Homework 5 Solution
&
Final Review

Final Exam

Time:
Thursday, April 20
7:00 - 9:00pm (the actual length might be shorter)

Location:
1017 Dow (Special considerations)
1670 BBB
STAMPS
(will announce the exact room assignment by Monday noon)

The exam will be closed book and closed notes. A single sheet is allowed per student (both sides can be used)

Requirements:

a. Completely hand-written
b. Your name should be on both sides
c. Cannot make any copies or share with others until after the exam
d. Cannot use other students' sheets to make yours

Calculators Allowed

You are allowed to bring a SAT-like calculator. The use of any other electronics (cell phones, ipads, etc) is strictly prohibited and will be considered a violation of the honor code. You cannot share a calculator with other students during the exam.

Exam Format

There will be some multiple choice questions on the final exam. These questions will be answered on a scantron form and will be auto graded. Therefore, it is your responsibility to bring your own No. 2 pencil (and an eraser) for the scantron questions AS WELL AS a pen for other questions. We will not be providing pens, pencils, erasers, etc.

The exam will cover:

Everything we’ve done after the midterm but also includes indexing and B+ tree (lectures, discussion, h/w, projects)

Question 1: Multiple choices

1.1 Which of the following statements about static hashing is (are) true:

i) number of primary bucket pages fixed
ii) allocated sequentially
iii) never de-allocated
iv) overflow pages if needed
v) use a directory of pointers to buckets
vi) works in round

A. i), ii), iii)
B. i), ii), iii), iv)
C. iii)
D. iii), iv)
E. iii), iv), v)
F. iii), iv), vi)

Question 1: Multiple choices

1.1 Which of the following statements about static hashing is (are) true:

i) number of primary bucket pages fixed
ii) allocated sequentially
iii) never de-allocated
iv) overflow pages if needed
v) use a directory of pointers to buckets
vi) works in round

Extendible hashing

Linear hashing

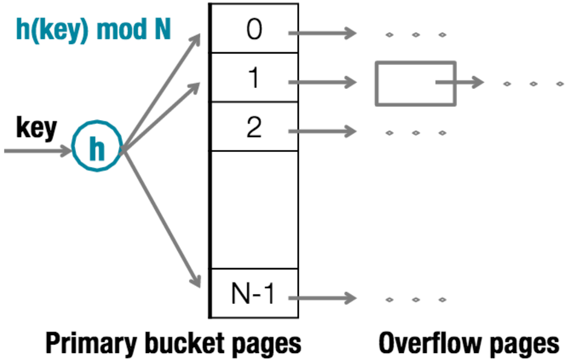
A. i), ii), iii)
B. i), ii), iii), iv)
C. iii)
D. iii), iv)
E. iii), iv), v)
F. iii), iv), vi)

Question 1: Multiple choices

1.1 Which of the following statements about static hashing is (are) true:

Static Hashing

- # primary bucket pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.
- h(key) mod N = bucket to which data entry with key belongs. (N = # of buckets)



What are downsides of static hashing?

Question 1: Multiple choices

1.2 Which of the following statements is true about sorting?

A. We need sorting for the first step to bulk-loading B+ tree, eliminating duplicates records, sort-merge join algorithm, the first step for grace hash join, etc.
B. Both using more buffers and using replacement sort at each step can reduce the number of passes, thus making the external sorting faster.
C. All of blocked I/O, double buffering and B+ tree indexing can make each pass cheaper, thus making external sorting faster.
D. None of above is true.

Question 1: Multiple choices

1.2 Which of the following statements is true about sorting?

A. We need sorting for the first step to bulk-loading B+ tree, eliminating duplicates records, sort-merge join algorithm, the first step for grace hash join, etc.
B. Both using more buffers and using replacement sort at each step can reduce the number of passes, thus making the external sorting faster.
C. All of blocked I/O, double buffering and B+ tree indexing can make each pass cheaper, thus making external sorting faster.
D. None of above is true.

Question 1: Multiple choices

1.2 Which of the following statements is true about sorting?

How to Make External Sorting Faster?

1. Reduce # of passes
   - Using more buffers at each step
   - Using replacement sort at Step 0
2. Make each pass cheaper
   - Blocked I/O
   - Double Buffering
3. Not use External Sort
   - Clustered B+ tree if available
   - Unclustered B+ tree if available and cheaper

Question 1: Multiple choices

1.3 Consider: A relation with 1M tuples, 100 tuples on a page and 500 (key, rid) pairs on a page

A. If the selectivity is 1%, the I/O cost by using non-clustered B+ tree index and the I/O cost by using non-clustered B+ tree index and sorted RIDs are similar.
B. If the selectivity is 10%, the I/O cost by using non-clustered B+ tree index and the I/O cost by using non-clustered B+ tree index and sorted RIDs are similar.
C. If non-clustered B+ tree index is used, no matter the selectivity is 1% or 10%, the I/O cost is similar.
D. When making choice of B+ tree index access plan, we consider the index selectivity and clustering. For hash index, the consideration is very different.

Question 1: Multiple choices

1.3 Consider: A relation with 1M tuples, 100 tuples on a page and 500 (key, rid) pairs on a page

When to Use a B+tree Index?

- Consider
  - A relation with 1M tuples
  - 100 tuples on a page
  - 500 (key, rid) pairs on a page

\# data pages
= 1M/100 = 10K pages
\# leaf idx pgs
= 1M / (500 * 0.67)
~ 3K pages

	1% SELECTION	10% SELECTION
CLUSTERED	~ 30 + 100	~ 300 + 1000
NON-CLUSTERED	~ 30 + 10,000	~ 300 + 100,000
NC + SORT RIDS	~ 30 + (~10,000)	~ 300 + (~10,000)

⇨ Choice of Index access plan, consider:
1. Index Selectivity 2. Clustering

⇨ Similar consideration for hash-based indices

Question 1: Multiple choices

1.3 Consider: A relation with 1M tuples, 100 tuples on a page and 500 (key, rid) pairs on a page

A. If the selection is 1%, the I/O cost by using non-clustered B+ tree index and the I/O cost by using non-clustered B+ tree index and sorted RIDs are similar.
B. If the selection is 10%, the I/O cost by using non-clustered B+ tree index and the I/O cost by using non-clustered B+ tree index and sorted RIDs are similar.
C. If non-clustered B+ tree index is used, no matter the selection is 1% or 10%, the I/O cost is similar.
D. When making choice of B+ tree index access plan, we consider the index selectivity and clustering. For hash index, the consideration is very different.

Question 1: Multiple choices

1.4 Which of the following statements is true about join algorithms?

A. Sort-merge join can be optimized to 3(|R| + |S|) I/Os when smaller relation |S| <= B^2, while hash join costs 3(|R| + |S|) I/Os too when smaller relation |R| <= B^2.
B. If the relation size differ greatly, or partitioning is skewed, hash join is better than sort-merge join.
C. Both BNL join and INL join with B+tree index work when we have inequality conditions (e.g. R.rname < S.sname), and BNL is usually better.
D. Hash join does not work if there are equalities over several attributes while BNL join, INL join, and sort-merge join work.

Question 1: Multiple choices

1.4 Which of the following statements is true about join algorithms?

A. Sort-merge join can be optimized to 3(|R| + |S|) I/Os when smaller relation |S| <= B^2, while hash join costs 3(|R| + |S|) I/Os too when smaller relation |R| <= B^2.

B. If the relation size differ greatly, or partitioning is skewed, hash join is better than sort-merge join.

C. Both BNL join and INL join with B+tree index work when we have inequality conditions (e.g. R.rname < S.sname), and BNL is usually better.

D. Hash join does not work if there are equalities over several attributes while BNL join, INL join, and sort-merge join work.

Question 1: Multiple choices

1.4 Which of the following statements is true about join algorithms?

How to Sort-Merge in 3(|R|+|S|) I/Os

- Before:
  - External sorting: $2.|R|.(1+\lceil \log_{B-1}\lceil |R|/B\rceil\rceil)+2.|S|.(1+\lceil \log_{B-1}\lceil |S|/B\rceil\rceil)$
  - Merge cost (no backups) = |R|+|S|
  - Merge cost (backups) = |R| * |S|
- **Can be done in 3(|R|+|S|):**
  - When larger relation |S| ≤ B, i.e. each relation fits in memory
- **Rationale**:
  - R fits in memory, sort in 2*|R| I/Os
  - S fits in memory, sort in 2*|S| I/Os
  - Assuming no backups are required, merge would take only |R|+|S|

Question 1: Multiple choices

1.4 Which of the following statements is true about join algorithms?

How to Sort-Merge in 3(|R|+|S|) I/Os

- Before:
  - External sorting: $2.|R|.(1+\lceil \log_{B-1}\lceil |R|/B \rceil \rceil) + 2.|S|.(1+\lceil \log_{B-1}\lceil |S|/B \rceil \rceil)$
  - Merge cost (no backups) = |R|+|S|
  - Merge cost (backups) = |R| * |S|
- **Can be done in 3(|R|+|S|):** square root of |S| fits in memory
  - When larger relation $|S| \leq B^2$, i.e. ~~each relation fits in memory~~
- **Example**: Assume |R| = 400, |S|=10,000 and B = 100
  - Use replacement sort to produce runs of 2*M=2*(B-2)=196 pages long ≈200=2*B for each relation.
  - Then we have at most |R|/(2B)≈2 runs of R and |R|/(2B)≈50 runs of S.
  - Note that since $|R| \leq |S| \leq B^2$ we have $|R|/(2B) \leq B/2$ and $|R|/(2B) \leq B/2$
  - Since total # of runs ≤B, we can read in memory one page per run (here 52 pages), merge, and apply join condition on-the-fly + one output page

Question 1: Multiple choices

1.4 Which of the following statements is true about join algorithms?

Join Algorithms: A Comparison

- Hash-Join vs. Sort-Merge Cost:
  - Sort-merge can be optimized to $3(|R|+|S|)$ I/Os
    - When larger relation $|S| \leq B^2$
    - **Memory requirements:** larger relation $|S| \leq B^2$
  - Hash join costs $3(|R|+|S|)$ I/Os
    - If each partition of the smaller relation fits in memory
    - **Memory requirements:** Smaller relation $|R| \leq B^2$

- Hash Join superior if relation sizes differ greatly
- Hash Join is highly parallelizable
- Hash join worse if partitioning is skewed
- Sort-Merge results already sorted (if matters)

Question 1: Multiple choices

1.4 Which of the following statements is true about join algorithms?

General Join Conditions

- Inequality conditions (e.g. `R.rname < S.sname`):
  - Block Nested Loop: still works
  - For Index Nested Loop, needs B+ tree index
    - Usually worse than Block Nested Loops
  - Sort-Merge and Hash-Join are not applicable

Question 1: Multiple choices

1.4 Which of the following statements is true about join algorithms?

General Join Conditions

- Equalities over several attributes is doable with all Joins!
  e.g. `R.sid=S.sid AND R.rname=S.sname`
  - Block Nested Loop
    - always works
  - Index Nested Loop:
    - index on *<sid, sname>* or sid or sname.
    - Usually more I/Os than BNL
  - Sort-merge join:
    - sort Reserves on <sid, rname> and Sailors on <sid, sname>
  - Hash join:
    - hash Reserves on <sid, rname> and Sailors on <sid, sname>

Question 1: Multiple choices

1.5 Which of the following RA equivalence is wrong?

A. $\sigma_{P1 \wedge P2 \ldots \wedge Pn}(R) \equiv \sigma_{P1}(\sigma_{P2}(\ldots \sigma_{Pn}(R)))$ (cascading σ)

B. $\prod_{a1}(R) \equiv \prod_{a1}(\prod_{a2}(\ldots\prod_{ak}(R)\ldots))$, a_{i+1} belongs to a_i (cascading ∏)

C. $\Pi_A(\sigma_c(R)) \equiv \sigma_c(\Pi_A(R))$ (if selection only involves attributes in the set A)

D. $\sigma_P(R \times S) \equiv \sigma_P(R) \times S$ (if p is only on R)

Question 1: Multiple choices

1.5 Which of the following RA equivalence is wrong?

A. $\sigma_{P1 \wedge P2 \ldots \wedge Pn}(R) \equiv \sigma_{P1}(\sigma_{P2}(\ldots \sigma_{Pn}(R)))$ (cascading σ)

B. $\prod_{a1}(R) \equiv \prod_{a1}(\prod_{a2}(\ldots\prod_{ak}(R)\ldots))$, a_{i+1} belongs to a_i (cascading ∏)

C. $\Pi_A(\sigma_c(R)) \equiv \sigma_c(\Pi_A(R))$ (if selection only involves attributes in the set A)

D. $\sigma_P(R \text{ X } S) \equiv \sigma_P(R) \text{ X } S$ (if p is only on R)

Question 2: Transactions

T1	T2	T3
R(A)		
W(A)		
	R(A)	
		R(B)
		W(B)
	R(B)	
	W(B)	
	COMMIT	
COMMIT		
		COMMIT

1. Is it recoverable? Is it in ACA?
2. Would this deadlock under 2PL?
3. Is it conflict serializable?

Question 2: Transactions

T1	T2	T3
R(A)		
W(A)		
	R(A)	
		R(B)
		W(B)
	R(B)	
	W(B)	
	COMMIT	
COMMIT		
		COMMIT

1. Is it recoverable? Is it in ACA? NO NO
2. Would this deadlock under 2PL? NO
3. Is it conflict serializable? YES

Question 2: Transactions

4. Describe two ways to address or avoid deadlock among transactions.

Question 2: Transactions

4. Describe two ways to address or avoid deadlock among transactions.

Any two of:

1) Simply abort and restart transactions that take too long
2) Check the waits-for graph. If there's a cycle, abort one
3) Wait-die: Assign each tx a priority. If Ti requests a lock held by Tj, then Ti only waits if it has a higher priority. Otherwise Ti aborts
4) Wound-wait: Assign each tx a priority. If Ti has a higher priority, abort Tj. Otherwise Ti waits.

Question 2: Transactions

4. Describe two ways to address or avoid deadlock among transactions.

Deadlock Detection

- Observation:
  - Deadlocks are rare
  - Often involve only a few transactions
  - Detect rather than prevent
- Approach #1: Lock Mgr maintains waits-for graph
  - Periodically check graph for cycles
  - Abort some Xact to break the cycle
- Approach #2 (Simpler hack): use *time-outs*
  - Abort if no progress made for a while

Question 2: Transactions

4. Describe two ways to address or avoid deadlock among transactions.

Deadlock Prevention

- Assign priorities to transactions
  - Use timestamp to assign priorities
- Ti requests a lock, held by Tj (in a conflicting mode)
  - Approach #1 (Wait-Die): If Ti has higher priority, it waits; else Ti aborts
  - Approach #2 (Wound-Wait): If Ti has higher priority, abort Tj; else Ti waits
  - After abort, restart with original timestamp
    - Guarantees the transaction eventually runs!

Question 2: Transactions

5. Imagine that an evil genie has told you that your database can be NO-FORCE or STEAL, but not both. Your entire database can easily fit into RAM. Which feature (NO-FORCE or STEAL) will likely yield the better performing system? Why?

Question 2: Transactions

5. Imagine that an evil genie has told you that your database can be NO-FORCE or STEAL, but not both. Your entire database can easily fit into RAM. Which feature (NO-FORCE or STEAL) will likely yield the better performing system? Why?

You should choose NO-FORCE.
If your database can fit into memory, there will be little or no pressure on the buffer pool and thus so no need for STEALing. NO-FORCE, on the other hand, will still be useful to make sure users' transactions enjoy low latency.

Question 3

Why is writing to the log faster than simply flushing dirty pages upon commit? Give 2 reasons.

Question 3

Why is writing to the log faster than simply flushing dirty pages upon commit? Give 2 reasons.

1) Updates to log are smaller than full pages, even when considering the before/after images

2) Log additions are sequential

Question 3

Why is writing to the log faster than simply flushing dirty pages upon commit? Give 2 reasons.

Write-Ahead Log (WAL)

- Why is WAL faster than FORCE?
  - Both need to FORCE data to disk as transactions are committed
- Two advantages:
  - WAL records typically **smaller** than database data pages
  - WAL is **sequential**. Traditional disks are great for sequential writes

Question 4: Recovery

LSN LOG

1. start T1
2. start T2
3. update: T1 writes to P1
4. update: T2 writes to P1
5. start T3
6. update: T3 writes to P2
7. update: T1 writes to P1
8. update T2 writes to P2
9. T1 commit
10. T1 end
11. begin checkpoint
12. end checkpoint
13. update: T3 writes to P1
14. update: T2 writes to P3
15. update: T3 writes to P2
16. CRASH

a) Draw the transaction table and dirty page table before the crash

Question 4: Recovery

LSN LOG

1. start T1
2. start T2
3. update: T1 writes to P1
4. update: T2 writes to P1
5. start T3
6. update: T3 writes to P2
7. update: T1 writes to P1
8. update T2 writes to P2
9. T1 commit
10. T1 end
11. begin checkpoint
12. end checkpoint
13. update: T3 writes to P1
14. update: T2 writes to P3
15. update: T3 writes to P2
16. CRASH

a) Draw the transaction table and dirty page table before the crash

TX_ID	LSN	Status
T3	15	U
T2	14	U

Page ID	LSN
P1	3
P2	6
P3	14

Question 4: Recovery

LSN LOG

1. start T1
2. start T2
3. update: T1 writes to P1
4. update: T2 writes to P1
5. start T3
6. update: T3 writes to P2
7. update: T1 writes to P1
8. update T2 writes to P2
9. T1 commit
10. T1 end
11. begin checkpoint
12. end checkpoint
13. update: T3 writes to P1
14. update: T2 writes to P3
15. update: T3 writes to P2
16. CRASH

b) Draw the transaction table and dirty page table after the analysis phase

Question 4: Recovery

LSN LOG

1. start T1
2. start T2
3. update: T1 writes to P1
4. update: T2 writes to P1
5. start T3
6. update: T3 writes to P2
7. update: T1 writes to P1
8. update T2 writes to P2
9. T1 commit
10. T1 end
11. begin checkpoint
12. end checkpoint
13. update: T3 writes to P1
14. update: T2 writes to P3
15. update: T3 writes to P2
16. CRASH

b) Draw the transaction table and dirty page table after the analysis phase

TX_ID	LSN	Status
T3	6	U
T2	8	U

Page ID	LSN
P1	3
P2	6

LSN LOG

1. start T1
2. start T2
3. update: T1 writes to P1
4. update: T2 writes to P1
5. start T3
6. update: T3 writes to P2
7. update: T1 writes to P1
8. update T2 writes to P2
9. T1 commit
10. T1 end
11. begin checkpoint
12. end checkpoint
13. update: T3 writes to P1
14. update: T2 writes to P3
15. update: T3 writes to P2
16. CRASH

c) At what LSN will the ANALYSIS phase start processing the log?

d) At what LSN will the REDO phase start processing the log?

e) Imagine that you are building a new database system which implements ARIES-style recovery. Your testing reveals that recovery is taking far too long. What is one well-established way you can speed up the ANALYSIS phase?

f) Again, you're building a new database system. You find a bug in the buffer manager that means dirty pages are rarely flushed. Which recovery phase --- ANALYSIS, REDO, OR UNDO --- will likely take a very long time to execute?

LSN LOG

1. start T1
2. start T2
3. update: T1 writes to P1
4. update: T2 writes to P1
5. start T3
6. update: T3 writes to P2
7. update: T1 writes to P1
8. update T2 writes to P2
9. T1 commit
10. T1 end
11. begin checkpoint
12. end checkpoint
13. update: T3 writes to P1
14. update: T2 writes to P3
15. update: T3 writes to P2
16. CRASH

c) At what LSN will the ANALYSIS phase start processing the log?

LSN 11

d) At what LSN will the REDO phase start processing the log?

LSN 3

e) Imagine that you are building a new database system which implements ARIES-style recovery. Your testing reveals that recovery is taking far too long. What is one well-established way you can speed up the ANALYSIS phase?

Checkpoint more frequently

f) Again, you're building a new database system. You find a bug in the buffer manager that means dirty pages are rarely flushed. Which recovery phase --- ANALYSIS, REDO, OR UNDO --- will likely take a very long time to execute?

REDO

Question 5: Join

Suppose we have two relations R and S we want to join on the condition R.a = S.a. R consists of 100 pages and S consists of 400 pages. Suppose we have 5 Buffer pages. Each page of R contains 75 tuples and each page of S contains 50 tuples. (using sort merge with <u>no replacement sort </u>optimization)

a) What is the cost of sorting each relation?

b) What is the cost of the join in the best case?

c) What is the cost of the join in the worst case? When would this occur?

d) What is one improvement to sort-merge join that would reduce the overall cost of the algorithm? How much is saved?

Question 5: Join

Suppose we have two relations R and S we want to join on the condition R.a = S.a. R consists of 100 pages and S consists of 400 pages. Suppose we have 5 Buffer pages. Each page of R contains 75 tuples and each page of S contains 50 tuples. (using sort merge with <u>no replacement sort</u> optimization)

a) What is the cost of sorting each relation?

Sorting R= 2*(100) * (ceil($\log_4$(100/5)) + 1) = **800 page I/Os**

Sorting S = 2*(400) * (ceil($\log_4$(400/5)) + 1) = **4000 page I/Os**

b) What is the cost of the join in the best case?

Best case would be only having to read through R and S once.

100 + 400 = **500 page I/Os**

c) What is the cost of the join in the worst case? When would this occur?

Worst case would be when R.a and S.a are all equal.

100*400 = **40,000 Page I/Os**

d) What is one improvement to sort-merge join that would reduce the overall cost of the algorithm? How much is saved?

Start the merge during the last pass of sorting rather than R/W the sorted relations.

Savings = 2*100+2*400 = **1000 I/Os** (saves cost of last pass)

Thursday, April 20
7:00 - 9:00pm

ALL THE BEST FOR THE EXAM

HW5 solution

Question 1

(A) **[4 points]** Is this schedule recoverable? Explain.

(B) **[4 points]** Does this schedule avoid cascading aborts? Explain.

(C) **[4 points]** Could it be generated by non-strict 2PL? Explain.

(D) **[4 points]** Could it be generated by strict 2PL? Explain.

T1	T2	T3	T4
		R(Z)	
		W(Z)	
	R(X)		
	W(X)		
	R(Y)		
R(X)			
			R(Y)
			COMMIT
COMMIT			
	COMMIT		

Question 1

A) No. T1 read value from T2 and commits before T2 commits
B) No. Not even recoverable. A schedule avoids cascading aborts if the transactions only read the changes of committed transactions.
C) Yes. Every Tx can release its locks at the moment it completes its operation.
D) No. T2 will not release its exclusive lock on X until it commits. So T1 could not have acquire shared lock.

Question 2

Determine if the following schedules are: conflict serializable, and/or serializable. If it is, provide a possible equivalent serial schedule.

Question 2A

- conflict serializable: no, T3 T1 cycle
- serializable: no (Considering the final writes, only T2T3T1, T3T2T1 are possible. However both modify A so R(A) from T1 will be affected)

T1	T2	T3
R(A)		
	W(A) COMMIT	
		R(A)
		W(A) COMMIT
W(A) COMMIT		

Question 2B

- conflict serializable: no, T2 T1 cycle
- serializable: no (final write on B is from T1, so T1 must be the last. But T2 will modify B that T1 read)

T1	T2	T3
R(A)		
		R(B) R(A)
	W(A)	
R(B)		
	W(B) COMMIT	
W(B) COMMIT		
		COMMIT

Question 2C

- conflict serializable: Yes. Following the definition, ignore aborted transaction, and other Tx has no cycle in precedence graph T1 T3
- serializable: yes T1 T3

T1	T2	T3
		R(A)
R(B) W(B) COMMIT		
		W(B)
	R(B) W(A)	
		W(A) COMMIT
	ABORT	

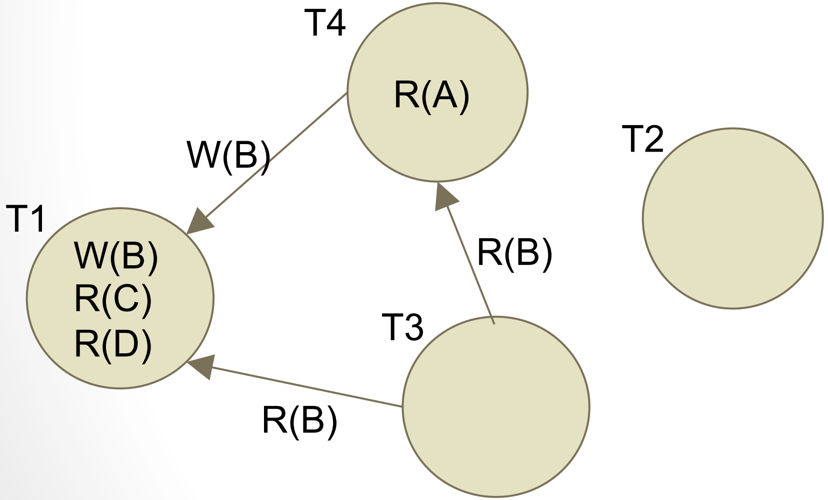
Question 2D

- conflict serializable: no, T3 T4 cycle, T1 T4 cycle
- serializable: yes, [T1T2T3]T4

T1	T2	T3	T4
			R(A)
R(B) R(D)			
			W(B)
		R(B) W(C)	
R(C) W(B) COMMIT			
	R(C) W(B) COMMIT		
		COMMIT	
			R(C) W(C) W(B) COMMIT

Question 3

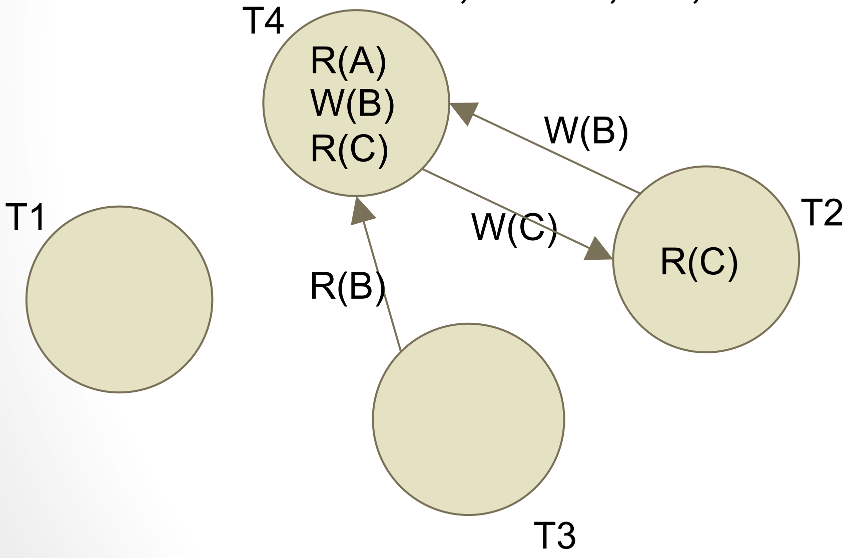
- B will not deadlock
- C will not
- D will deadlock, on T2, T3, T4



T1	T2	T3	T4
			R(A)
R(B) R(D)			
			W(B)
		R(B) W(C)	
R(C) W(B) COMMIT			
	R(C) W(B) COMMIT		
		COMMIT	
			R(C) W(C) W(B) COMMIT

Question 3

- B will not deadlock
- C will not
- D will deadlock, on T2, T3, T4



T1	T2	T3	T4
			R(A)
R(B) R(D)			
			W(B)
		R(B) W(C)	
R(C) W(B) COMMIT			
	R(C) W(B) COMMIT		
		COMMIT	
			R(C) W(C) W(B) COMMIT

Question 4

Consider the following example of operations on a DBMS:

- Transaction 1 modifies Page 1
- Transaction 1 modifies Page 3
- Transaction 2 modifies Page 2
- Crash!!!!

Assume that:

The memory can only accommodate 2 pages

Avoid writing pages back to disk if not necessary (do not force)

No logging or any other kind of book keeping technique is used here

(A) At timestamp 3, What does the DBMS have to do to make sure there is space to modify Page 2?

(B) After timestamp 4, the DBMS came back without running any recovery protocol. Which property in ACID does it violates now? Explain it in 1~2 sentences

Question 4

Now consider another scenario:

- Transaction 1 modifies Page 1
- Transaction 1 modifies Page 3
- Transaction 1 commits
- Transaction 2 modifies Page 2
- Crash!!!!

Again, assume that:

The memory can accommodate 4 pages

Avoid writing pages back to disk if not necessary (do not force)

No logging or any other kind of book keeping technique is used here

(A) After timestamp 5., the DBMS came back without running any recovery protocol. Which property in ACID does it violates now? Explain it in 1~2 sentences.

(B) How can we restore this property without using logging? Is there any tradeoff?

Question 4

(A) DBMS has to write a uncommitted page to disk(steal)
(B) Atomicity, only a part of Tx1 persists
(C) Durability, committed data loss
(D) Force write when a Tx commits. Higher cost.

Question 5

Consider the following log file, which was found on disk, and the ARIES recovery protocol:

(A) Describe the dirty page table at the end of the ANALYSIS phase of recovery.
(B) Describe the transaction table at the end of the ANALYSIS phase of recovery
(C) Which transactions are "loser" transactions
(D) Are there any new compensation log records (CLRs) written during the REDO phase of recovery? Explain.
(E) How many CLR records are added in the UNDO phase? Briefly describe them.

LSN	LOG
1	T1 writes to P1
2	T2 writes to P2
3	T3 writes to P3
4	T1 writes to P1
5	T2 writes to P2
6	T1 commit
7	T1 end
8	begin checkpoint
9	end checkpoint (along with the dirty page table and transaction table)
10	T3 writes to P1
11	T3 commit
12	T3 end
13	T2 writes to P3
14	CRASH

Question 5

(A)

Page ID	recLSN
P1	1
P2	2
P3	3

XID	Last LSN	Status
T2	13	U

(B) T2

(C) No, REDO is applied only to logged actions, so there is no additional logging required. (A CLR is added only during the UNDO phase if there is a need to undo an update.)

(D) 3 - 3 CLRs for undoing T2's update to P3, P2(twice)