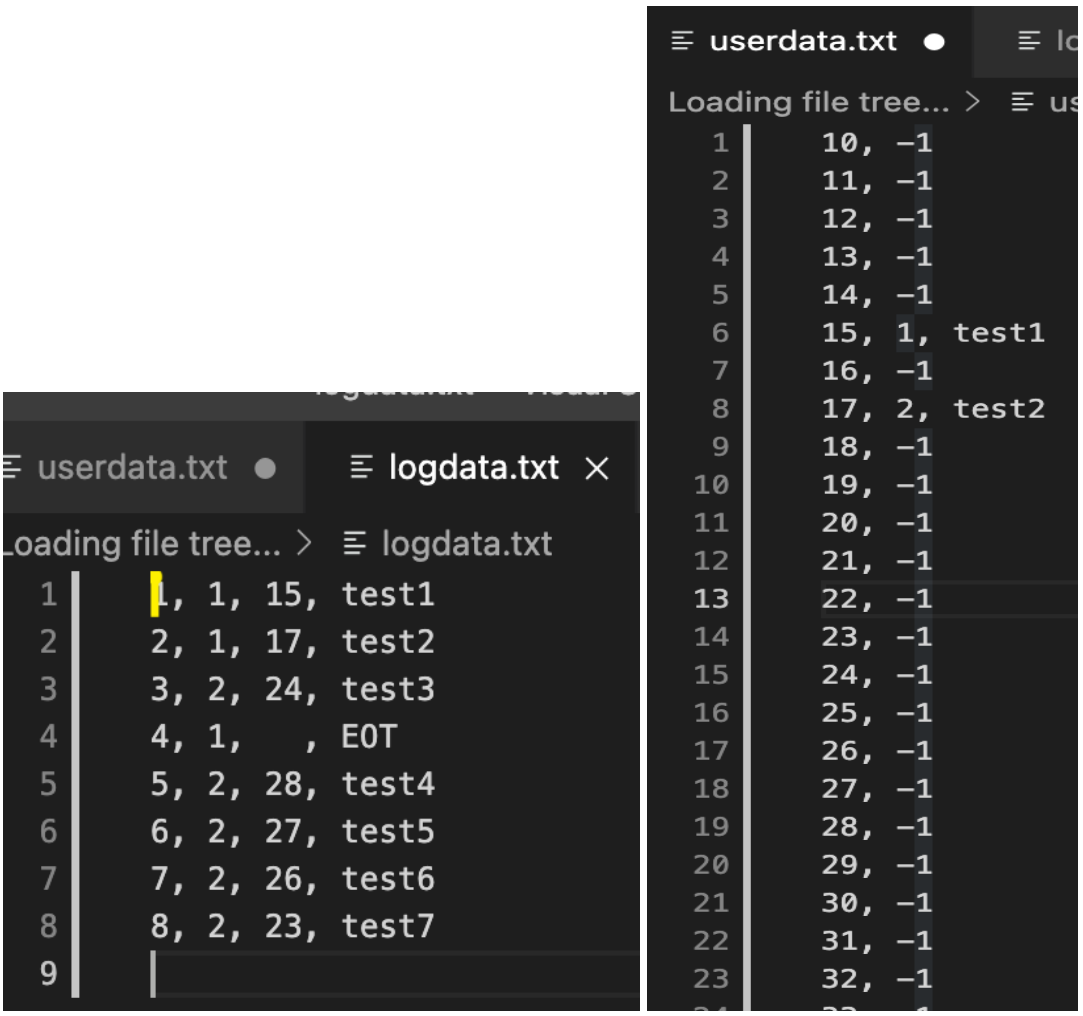
DIS Sheet 4

a) Describe how the program works and how the persistence manager accomplishes logging and recovery
   - We have a persistence manager, who inhabits the three important functions “write”, “commit” and “safe_persistent”. Whenever a Client begins a transaction he gets his own transactionnumber (taid). Then when he gives a write in order this write gets his own lognumber (lsn) and is being safed with lsn, taid, pageid and data in the hashtable, which is our buffer, and the logdata. If the client wants to commit, this commit is safed with “EOT” in the logfile, but not in the buffer. Now it doesn't matter which client has given any writes in order, but as long as the amount of writes safed in the buffer exceeds 5, when a new write is being ordered, the buffer automatically checks wether the write orders in the buffer have already been committed. This is done by looking in the logfile and checking if there are any commits (can be seen by “EOT”) for a specific taid. If there are commits for write commands in the buffer then we start the function safe_persistent. This function safes the write commands in the userdata. After a write command is safed persistent in the userdata we delete this writecommand in the buffer. If we don't find any commits, but the number of writes in the buffer is bigger than 5, is tdoesnt matter, since we will check again when a new write command is being ordered. At the end of the system all write operations with a commit are being safed, since we let the recoverymode alo run at the end of the system.
   - the recovery_mode starts whenever we start the program. It checks the logfile for write orders which havent been safed persistent and also haven’t been overwritten by any other write orders in the userdata 8this is checked by comparing lsn in userdata and in logfile). For these write commands the system will then place a new write order with the same taid, pageid and data (only lsn will be new). By doing this the write commands will then be safed when a commit for the taid is/was placed. When we excecute a write function from the recoverymode we don't have to wait till the buffer exceeds a number of entries, since we let the writefunction know we come from the recoverymode by giving him a Boolean.

b) Tested logging and buffering (with screenshots)

i) uncommited transactions in the log file

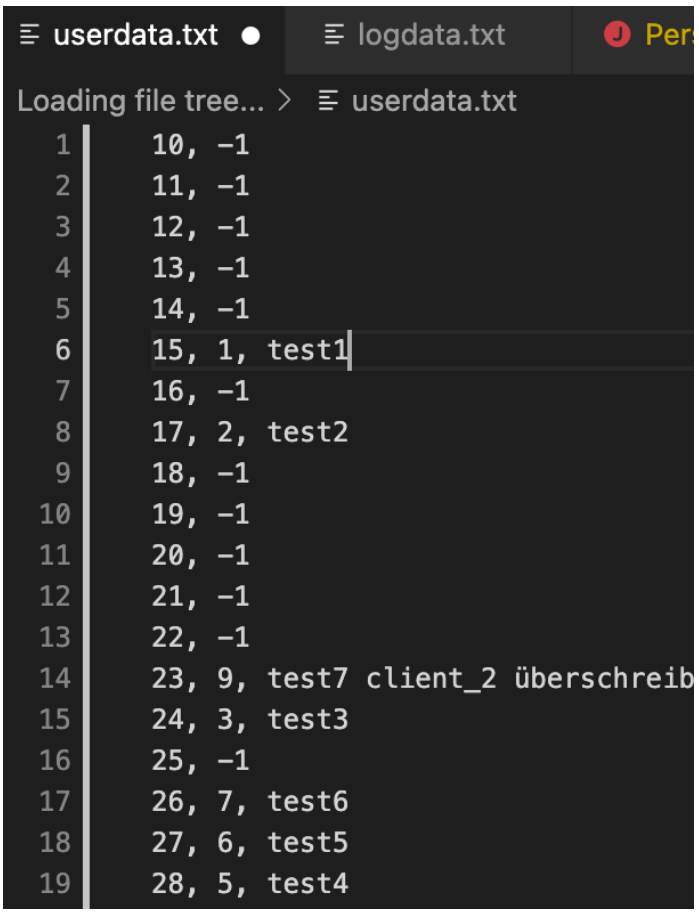
```
RecoveryTool.recoverymode();
Integer client_1_taid = client_1.beginTransaction();
Integer client_2_taid = client_2.beginTransaction();
Integer client_3_taid = client_3.beginTransaction();
Integer client_4_taid = client_4.beginTransaction();
Integer client_5_taid = client_5.beginTransaction();

client_1.write(client_1_taid, 15, "test1");
client_1.write(client_1_taid, 17, "test2");
client_2.write(client_2_taid, 24, "test3");
client_1.commit(client_1_taid);
client_2.write(client_2_taid, 28, "test4");
client_2.write(client_2_taid, 27, "test5");
client_2.write(client_2_taid, 26, "test6");
client_2.write(client_2_taid, 23, "test7");
// client_2.write(client_2_taid, 23, "test7 client_2 üb
```



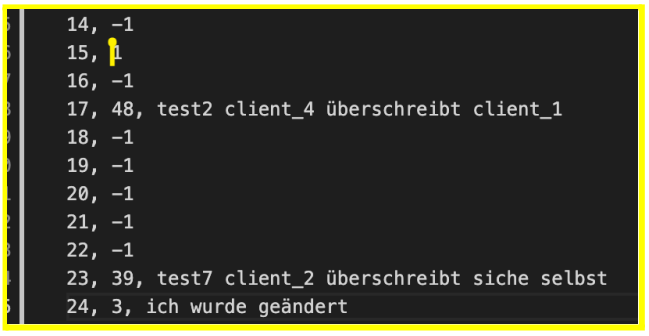
c) Recovery

i) Total program run (with changed pages)

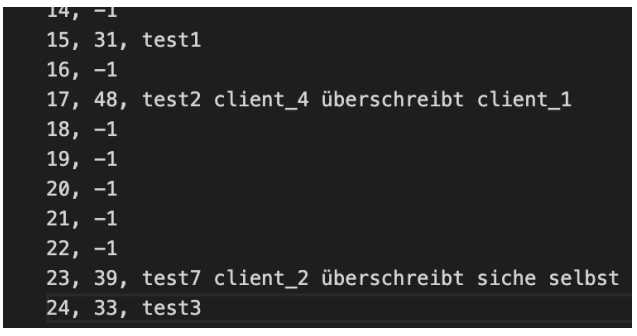
before the change:



after the change :



Result:



ii) changed buffer size (current state of the page)
user data is still empty

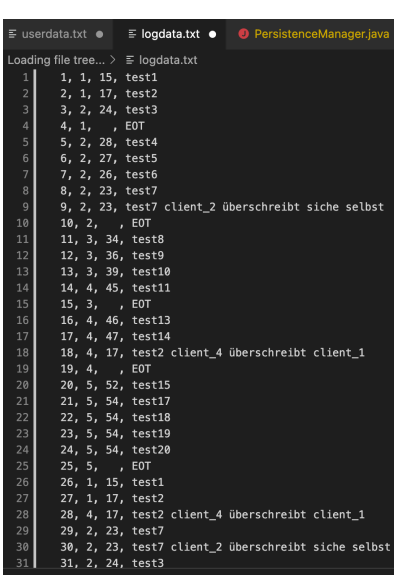
and log data:

userdata.txt | logdata.txt | PersistenceManager.java

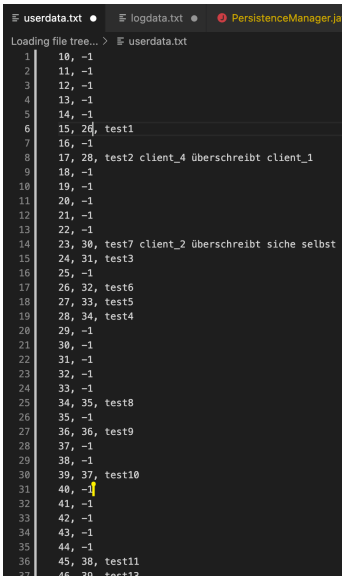
Loading file tree... > logdata.txt

```
1, 1, 15, test1
2, 1, 17, test2
3, 2, 24, test3
4, 1,   , EOT
5, 2, 28, test4
6, 2, 27, test5
7, 2, 26, test6
8, 2, 23, test7
9, 2, 23, test7 client_2 überschreibt siche selbst
10, 2,   , EOT
11, 3, 34, test8
12, 3, 36, test9
13, 3, 39, test10
14, 4, 45, test11
15, 3,   , EOT
16, 4, 46, test13
17, 4, 47, test14
18, 4, 17, test2 client_4 überschreibt client_1
19, 4,   , EOT
20, 5, 52, test15
21, 5, 54, test17
22, 5, 54, test18
23, 5, 54, test19
24, 5, 54, test20
25, 5,   , EOT
```

Result:

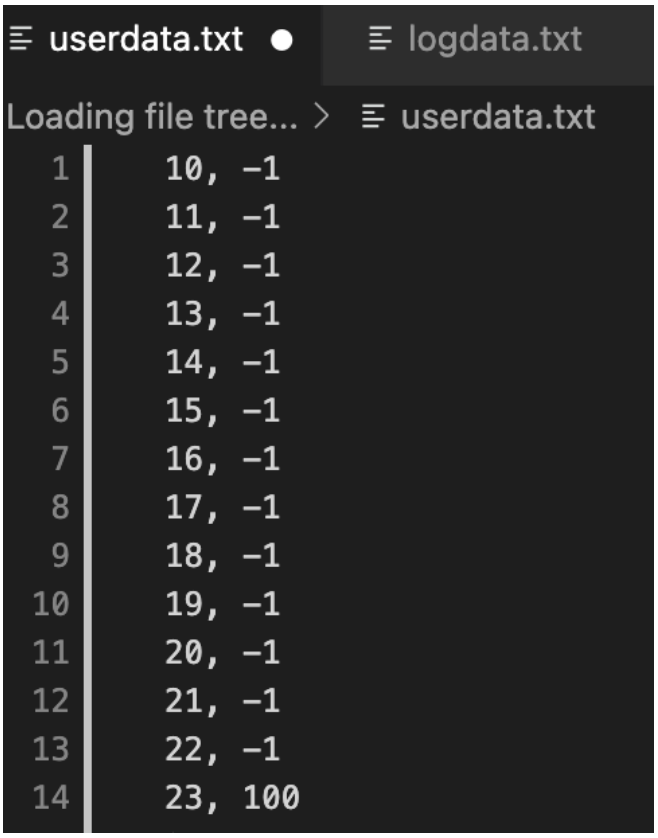


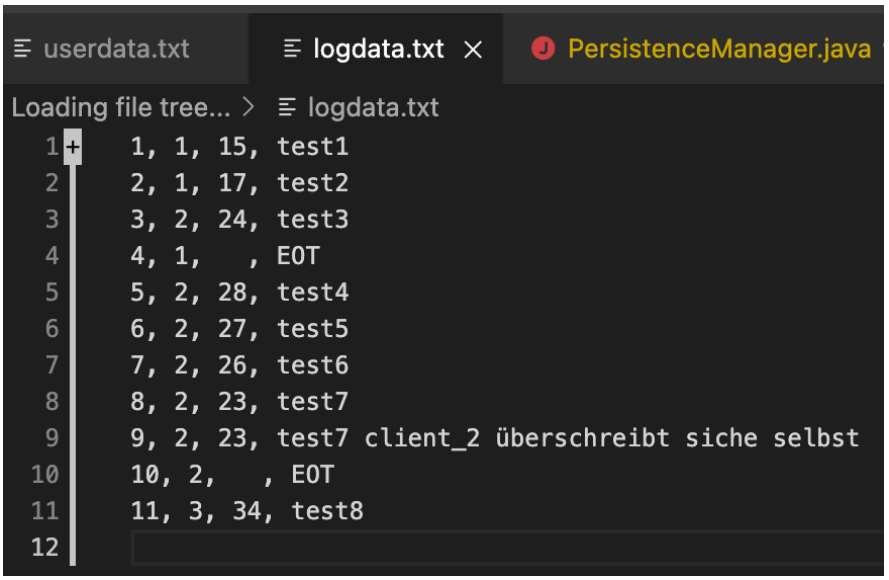
```
1, 1, 15, test1
2, 1, 17, test2
3, 2, 24, test3
4, 1,   , EOT
5, 2, 28, test4
6, 2, 27, test5
7, 2, 26, test6
8, 2, 23, test7
9, 2, 23, test7 client_2 überschreibt siche selbst
10, 2,   , EOT
11, 3, 34, test8
12, 3, 36, test9
13, 3, 39, test10
14, 4, 45, test11
15, 3,   , EOT
16, 4, 46, test13
17, 4, 47, test14
18, 4, 17, test2 client_4 überschreibt client_1
19, 4,   , EOT
20, 5, 52, test15
21, 5, 54, test17
22, 5, 54, test18
23, 5, 54, test19
24, 5, 54, test20
25, 5,   , EOT
26, 1, 15, test1
27, 1, 17, test2
28, 4, 17, test2 client_4 überschreibt client_1
29, 2, 23, test7
30, 2, 23, test7 client_2 überschreibt siche selbst
31, 2, 24, test3
```



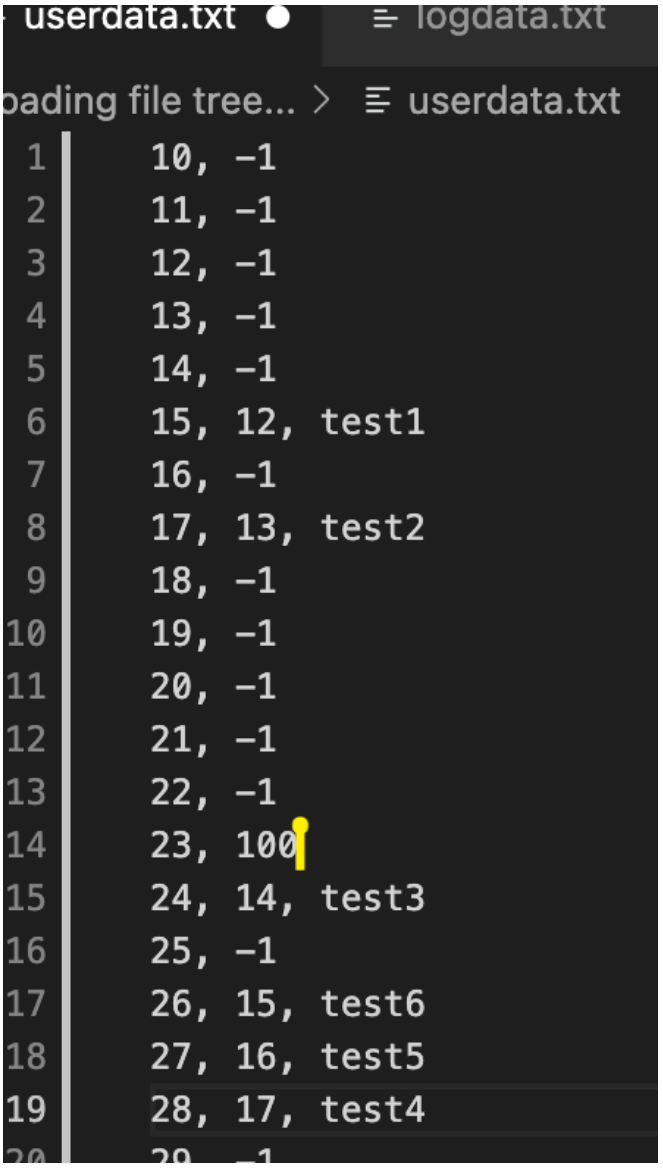
```
10, -1
11, -1
12, -1
13, -1
14, -1
15, 26, test1
16, -1
17, 28, test2 client_4 überschreibt client_1
18, -1
19, -1
20, -1
21, -1
22, -1
23, 30, test7 client_2 überschreibt siche selbst
24, 31, test3
25, -1
26, 32, test6
27, 33, test5
28, 34, test4
29, -1
30, -1
31, -1
32, -1
33, -1
34, 35, test8
35, -1
36, 36, test9
37, -1
38, -1
39, 37, test10
40, -1
41, -1
42, -1
43, -1
44, -1
45, 38, test11
```

iii) changed buffer size (changed lsn)





Result: The user data is still empty at pageid 23



userdata for pageid 23 is empty because the write command didnt get pushed into the hashtable again, since the lsn in the page is higher than in the logdata