CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (FALL 2025)
PROF. ANDY PAVLO

Homework #6 (by Will) – Solutions

Due: **Sunday December 7, 2025 @ 11:59pm**

IMPORTANT:

- Enter all of your answers into **Gradescope by 11:59pm on Sunday December 7, 2025**.
- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.
- **You have to use this PDF for all of your answers.**

For your information:

- Graded out of **100** points; **3** questions total
- Each part is all or nothing. There is no partial credit.

Revision : 2025/12/09 19:06

Question	Points	Score
ARIES	48	
Logging and Recovery	32	
Distributed Databases	20	
Total:	100	

Question 1: ARIES . . . . [48 points]

Consider the sequence of ARIES undo/redo log records with fuzzy checkpointing in Figure 1. The transaction ids are T1, T2, T3, and T4. The object ids are A, B, C, D, and E. Each object is stored in a separate page, and the page id for an object is the same as its object id.

Reminder: Recall the following information about the ARIES log record format:

- The number at the start of each log record indicates the log sequence number (LSN).
- Each record for a transaction write includes the LSN, the transaction id, the log record type, and the previous LSN.
- For updates, the record also includes the object id, the before-image (former value) of the object and the after-image (new value) of the object, in that order.
- Compensation log records contain the same information as update records and also the `undoNextLSN` (i.e., the next LSN that needs to be undone).

LSN	RECORD
1000	`<T1, BEGIN, prevLSN=∅>`
1001	`<T2, BEGIN, prevLSN=∅>`
1002	`<T3, BEGIN, prevLSN=∅>`
1003	`<T1, UPDATE, prevLSN=1000, A, 1, 2>`
1004	`<T3, UPDATE, prevLSN=1002, C, 1, 2>`
1005	`<T3, COMMIT, prevLSN=1004 >`
1006	<**CHECKPOINT-BEGIN**>
1007	`<T1, UPDATE, prevLSN=1003, B, 1, 2>`
1008	`<T4, BEGIN, prevLSN=∅>`
1009	`<T4, UPDATE, prevLSN=1008, C, 2, 3>`
1010	<**CHECKPOINT-END**, DPT={C}, ATT={???}>
1011	`<T2, UPDATE, prevLSN=1001, D, 1, 2>`
1012	`<T4, UPDATE, prevLSN=1009, E, 1, 2>`
1013	`<T3, TXN-END, prevLSN=1005 >`
1014	<**CHECKPOINT-BEGIN**>
1015	`<T1, UPDATE, prevLSN=1007, A, 2, 3>`
1016	`<T4, UPDATE, prevLSN=1012, E, 2, 3>`
1017	<**CHECKPOINT-END**, DPT={???}, ATT={???}>
1018	`<T1, UPDATE, prevLSN=1015, A, 3, 4>`
1019	`<T4, ABORT, prevLSN=1016 >`
1020	<**CHECKPOINT-BEGIN**>
1021	`<T1, COMMIT, prevLSN=1018 >`
1022	`<T4, CLR, prevLSN=1019, E, 3, 2, undoNextLSN=1012 >`
1023	`<T1, TXN-END, prevLSN=1021 >`
	CRASH!

Figure 1: ARIES Write-Ahead Log

Question 1 continues…

(a) Suppose the database started with a clean state (no active transactions and no dirty pages) at the start of the log, a background thread periodically flushes dirty buffer pool pages to reduce eviction and recovery costs, and a crash occurs after the log record `1010` is flushed to disk.

i. **[4 points]** For **A**, which states are possibly stored on disk before recovery starts?
☐ `A=1` [✓ **`A=2`**] ☐ `A=3` ☐ Cannot be determined

Solution: Page A was not in the DPT. Hence `A=2`.

ii. **[4 points]** For **B**, which states are possibly stored on disk before recovery starts?
[✓ **`B=1`**] [✓ **`B=2`**] ☐ Cannot be determined

Solution: Page B was not in the DPT, but there has been a change after the checkpoint starts. Hence `B=1` or `B=2`.

iii. **[4 points]** For **C**, which states are possibly stored on disk before recovery starts?
[✓ **`C=1`**] [✓ **`C=2`**] [✓ **`C=3`**] ☐ Cannot be determined

Solution: Page C was in the DPT at LSN=10. Hence it can be any value.

iv. **[4 points]** Select the transactions that would be in the ATT at `1010`.
[✓ **`T1`**] [✓ **`T2`**] [✓ **`T3`**] ☐ `T4` ☐ Cannot be determined

Solution: `T4` has not started as of `1006`. `T3`'s TXN-END log has not been written yet so it is still in the ATT.

(b) Suppose the database started with a clean state (no active transactions and no dirty pages) at the start of the log, a background thread periodically flushes dirty buffer pool pages to reduce eviction and recovery costs, and a crash occurs after the log record `1017` is flushed to disk.

i. **[4 points]** Select the transactions that would be in the ATT at `1017`.
[✓ **`T1`**] [✓ **`T2`**] ☐ `T3` [✓ **`T4`**] ☐ Cannot be determined

Solution: `T3`'s TXN-END log has been written so it is no longer in the ATT.

ii. **[6 points]** Select all possible values of DPT at `1017`.
☐ `A` [✓ **`B`**] [✓ **`C`**] [✓ **`D`**] [✓ **`E`**] [✓ **`B, C`**] [✓ **`C, D`**] [✓ **`D, E`**]
☐ `A, B, C` [✓ **`B, C, D`**] [✓ **`C, D, E`**]
☐ `A, B, C, D` ☐ `A, C, D, E` [✓ **`B, C, D, E`**]
☐ `A, B, C, D, E` [✓ **Empty**]

Solution: We know `A` was clean. `B, C, D, E` were dirtied between `1006` and `1014`. Since the system can flush dirty pages at any time, any subset of {`B, C, D, E`} is possible.

(c) Suppose the database started with a clean state (no active transactions and no dirty pages) at the start of the log, a background thread periodically flushes dirty buffer pool pages to reduce eviction and recovery costs, and a crash occurs after the log record `1023` is flushed to disk. Assume that at `1017`, the DPT is **Empty**.

Question 1 continues…

i. **[2 points]** Which LSN should be recorded in the DBMS's *MasterRecord*, which records the first LSN to be analyzed for recovery?
☐ `1006` ☐ `1010` [✓ **`1014`**] ☐ `1017` ☐ `1020` ☐ None of the above

Solution: `1014`, the LSN for the last CHECKPOINT-BEGIN corresponding to an CHECKPOINT-END that has been flushed.

ii. **[2 points]** Which pages are *definitely not dirty* at the time the last WAL record was written?
☐ A [✓ **B**] [✓ **C**] [✓ **D**] ☐ E ☐ None of the above

Solution: Because pages for `B, C, D` not appear in the dirty page table in the CHECKPOINT-END record (LSN `1017`), we know that `B, C, D` were flushed at some point before that checkpoint completed. Because the pages were not written again later in the log, we know it was still not dirty at the time the WAL was last written.

iii. **[2 points]** Which pages are *definitely dirty* at the time the database crashed? Select all correct answers.
☐ A ☐ B ☐ C ☐ D ☐ E [✓ **None of the above**]

Solution: Independent of the information in the WAL, the background thread that flushes dirty buffer pool pages could have flushed any (or all) buffer pool pages to disk before the crash. It's possible that no buffer pool pages are dirty at the time of the crash.

iv. **[2 points]** Which pages are *maybe dirty* at the time the database crashed? Select all correct answers.
[✓ **A**] ☐ B ☐ C ☐ D [✓ **E**] ☐ None of the above

v. **[2 points]** Which log records will *definitely be redone* during the ARIES **REDO** phase? Select all correct answers.
☐ `1003` ☐ `1004` ☐ `1007` ☐ `1009` ☐ `1011` ☐ `1012` ☐ `1015`
☐ `1016` ☐ `1018` ☐ `1022` [✓ **None of the above**]

Solution: Analysis of the log will identify pages A and E as potentially dirty at the time of the crash. The DBMS will then read those pages from disk and inspect the `pageLSN` on each page to determine which log entries need to be redone. Because the background thread periodically flushes dirty buffer pool pages to disk, it's possible that all pages on disk include their last logged update. In that case, none of the logged updates will be redone.

vi. **[2 points]** Which log records will *definitely not be redone* during the ARIES **REDO** phase? Select all correct answers.
[✓ **`1003`**] [✓ **`1004`**] [✓ **`1007`**] [✓ **`1009`**] [✓ **`1011`**] [✓ **`1012`**] ☐ `1015`
☐ `1016` ☐ `1018` ☐ `1022` ☐ None of the above

Solution: Because the DPT at `1017` is empty, we know that updates before `1014` must have been flushed before the checkpoint finished. Those updates will definitely not be redone.

Question 1 continues…

vii. **[2 points]** Which log records will *maybe be redone* during the ARIES **REDO** phase? Select all correct answers.
☐ `1003` ☐ `1004` ☐ `1007` ☐ `1009` ☐ `1011` ☐ `1012` [✓ **`1015`**] [✓ **`1016`**] [✓ **`1018`**] [✓ **`1022`**] ☐ None of the above

Solution: ARIES might need to REDO every log record except those identified above.

viii. **[4 points]** Which transactions will be undone in the **UNDO** phase of ARIES? Select all correct answers.
☐ T1 [✓ **T2**] ☐ T3 [✓ **T4**] ☐ None of the above

Solution:

- T1: Committed at `1021`.
- T2: Not committed before crash, so DBMS has to undo.
- T3: Committed at `1005`.
- T4: Aborted at `1019`, so DBMS has to undo.

ix. **[1 point]** For Transaction **T1**, how many new CLR entries will the DBMS add to the WAL during ARIES recovery for each transaction? Do not count any CLR entries that already exist in the WAL before the crash in your answer. [✓ **0**] ☐ 1 ☐ 2 ☐ 3

x. **[1 point]** For Transaction **T2**, how many new CLR entries will the DBMS add to the WAL during ARIES recovery for each transaction? Do not count any CLR entries that already exist in the WAL before the crash in your answer. ☐ 0 [✓ **1**] ☐ 2 ☐ 3

xi. **[1 point]** For Transaction **T3**, how many new CLR entries will the DBMS add to the WAL during ARIES recovery for each transaction? Do not count any CLR entries that already exist in the WAL before the crash in your answer. [✓ **0**] ☐ 1 ☐ 2 ☐ 3

xii. **[1 point]** For Transaction **T4**, how many new CLR entries will the DBMS add to the WAL during ARIES recovery for each transaction? Do not count any CLR entries that already exist in the WAL before the crash in your answer. ☐ 0 ☐ 1 [✓ **2**] ☐ 3

Question 2: Logging and Recovery .... [32 points]

(a) **[4 points]** The STEAL policy means that the pages modified by a transaction *could* be written to non-volatile storage before that transaction commits.

- ■ **True**
- ☐ False

Solution: STEAL allows the DBMS to write uncommitted pages to disk.

(b) **[4 points]** In reference to recovery algorithms that use a write-ahead log (WAL). Under NO-STEAL + FORCE policy, a DBMS will have to undo the changes of an aborted transaction during recovery.

- ☐ True
- ■ **False**

Solution: The FORCE policy guarantees that all changes from committed transactions are immediately written to disk, ensuring durability. It is the NO-STEAL policy that prevents changes by uncommitted transactions from being written to disk, avoiding the need for undo operations on disk for those transactions.

(c) **[4 points]** If the DBMS is using the STEAL and NO-FORCE policies, then a transaction is considered committed if all of the data pages that it modified have been written to non-volatile storage.

- ☐ True
- ■ **False**

Solution: Under STEAL + NO-FORCE, a txn is considered committed when all of the WAL entries for the pages that it modified and the COMMIT record are flushed to disk.

(d) **[4 points]** If the DBMS uses the FORCE policy, it could potentially have to redo the changes of previously committed transactions during recovery.

- ☐ True
- ■ **False**

Solution: If we flush all of the changes that a txn makes to disk atomically when it commits, then we will not need to redo anything during recovery.

(e) **[4 points]** In ARIES, only transactions that commit will have an associated "TXN-END" record in the log.

- ☐ True
- ■ **False**

Solution: Transactions will commit when the COMMIT log is written, TXN-END is an optimization in the ARIES protocol that indicates it can be removed from the transaction table.

Question 2 continues…

(f) **[4 points]** Fuzzy checkpoints do not need to block the execution of all transactions while a consistent snapshot is written to disk.

- [✓ **True**]
- ☐ False

Solution: See motivation of checkpointing.

(g) **[4 points]** After the DBMS begins a non-blocking (i.e., fuzzy) checkpoint, the DBMS cannot flush dirty pages to disk until the checkpoint ends to ensure correctness.

- ☐ True
- [✓ **False**]

Solution: False. Fuzzy checkpoint does not block the system from flushing dirty pages.

(h) **[4 points]** If the DBMS uses a *physical logging* scheme for its WAL, then it does not need to maintain additional WAL records for updates to indexes.

- ☐ True
- [✓ **False**]

Solution: False. Physical logging records page-level changes to tuples. There still needs to be separate log records for index updates. Only logical logging does not require separate log records for indexes.

Homework #6 continues…

Question 3: Distributed Databases .... [20 points]

(a) Consider a database that has a two tables `R(a,b,c)` `S(a,d,e)`, where `S.a` is a foreign key reference to `R.a`. Assume that `R` contains 100,000,000 tuples and `S` contains 1,000,000 tuples. Also assume that the size of a tuple in `R` is the same as a tuple in `S`.

Consider the following query:

```
SELECT * FROM R JOIN S ON R.a = S.a
WHERE R.b BETWEEN 1 AND 100;
```

Answer the following questions about the best way to execute this query given the different configurations in a distributed, shared-nothing DBMS with 10 nodes. You can assume that the DBMS executes a local hash join at each node. You can also assume that it is 10× faster for a node to read data from its local disk than it is to retrieve data from another node over the network. If a table is partitioned, then you can assume that each partition contains the same number of tuples (i.e., the data is not skewed).

Consider the following database configuration:

- Table `R` is partitioned on `R.a`. Each partition fits in memory on a single node.
- Table `S` is replicated at every node. The entire table fits in memory on a single node.

i. **[4 points]** What is the most optimal join execution strategy for table `R`?

■ **Leave R as it exists on every node.**
☐ Shuffle `R` on `R.a` to every node.
☐ Shuffle `R` on `R.b` to every node.
☐ Broadcast `R` to every node.
☐ None of the above

Solution: Table R is partitioned on the join key, so there is no need to move it.

ii. **[4 points]** Given your choice for `R`, what is the most optimal join execution strategy for table `S`?

■ **Leave S as it exists on every node.**
☐ Shuffle `S` on `S.a` to every node.
☐ Shuffle `S` on `S.d` to every node.
☐ Broadcast `S` to every node.
☐ None of the above

Solution: Table S is already replicated at each node, so the DBMS can just do the join on the local data at each partition.

Question 3 continues...

(b) **[4 points]** A distributed DBMS can immediately commit a transaction under network partitioning without any loss of data consistency.

- ☐ True
- ■ **False**

Solution: False. This is a violation of the CAP theorem.

(c) **[4 points]** A DBMS can use either the PAXOS or Two-Phase Commit protocols to support strong consistency for distributed transactions (i.e., transactions that modify data on multiple nodes).

- ■ **True**
- ☐ False

Solution: 2PC is a degenerate case of Paxos, which is a consensus protocol for distributed transactions.

(d) **[4 points]** With consistent hashing, if a node fails, then only a subset and not all keys will be reshuffled among the remaining nodes.

- ■ **True**
- ☐ False

Solution: See motivation of consistent hashing.