351/751 Database Systems, 2022, Semester 1

Lab 07

Name: Yixuan Li

API:yil845

1. **Transactions, 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 objects phenomena occur; describe the phenomena. State the highest isolation level that the schedule can be performed on.**

   (a) s1 : r1[x], r2[y], r3[y], w1[x], w3[y], c3, r2[x], c2, c1

   Read-uncommitted:

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

   TA2: r2[y], r2[x],c2

   TA3: r3[y], w3[y],c3

   It contains phenomena 'dirty read'. r2[x] reads uncommitted value of w1[x]. It violates of the write-disjoint locking rules of the common scheduler. The highest isolation level is serializable common scheduler.

   (b) s2 : r1[x], r2[y], r3[y], w1[x], w3[y], c3, r2[y], c2, c1

   Read-uncommitted:

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

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

   TA3: r3[y], w3[y],c3

   It contains phenomena 'fuzzy read'. The first r2[y] reads the old value of y, while the second r2[y] reads the updated and committed value of y after TA3. It violates of the write-disjoint locking rules of the common scheduler, thus the second r2[y] is not equal to the first r2[y] in TA2. The highest isolation level is serializable common scheduler.

   (c) s3 : r1[x], r2[y], r3[y], w1[x], w3[y], c3, w2[y], c2, c1

   Read-uncommitted:

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

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

   TA3: r3[y], w3[y],c3

It contains phenomena 'Lost update'. The second r2[y] read the y before TA3 committed, but w2[y] executed after w3[y] in TA2 committed but still based on old value of y, thus causing serious problem of lost update. It violates of the write-disjoint locking rules of the common scheduler. The highest isolation level is serializable common scheduler, in which TA2 and TA3 will run into deadlocks. It can be solved by changing schedule to R[y].

(d) s4 : r1[x], r2[y], r1[y], r3[y], r2[x], r3[x], w1[x], c3, w2[y], c2, c1

Read-uncommitted:

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

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

TA3: r3[y], r3[x], c3

It does not contain any phenomena. But it violates the write-disjoint locking rules of the common scheduler. In common scheduler, TA1, TA2 will run into deadlocks, only TA3 can be committed.

2. **Transactions, Deadlocks. Consider the following set of transactions.**

   TA1: r1[k], r1[x], w1[x], r1[u], c1

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

   TA3: r3[x], r3[z], c3

   (a) Can this set of transactions run into a deadlock if we use the simple scheduler? If yes, give a scheduling diagram showing the deadlock, if no say why not.

   Simple scheduler: will run into a deadlock, due to violation of data-disjoint rules on object x and z.

   TA1: r1[k], r1[x]----------(deadlock)
   TA2: r2[z], r2[x]----------
   TA3: r3[x], r3[z]----------

   (b) Can this set of transactions run into a deadlock if we use the common scheduler? If yes, give a scheduling diagram showing the deadlock, if no say why not.

   Common scheduler: will run into a deadlock due to violation of write-disjoint rules. X is read and written by both TA1 and TA2. Only TA3 can be executed because of non-write actions.

   TA1: r1[k], r1[x],w1[x]----------(deadlock)
   TA2: r2[z], r2[x],w2[z], w2[x]------
   TA3: r3[x], r3[z], c3

3. **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:**

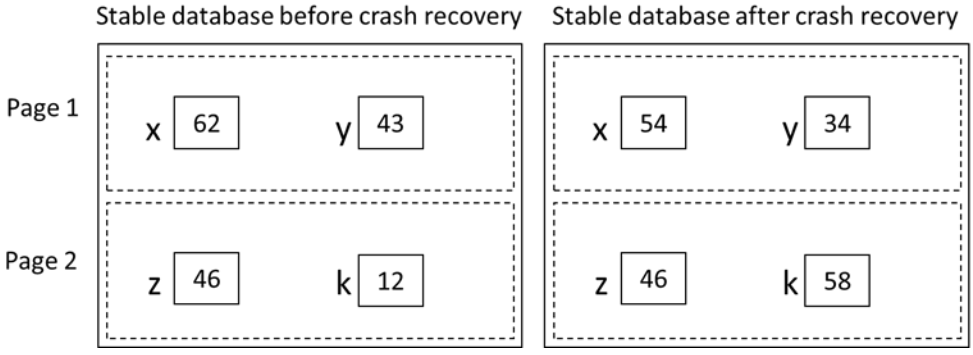
   Page 1: x = 62
   y = 43
   Page 2: z = 46

k = 12

The following is the list of the most recent stable log records at the same point in time. The database uses the steal, no-force policy.

[nr: 321, ta: 62, obj: x, b: 31, a: 62]
[nr: 324, ta: 62, obj: k, b: 12, a: 58]
[nr: 322, ta: 63, obj: y, b: 34, a: 43]
[nr: 323, ta: 62, obj: x, b: 62, a: 54]
[nr: 325, ta: 62, commit]
[nr: 327, ta: 64, obj: z, b: 46, a: 89]
[nr: 328, ta: 64, obj: k, b: 58, a: 91]

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.
We need to do first redo committed transaction and then undo uncommitted transaction on stable log as shown above.
[nr: 321, ta: 62, obj: x, b: 31, a: 62] redo1
[nr: 324, ta: 62, obj: k, b: 12, a: 58] redo2
[nr: 322, ta: 63, obj: y, b: 34, a: 43] undo1
[nr: 323, ta: 62, obj: x, b: 62, a: 54] redo3
[nr: 325, ta: 62, commit] redo4
[nr: 326, redoLsn: 321, undoLsn: 322]
[nr: 327, ta: 64, obj: z, b: 46, a: 89]



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, object y in the stable database was written by an uncommitted value '43'. In page 1, Lsn 322 it is written to the stable database. The before image of y is 34, after image is 43. Here is the log record [nr: 322, ta: 63, obj: y, b: 34, a: 43], it needs to be undone.