1. **在数据库中为什么要并发控制?并发控制技术能保证事务的哪些特性?**

数据库是共享资源,通常有多个事务同时在运行。当多个事务并发地存取数据库时就会产生同时读取或修改同一数据的情况。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,破坏数据库的一致性。所以数据库管理系统必须提供并发控制机制。

并发控制可以保证事务的一致性和隔离性。

**2.并发操作可能会产生哪几类数据不一致?用什么方法能避免各种不一致的情况?**

并发操作带来的数据不一致性包括三类:

(1)丢失修改

两个事务T1和T2读入同一数据并修改,T2提交的结果破坏了(覆盖了)T1提交的结果,导致T1的修改被丢失。

(2)不可重复读

不可重复读是指事务T1读取某一数据后,事务T2对其执行更新操作,使T1无法再现前一次读取结果。不可重复读包括三种情况:

①事务T1读取某一数据后,事务T2对其做了修改,当事务T1再次读该数据时，得到与前一次不同的值。

②事务T1按一定条件从数据库中读取了某些数据记录后,事务T2删除了其中部分记录，当T1再次按相同条件读取数据时,发现某些记录消失了。

③事务T1按一定条件从数据库中读取某些数据记录后,事务T2插入了一些记录,当T1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。

(3)读“脏”数据

读“脏"数据是指事务T1修改某一数据,并将其写回磁盘,事务T2读取同一数据后,T1由于某种原因被撤销,这时T1已修改过的数据恢复原值，T2读到的数据就与数据库中的数据不一致,则T1读到的数据就为“脏”数据,即不正确的数据。

避免不一致性的方法就是并发控制。常用的并发控制技术包括封锁方法、时间戳方法、乐观控制方法和多版本并发控制方法等。

**3.什么是封锁?基本的封锁类型有几种?试述它们的含义。**

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,其他的事务不能更新或读取此数据对象。

基本的封锁类型有两种:排他锁(简称X锁)和共享锁(简称S锁)。

排他锁又称为写锁。若事务T对数据对象A加上X锁,则只允许T读取和修改A,其他任何事务都不能再对A加任何类型的锁,直到T释放A上的锁。这就保证了其他事务在T释放A上的锁之前不能再读取和修改A。

共享锁又称为读锁。若事务T对数据对象A加上S锁,则事务T可以读A但不能修改A,其他事务只能再对A加S锁,而不能加X锁,直到T释放A上的S锁。这就保证了其他事务可以读A,但在T释放A 上的S锁之前不能对A做任何修改。

**4．如何用封锁机制保证数据的一致性?**

封锁机制是一种用于保证数据一致性的常见方法。在多线程或多进程环境下，封锁机制可以确保同时只有一个线程或进程可以访问共享数据，从而避免数据的不一致性和竞争条件。通过使用封锁机制，可以保证并发事务之间访问资源的顺序和时序，从而提高系统的并发度和性能，并避免数据冲突和并发问题。

**5.什么是活锁?试述活锁的产生原因和解决方法。**

活锁（Livelock）是一种类似于死锁的情况，其中线程在竞争资源时无法继续执行，但它们并没有被阻塞，而是不断地改变自己的状态，导致无法取得进展。在活锁中，线程一直处于运行状态，但无法完成其任务。

活锁产生的原因:当一系列封锁不能按照其先后顺序执行时,就可能导致一些事务无限期等待某个封锁,从而导致活锁。

避免活锁的简单方法是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事务排队，数据对象上的锁一旦释放就批准申请队列中第一个事务获得锁。

**6.什么是死锁?请给出预防死锁的若干方法。**

死锁指的是在并发执行的多个事务中，各自持有一些资源并且都等待其他事务释放它们持有的资源，导致它们都无法继续向前执行的一种状态，形成了一种相互等待对方释放资源的僵局。

预防死锁通常有两种方法:

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。

②顺序封锁法

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。

**7. 请给出检测死锁发生的一种方法,当发生死锁后如何解除死锁?**

数据库系统一般采用的方法是允许死锁发生,DBMS检测到死锁后加以解除。DBMS中诊断死锁的方法与操作系统类似,一般使用超时法或事务等待图法。

超时法是指事务的等待时间超过了规定的时限,就认为发生了死锁。

DBMS并发控制子系统检测到死锁后,就要设法解除。通常采用的方法是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锁,使其他事务得以继续运行下去。

**8.什么样的并发调度是正确的调度?**

可串行化的调度是正确的调度。

可串行化的调度的定义:多个事务的并发执行是正确的，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，称这种调度策略为可串行化的调度。

**9.设T1、T2、T3是如下的三个事务,设A的初值为0。**

**T1:A:=A+2;**

**T2:A:=A\*2;**

**T3:A:=A\*A;**

1. **若这三个事务允许并行执行,则有多少可能的正确结果,请列举出来**

A的最终结果可能有2、4、8、16。

因为串行执行次序有T1T2T3;T1T3T2；T2T1T3;T2T3T1;T3T1T2;T3T2T1;

对应的执行结果是16;8;4;2;4;2。

**②请给出一个可串行化的调度,并给出执行结果**

|  |  |  |
| --- | --- | --- |
| T1 | T2 | T3 |
| Slock A  Y=A=0.  Unlock A  Xlock A  A=Y+2  写回A( =2)  Unlock A | Slock A  等待  等待  等待  Y=A=2  Unlock A  Xlock A  A=Y\*2  写回A(=4)  Unlock A | Slock A  等待  等待  等待  Y=A=4  Unlock A  Xlock A  A=Y\* Y  写回A(=16)  Unlock A |

最后结果A为16,是可串行化的调度。

**③请给出一个非串行化的调度,并给出执行结果。**

|  |  |  |
| --- | --- | --- |
| T1 | T2 | T3 |
| Slock A .  Y=A=0.  Unlock A  Xlock A  等待  A=Y+2.  写回A( =2)  Unlock A | Slock A  Y=A=0  Unlock A  Xlock A  等待  等待  等待  A=Y\*2  写回A(=0)  Unlock A | Slock A  等待  Y=A= 2  Unlock A  Xlock A  Y=Y\*\*2  写回A( =4)  Unlock A |

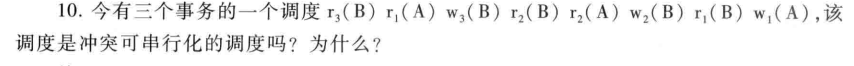
最后结果A为0,为非串行化的调度。

**④若这三个事务都遵守两段锁协议,请给出一个不产生死锁的可串行化调度**

|  |  |  |
| --- | --- | --- |
| T1 | T2 | T3 |
| Slock A  Y=A=0  Xlock A  A=Y+2.  写回A( =2)  Unlock A  Unlock A | Slock A，  等待  等待  Y=A=2  Xlock A  等待  A=Y\*2  写回A( =4)  Unlock A  Unlock A | Slock A  等待  等待  等待  Y=A=4.  Xlock A  A=Y\*\*2  写回A(=16)  Unlock A  Unlock A |

**⑤若这三个事务都遵守两段锁协议,请给出一个产生死锁的调度**

|  |  |  |
| --- | --- | --- |
| T1 | T2 | T3 |
| Slock A  Y=A=0  Xlock A  等待 | Slock A  Y=A=0  Xlock A  等待 | Slock A  Y=A= 0  Xlock A  等待 |



是冲突可串行化的调度。

Sc1=r3(B) r1(A) w3(B) r2(B) r2(A) w2(B) r1(B) w1(A),交换r1(A)和w3(B) ,得到

R3(B) w3(B) r1(A) r2(B) r2(A) w2(B) r1(B) w1(A)

再交换r1(A)和r2(B) r2(A) w2(B) ,得到

Se2=r3(B) w3(B) r2(B) r2(A) w2(B) r1(A) r1(B) w1(A)

由于Sc2是串行的,而且两次交换都是基于不冲突操作的,所以

Sc1=r3(B) r1(A) w3(B) r2(B) r2(A) w2(B) r1(B) w1(A)

是冲突可串行化的调度。

**12.举例说明,对并发事务的一个调度是可串行化的,而这些并发事务不一定遵守两段锁协议。**

假设有两个事务 T1 和 T2，它们分别进行如下操作：

- T1: 读取 A

- T2: 修改 A

- T1: 修改 B

- T2: 读取 B

如果在并发执行这些操作时，调度为 T1 读取 A --> T2 修改 A --> T1 修改 B --> T2 读取 B，则此调度是不可串行化的。

但是，可以通过修改该调度来使其变得可串行化。例如，我们可以将原调度改为 T2 修改 A --> T2 读取 B --> T1 读取 A --> T1 修改 B。这个新的调度中，先执行了 T2 中的所有操作，然后再执行了 T1 中的所有操作。虽然这个调度中的事务没有遵守两段锁协议，但它是可串行化的。

**13.考虑如下的调度,说明这些调度集合之间的包含关系。**

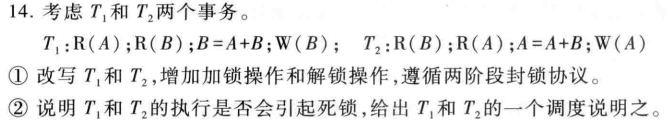
**①正确的调度。**

**②可串行化的调度。**

**③遵循两阶段封锁(2PL)的调度。**

**④串行调度**

③遵循两阶段封锁(2PL)的调度⊂①正确的调度=②可串行化的调度④串行调度⊂①正确的调度



①

|  |  |
| --- | --- |
| T1 | T2 |
| Slock A  R(A)  Xlock B  R( B)  B=A+B  W( B)  Unlock A  Unlock B | Slock B  R(B)  Xlock A .  R(A)  A=A+B  W(A)  Unlock B  Unlock A  Unlock A  Xlock B .  B=X+ l  写回B=4  Unlock B . |

②可能产生死锁

|  |  |
| --- | --- |
| T1 | T2 |
| Slock A  R(A)  Xlock B | Slock B .  R( B)  Xlock A |

1. **为什么要引进意向锁?意向锁的含义是什么?**

引进意向锁是为了提高封锁子系统的效率。

原因是:在多粒度封锁方法中,一个数据对象可能以两种方式加锁—―显式封锁和隐式封锁。因此系统在对某一数据对象加锁时,不仅要检

查该数据对象上有无(显式和隐式)封锁与之冲突,还要检查其所有上级结点和所有下级结点,看申请的封锁是否与这些结点上的(显式和隐式)封锁冲突。显然,这样的检查方法效率很低。为此引进了意向锁。

意向锁的含义是:对任一结点加锁时,必须先对它的上层结点加意向锁。引进意向锁后,系统对某一数据对象加锁时不必逐个检查与下一级结点的封锁冲突了。

1. **试述常用的意向锁:IS锁、IX锁、SIX锁,给出这些锁的相容矩阵。**

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。例如,要对某个元组加S锁,则要首先对关系和数据库加IS锁

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。例如,要对某个元组加X锁,则要首先对关系和数据库加IX锁。

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。

相容矩阵：

