《数据库概论》第三次课后作业

题目：

1. The following is a sequence of undo/redo-log records written by two transactions T and U:

```
<START U>
<U, A, 10, 11>
<START T>
<T, B, 20, 21>
<U, C, 30, 31>
<T, D, 40, 41>
<COMMIT T>
<U, E, 50, 51>
<COMMIT U>
```

Describe the action of the recovery manager, including changes to both disk and the log, if there is a crash and the last log record to appear on disk is:

```
(a)  <START T>
(b)  <COMMIT T>
(c)  <U, E, 50, 51>
(d)  <COMMIT U>
```

(a)

逆向扫描，撤销所有未提交和未结束的事务（U&T），直至日志的起点。

此时磁盘中修改为：

```
A = 10
```

正向扫描，重做所有已提交事务（），直至日志的终点。

此时磁盘中修改为：

```
无
```

故障恢复结束后，在日志文件的尾部插入 U 和 T 的结束标志：

```
<START U>
<U, A, 10, 11>
<START T>
<Abort U>
<Abort T>
```

(b)

逆向扫描，撤销所有未提交和未结束的事务（U），直至日志的起点。

此时磁盘中修改为：

```
A = 10
C = 30
```

正向扫描，重做所有已提交事务（T），直至日志的终点。

此时磁盘中修改为：

```
B = 21
D = 41
```

故障恢复结束后，在日志文件的尾部插入 U 的结束标志：

```
<START U>
<U, A, 10, 11>
<START T>
<T, B, 20, 21>
<U, C, 30, 31>
<T, D, 40, 41>
<COMMIT T>
<Abort U>
```

(c)

逆向扫描，撤销所有未提交和未结束的事务（U），直至日志的起点。

此时磁盘中修改为：

```
A = 10
C = 30
E = 50
```

正向扫描，重做所有已提交事务（T），直至日志的终点。

此时磁盘中修改为：

```
B = 21
D = 41
```

故障恢复结束后，在日志文件的尾部插入 U 的结束标志：

```
<START U>
<U, A, 10, 11>
<START T>
<T, B, 20, 21>
<U, C, 30, 31>
<T, D, 40, 41>
<COMMIT T>
<U, E, 50, 51>
<Abort U>
```

（d）

逆向扫描，撤销所有未提交和未结束的事务（），直至日志的起点。

此时磁盘中修改为：

```
无
```

正向扫描，重做所有已提交事务（U&T），直至日志的终点。

此时磁盘中修改为：

```
A = 11
B = 21
C = 31
D = 41
E = 51
```

2. 请考虑下面两个事务：

```
T1: read(A);
    read(B);
    if A = 0 then B := B + 1;
    write B;
```

```
T2: read(B);
    read(A);
    if B = 0 then A := A + 1;
    write A;
```

请给事务 T1 与 T2 增加封锁和解锁指令，使它们遵从两阶段封锁协议。这两个事务的执行会导致死锁吗?

增加封锁和解锁指令后：

```
T1: sl_1(A)
    read(A);
    sl_1(B)
    read(B);
    if A = 0 then B := B + 1;
    xl_1(B);
    write B;
    u_1(A);
    u_1(B);
```

```
T2: sl_2(B)
    read(B);
    sl_2(A)
    read(A);
    if B = 0 then A := A + 1;
    xl_2(A);
    write A;
    u_2(A);
    u_2(B);
```

这两个事务的执行，

有可能会死锁，也可能不会死锁。

（1）考虑如下调度，将产生死锁：

$$r_1(A);r_1(B);r_2(B);w_1(B);r_2(A);w_2(A);$$

具体如下：

	事务T1	事务T2	A的封锁状态	B的封锁状态
1	sl_1(A);		S(T1)	S(T1)
2	r_1(A);		S(T1)	S(T1)
3	sl_1(B);		S(T1)	S(T1)
4	r_1(B);		S(T1)	S(T1)
5		sl_2(B);	S(T1)	S(T1,T2)
6		r_2(B);	S(T1)	S(T1,T2)
7	xl_1(B);		S(T1)	S(T1,T2)
8	Wait...	sl_2(A)	S(T1,T2)	S(T1,T2)
9	Wait...	r_2(A);	S(T1,T2)	S(T1,T2)
10	Wait...	xl_2(A);	S(T1,T2)	S(T1,T2)
11	Wait...	Wait...	S(T1,T2)	S(T1,T2)
12	Wait...	Wait...	S(T1,T2)	S(T1,T2)

当事务 T1 申请 B 的排他锁时，由于 B 上有事务 T2 加的共享锁，因而陷入等待

当事务 T2 申请 A 的排他锁时，由于 A 上有事务 T1 加的共享锁，因而陷入等待

因此事务 T1 和 T2 陷入循环等待，产生死锁现象

（2）考虑如下调度，将不会产生死锁：

$$r_1(A);r_1(B);w_1(B);r_2(B);r_2(A);w_2(A);$$

具体如下：

	事务T1	事务T2	A的封锁状态	B的封锁状态
1	sl_1(A);		S(T1)	S(T1)
2	r_1(A);		S(T1)	S(T1)
3	sl_1(B);		S(T1)	S(T1)
4	r_1(B);		S(T1)	S(T1)
5	xl_1(B);		S(T1)	X(T1)
6	w_1(B);		S(T1)	X(T1)
7	u_1(A);			X(T1)
8	u_1(B);			
9		sl_2(B);		S(T2)
10		r_2(B);		S(T2)
11		sl_2(A);	S(T2)	S(T2)
12		r_2(A);	S(T2)	S(T2)
13		xl_2(A);	X(T2)	S(T2)
14		w_2(A);	X(T2)	S(T2)
15		u_2(A);		S(T2)
16		u_2(B);		

当事务 T1 申请 B 的排他锁时，B 上没有任何封锁，因而申请成功

当事务 T2 申请 A 的排他锁时，A 上没有任何封锁，因而申请成功