Nicholas Jake Jeffreys
932-221-702
CS 440 Database Management Systems
25 February 2018

Assignment 6

Problem 1

(a)

At time 1:

pageID	recLSN
P7	00

transID	lastLSN
T1	00

At time 2

pageID	recLSN
P7	00

transID	lastLSN
T1	00
T0	10

At time 3

pageID	recLSN
P7	00

transID	lastLSN
T1	20
T0	10

** checkpoint **

At time 6:

pageID	recLSN
P9	50

transID	lastLSN
T1	50

At time 7:

pageID	recLSN
P9	50
P6	60

transID	lastLSN
T1	50
T2	60

At time 8:

pageID	recLSN
P9	50
P6	60
P5	70

transID	lastLSN
T1	70
T2	60

At time 9:

pageID	recLSN
P9	50
P5	70
P7	80

transID	lastLSN
T1	80
T2	60

(b)

We first reconstruct state at checkpoint via end_checkpoint record at LSN 40. We then scan log forward from checkpoint.

At LSN 50:

pageID	recLSN
P9	50

transID	lastLSN
T1	50

At LSN 60:

pageID	recLSN
P9	50
P6	60

transID	lastLSN
T1	50
T2	60

At LSN 70:

pageID	recLSN
P9	50
P6	60
P5	70

transID	lastLSN
T1	70
T2	60

At LSN 80:

pageID	recLSN
P9	50
P5	70
P7	80

transID	lastLSN
T1	80
T2	60

(c)

Redo phase states from the smallest recLSN which would be 50.

All of these steps after the checkpoint need to be repeated excluding T2 updating P6 because that was committed before the crash:
At LSN 50 -> update: T1 updates P9
At LSN 70 -> update: T1 updates P5
At LSN 80 -> update: T1 updates P7

(d)

The Undo phase starts at the end of the log and will undo all loser transaction T1 in reverse order:

At LSN 80 -> update: T1 updates P7
At LSN 70 -> update: T1 updates P5
At LSN 50 -> update: T1 updates P9

Problem 2

(a)

From assignment 3 I used the sort-merge algorithm to merge these two relations. The only difference here is that the both relations are initially sorted now so we can ignore that aspect of the algorithm. For the join aspect of the problem we will want to pull one tuple into the buffer from each run. At this point we will compare the managerid brought in by Dept to the eid brought in by Emp. If managerid is more than eid then iterate to next eid. If managerid is less than eid then iterate to next managerid. If they are equal, then combine tuples and write to file. Continue to pull in new tuples from Dept and Emp runs until all have been read and compared. In order to handle a crash during the join we will need to keep track of each action and comparison within the merge. Once we have a log file we can use this file to detect if there was a crash in the previous run. If the join is successful without a crash, then we will write "SUCCESS" at the end of the log file. We also have to make sure we write what comparison is going to occur before it occurs as to not lose track of any joins.

(b)
- Compilation Instruction

    g++ -std=c++11 main.cpp -o main

- Const bool controls whether or not crash will occur (line 13) - > true for crash, false for no crash

    const bool CRASH_FLAG

- When const is set to true there is the possibility of sequential crashes. Run program again to continue again.