Homework #7 (12.31)

Transaction Processing

1. 下面是一个数据库系统开始运行后的日志记录，该数据库系统支持检查点。

1) <T1, Begin Transaction>
2) <T1, A, 10, 40>
3) <T2, Begin Transaction>
4) <T1, B, 20, 60>
5) <T1, A, 40, 75>
6) <T2, C, 30, 50>
7) <T2, D, 40, 80>
8) <T1, Commit Transaction>
9) <T3, Begin Transaction>
10) <T3, E, 50, 90>
---------- ①
11) <T2, D, 80, 65>
12) <T2, C, 50, 75>
13) <T2, Commit Transaction>
---------- ②
14) <T3, Commit Transaction>
15) **<CHECKPOINT>**
16) <T4, Begin Transaction>
17) <T4, F, 60, 120>
18) <T4, G, 70, 140>
---------- ③
19) <T4, F, 120, 240>
20) <T4, Commit Transaction>

设日志修改记录的格式为 <Tid, Variable, Old value, New value>，请给出对于题中所示①、②、③三种故障情形下，数据库系统恢复的过程以及数据元素A, B, C, D, E, F 和G 在执行了恢复过程后的值。

2. 证明：如果一个并发调度S 中的所有事务都遵循2PL，则该调度必定是可串化调度。

3. 在锁的相容性矩阵中，<S, U>（即 T1 先持有了某对象上的 S 锁，T2 再申请同一对象上的U 锁时）是相容的，而<U, S>是不相容的。请解释一下 DBMS 为什么要设计这样不对称的锁相容性规则。

4. 采用了两阶段锁协议的事务还会出现脏读问题吗？如果会，请给出一个例子;如果不会，请解释理由。

5. 判断下面的并发调度是否冲突可串？如果是，请给出冲突等价的串行调度事务顺序;如果不是，请解释理由。

w3(D); r1(A); w2(A); r4(A); r1(C); w2(B); r3(B); r3(A); w1(D); w3(B); r4(B); r4(C); w4(C); w4(B)

解：

1.

基于Undo/Redo日志的恢复过程为：

1）正向扫描日志，将<commit>的事务放入Redo列表中，将没有结束的事务放入Undo列表;

2）反向扫描日志，对于<T,x,v,w>，若T在Undo列表中，则Write(x,v); Output(x);

3）正向扫描日志，对于<T,x,v,w>，若T在Redo列表中，则Write(x,w); Output(x);

4）对于Undo列表中的T，写入<abort,T>。

①恢复过程：

1）Redo列表{T1};Undo列表 {T2,T3}

2）Undo

T3:<T3,E,50>

T2:<T2,D,40>;<T2,C,30>

3）Redo

T1:<T1,A,40>;<T1,B,60>;<T1,A,75>

4）Write log

<Abort,T2>;<Abort,T3>

恢复后，A=75,B=60,C=30,D=40,E=50.

②恢复过程：

1）Redo列表{T1,T2};Undo列表 {T3}

2）Undo

T3:<T3,E,50>

3）Redo

T1:<T1,A,40>;<T1,B,60>;<T1,A,75>

T2:<T2,C,50>;<T2,D,80>;<T2,D,65>;<T2,C,75>

4）Write log

<Abort,T3>

恢复后，A=75,B=60,C=75,D=65,E=50.

③T1、T2、T3已经Commit，无需Undo/Redo。

T4恢复过程：

1）Undo列表 {T4}

2）Undo

T4:<T4,G,70>;<T4,F,60>;

3）Redo(Null)

4）Write log

<Abort,T4>

恢复后，A=75,B=60,C=75,D=65,E=90,F=60,G=70

2.

首先以两个并发事务Tl和T2为例，存在多个并发事务的情形可以类推。根据可串行化定义可知，事务不可串行化只可能发生在下列两种情况：

（1）事务 Tl 写数据A_p，T2读或写A_p

（2）事务 Tl 读或写数据A_q，T2写 A_q

设 Tl 和T2访问的公共对象为{A_1，A_2 …，A_n }。假设这组对象中 X ＝(A_1 , A_2 ，…，A_i} 均符合情况（1）。 Y ＝{A_{i+1}，…，A_n} 符合情况（2）。

对于情况（1），$\forall A \in X$, ①Tl 需要 Xlock A，②T2 需要 Slock A 或 Xlock A，若操作①先执行，则 Tl 获得锁，T2等待。

由于遵守两段锁协议， Tl 在成功获得X中全部对象的锁后，才会释放锁。

若不出现死锁， Tl 在对 X 中对象依次处理完毕释放锁后，T2才能获得对应对象的锁继续执行。这相当于按 Tl 、T2的顺序串行执行，根据可串行化定义， Tl 和T2的调度在对象X中是可串行化的。

对于情况（2），$\forall A \in Y$，①Tl 需要 Slock A 或 Xlock A，②T2 需要Xlock A

同理可得，Tl 和T2的调度在对象Y中是可串行化的。

综上，在设 Tl 和T2访问的公共对象集合{A_1，A_2 …，A_n } 中，若并发事务遵守两段锁协议，在不发生死锁的情况下，对这些事务的并发调度一定是可串行化的。证毕。

3.

更新锁U(R)只给予事务T读R而不是写R的权限。但是，只有U锁能在以后升级为X锁，读锁是不能升级的。当R上已经有S锁时，可以授予R上的U锁，因为U锁可以在以后升级为X锁，S锁不能升级， 这样就不会有两个X锁互相等待释放资源出现死锁的问题，并且获得较好的并发性。但是一旦R上有了U锁，就禁止在R上加其他任何种类(S、U或X)的锁。其原因是，如果不拒绝这样的锁，那么U锁可能由于R上总有其他的锁而永远没有机会升级到排他锁。

这一规则导致一个不对称的相容性矩阵，因为U锁在申请它时看起来像S锁，而当已经持有它时看起来像X锁。因此，关于S和U的列相同，关于U和X的行相同。

4.

可能会脏读，这取决于DBMS实现2PL时使用几级封锁协议。

封锁协议有**一级封锁协议**（要求修改数据时必须加X锁，直到事务结束才释放锁，读数据是不需要加锁的）、**二级封锁协议**（要求修改数据时必须加X锁，直到事务结束才释放锁，要求读取数据时必须加S锁，读取完马上释放S锁，无需等待事务结束。）**三级封锁协议**（要求修改数据时必须加X锁，直到事务结束才释放锁，要求读取数据时必须加S锁，直到事务结束才释放S锁）

如果采用**一级封锁协议，**假设有两个事务T1和T2：

T1	**T2**
	Lock(A)
	A=A+10
Read(A)	
	rollback
commit	**commit**

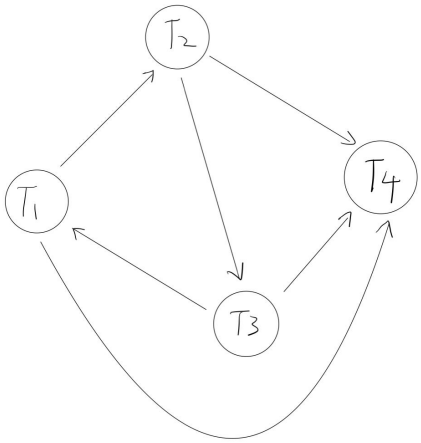
假设初始时数据库中A记录的值为50，T2事务正在对A记录做修改，修改后A=60,在T2完成并提交之前，A=60是处于待定状态的（可能commit也可能rollback），这时，T1事务来读取这条没有提交的A记录，得到的是更新之后的A=60，但随后T2进行rollback并commit，此时数据库中的A=50，但T1事务读到的A=60，产生了未提交的数据依赖关系，产生了脏读。

二级封锁协议可以避免脏读：读数据的时候必须加S锁，但因为T2事务在修改的时候已经加了 X锁，在X锁的基础上是不能加S锁的，所以T1事务无法获取S锁，就会导致B事务无法读取A事务中正在操作的数据，从而避免了脏读的发生。

所以采用**二级封锁协议**和**三级封锁协议**时不会产生脏读问题，采用**一级封锁协议**可能会产生脏读问题。

5.

不是冲突可串的，优先图为：



r1(A)、w2(A)和w2(A)、r3(A)和w3(D)和 w1(D)导致T1,T2,T3之间有回路，所以这个并发调度不是冲突可串的。