Exercise 4 Report

Gruppe 16

Anastasiia Rubanovych Sebastian Funck

June 9, 2020

a)

- **Describe in your own words how the program works and how the persistence manager accom- plishes logging and recovery.**

  The persistance manager provides the interface to create transactions that modify an underlying data struture consisting of pages and user data. It is implemented as a Singleton so it has sole control over write and read operations on the respective pages and their content. The main functionality is divided in 3 parts: Handling Transactions, Logging and Recovery.

- **Transactions:** The `PersitenceManager` buffers write operation in a `ConcurrentHashMap<Integer, PageWrite>` with page-id as key. Comitted transactions aren't persisted right after commit, but rather at a later point when the `PersitenceManager` decides the buffer has become too large. This can produce the effect of dirty writes, because if Transaction A writes to Page 10, then Transaction B writes to Page 10 and then A commits, the changes aren't persisted, because the change Transaction A to page 10 has been overridden. Only when B is comitted it is eventually written.

- **Logging:** Before any operation, the `PersitenceManager` makes sure the operation is logged and written to the logfile, so that in case a failure occurs, the recovery can recover the system. The operations that are logged are `write` and `commit`. begin_transaction is not persisted, because we assume that only valid transaction ids make it into the log.

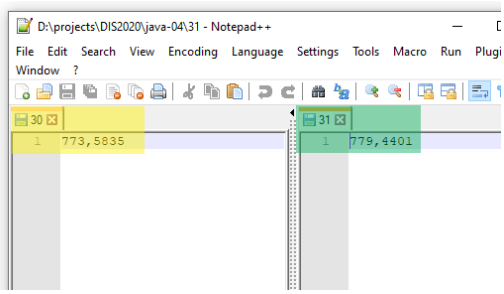
- **Recovery:** Because the `PersitenceManager` is *nosteal* and *noforce* we are doing a redo-recovery. The recovery is done on program start, where the `PersitenceManager` reads for every page id through the logs and determines winner and loser transactions. Loser transactions are those that don't have an EOT logged. When the winner transactions have been determined the logs of the winner transactions modifying the respective page are replayed starting from the first LSN that is greater than the current Page LSN.

c.1)

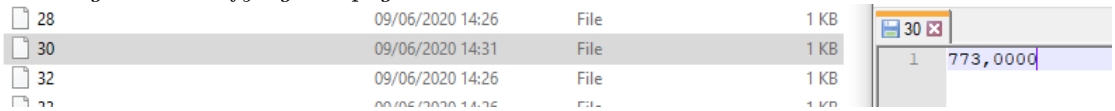
- **After program completion delete a written page and change the value of a written page. Start the recovery and check if it was successful?**

After first execution:

```
LSN : 769, TransactionId : 48, IsEoT : true
LSN : 770, TransactionId : 50, IsEoT : false, PageId : 34, Data : 5857
LSN : 771, TransactionId : 49, IsEoT : false, PageId : 13, Data : 4802
LSN : 772, TransactionId : 49, IsEoT : false, PageId : 10, Data : 8841
LSN : 773, TransactionId : 50, IsEoT : false, PageId : 30, Data : 5835
LSN : 774, TransactionId : 49, IsEoT : false, PageId : 13, Data : 5681
LSN : 775, TransactionId : 50, IsEoT : false, PageId : 32, Data : 4560
LSN : 776, TransactionId : 49, IsEoT : false, PageId : 16, Data : 28
LSN : 777, TransactionId : 50, IsEoT : false, PageId : 38, Data : 9309
LSN : 778, TransactionId : 49, IsEoT : false, PageId : 10, Data : 4193
LSN : 779, TransactionId : 50, IsEoT : false, PageId : 31, Data : 4401
LSN : 780, TransactionId : 49, IsEoT : false, PageId : 13, Data : 1777
LSN : 781, TransactionId : 50, IsEoT : true
LSN : 782, TransactionId : 49, IsEoT : false, PageId : 17, Data : 7397
LSN : 783, TransactionId : 49, IsEoT : true
```



Deleting and modifying the pages:



After recovery:

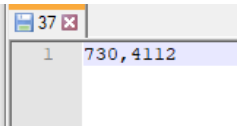
The recovery failed for page `30` because the redo-recovery only checks for writes that satisfy `LSN(LS) > LSN(Page)`. We were wondering why `>` is not a `>=` because this small change would enable us to also recover from that error as well. For page `31` the recovery worked.

c.2)

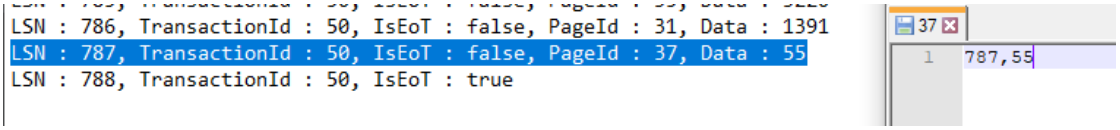
After program completion find an unwritten change and start the recovery. Was the recovery successful?

After first execution:

```
LSN : 786, TransactionId : 50, IsEoT : false, PageId : 31, Data : 1391
LSN : 787, TransactionId : 50, IsEoT : false, PageId : 37, Data : 55
LSN : 788, TransactionId : 50, IsEoT : true
```



After recovery:

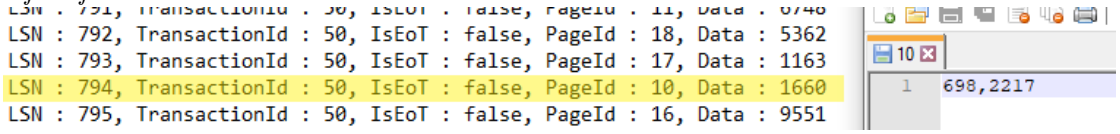


The recovery was successful because the value of page 37 corresponds now to the latest comitted log (55).

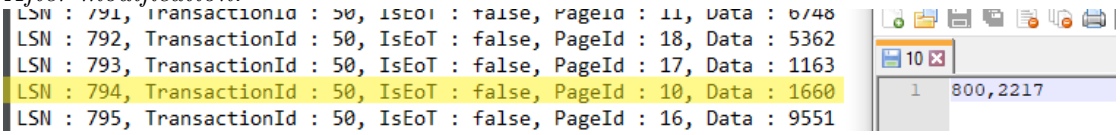
c.3)

After program completion find an unwritten change. Increase the LSN to a value higher than the unwritten change and start the recovery. Was the recovery successful?

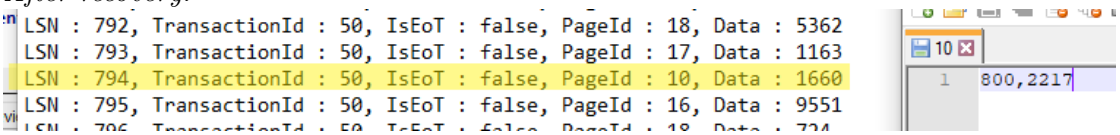
After first execution:



After modification:



After recovery:



The recovery failed, because the redo-recovery assumed the written change is newer than the latest logged write. In theory the recovery could check if this LSN even exists and rollback from there, but this would be an undo-recovery.