COMPSCI 751 S1 C – Assignment 3

Name: Yixuan Li

UPI: yil845

1. **ACID Properties: Jo's organization uses an established transactional system BalnC, which manages accounts. Each account has an integer value at any time. All transactions in BalnC are transfers between accounts, so after each transaction the sum of all the accounts in BalnC is supposed to be zero. One week ago Jo's team installed a new version of the ACID transaction management used by BalnC. Today Jo observes that the sum of all accounts is not zero any more. Jo suspects the recently installed new version violates one or more ACID properties. We do not consider consistency and durability here. For each of the two remaining ACID properties, say in about 3 sentences whether and how a violation of that ACID property could have caused the observed problem.**

   Excluding Consistency and Durability, this imbalance can be caused by violation of Atomicity and Isolation properties. In case of atomicity, the imbalance of accounts can occur if the database did perform rollback correctly in case of abortion of transactions. If changing the Isolation level of transactions, lost update could happen, therefore the overall balance is incorrect.

2. **Deadlocks. For each of the following transaction sets, say whether they can run into a deadlock in the common scheduler. If yes, give a possible partial schedule for them that leads to a deadlock. If no, give a reason why they cannot run into a deadlock.**

   (a) This schema will run into a deadlock in common scheduler, because x is written by TA1 and TA2 at non- write-disjoint conditions.

   [Case1] TA1: r1[x], r1[y], w1[x]----------(deadlock)

   TA2: r2[y], r2[x], w2[x]----------(deadlock)

   [Case2] TA1: r1[x], r1[y], w1[x]---------- (deadlock)

   TA2: r2[y], r2[x], w2[x]----------(deadlock)

   (b) This schema will not run into a deadlock because R[] is 'select…for update', that means TA1 gets an exclusive first-to-write-lock before any other reading on x and y. r2[y] in TA2 needs to wait for TA1 to finish firstly.

   [Case1] TA1: R1[x], R1[y], w1[x], c1

   TA2: r2[y]----------→, r2[x],w2[x], c2

   [Case2] TA1: R1[x],R1[y], w1[x], c1

   TA2: r2[y]----------→, r2[x],w2[x], c2

(c) This schema will not run into a deadlock. It is similar to (b), the only difference is y does not have R[]exclusive first-to-write-lock before any other reading access, but it is also not necessary because of non-writing-action on y.

[Case1] TA1: R1[x], r1[y], w1[x], c1

TA2: r2[y], r2[x]--------→,w2[x], c2

[Case2] TA1: R1[x], r1[y], w1[x], c1

TA2: r2[y], r2[x]----------→,w2[x], c2

3. **Transactions and Phenomena. Say for each of the following schedules: does the schedule contain phenomena or any other violation of the locking rules of the common scheduler? If not, give an explanation why not. If yes, say on which data object the phenomenon occurs; describe the phenomenon and using this example, explain why this phenomenon or violation of locking rules can be a problem. State the highest isolation level that the schedule can be performed on.**

(a) s1 : r1[z], r3[y], r2[y], c3, w2[z], w2[y], r1[z], c2, r1[x], c1.

Dirty read:

TA1:r1[z], r1[z], r1[x],c1

TA2: r2[y], w2[z],w2[y], c2

TA3: r3[y],c3

This schedule contains phenomena of dirty reads. The second r1[z] in TA1 reads uncommitted w2[z] in TA2. Dirty read is at relaxation level of Read-uncommitted.

w2[z] might be a part of a transaction that tries to transfer some money from account x to another account y, but r1[z] is reading the balance of z before TA2 can be committed/aborted, thus the balance read by TA1 could be inconsistent.

This level of relaxation 'read-uncommitted' violates the write-disjoint locking rules of the common scheduler. The highest isolation level is serializable common scheduler, in which TA2 will be executed after TA1 committed as depicted underlying:

TA1:r1[z], r1[z],r1[x],c1

TA2: r2[y], w2[z]----------→,w2[y],c2

TA3: r3[y],c3

(b) s2 : r1[x], r3[y], r2[y], c3, r1[y], w2[z], w2[y], c2, r1[z], r1[y], w1[x], c1.

Fuzzy read:

TA1:r1[x], r1[y], r1[z],r1[y], w1[x],c1

TA2: r2[y], w2[z],w2[y],c2

TA3: r3[y],c3

This transaction can be done in phenomena 'Fuzzy read', it is at the relaxation level of Read-committed. It happens when TA2 did not observe the read-lock of y in TA1 but TA1 sees the write lock of y in TA2. In fuzzy read both committed and uncommitted values can be read. The second r1[y] is not equal to the first r1[y] in TA1, it can read the updated w2[y]. This violates the locking rules of the common scheduler. The highest isolation level is serializable, in which w2[y] in TA2 will be delayed till TA1 is committed as shown underlying:

TA1:r1[x], r1[y], r1[z],r1[y], w1[x],c1

TA2: r2[y], w2[z],w2[y]----------→,c2

TA3: r3[y],c3

(c) s3 : r1[x], r3[y], r2[y], c3, r1[y], w2[z], w2[y], c2, r1[z], w1[x], w1[y], c1

Lost update:

TA1:r1[x], r1[y], r1[z],w1[x],w1[y],c1

TA2: r2[y], w2[z],w2[y],c2

TA3: r3[y],c3

In the isolation level of "read-committed", phenomena 'Lost update' will happen, r1[y] in TA1 reads and writes on the old value of y, but w2[y] in TA2 already modified object y and committed. It would cause a serious problem of lost update. That y object after TA1 did not take into account of updated y object in TA2.This violates the write-disjoint locking rules of the common scheduler. To prevent this happening, we can increase the isolation level to the highest – serializable common scheduler, but it will be deadlock as following:

TA1:r1[x], r1[y], r1[z],w1[x],w1[y]-------(deadlock)

TA2: r2[y], w2[z],w2[y]--------------------------

TA3: r3[y],c3

4. **Transactions, lock-based scheduling. Give one example schedule that violates the common scheduler locking rules but contains no phenomenon. Say at which operation which locking rules are violated and explain why the schedule still contains no phenomenon.**

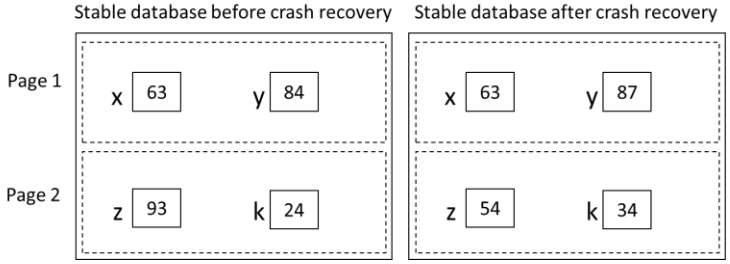
This simple schedule violates the common scheduler locking rules but contains no phenomenon,
s: r1[x],r2[y],w1[y],w2[y],c1,c2
TA1:r1[x], w1[y]--------------------(deadlock)
TA2: r2[y], w2[y]----------------

Both transactions will run into a deadlock because of violation of write-disjoint requirement of common scheduler, but it does not contain any phenomenon, like dirty read, fuzzy read or lost update, it does not have committed / uncommitted reads in between, also no lost update.

5. Crash recovery for steal, no-force policy: The following list gives pages, objects on these pages and their values in the stable database at a certain point in time:

a) You are supposed to perform crash recovery. What operations do you have to perform on which transactions? Give the content of the stable database after the crash recovery.
[nr: 320, ta: 12, obj: z, b: 23, a: 93] Redo1
[nr: 321, ta: 13, obj: k, b: 34, a: 24] undo1
[nr: 322, ta: 12, obj: z, b: 93, a: 54] Redo2
[nr: 323, ta: 12, obj: y, b: 84, a: 87] Redo3
[nr: 324, ta: 12, commit] Redo4
[nr: 325, ta: 14, obj: x, b: 63, a: 64]
[nr: 326, redoLsn: 320, undoLsn: 321]

We need to do first redo committed transaction and then undo uncommitted transaction on stable log as shown above.



b) Was a database buffer page with an uncommitted write written to the stable database? If yes, say which page and identify the time interval when it was written to the stable database. Give the interval as two log sequence numbers before and after, and say how you came to that conclusion. If no, give reasons for your answer.

- Yes, an uncommitted write is written to the stable database, that is object k in page2, in nr:321. The checkpoint [nr: 326, redoLsn: 320, undoLsn: 321] says undoLsn 321. In log [nr: 321, ta: 13, obj: k, b: 34, a: 24] it shows object k before image is 34 and after image is 24, therefore rollback needs to be done to get before image number of 34 for this uncommitted transaction.