数据库系统原理

李瑞轩

华中科技大学计算机学院

第十二章 并发控制

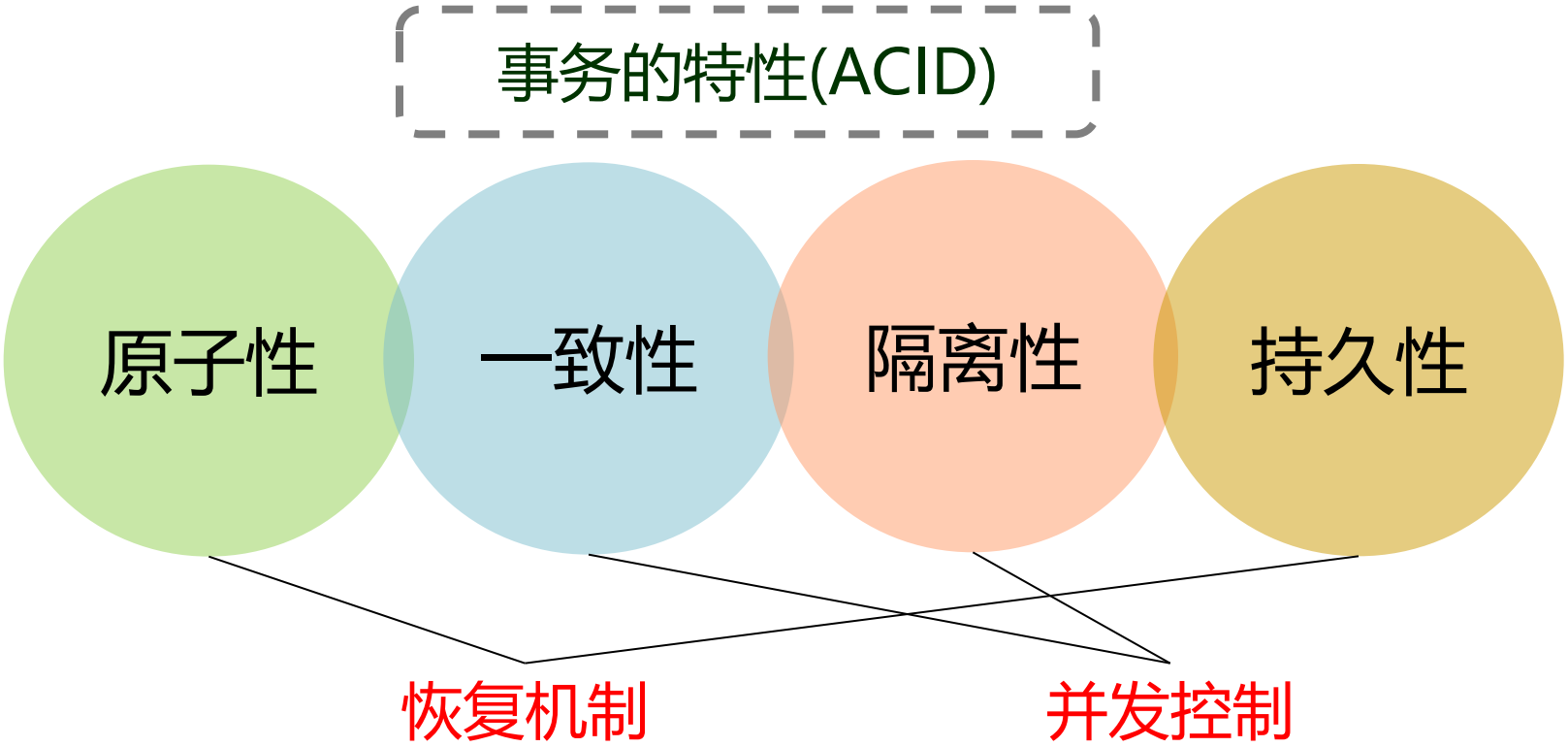
- 12.1 并发控制概述
- 12.2 并发调度的可串行性
- 12.3 封锁
- 12.4 活锁和死锁
- 12.5 两段锁协议
- 12.6 封锁的粒度

• 学习目标

- 理解并掌握并发控制的基本概念及其必要性
- 理解并掌握并发调度的可串行性原则
- 理解并基本掌握基于封锁的并发控制方法

回顾：事务的基本概念和ACID特性

- 事务：用户定义的一个操作序列，这些操作要么都执行，要么都不执行，是一个不可分割的工作单位

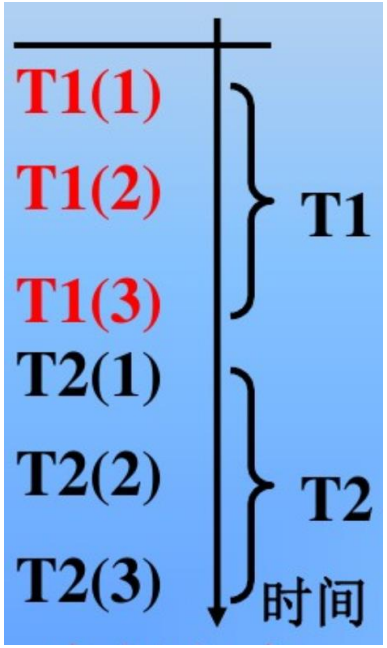


多用户数据库系统

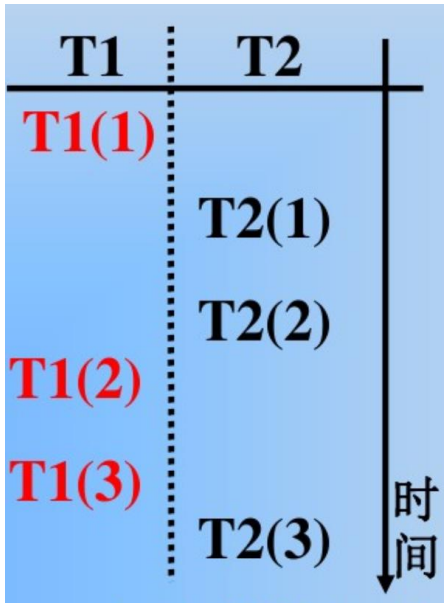
- 允许多个用户同时使用数据库系统
  - 飞机订票数据库系统
  - 银行数据库系统
  - 电子商务数据库系统
  - ……

  特点：在同一时刻并发运行的事务数可达数百上千个

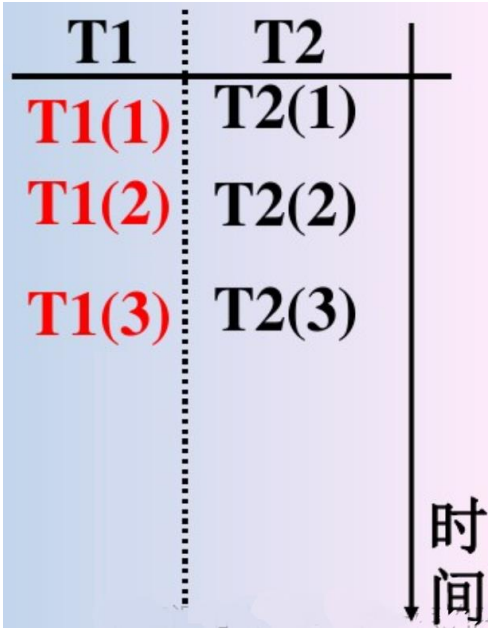
多个事务并发执行



串行方式



交叉并发方式
(单处理机系统)



同时并发方式
(多处理机系统)

事务的并发执行

银行转账示例

- T_1：用户1要从账户A转账50元到账户B
  T_2：用户2要从账户B转账10元到账户A

T_1	T_2
read(A)	
A:=A-50	
write(A)	
	read(B)
	B:=B-10
	write(B)
read(B)	
B:=B+50	
write(B)	
commit T_1	
	read(A)
	A:=A+10
	write(A)
	commit T_2

初始值：

A = 100

B = 100

转账后：

A = 60

B = 140

银行转账示例

- T_1：用户1要从账户A转账50元到账户B
  T_2：用户2要从账户B转账10元到账户A

T_1	T_2
read(A)	
A:=A-50	
write(A)	
	read(B)
read(B)	
	B:=B-10
	write(B)
B:=B+50	
write(B)	
commit T_1	
	read(A)
	A:=A+10
	write(A)
	commit T_2

初始值：

A = 100

B = 100

转账后：

A = 60

B = 150

12.1 并发控制概述

12.1.1 问题的提出

- 事务并发执行的必然性
  - 系统吞吐率（System throughput）
  - 响应时间（Response time）
- 数据库管理系统必须提供并发控制机制
- 并发控制机制是衡量一个数据库管理系统性能的重要标志之一

12.1.1 问题的提出

- 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
  - 可能会存取和存储不正确的数据，破坏事务隔离性和数据库的一致性
- 不正确并发控制的后果
  - 读“脏”数据(W-R conflict)
  - 不可重复读(R-W conflict)
  - 丢失更新(W-W conflict)

读“脏”数据(W-R conflict)

- 1. 定义
  - 读修改后未提交的随后又被撤消(Rollback)的数据。

T_1	T_2
read(A)	
write(A)	
	read(A)
	read(B)
	write(B)
	Commit T_2
read(B)	
write(B)	
Abort T_1	

不可重复读(R-W conflict)

- 1. 定义
  - 同一事务重复读同一数据，但获得结果不同。

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
	Commit T_2
read(A)	
Commit T_1	

三种情况：

(1) 数据修改

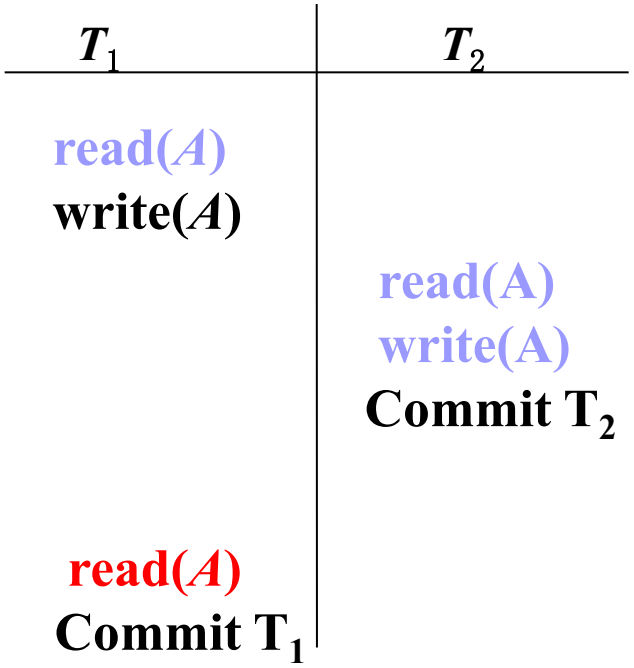
(2) 删除记录

(3) 插入记录

幻影现象：Phantom row

不可重复读(R-W conflict)

- 1. 定义
  - 同一事务重复读同一数据，但获得结果不同。

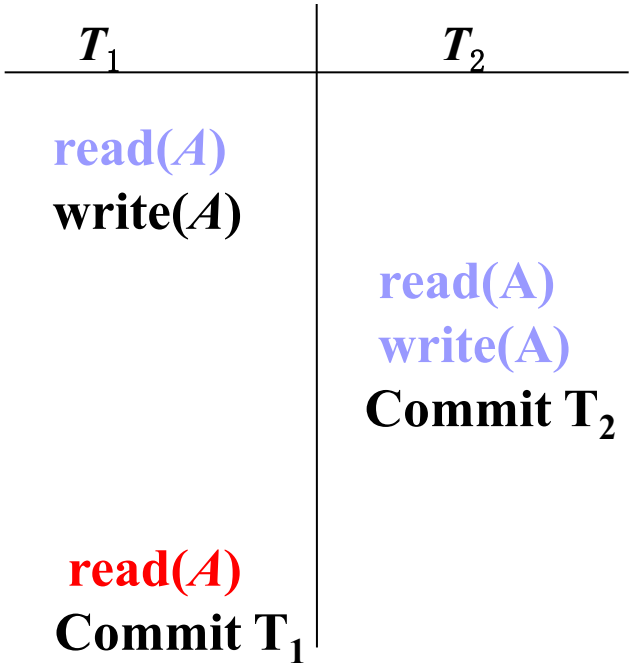


三种情况：

- 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。

不可重复读(R-W conflict)

- 1. 定义
  - 同一事务重复读同一数据，但获得结果不同。



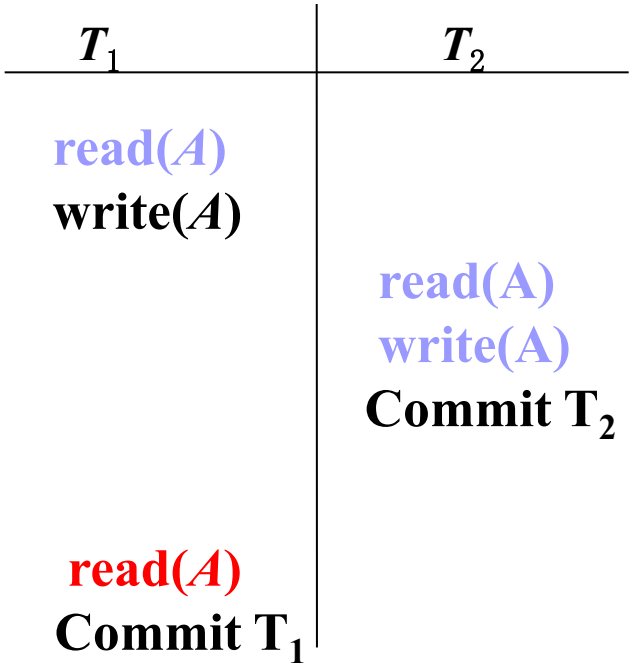
三种情况：

- 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。

幻影现象：Phantom row

不可重复读(R-W conflict)

- 1. 定义
  - 同一事务重复读同一数据，但获得结果不同。



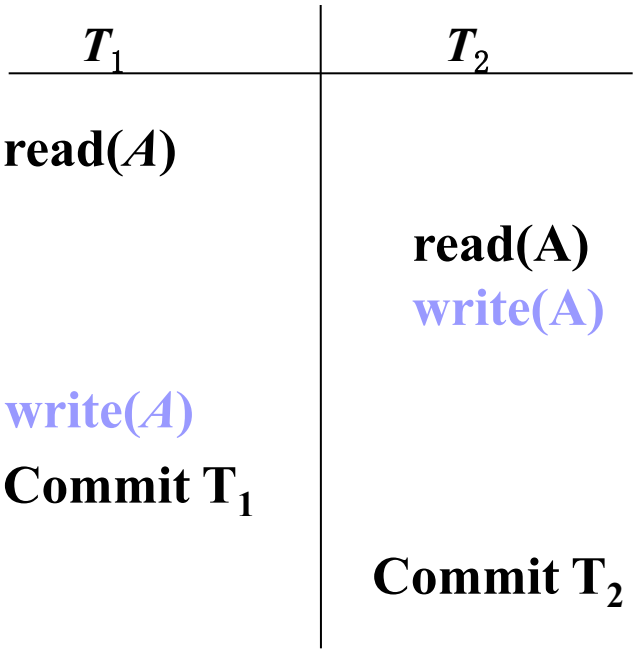
三种情况：

- 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。

幻影现象：Phantom row

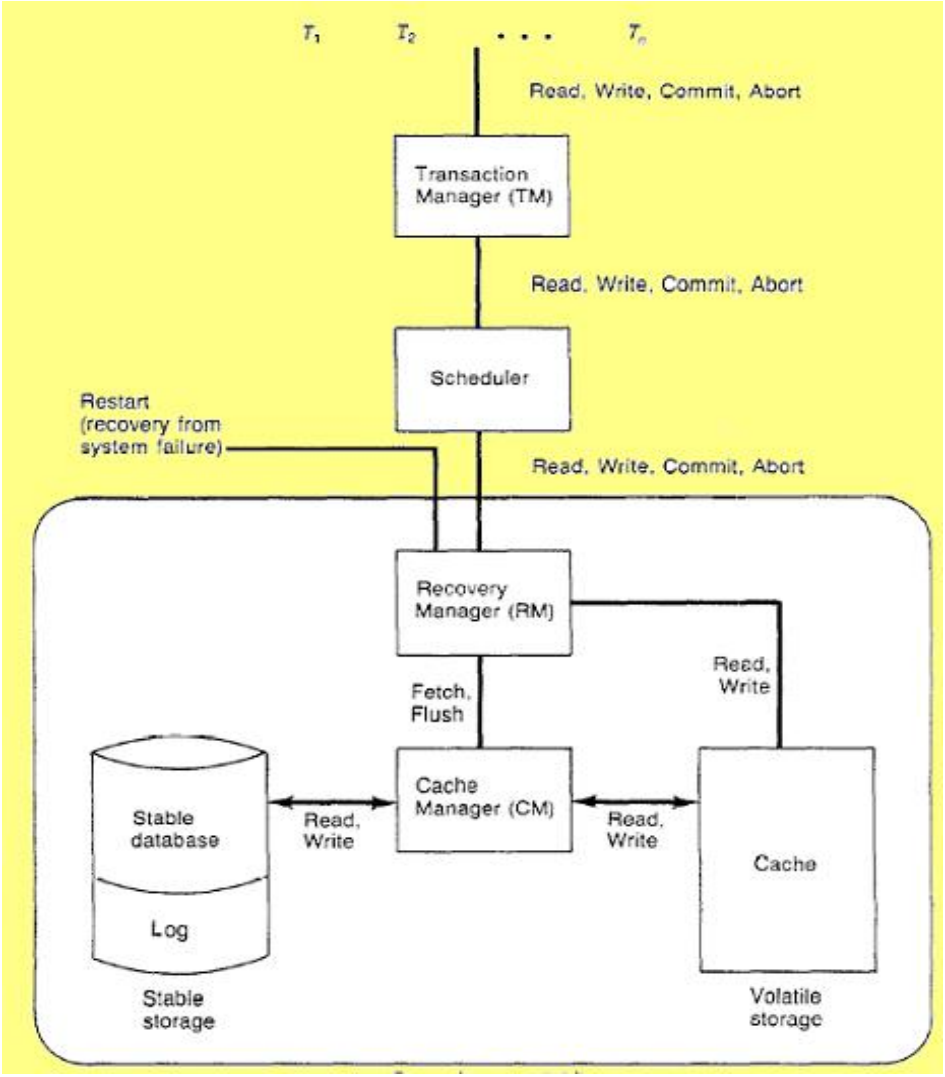
丢失更新(W-W conflict)

- 定义
  - 两个以上事务从DB中读入同一数据并修改之，其中一事务的提交结果破坏了另一事务的提交结果，导致该事务对DB的修改被丢失。



12.1.2 并发控制的定义

- 并发控制的定义
  - 当多个事务并发执行时，必须采取一定的控制措施以使某个事务的执行不会对其他事务产生不良影响，这称为并发控制。



12.1.3 并发控制的主要技术

- 并发控制的主要技术
  - 封锁(Locking)
  - 时间戳(Timestamp)
  - 乐观控制法
- 商用的DBMS一般都采用封锁方法

阿里OceanBase： TPC-C测试全球领先 (2019)

Hardware Vendor	System	Performance (tpmC)	Price/tpmC	Watts/KtpmC	System Availability	Database	Operating System	TP Monitor	Date Submitted
ANT FINANCIAL	Alibaba Cloud Elastic Compute Service Cluster	60,880,800	6.25 CNY	NR	10/02/19	OceanBase v2.2 Enterprise Edition with Partitioning, Horizontal Scalab	Aliyun Linux 2	Nginx 1.15.8	10/01/19
ORACLE	SPARC SuperCluster with T3-4 Servers	30,249,688	1.01 USD	NR	06/01/11	Oracle Database 11g R2 Enterprise Edition w/RAC w/Partitioning	Oracle Solaris 10 09/10	Oracle Tuxedo CFSR	12/02/10
IBM	IBM Power 780 Server Model 9179-MHB	10,366,254	1.38 USD	NR	10/13/10	IBM DB2 9.7	AIX Version 6.1	Microsoft COM+	08/17/10
ORACLE	SPARC T5-8 Server	8,552,523	.55 USD	NR	09/25/13	Oracle 11g Release 2 Enterprise Edition with Oracle Partitioning	Oracle Solaris 11.1	Oracle Tuxedo CFSR	03/26/13
ORACLE	Sun SPARC Enterprise T5440 Server Cluster	7,646,486	2.36 USD	NR	03/19/10	Oracle Database 11g Enterprise Edition w/RAC w/Partitioning	Sun Solaris 10 10/09	Oracle Tuxedo CFSR	11/03/09
IBM	IBM Power 595 Server Model 9119-FHA	6,085,166	2.81 USD	NR	12/10/08	IBM DB2 9.5	IBM AIX 5L V5.3	Microsoft COM+	06/10/08
BULL	Bull Escala PL6460R	6,085,166	2.81 USD	NR	12/15/08	IBM DB2 9.5	IBM AIX 5L V5.3	Microsoft COM+	06/15/08
ORACLE	Sun Server X2-8	5,055,888	.89 USD	NR	07/10/12	Oracle Database 11g R2 Enterprise Edition w/Partitioning	Oracle Linux w/Unbreakable Enterprise Kernel R2	Oracle Tuxedo CFSR	03/27/12
ORACLE	Sun Fire X4800 M2 Server	4,803,718	.98 USD	NR	06/26/12	Oracle Database 11g R2 Enterprise Edition	Oracle Linux w/Unbreakable Enterprise Kernel R2	Oracle Tuxedo CFSR	01/17/12
Hewlett Packard Enterprise	HP Integrity Superdome-Itanium2/1.6GHz/24MB iL3	4,092,799	2.93 USD	NR	08/06/07	Oracle Database 10g R2 Enterprise Edition w/Partitioning	HP-UX 11.i v3	BEA Tuxedo 8.0	02/27/07

国际事务处理性能委员会（TPC），TPC-C是在线事务处理(OLTP)的基准程序

华为GaussDB：TPC-C性能高30%

TPC-C模型多核性能对比测试(X86)

TPC-C模型多核性能对比测试(Kunpeng)

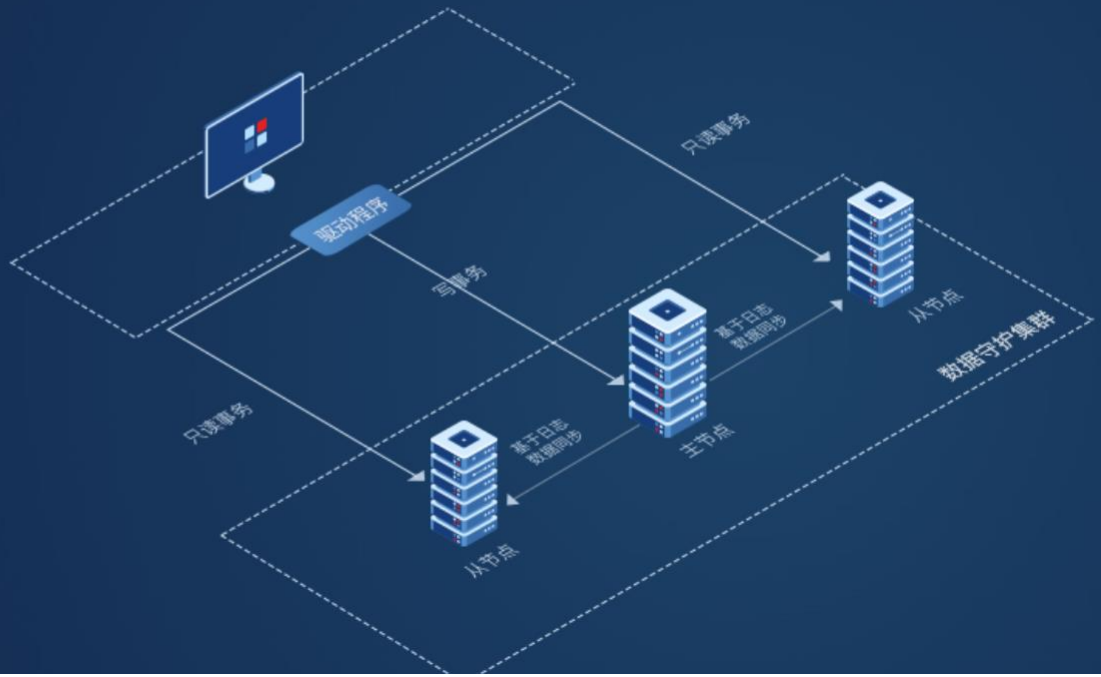
GaussDB@Kunpeng920
VS
O@Intel6148

TPC-C 性能高30%

达梦DM8：读写分离集群DMRWC

达梦读写分离集群DMRWC

达梦读写分离集群（DMRWC）是DM8提供的一个用于提升并发事务处理性能的集群组件。在一个高并发的事务型系统中，当写事务占的比例相对读事务较小时，DM8提供的这种独具创新的方案DMRWC，可通过客户端来实现读、写事务的自动分离，读事务在备机执行，写事务在主机执行，减轻主机的负载。



达梦数据库

12.2 并发调度的可串行性

- 12.2.1 事务调度的基本概念

  - 1. 定义

    - 调度是语句在系统中执行的时间顺序。一组事务的一个调度必须包含这一组事务的全部语句，并且必须保持语句在各个事务中出现的顺序。

事务调度示例

T_1从帐户A过户￥50到帐户B

T_1: read(A);
A:=A-50;
write(A);
read(B);
B:=B+50;
write(B).

T_2从帐户A过户10%的存款余额到帐户B.

T_2: read(A);
$temp$:=A*0.1;
A:=A-$temp$;
write(A);
read(B);
B:=B+$temp$;
write(B).

执行前　A：￥1000，　B：￥2000

事务调度示例(续)

调度1：串行调度，T_2跟在T_1之后

执行后 A：￥855， B：￥2145

调度2：串行调度，T_1跟在T_2之后

执行后 A：￥850， B：￥2150

事务调度示例(续)

调度3：并发调度

等价于调度1，执行后

A：￥855

B：￥2145

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 T_1	
	read(*B*)
	B:=*B*+*temp*
	write(*B*)
	Commit T_2

事务调度示例(续)

调度4：并发调度

不等价于调度1，执行后

A：￥950

B：￥2100

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 T_1	
	B:=B+temp
	write(B)
	Commit T_2

12.2 并发调度的可串行性(续)

- 2. 并发调度的正确性标准
  - 单个事务
    - 每个事务都能保证DB的正确性
  - 多个事务
    - 多个事务以任意串行方式执行都能保证DB的正确性

T_1，T_2，T_3：

$T_1 \rightarrow T_2 \rightarrow T_3$
$T_1 \rightarrow T_3 \rightarrow T_2$
$T_2 \rightarrow T_1 \rightarrow T_3$
$T_2 \rightarrow T_3 \rightarrow T_1$

……

12.2 并发调度的可串行性(续)

- 2. 并发调度的正确性标准
  - 可串行化
    - 考虑哪些调度能保证一致性，哪些不能的问题
  - 若某并发调度S与一个串行调度的执行有相同的效果，则称调度S是可串行化的
  - 等价概念
    - 冲突可串行化(调度优先图)
    - 视图可串行化

12.2.2 冲突可串行化调度

- 可串行化调度的**充分条件**
  - 一个调度Sc在保证**冲突操作**的次序不变的情况下，通过交换两个事务不冲突操作的次序得到另一个调度Sc’，如果Sc’是串行的，称调度Sc为冲突可串行化的调度
  - 一个调度是冲突可串行化，一定是可串行化的调度

冲突可串行化调度（续）

冲突操作

- 冲突操作是指不同的事务对同一个数据的读写操作和写写操作
  - Ri (x)与Wj(x)　　/* 事务Ti读x，Tj写x*/
  - Wi(x)与Wj(x)　　/* 事务Ti写x，Tj写x*/
- 其他操作是不冲突操作
- 不同事务的冲突操作和同一事务的两个操作不能交换(Swap)

冲突可串行化调度（续）

例：今有调度

Sc1=r1(A)w1(A)r2(A)w2(A)r1(B)w1(B)r2(B)w2(B)

- 把w2(A)与r1(B)w1(B)交换，得到：

  r1(A)w1(A)r2(A)r1(B)w1(B)w2(A)r2(B)w2(B)

- 再把r2(A)与r1(B)w1(B)交换：

  Sc2 = r1(A)w1(A)r1(B)w1(B)r2(A)w2(A)r2(B)w2(B)

- Sc2等价于一个串行调度T1，T2，即Sc1冲突可串行化的调度

冲突可串行化调度（续）

- 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]有3个事务

  T1=W1(Y)W1(X)，T2=W2(Y)W2(X)，T3=W3(X)

  - 调度L1=W1(Y)W1(X)W2(Y)W2(X) W3(X)是一个串行调度。
  - 调度L2=W1(Y)W2(Y)W2(X)W1(X)W3(X)
    - 不满足冲突可串行化，但是调度L2是可串行化的，因为L2执行的结果与调度L1相同，Y的值都等于T2的值，X的值都等于T3的值。

12.2.3 可恢复的并发调度

- 12.2.3 可恢复的并发调度

  - 可恢复调度应满足：对于每一对事务 T_i 和 T_j，如果 T_j 读取了由 T_i 所写的数据项，则 T_i 先于 T_j 提交。

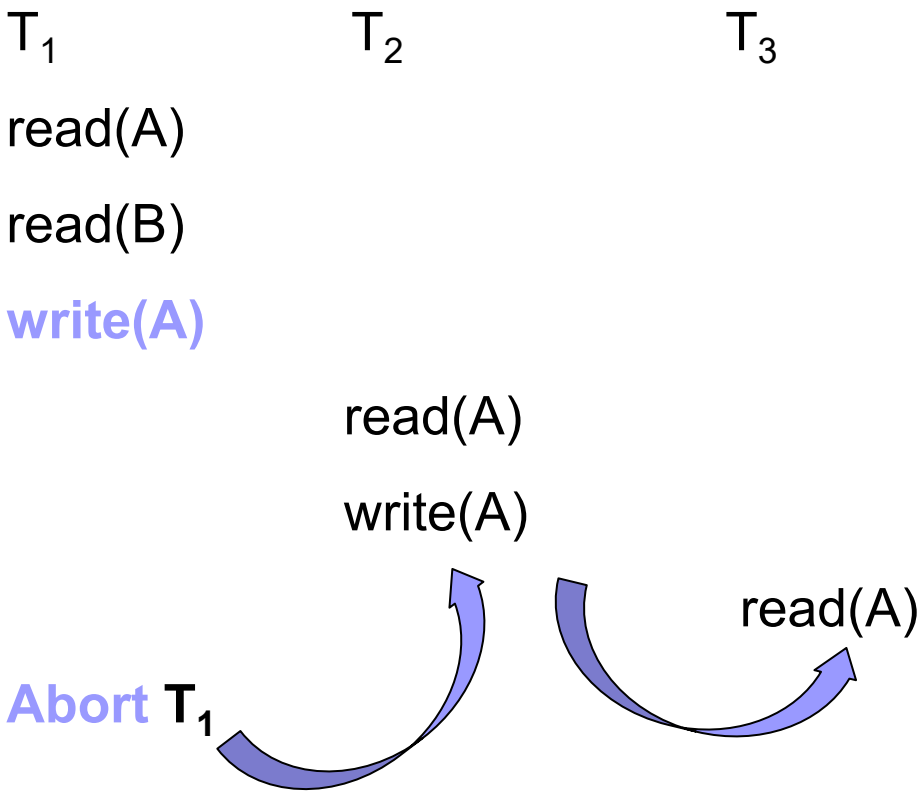
一个不可恢复的调度

T1	T2
read(A)	
write(A)	
	read(A)
	commit
read(B)	

可恢复的并发调度(续)

1. 可恢复的并发调度

- 级联回滚：因一个事务故障导致一系列事务回滚的现象



可恢复的并发调度(续)

2. 可恢复的并发调度

- 无级联调度应满足
  - 对于每对事务 T_i和 T_j，如果 T_j读取了由 T_i所写的数据项，则 T_i必须在 T_j这一读取前提交

- 若某调度是无级联调度，则该调度一定是可恢复调度。

12.3 封锁

并发控制的各种策略：

- 1. 思想
  - 通过并发控制机制使得多个事务的并发调度是可串行化的

- 2. 方法

基于封锁的方法(2PL) → 悲观方法

基于时间戳的方法(TO)
基于有效性确认的方法(SGT)
多版本模型(MVCC)
} → 乐观方法

12.3 封锁

- 12.3.1 封锁的基本概念
- 12.3.2 封锁协议

12.3.1 封锁的基本概念

- 1. 封锁的定义
  - 事务T可以向系统发出请求，对某个数据对象加锁(Lock)，于是事务T对这个数据对象就有一定的控制，直到T释放它的锁为止(Unlock)
- 2. 锁类型
  - 排它锁(X锁)：若事务T对数据D加上X锁，则其他事务不能对D进行任何封锁，保证了其他事务不能再读取和修改D
  - 共享锁(S锁)：若事务T对数据D加上S锁，则其他事务能对D加S锁，保证了其他事务能读取但不能修改D

12.3.1 封锁的基本概念(续)

- 3.锁的相容矩阵

	X	S	-
X	false	false	true
S	false	true	true
—	true	true	true

- 4. 封锁的粒度
  - 封锁对象的大小称为封锁的粒度
  - 关系数据库的封锁对象：属性值、元组、关系、索引

12.3.2 封锁协议

- 1. 封锁协议的定义
  - 系统中的每一个事务都必须遵从的关于何时对数据项加何种锁，何时解锁的一组规则
- 2. 三级封锁协议
  - 一级
  - 二级
  - 三级

3. 一级封锁协议

- 策略
  - 事务Ti在修改数据Di之前须先对Di加X锁，直到事务Ti结束才释放。
- 功能
  - 防止丢失修改
  - 保证Ti可恢复

时间	T_A	X值	T_B	T_B对X的修改丢失了T_A对X的修改，所以结果均为99。
t1	X=100	100		
t2			X=100	
t3	X：=X-1 COMMIT	99		
t4		99	X：=X-1 COMMIT	

一级封锁协议

时间	T_A	X值	T_B	说明
t1	X Lock X X=100	100		T_A对X加锁成功后读X
t2			X Lock X 等待	T_B对X加X锁未成功则等待
t3	X:=X-1 COMMIT UNLOCK X	99	等待	T_A修改，X结果写回DB释放X锁
t4			X Lock X X=99 X：=X-1 COMMIT UNLOCK X	T_B获得X的X锁读X得99（已更新后结果）将修改后X（98）写回DB

4. 二级封锁协议

- 策略
  - 在一级封锁协议加上事务Ti在读取Di之前须先对Di加S锁，读完后即可释放该S锁。
- 功能
  - 防止丢失修改
  - 防止读脏

时间	T_A	X值	T_B	说明
t1	X Lock X X=100 X：=X*2	100 200		T_A先获得X锁
t2			S Lock X 等待	T_B申请S锁 T_A未释放， T_B等待
t3	Rollback UNLOCK X	100		T_A因故撤消 X值恢复为100
t4			S Lock X X=100 Unlock X	T_B获得S锁 T_B读到值与DB中值一致 防止了读脏

5. 三级封锁协议

- 策略
  - 在一级封锁协议上加上Ti读Di前须先对Di加S锁，直至Ti结束后才释放该S锁。
- 功能
  - 防止丢失修改
  - 防止读脏
  - 防止不可重复读

时间	T_A	DB中值	T_B
t1	A=50 B=100 C：=A+B	50（A） 100（B）	
t2		50（A） 200（B）	B=100 B：=B*2 COMMIT
t3	A=50 B=200 D：=A+B	50（A） 200（B）	

三级封锁协议

时间	T_A	DB中A、B值	T_B	说明
t1	S Lock A A=50 S Lock B B=100 C：=A+B	A：50 B：100 A：50 C：150		T_A对A、B加S锁 T_B不能再对之加X锁
t2			X Lock B 等待	T_B对B加锁不成功 T_B等待
t3	A=50 B=100 D：=A+B COMMIT UNlock A UNlock B	A：50 B：100 C：150 D：150		T_B等待 T_B重读A、B计算、结果相同
t4		A：50 B：200 C：150 D：150	X Lock B B=100 B：=B*2 写回B=200	T_B获得B的X锁

• 三级封锁协议小结

	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”	可重读
1级协议		√			√		
2级协议		√	√		√	√	
3级协议		√		√	√	√	√

- 事务隔离级别与并发问题

	Lost Updates	Dirty Reads	Non-repeating Reads	Phantom Reads
READ UNCOMMITTED				
READ COMMITTED		✓		
REPEATABLE READ	✓	✓	✓	
SERIALIZABLE	✓	✓	✓	✓

12.4 活锁和死锁

- 1. 活锁
  - 定义
    - 某事务T等待对数据项加排它锁，而另一个事务序列中的每一个事务都对该数据项加共享锁，则事务T永远得不到进展
  - 解决方法
    - FCFS（先来先服务）

12.4 活锁和死锁(续)

- 活锁示例

T_1	T_2	T_3	T_4
S lock R	.	.	.
.	X lock R	.	.
.	等待	S Lock R	.
Unlock	等待	.	S Lock R
.	等待	Lock R	等待
.	等待	.	等待
.	等待	Unlock	等待
.	等待	.	Lock R
.	等待	.	.

12.4 活锁和死锁(续)

- 2. 死锁定义
  - 两个或两个以上事务均处于等待状态，每个事务都在等待其中另一个事务封锁的数据，导致任何事务都不能继续执行的现象称为死锁。

时间	T_A	T_B
t1	X Lock A	
t2		X Lock B
t3	X Lock B 等待	
t4		X Lock A 等待
t5	等待	等待

12.4 活锁和死锁(续)

- 3. 死锁原因
  - 互斥（排它性控制）
  - 不可剥夺（释放前，其它事务不能剥夺）
  - 部分分配（每次申请一部分，申请新的数据封锁时，又占用已获得者）
  - 环路（循环链中，每个事务获得封锁的数据同时又被另一事务请求）

12.4 活锁和死锁(续)

- 4. 死锁的解决办法
  - 预防
    - 一次封锁法
    - 顺序封锁法
  - 诊断（实际中更多采用的方法）
    - 超时法
    - 等待图法

A. 死锁的预防

- (1) 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
  - 存在的问题
    - 降低系统并发度
    - 难于事先精确确定封锁对象

死锁的预防(续)

- (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：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

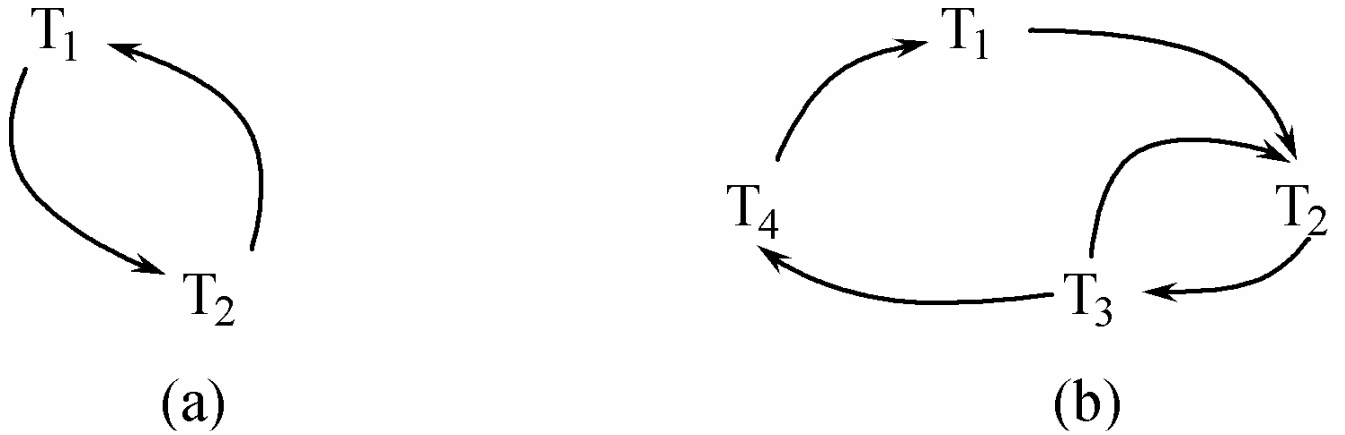
B. 死锁的诊断与解除

- (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：
    - 时限设置不合理，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

死锁的诊断与解除(续)

- (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

等待图法（续）



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

等待图法（续）

- 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。

死锁的诊断与解除（续）

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

12.5 两段锁(2PL)协议

- 1. 定义
  - 要求每个事务分两个阶段提出加锁和解锁申请
  - 扩展阶段：事务可以获得锁，但不能释放锁
  - 收缩阶段：事务可以释放锁，但不能获得新锁

- 2.策略
  - 在对任何数据读、写之前，须先获得该数据锁
  - 在释放一个封锁之后，该事务不能再申请任何其它锁

两段锁(2PL)协议（续）

示例：

事务T_i遵守两段锁协议，其封锁序列是：

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

|←　扩展阶段　→|　|←　收缩阶段　→|

事务T_j不遵守两段锁协议，其封锁序列是：

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

两段锁(2PL)协议（续）

- 3. 理论基础
  - 若所有并行事务都遵守2PL协议，则对这些事务的所有并行调度策略都是可串行化的。

- 4. 说明
  - 2PL协议是可串行化的充分条件，不是必要条件。

- 符合2PL协议的调度

时间	T_A	DB值	T_B	
t1	S Lock B B=2 Y=B X Lcok A	B：2		
t2			X Lock B 等待	T_A未释放A的X锁 T_B等待
t3	A：=Y+1 写回A=3 UNLock B UNLock A	B：2 A：3		T_A释放B、A锁
t4		B：2 A：3	S Lock A A=3 Y=A	T_B获得锁
t5		B：4 A：3	X Lock B B=Y+1 写回B=4 UNLockB UNLockA	

- 不符合2PL协议的调度

时间	T_A	DB值	T_B
t1	S Lock B B=2 Y：=B UNLock B X Lock A	B=2	
t2			S Lock A 等待
t3	A：=Y+1 写回A=3 UNLock A		等待
t4		A=3 B=4	S Lock A A=3 Y：=A UN Lock A X Lock B B：=Y+1 写回B=4 UNLock B

两段锁协议（续）

- 两段锁协议与防止死锁的一次封锁法
  - 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，因此一次封锁法遵守两段锁协议
  - 两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁，因此遵守两段锁协议的事务可能发生死锁

两段锁协议（续）

遵守两段锁协议的事务可能发生死锁

T_1	T_2
Slock B	
R(B)=2	
	Slock A
	R(A)=2
Xlock A	
等待	Xlock B
等待	等待

遵守两段锁协议的事务可能发生死锁

12.6 封锁的粒度

- 封锁对象的大小称为封锁粒度(Granularity)
- 封锁的对象：逻辑单元 | 物理单元

  例：在关系数据库中，封锁对象：

  - 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
  - 物理单元：页（数据页或索引页）、物理记录等

选择封锁粒度的原则

- 封锁粒度与系统的并发度和并发控制的开销密切相关。
  - 封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；
  - 封锁的粒度越小，并发度较高，但系统开销也就越大。

选择封锁粒度的原则（续）

- 多粒度封锁 (Multiple Granularity Locking)

  在一个系统中同时支持多种封锁粒度供不同的事务选择。

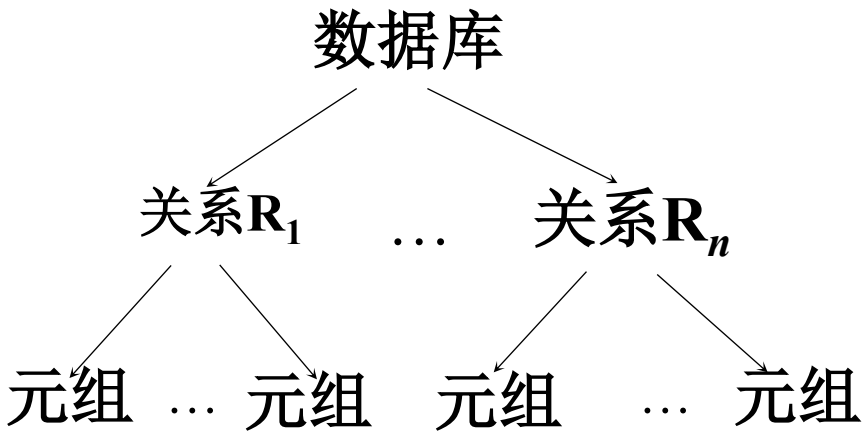
- 选择封锁粒度

  同时考虑封锁开销和并发度两个因素，适当选择封锁粒度。

  - 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
  - 需要处理大量元组的用户事务：以关系为封锁单元
  - 只处理少量元组的用户事务：以元组为封锁单位

12.6.1 多粒度封锁

- 多粒度树
  - 以树形结构来表示多级封锁粒度
  - 根结点是整个数据库，表示最大的数据粒度
  - 叶结点表示最小的数据粒度



三级粒度树

多粒度封锁协议

- 允许多粒度树中的每个结点被独立地加锁
  - 对一个结点加锁意味着这个结点的所有后裔结点也被加以同样类型的锁
- 在多粒度封锁中一个数据对象可能以两种方式封锁
  - 显式封锁: 直接加到数据对象上的封锁
  - 隐式封锁: 该数据对象没有独立加锁，是由于其上级结点加锁而使该数据对象加上了锁
  - 显式封锁和隐式封锁的效果是一样的

显式封锁和隐式封锁（续）

- 对某个数据对象加锁，系统要检查
  - 该数据对象
    - 有无显式封锁与之冲突
  - 所有上级结点
    - 检查本事务的显式封锁是否与该数据对象上的隐式封锁冲突：(由上级结点已加的封锁造成的)
  - 所有下级结点
    - 看上面的显式封锁是否与本事务的隐式封锁（将加到下级结点的封锁）冲突

12.6.2 意向锁

- 引进意向锁（Intention Lock）目的
  - 提高对某个数据对象加锁时系统的检查效率
- 意向锁的使用
  - 如果对一个结点加意向锁，则说明该结点的下层结点正在被加锁
  - 对任一结点加基本锁，必须先对它的上层结点加意向锁
    - 例如，对任一元组加锁时，必须先对它所在的数据库和关系加意向锁

常用意向锁

- 意向共享锁(Intent Share Lock，简称IS锁)
  - 如果对一个数据对象加IS锁，表示它的后裔结点拟（意向）加S锁。
- 意向排它锁(Intent Exclusive Lock，简称IX锁)
  - 如果对一个数据对象加IX锁，表示它的后裔结点拟（意向）加X锁。
- 共享意向排它锁(Share Intent Exclusive Lock，简称SIX锁)
  - 如果对一个数据对象加SIX锁，表示对它加S锁，再加IX锁，即SIX = S + IX。

意向锁的相容矩阵

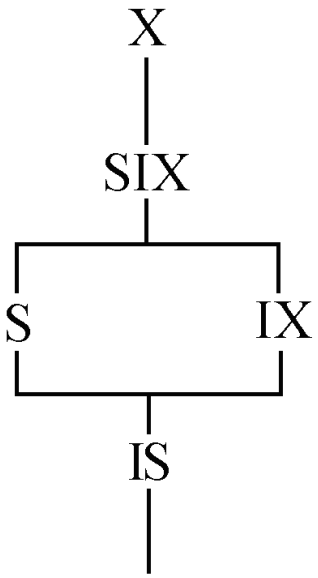
T_1 \ T_2	S	X	IS	IX	SIX	–
S	Y	N	Y	N	N	Y
X	N	N	N	N	N	Y
IS	Y	N	Y	Y	Y	Y
IX	N	N	Y	Y	N	Y
SIX	N	N	Y	N	N	Y
–	Y	Y	Y	Y	Y	Y

Y=Yes，表示相容的请求　　N=No，表示不相容的请求

(a) 数据锁的相容矩阵

引入意向锁后的锁强度

- 锁的强度
  - 锁的强度是指它对其他锁的排斥程度
  - 一个事务在申请封锁时以强锁代替弱锁是安全的，反之则不然



(b) 锁的强度的偏序关系

具有意向锁的多粒度封锁

- 具有意向锁的多粒度封锁方法
  - 申请封锁时应该按自上而下的次序进行
  - 释放封锁时应该按自下而上的次序进行

例如：事务T1要对关系R1加S锁

- 要首先对数据库加IS锁
- 检查数据库和R1是否已加了不相容的锁(X或IX)
- 不再需要搜索和检查R1中的元组是否加了不相容的锁(X锁) （为什么？）

具有意向锁的多粒度封锁

- 具有意向锁的多粒度封锁方法
  - 提高了系统的并发度
  - 减少了加锁和解锁的开销
  - 在实际的数据库管理系统产品中得到广泛应用

12.7 小结

- 数据共享与数据一致性是一对矛盾。
- 数据库的价值在很大程度上取决于它所能提供的数据共享度。
- 数据共享在很大程度上取决于系统允许对数据并发操作的程度。
- 数据并发程度又取决于数据库中的并发控制机制。
- 数据的一致性也取决于并发控制的程度。施加的并发控制愈多，数据的一致性往往愈好。

小结（续）

- 数据库的并发控制以事务为单位
- 数据库的并发控制通常使用封锁机制
  - 两类最常用的封锁（基本封锁）

小结（续）

- 并发控制机制调度并发事务操作是否正确的判别准则是可串行性
  - 并发操作的正确性通常由两段锁协议来保证
  - 两段锁协议是可串行化调度的充分条件，但不是必要条件

小结（续）

- 对数据对象施加封锁，会带来问题
- 活锁：先来先服务
- 死锁：
  - 预防方法
    - 一次封锁法
    - 顺序封锁法
  - 死锁的诊断与解除
    - 超时法
    - 等待图法

本课程结束了。。。

探索

我有一个新的构想。。。