖

并发操作的调度 ⊖

封锁 ⊖

死锁和活锁 ⊖

事务

数据不一致问题

可串行化调度

封锁类型

封锁粒度

封锁协议

死锁

活锁