CS W186 Fall 2019 Final

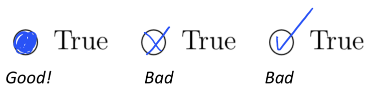
Do not turn this page until instructed to start the exam.

Contents:

- You should receive one *double-sided answer sheet* and a 37-page *exam packet.*
- The midterm has *11 questions*, each with multiple parts, and worth a total of *153.5 points.*

Taking the exam:

- You have *170 minutes* to complete the midterm.
- All answers should be written on the answer sheet. The exam packet will be collected but not graded.
- For each question, place only your *final answer* on the answer sheet; *do not show work.*
- For multiple choice questions, please *fill in the bubble or box completely* as shown on the left below. ***Answers marking the bubble or box with an X or check mark may receive a point penalty.***



- A blank page is provided at the end of the exam packet for use as scratch paper.

Aids:

- You are allowed **three handwritten** 8.5” × 11” double-sided pages of notes.
- *No electronic devices are allowed on this exam.* No calculators, tablets, phones, smartwatches, etc.

Grading Notes:

- All I/Os must be written as integers. There is no such thing as 1.02 I/Os – that is actually 2 I/Os.
- 1 KB = 1024 bytes. We will be using powers of 2, not powers of 10
- Unsimplified answers, like those left in log format, will receive a point penalty.

0 Pre-Exam Questions (0 points)

1. (0.0001 points) How many hours of week should a head TA work while concussed on medical leave?

Solution: 30 hours a week to handle academic dishonesty cases.

2. (0.0001 points) What was the maximum attainable selection index for the PSAT in 2013-14?

Solution: 240. We're old.

1 Broken Joins (12 points)

1. (1 point) Select all possible implementations that can be used for the following join:

   SELECT * FROM R, S WHERE R.a != S.b;

   A. Page Nested Loop Join
   B. Block Nested Loop Join
   C. Index Nested Loop Join
   D. Grace Hash Join
   E. Sort Merge Join

   Solution: INLJ, Grace Hash Join, and Sort Merge Join can only be used for equijoins (=). PNLJ and BNLJ can be used to implement joins with any join condition (!=, >, <, etc.).

2. (3 points) The inventor of Block Nested Loop Join has decided that block sizes are now always going to be B/2. What is the min I/O cost of joining R, S where [R] = 70 pages, [S] = 50 pages, and B = 4 pages.

   Solution: The join order that results in the min I/Os is S join R.

   The total I/O cost = 1800 I/Os. 50 + 50/(4/2) * 70 = 1800.

3. (4 points) From the depths of the 186 archives, we have retrieved 2 files G, S dating back to 1998 where G = grades and S = students. [G] = 110 pages, [S] = 25 pages, and B = 6 pages. 80% of values in G's join column are duplicates while the other 20% are unique values. Assume a perfect hash function is used to partition G and that its duplicates are all mapped to partition 1. S's join column consists of unique values and an imperfect hash function is used to create the following partitions of S:

   - partition 1: 6 pages
   - partition 2: 3 pages
   - partition 3: 4 pages
   - partition 4: 8 pages
   - partition 5: 4 pages

   What is the total I/O cost of performing Grace Hash Join on G, S assuming that perfect hash functions are used on each partitioning phase following the initial partitioning phase?

   Solution: There are 2 answers that were accepted since the range of values were not given:

   Answer 1:
   Using a uniform hash function, G will have the following partitions:

   - partition 1: 88 + 5 = 93

- partition 2: 5
- partition 3: 5
- partition 4: 5
- partition 5: 5

Since B = 6, only the 1st and 4th pair of partitions need to be recursively partitioned again. After recursively partitioning the 1st pair of partitions, 5 partitions of [G] = 19, [S] = 2 are returned. After recursively partitioning the 4th pair of partitions, 5 partitions of [G] = 1, [S] = 2 are returned.

Therefore, the total cost is (110 + 25) (divide phase in) + (113 + 25) (divide phase out) + (93 + 6) (recursively partition 1 in) + (19*5 + 2*5) (recursively partition 1 out) + (5 + 8) (recursively partition 4 in) + (1*5 + 2*5) (recursively partition 4 out) + (115 + 31) (conquer phase in) = **651** I/Os

Answer 2:
Using a uniform hash function, G will have the following partitions:

- partition 1: 88
- partition 2: 6
- partition 3: 6
- partition 4: 6
- partition 5: 6

Since B = 6, only the 1st and 4th pair of partitions need to be recursively partitioned again. After recursively partitioning the 1st pair of partitions, 5 partitions of [G] = 18, [S] = 2 are returned. After recursively partitioning the 4th pair of partitions, 5 partitions of [G] = 2, [S] = 2 are returned.

Therefore, the total cost is (110 + 25) (divide phase in) + (112 + 25) (divide phase out) + (88 + 6) (recursively partition 1 in) + (18*5 + 2*5) (recursively partition 1 out) + (6 + 8) (recursively partition 4 in) + (2*5 + 2*5) (recursively partition 4 out) + (118 + 31) (conquer phase in) = **649** I/Os

4. (3 points) Due to old age, our machine has forgotten how to sort. Luckily, we have indexes on the join columns between R and S. R's index is an Alt 2 unclustered index with a height of 2 and contains 10 leaf pages. S's index is an Alt 2 clustered index with a height of 2 and contains 12 leaf pages. R consists of 30 pages with 4 records on each page and S consists of 40 pages with 5 records on each page. What is the average cost of performing Sort Merge Join on R and S?

   You can assume that resetting to a record will never require accessing a previous leaf page or a previous data page and that there are enough memory pages to satisfy any operation.

Solution: Sort Merge Join on R,S normally iterates through both relations, compares records, and costs N + M I/Os where N = # of pages of R and M = # of pages of S. This same logic can be applied for indexes since leaf pages can be iterated over as each page contains a pointer to the next leaf page.

Iterating over S's sorted records takes a total of 132 I/Os: 3 for traversing to a leaf, 9 for traversing all the leaf pages, and 30 * 4 for accessing every record since the index is unclustered.

Iterating over R's sorted records takes a total of 54 I/Os: 3 for traversing to a leaf, 11 for traversing the data pages, and 40 for accessing every data page.

The total I/O cost for the average case = **186** I/Os.

5. (1 point) Grace Hash Join uses exactly two hash functions: 1 for partitioning and 1 for the in-memory hash table.

   A. True

   B. False

Solution: False because recursive partitioning would necessitate the use of an extra hash function per recursive pass.

2 Concussion Recovery (@lakya) (11 points)

Here's a list of buzzwords about databases that investors love to hear:

- Atomicity
- Durability
- Flush
- Force
- Memory Usage
- No-Force
- No-Steal
- Performance
- Redo
- Recovery
- Steal
- Undo

Use the given database buzzwords to fill in the blanks. A word may be used more than once.

1. (0.5 points) A property of transactions is ________, which refers to the idea that we will never lose the result of some transaction.

   Solution: durability

2. (0.5 points) One way to ensure this property is to use the ________ policy.

   Solution: force

3. (0.5 points) However, the downside to using this policy is ________.

   Solution: performance

4. (0.5 points) In the forward processing portion of ARIES recovery (as covered in this course), we choose to use the ________ policy, which allows us to avoid immediately flushing dirty pages to disk upon commit.

Solution: no-force

5. (0.5 points) However, our decision to use this policy complicates ________ which is a key part of ACID.

Solution: durability

6. (0.5 points) To overcome this problem, we will need to ________ operations during recovery.

Solution: redo

Fill in the following blanks with your favorite (correct) inequality symbols:

7. (1 point) Log records must be written to disk when a transaction t commits. The inequality that must be true in order to achieve this is: lastLSN_t __ flushedLSN.

Solution: lastLSNt $\leq$flushedLSN

8. (1 point) Log records must be on disk before data page i gets written to disk. The inequality that must be true in order to achieve this is: pageLSN_i __ flushedLSN.

Solution: pageLSN $\leq$flushedLSN

You're a database. You're managing transactions and pages and logs and you oop... you just crashed. You get the log up to the crash and see the following:

LSN	Record	prevLSN
0	master: checkpoint at LSN 80	-
10	update: T1 writes P1	0
20	update: T2 writes P2	0
30	update: T3 writes P3	0
40	update: T1 writes P1	10
50	update: T2 writes P4	20
60	abort: T1	40
70	CLR: undo T1 LSN 40, undoNextLSN 10	60
80	begin checkpoint	-
90	update: T2 writes P4	50
100	abort: T2	90
110	commit: T3	30
120	end: T3	110
130	end checkpoint	-
	C R A S H	

The end checkpoint also contains the following tables:

TID	Status	lastLSN
T1	Aborting	70
T2	Running	50
T3	Running	30

PID	recLSN
P1	10
P2	20
P3	30
P4	50

9. (2 points) Fill in the following dirty page table and transaction table as they would appear at the end of the analysis phase of ARIES. If an entry should not appear in the table then leave the row blank.

TID	Status	lastLSN
T1		
T2		
T3		

PID	recLSN
P1	
P2	
P3	
P4	

Solution:

TID	Status	lastLSN
T1	Aborting	70
T2	Aborting	100

PID	recLSN
P1	10
P2	20
P3	30
P4	50

We were given the LSN of the most recent checkpoint in the master record, so we can take the tables we got from the checkpoint and start analysis from LSN 80. We see that the only page modified since the checkpoint is P4, which is already in the DPT, so we don't make any changes to the DPT. We see that only T2 and T3 had log records emitted since the checkpoint, so we update the transaction table for T2 and T3 to reflect those log records, specifically, that T2's lastLSN is now 100 and its status is aborting, and since there is an End record at LSN 120, we remove T3 from the transaction table altogether.

10. (4 points) Assume that we've already completed the redo phase. Complete the undo phase, specifically, what log records are emitted? Use the table provided in the answer sheet to fill out all records emitted during this phase. You may not need all the rows provided in the table.

Solution:

LSN	Record	prevLSN	undoNextLSN
130	CLR: undo T2 LSN 90	100	50
140	CLR: undo T2 LSN 50	130	20
150	CLR: undo T2 LSN 20	140	0
160	end: T2	150	
170	CLR: undo T1 LSN 10	70	0
180	end: T1	170	

We add the two aborting transactions at the end of analysis (T1 and T2) along with their lastLSN (as the weight) into a priority queue. We take the highest LSN in the priority queue and process it. The order of processing is as follows:

- LSN 100 (T2): This is an abort record, we get the prevLSN and add this back to the priority queue (add LSN 90, T2 back into the priority queue).
- LSN 90 (T2): We can undo this record at LSN 90, giving us the CLR with LSN 130. We get the undoNextLSN using the prevLSN (LSN 50) of this update record at LSN 90. We get the prevLSN through the lastLSN of T2 in the transaction table, and update the lastLSN in the transaction accordingly. We then requeue this transaction back into the PQ with a weight of 50.
- LSN 70 (T1): This is a CLR, we don't do anything. We get the undoNextLSN (10) and requeue this transaction.
- LSN 50 (T2): We undo this record at LSN 50, with the prevLSN as 130 and undoNextLSN as 20. Requeue T2 with weight 20.
- LSN 20 (T2): We undo this record at LSN 20, with prevLSN as 140 and undoNextLSN as 0. At this point we are done with the rollback process so we write an end record with prevLSN as 150.
- LSN 10 (T1): We undo this record at LSN 10, with prev LSN as 70 and undoNextLSN as 0. At this point we are done with the rollback process so we write an end record with prevLSN as 170.

3 Octograder Queries (13 points)

There has been an increasing amount of interest in the Introduction to Database Systems course, and this semester, CS W186 expanded to a class size larger than ever. To accommodate the expected increase in volume of homework submissions, the TAs set up a new autograder, the Octograder, for the course homeworks. Octograder is accompanied by Octobot, which is a bot that manages homework submissions (and more).

Consider the following schema for Octograder:

```
CREATE TABLE edx_students (
    edx_id INTEGER PRIMARY KEY,
    name VARCHAR,
    email VARCHAR UNIQUE
);

CREATE TABLE assignments (
    assignment_name VARCHAR PRIMARY KEY,
    due_date TIMESTAMP
);

CREATE TABLE submissions (
    submission_id INTEGER PRIMARY KEY,
    edx_id INTEGER REFERENCES edx_students(edx_id),
    assignment_name VARCHAR REFERENCES assignments(assignment_name),
    submission_time TIMESTAMP,
    submission_file VARCHAR
);
```

As a reminder, students can submit assignments multiple times. Also, assume all tables only contain information about the current semester.

The TAs have been very busy this semester, so Octobot, being a highly intelligent and sentient bot, wants to create some SQL queries to help the TAs.

1. (3 points) To get an idea of students' progress on `hw5`, it is useful to know how many students have submitted the homework.

   Select the queries that find the number of students who submitted `hw5`.

   There may be zero, one, or more than one correct answers.

   A.
   ```
   SELECT COUNT(*)
   FROM submissions
   WHERE assignment_name = 'hw5';
   ```

   B.
   ```
   SELECT COUNT(edx_id)
   FROM submissions
   WHERE assignment_name = 'hw5';
   ```

C.
```
SELECT COUNT(DISTINCT edx_id)
FROM submissions
GROUP BY assignment_name
HAVING assignment_name = 'hw5';
```

> **Solution:** A and B are wrong because students can have multiple submissions. C is correct.

2. (3 points) Since students are allowed to use slip minutes on homeworks, Octobot wants to write a query that fetches late submissions.

   Select the queries that find the `submission_id` for all the `hw1` submissions that are late.

   There may be zero, one, or multiple correct answers.

A.
```
SELECT submission_id
FROM submissions S, assignments A
WHERE S.assignment_name = A.assignment_name
AND S.assignment_name = 'hw1'
AND S.submission_time > A.due_date;
```

B.
```
SELECT submission_id
FROM submissions S INNER JOIN assignments A
ON S.assignment_name = A.assignment_name
AND S.submission_time > A.due_date
WHERE S.assignment_name = 'hw1';
```

C.
```
SELECT submission_id
FROM (
    SELECT due_date
    FROM assignments
    WHERE assignment_name = 'hw1'
) AS A1, submissions S
WHERE S.submission_time > A1.due_date;
```

> **Solution:** C is wrong because it also includes all the submissions that are not hw1.

3. (3 points) Octobot wants to send an email to the students who didn't submit hw5 to ask why they didn't submit.

   Select the queries that find all the emails for all the students who didn't submit hw5.

   There may be zero, one, or multiple correct answers.

A.
```
SELECT email
FROM (
    SELECT edx_id
    FROM submissions
    WHERE assignment_name = 'hw5'
) AS S FULL OUTER JOIN edx_Students E
ON S.edx_id = E.edx_id
WHERE S.edx_id IS NULL;
```

B.
```
SELECT email
FROM edx_students E LEFT OUTER JOIN (
    SELECT edx_id
    FROM submissions S
    WHERE S.assignment_name = 'hw5'
) AS S1
ON E.edx_id = S1.edx_id
WHERE E.edx_id IS NULL;
```

C.
```
SELECT email
FROM edx_students
WHERE email NOT IN (
    SELECT email
    FROM edx_students E, submissions S
    WHERE E.edx_id = S.edx_id
    AND S.assignment_name = 'hw5');
```

Solution: B is wrong. E.edx_id is never null from the left outer join.

Octobot, being an ambitious bot, wants to do more than just querying the submissions database. He wants to help with grading the submissions! The first step is to find the `submission_id` of all the submissions to grade. Only the latest submissions for each assignment for every student will be graded.

In order to find all the `submission_id` of the latest submission for each assignment for every student, Octobot comes up with the following query:

```
-- Octobot's query
SELECT S.submission_id
FROM Submissions S INNER JOIN (
    SELECT edx_id, assignment_name, MAX(submission_time) as latest_sub_time
    FROM Submissions
    GROUP BY edx_id, assignment_name) AS S1
ON S.submission_time = S1.latest_sub_time
AND S.edx_id = S1.edx_id
AND S.assignment_name = S1.assignment_name;
```

4. (1.5 points) Does Octobot's query output the correct result?

   A. Yes, it outputs the `submission_id` of all the latest submissions for each assignment for every student.

   B. No, it might output too few results.

   C. No, it might output too many results.

   D. No, this query errors.

Octobot is not very confident about its query, so it asks its friend, Kiwibot. You might ask, how does a food delivery bot know about SQL? Well, Kiwibot has been wandering around the campus of the top institution in Computer Science for more than two years now, and it actually gained some knowledge of SQL by overhearing the conversations of EECS students outside Soda Hall!

Kiwibot comes up with the following query:

```
-- Kiwibot's query
SELECT S1.submission_id
FROM Submissions S1
WHERE S1.submission_time >= ANY (
    SELECT S2.submission_time
    FROM Submissions S2
    WHERE S1.edx_id = S2.edx_id
    AND S1.assignment_name = S2.assignment_name);
```

5. (1.5 points) Does Kiwibot's query output the correct result?

   A. Yes, it outputs the submission_id of all the latest submissions for each assignment for every student.

   B. No, it might output too few results.

   C. No, it might output too many results.

   D. No, this query errors.

   Solution: This query basically outputs every submission_id. So unless every student submits at most once for every assignment, this query would output too many results. It would be correct if we change ANY to ALL.

6. (1 point) Kiwibot's query contains a subquery (lines 4-7). Is it a correlated subquery?

   A. Yes.

   B. No.

7. (0.0001 points) How many octopuses are there in the "HW5 Grades Released" piazza post?

   Solution: 7

4 Disks, Fires, Buffalo (5 points)

1. (2 points) Fill in the corresponding box on the answer sheet for True or False.
   - A. For the current generation of flash storage (SSDs), reads are more costly than writes.
   - **B. In a heap file, some pages may not be filled to capacity.**
   - **C. Fragmentation is not an issue with packed variable-length record page format**
   - D. For the heap file implementations, the main advantage of linked lists over page directories is that inserting records is more efficient

Solution: B and C. There is no capacity requirement on heap files and packed page formats have no fragmentation issues. A is wrong because flash has fine-grain reads and coarse-grain writes. D is wrong because inserting records is more efficient with page directories rather than with linked lists.

2. (3 points) One day, the CS W186 TAs got together in a very productive meeting and made the decision to host weekly raffles for their students, with yet-to-be-determined mystery prizes. In this raffle, a new file is created every week. Throughout the week, every student who goes to office hours (assume there are many) gets entered into the file as a record containing their edX id and name. At the end of the week, the TAs draw 5 edX ids to be the winners.

   In this scenario, which choices (**one letter per each column**) would be appropriate and optimize the performance of the raffle execution?

Page Format	File Layout	Index
(a) Fixed Length (Packed)	(d) Sorted File	(f) Clustered Index on edX ids
(b) Fixed Length (Unpacked)	(e) Heap File	(g) Unclustered Index on edX ids
(c) Slotted Page		(h) No Index

Solution: The raffle execution involves many insert operations and relatively few lookup operations. Thus, we want to optimize insertions.

In terms of the records, edX ids and names are variable length, so we need (c) slotted pages.

Because we have few lookup operations, (e) heap file is appropriate for fast insertions. The lookup benefit from a sorted file is not worth the overhead for maintaining the order.

Building any index would add overhead to the insertion operations so (h) no index is appropriate.

5 More Query Pessimization (13 points)

For the following questions, assume the following:

- The System R assumptions about uniformity and independence from lecture hold.
- We use System R defaults when selectivity estimation is not possible.

Table Schema	Records	Pages	Notes
`CREATE TABLE Customers ( cid INTEGER PRIMARY KEY, name VARCHAR, address VARCHAR, email VARCHAR, );`	250	25	• Primary key `cid` is sequential in this table, starting from 1, and there are no gaps in the `cid`.
`CREATE TABLE Purchases ( pid INTEGER PRIMARY KEY, cid REFERENCES Customers(cid), item_id REFERENCES Items(iid), );`	10,000	1,000	None
`CREATE TABLE Items ( iid INTEGER PRIMARY KEY, price INTEGER, quantity INTEGER, );`	5000	500	• There is a clustered Alternative 3 index on `iid` of height 2. • `iid` ranges from 1 to 7500. • Prices for each item is unique. • Prices range from 1 to 10,000.

1. (1 point) What is the selectivity of `cid <= 100` from the `Customers` table?

   A. $\frac{1}{2}$

   B. $\frac{1}{4}$

   C. $\frac{1}{5}$

   [✓ D. $\frac{2}{5}$]

   E. $\frac{1}{10}$

Solution: We get the selectivity as $\frac{100-1+1}{250} = \frac{2}{5}$

2. (1 point) What is the selectivity `price = 10000 AND quantity < 10`?

   A. $\frac{1}{100000}$

   B. $\frac{1}{50000}$

   C. $\frac{1}{10000}$

   D. $\frac{1}{5000}$

   E. $\frac{1}{10}$

Solution: We know that the selectivity of `price = 10000` is $\frac{1}{5000}$ because prices are unique. We don't know the selectivity of `quantity < 10` so we can assume it is $\frac{1}{10}$. Therefore, our resulting selectitivy is $\frac{1}{5000} * \frac{1}{10} = \frac{1}{50000}$

3. (2 points) After applying the System R optimizer, which query has a lower estimated final cost for the optimal query plan.

   A. `SELECT * FROM Customers;`

   B. `SELECT * FROM Purchases WHERE pid > 1000;`

   C. `SELECT * FROM Items WHERE iid > 1500;`

Solution: Choice A would be a full table scan of 25 pages. Choice B would be a full table scan of 1,000 pages. Choice C would be an index scan on iid. We have to scan at least 400 pages because the selectivity of `iid > 1500` is $\frac{4}{5}$. Therefore, choice A has the lowest cost.

4. (5 points) Select all of the following statements that are true:

   A. If there is an eligible index exists, an index scan will always be selected over a sequential scan.

   B. We can always determine the resulting cardinality of a join.

   C. We should always push down selections involving a single table.

   D. We use the pass 1 table and previous pass (n-1) table to get the table for the nth pass.

   E. In our pass 1 table, we only keep one type of scan for each relation.

Solution: Choice A is false because an index scan can be worse than a sequential scan. Choice B is false because sometimes we can only estimate the cardinality. Choice C is true because we want to push down selections to eliminate rows early on to reduce our IO cost. Choice D is true because we try to join n-1 tables to a new table. Choice E is false because we can keep multiply scans for each relation if there are interesting orders.

Join	Left Relation	Left Ordering	Right Relations	Right Ordering	Join Type
(A)	C	none	P	none	BNLJ
(B)	I	iid	P	none	SMJ
(C)	C	none	I	iid	BNLJ
(D)	C	cid	P	cid	SMJ

5. (2 points) Which of the joins from the table above will **NOT** result in an interesting order for the next pass of the System R algorithm for the following query?

```
SELECT *
FROM Customers as C, Purchases as P, Items as I
WHERE C.cid = P.cid AND I.iid = P.item_id AND C.cid < 100
ORDER BY C.cid;
```

Solution: Choice A and C will both not result interesting orders because they are block nested loop joins. Choice B will not result in an interesting order because the `iid` column is not used in a later join or for an order by. Choice D does result in an interesting order because it can be used later on for the order by clause.

6. (2 points) Which of the following join orders will **NOT** be considered for Pass 3 in the System R algorithm for the query above?

A. $C \bowtie (P \bowtie I)$

B. $(C \bowtie I) \bowtie P$

C. $(I \bowtie P) \bowtie C$

D. $P \bowtie (C \bowtie I)$

E. $(C \bowtie P) \bowtie I$

F. $I \bowtie (P \bowtie C)$

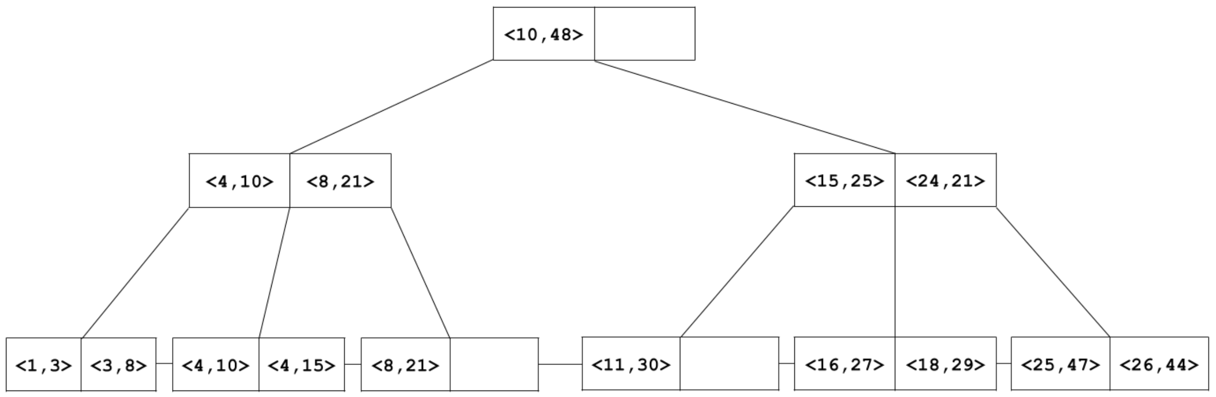
G. $(P \bowtie I) \bowtie C$

Solution: Choice A, D, and F are not left-deep joins so they will not be considered. Choice B has a cross join between C and I so it will not be considered.

6 B+ tree is no more. (16 points)

Kiwibot is ending its delivery service (for the semester, not for good) and plans to better balance workload across their fleet of robots for next semester so no robot malfunctions, shuts down, or catches on fire due to being overworked. They want to first know how many deliveries each robot made and how many miles each robot traveled this semester.

Luckily, the startup hired an intern who happened to take CS W186 and is an expert in tree indexes. The intern decides to build the following order $d = 1$ Alternative 3 unclustered B+ tree index on the composite key `<Deliveries, Miles>`. Assume B = 20.



1. (1 point) How many I/Os will it take to check if there exists a Kiwibot that made 3 deliveries this semester?

Solution: 3 I/Os. 3 to traverse from root to leaf.

2. (1 point) If the intern wants to access records of all Kiwibots that have made 8 deliveries and traveled 21 miles, how many I/Os would this take, assuming there are 5 such Kiwibots with all their references on the same page?

Solution: 8 I/Os. 3 to traverse from root to leaf, 5 to read records since the B+ tree is unclustered.

3. (2 points) One of the company's field testers found a lost Kiwibot in a bush but was able to repair it and retrieve its delivery and mileage data. The tester tells the intern to insert a new record into the B+ tree with `Deliveries = 13, Miles = 29`. How many I/Os will this take?

Solution: 6 I/Os. 3 to traverse from root to leaf, 2 to read in new data page and write record to it, 1 to write record to leaf.

4. (4 points) In the worst case, how many I/Os would it take to insert a new record into the B+ tree in the original diagram? Assume we have 20 buffer pages and we do **not** need to read in a new page before writing to it.

Solution: 10 I/Os. If we insert into the rightmost leaf node (involves largest possible number of splits): 3 to traverse from root to leaf, 2 to read in data page and write record to it, 1 to write leaf (changed due to split), 1 to write new leaf, 1 to write updated inner node (also changed due to split), 1 to write new inner node, 1 to write updated root (key pushed up from inner node).

5. (6 points) Which of the following statements about the intern's index design are true? **There may be zero, one, or more than one correct answer.**
   A. The B+ tree in the original diagram will always be able to handle 8 new record insertions without growing its height.
   B. We can lower the I/O cost of accesses over time by saving the root and all inner nodes in memory.
   C. The range `Deliveries < 20` will require at least a full scan.
   D. The range `Miles > 50` will require at least a full scan.
   E. `Deliveries < 8 && Miles > 10` is a lexicographic range.
   F. Next semester, Kiwibot implements better workload balancing and discovers a lot of robots have duplicate or very similar keys. Modifying the B+ tree to be Alternative 3 clustered will lower the I/O cost for some range scans.

Solution: Choice A is false because it is possible to insert fewer than 8 records to increase the tree's height. Choice B is true since traversal from root to leaf now only takes 1 I/O instead of 3 I/Os. Choice C is false because we can traverse the tree on `Deliveries` and then scan through the leaf nodes. Choice D is true because we cannot traverse the tree on `Miles.` Choice E is false because the range is not guaranteed to be continuous. Choice F is true being clustered helps with lexicographic range scans.

Great news! Kiwibot secured their Series A funding and expanded incredibly fast in Spring 2020. The company now has delivery service in over 100 college campuses across the country. Try to manage so many new robots, its CIO instructs the intern to build a new B+ tree that is now order $d = 4$.

6. (2 points) What is the maximum number of keys this new B+ tree can hold if its height is 3?

Solution: Maximum number of keys: $2d \cdot (2d + 1)^h = 8 \cdot 9^3 = 5832$

7 Zero Waste Sorting and Hashing (18 points)

It's almost 2020! That's bad news for Berkeley who is rushing to reach its goal of "Zero Waste by 2020." What's more is that they just found more waste in their external sorting and hashing algorithms! It's your job to find out how much is being wasted and what can be done to reduce the waste.

1. (5 points) First, we need to sort N = 1000 pages using B = 30 buffer frames. For each instance of creating runs or merging in a pass, find the number of frames that go unused then find the sum of unused frames across all passes. How many total frames will go unused in the process of sorting?

Solution: Pass 0: Use all 30 buffer frames to create sorted runs. Creates 33 runs of 30 pages and one run of 10 pages. When creating this last run, 20 buffer frames will be unused.

Pass 1: Use 29 buffer frames (and 1 frame set aside for output) to merge the runs together. First, merge 29 runs of 30 pages to create a run of 870 pages. All frames will be used in this step. Next, merge the last 4 runs of 30 pages and the 10 pg run into a run of 130 pages, which will leave 24 frames unused (don't forget about the output buffer!).

Pass 2: Merge the two large runs together, creating the final 1000 pg sorted run. This uses two buffer frames to hold the runs and another for the output buffer, leaving 27 frames unused.

In total, 71 frames will go unused over the course of sorting.

2. (1.5 points) Pertaining to the question above, select all of the following that could have an effect on reducing the number of unused frames. Assume we decrease the number of buffer frames to where we can still sort the data.

   A. Increasing the number of buffer frames

   B. Decreasing the number of buffer frames

   C. The values of data we're sorting

Solution: a,b. For a, if we increase the number of buffer frames to 1000, we don't have to external sort at all, and all frames can be used for an in-memory merge sort. For b, if we, for instance, decrease the number of buffer frames to 10, we can decrease the number of unused frames to 14. For c, external merge sort isn't vulnerable to key skew, so the values of the data have no effect on the number of unused frames.

3. (3 points) Let's now try to sort an N=120 page file. Let's assume that between pass 0 and pass 1, someone magically shrunk our buffer pool; in pass 0, we had B=10 buffer frames, but from pass 1 onwards, we had B=3. How many I/Os will it take to sort the file?

Solution: In pass 0, we generate $\lceil \frac{120}{10} \rceil = 12$ sorted runs of 10 pages each. From pass 1 onwards, we have B=3; using the formula for mergesort, we can see that we have $(1 + \lceil \log_2 12 \rceil) * 2 * 120 = 5 * 240 = 1200$ I/Os

4. (4 points) Now, we need to hash a N = 50-page file using B = 6 buffer frames. For the first partitioning phase, assume two partitions contain 18 pages worth of data evenly between them and all other partitions are uniformly partitioned. Assume uniform partitioning. How much total empty space will be written to disk when partitioning this data across all partitioning passes?

Solution: 16 pages worth of empty space. In the first partitioning pass, we'll set one buffer aside as our input buffer, giving us 5 partitions to split the data up. We're told that two partitions contain 18 pages worth of data between them, so that means 18 pages are taken care of for us already. We'll split the remaining 32 pages among the last three partitions evenly. We have to write entire pages to memory, so each of these partitions will write 11 pages worth of data, with each partition leaving 1/3 of a page blank. Across these three partitions, 1 page of empty space will be written to disk.

For the two partitions of 9 pages, each partition will write 9 full pages to disk, so there isn't empty space for the first partitioning pass. However, they're too large to make in-memory hash tables, so they'll each need to be recursively partitioned. In the second partitioning phase, consider one partition of 9 pages. 9 pages will be spread across 5 partitions, making each partition hold 1.8 pages worth of data and write 2 pages of data to disk per partition. Thus, in total, as we take 9 full pages in and write 10 pages out, 1 page worth of extra space will be written to disk for each of the 9-page partitions (giving us 2 pages worth of extra space written here).

For each of the three 11-page partitions, they will be spread across 5 partitions – each partition must thus write out 3 pages worth of data. Thus, at the end, we will have written out 45 pages worth of data to disk (15 per partition). Here, however, there is a caveat – initially, across the 3 partitions, there were 32 pages of data, and we have written 45 pages out; therefore, there is 13 pages worth of empty data written to disk for these partitions.

In total, our answer is 1 + 2 + 13 = 16 pages worth of empty pages.

In total, we'll waste 5 pages of space across our hashing process.

5. (1.5 points) Pertaining to the question above, select all of the following that could have an effect on reducing the amount of empty space on pages written to partitions. Assume uniform partitioning and that we only decrease the number of buffer frames to where we can still hash the data.

A. Increasing the number of buffer frames

B. Decreasing the number of buffer frames

C. The values of data we're hashing

Solution: a,b,c. For a, if we increase the number of buffer frames to 50, we can immediately form an in-memory hash table. For b, we can't decrease the number of buffer frames to 1 or 2, otherwise we wouldn't be able to hash. However, decreasing the number of buffer frames to 4, we waste 4 pages of space (with uniform partitioning). For c, if we did change the values of data we are hashing, we could introduce key skew and possibly change the division of values nicely. However, due to the typo that some exams saw (where it said sorting instead of hashing), everyone received credit for this answer choice.

6. (3 points) Let us assume that we have 2 machines, M_1 with B=5 and M_2 with B=10. Partitioning is done on the machine where B=5, and the partitions are then sent over to the B=10 machine for the conquer phase. How many I/Os will it take to hash an N=100 page file, assuming uniform partitioning and assuming that partitions are streamed directly to memory on M_2?

Solution: We partition until our partitions are <= 10 pages large. In the first partitioning pass, the file is split into 4 partitions of 25 pages each. This will be 100 I/Os to read in and 100 I/Os to write out. In pass 2, each partition of 25 pages is split into 4 partitions of 7 pages each, resulting in 16 partitions of 7 pages each in total. The cost here is 100 I/Os in and 112 I/Os out. They are then sent over to M_2, partition by partition (so 112 I/Os in to read it in to memory for the purpose of sending it to M_2), but as they are streamed directly into memory for M_2, there are only 112 I/Os on M_2. So the total I/O cost is 100 + 100 + 100 + 112 + 112 + 112 = 636.

8 Parallelograms (10.5 points)

We have 3 machines: m1, m2, and m3. The data for the `Teachers` table is range partitioned on the `age` column such that m1 has 40 pages of data, m2 has 15 pages, and m3 has 50 pages. The ranges for each machine are: m1 has values 1-25, m2 has values 26-30, and m3 has values 31-120. The data is not stored in sorted files on any machine. For questions 1-3, how many I/Os across all machines will it take to execute each query? Assume that each machine has 742 pages of memory.

1. (2 points) `SELECT MIN(age) FROM Teachers;`

   Solution: 40 I/Os - just need to ask the machine (m1) that can have the min.

2. (2 points) `SELECT COUNT(DISTINCT age) FROM Teachers;`

   Solution: 105 I/Os - need to look at every page

3. (1.5 points) Which queries, if any, would run faster if we round-robin partitioned the data instead of the range partitioning scheme described in the problem?

   A. Query 1

   B. Query 2

   C. None of them

   Solution: All of them. The first query takes 40ms because you have to wait for m1 to finish, and the second query takes 50ms because it needs to wait for m3. If you round robin partitioned the data, each machine would get 105 / 3 = 35 pages of data, so all 3 queries would take 35ms to execute.

We want to join the `Students` table with the `Classes` table using the join condition: `S.cid = C.id`. The `Students` table has 90,000 pages and the `Classes` table has 90 pages. We have three machines: m1, m2, and m3. Currently the `Students` table is round robin partitioned across the 3 machines and the `Classes` table is all on m1. Each page is 1KB.

4. (1 point) How many KB of data from the `Students` table will be sent across the network in a broadcast join?

   Solution: 0. In broadcast join the bigger table does not get repartitioned.

5. (1 point) How many KB of data from the `Classes` table will be sent across the network in a broadcast join?

Solution: 180KB. The smaller table must be sent to both machines that don't have it in broadcast join.

6. (1 point) How many KB of data from the `Classes` table will be sent across the network in a parallel hash join in the average case? Assume that we have perfect hash functions that divide up the data evenly.

Solution: 60KB. 2/3 of the pages need to move in the average case, so 90 * 2/3 = 60.

7. (1 point) How many KB of data from the `Students` table will be sent across the network in a parallel hash join in the average case? Again assume that the data is distributed uniformly and that we have hash functions that divide up the data completely evenly.

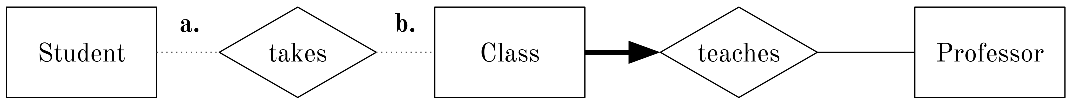
Solution: 60,000KB. 2/3 of the pages need to move again

8. (1 point) If disk access cost and CPU cost are negligible compared to the network cost, which join should we pick to optimize for performance?

   A. Broadcast Join

   B. Hash Join

9 Emergency Rooms & Fire Departments (21 points)



Complete the ER diagram to enforce that every student must be enrolled in at least one class and that classes can have any number of students.

1. (1 point) Fill in **a.** (between **Students** and **takes**):
   - A. Thin Line
   - **B. Bold Line**
   - C. Thin Arrow
   - D. Bold Arrow

   Solution: The students need to have at least one class, and thus a bold line is appropriate.

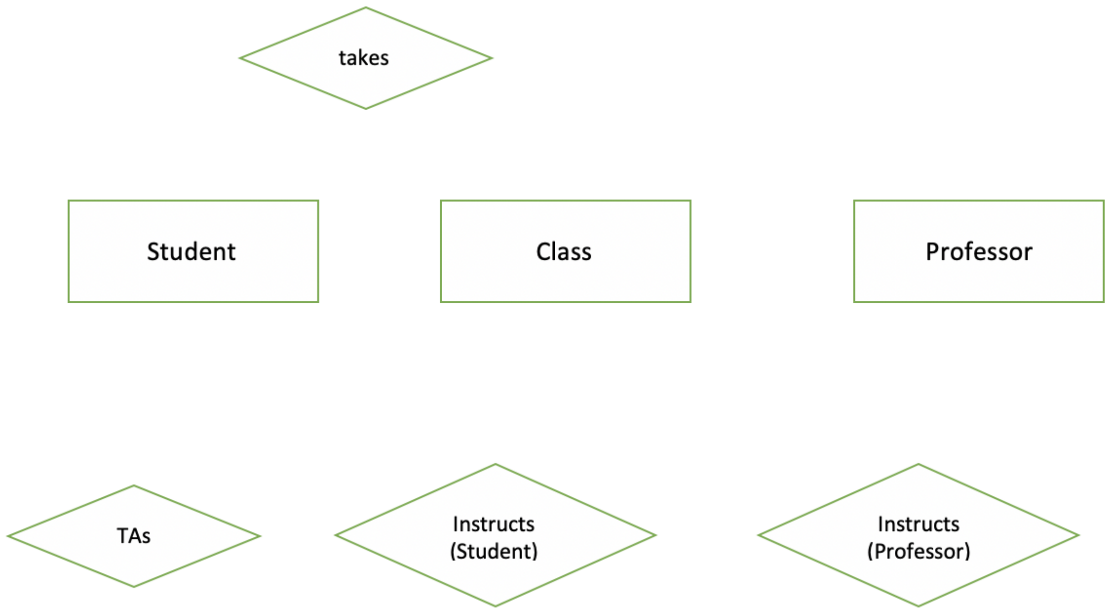
2. (1 point) Fill in **b.** (between **takes** and **Class**):
   - **A. Thin Line**
   - B. Bold Line
   - C. Thin Arrow
   - D. Bold Arrow

   Solution: There are no constraints on the classes.

3. (4 points) Which of the following statements are true? **There may be zero, one, or more than one correct answer.**
   - **A. Josh Hug, under the current design, can teach two classes.**
   - B. Weak Entities have a directly attached primary key
   - **C. Josh Hug and Joe Hellerstein cannot both teach CS W186.**
   - D. ER Diagrams are purely engineering and have no 'policy' implications.

   Solution: A single Professor can participate in multiple `teaches` relationships. Weak Entities have *partial* keys. Since a class can only exist in the binary `teaches` relationship, there can only be one professor. ER Diagrams are based heavily on 'policy'!

The existing database design does not allow for students to teach classes, and the department decides to change this to allow a qualified student to co-*instruct* a course with a professor.



Implement the following constraints:

- Students may instruct up to one class
- A course cannot be instructed by multiple students.
- A student cannot instruct a course on their own.
- Classes may have multiple TAs.
- Professors cannot opt out of instructing.
- Students have to take at least one course.
- Classes must have an enrollment > 0.

4. (1 point) What line goes between **Student** and **Instructs (Student)**?
   A. Thin Line
   B. Bold Line
   C. Thin Arrow
   D. Bold Arrow

Solution: Students can instruct at most one course.

5. (1 point) What line goes between **Class** and **Instructs (Student)**?
   - A. Thin Line
   - B. Bold Line
   - **C. Thin Arrow**
   - D. Bold Arrow

Solution: A course has at most one student instructing it; it cannot have more than one student instructor.

6. (1 point) What line goes between **Class** and **takes**?
   - A. Thin Line
   - **B. Bold Line**
   - C. Thin Arrow
   - D. Bold Arrow

Solution: A class has to have at least one student, but can have multiple.

7. (1 point) What line goes between **Student** and **takes**?
   - A. Thin Line
   - **B. Bold Line**
   - C. Thin Arrow
   - D. Bold Arrow

Solution: A student has to take at least one class, but can take multiple.

8. (1 point) What line goes between **Class** and **TAs**?
   - **A. Thin Line**
   - B. Bold Line
   - C. Thin Arrow
   - D. Bold Arrow

Solution: A class doesn't even have to have a TA.

9. (1 point) What line goes between **Student** and **TAs**?
   A. Thin Line
   B. Bold Line
   C. Thin Arrow
   D. Bold Arrow

Solution: A student doesn't have to be a TA. They could also TA for multiple courses if they so wish. Remember that there is no constraint listed for students and TAs, and in the absence of constraints, you cannot impose any in an ER diagram.

10. (1 point) What line goes between **Professor** and **Instructs (Professor)**?
   A. Thin Line
   B. Bold Line
   C. Thin Arrow
   D. Bold Arrow

Solution: A professor must instruct at least one course but could teach many.

11. (1 point) What line goes between **Class** and **Instructs (Professor)**?
   A. Thin Line
   B. Bold Line
   C. Thin Arrow
   D. Bold Arrow

Solution: A class must have at least one professor teaching it.

In the final part we will be using the following attribute set: $R = (SFLNGCRWH)$ = (S: SID, F: First Name, L: Last Name, N: Number of Units, G: GPA, C: Course (of teaching), R: Rating for class, W: Wage, H: Hours per week of work)

12. (3 points) Given the following FDs: $F = \{S \longrightarrow FL, S \longrightarrow NG, SC \longrightarrow R, CH \longrightarrow W, FL \longrightarrow R, C \longrightarrow H, H \longrightarrow C\}$, which of the following additional FDs (added to the set in isolation from the other answer choices) would reduce the size of the candidate key if it were the only one applied? **There may be zero, one, or more than one correct answer.**
   A. SID determines Rating
   B. Rating determines Wage
   C. Rating determines Course
   D. Rating and First Name determine Wage and Hours of Work

Solution: SID already determines Rating because $S \rightarrow FL$ and $FL \rightarrow R$.

$R \rightarrow W$ does not help because Ratings and Wage do not appear o n the left side of any existing FD.

The current candidate key is SC, adding $R \rightarrow C$ means that $S \rightarrow C$, which reduces the candidate key to S

Similarly, $RF \rightarrow WH$ means that $S \rightarrow H$ (and $H \rightarrow C$ is given) which reduces the candidate key to S again!

13. (4 points) What is the BCNF decomposition of the following FDs $F = \{S \longrightarrow FL, FL \longrightarrow NG, S \longrightarrow C, SC \longrightarrow RH, CH \longrightarrow W\}$? Please check the appropriate boxes to indicate the attribute and its table in BCNF; you might not use all 4 columns on the answer sheet. Please reserve R1 for the largest table (in terms of number of attributes), R2 for the second largest, and so forth.

Solution: The decomposition leads to $CFHLRS, FGLN, CHW$

10 Congo Web Services (19 points)

1. (4 points) Which of the following statements are true? **There may be zero, one, or more than one correct answer.**

   A. There exists a way to check in polynomial time if any schedule is view serializable.

   B. Starting from a node on a dependency graph with only outgoing edges and running Depth First Search will always return a conflict equivalent serial schedule.

   C. Two Phase Locking prevents cascading aborts, but does not guarantee conflict serializability.

   D. According to HW4, you cannot substitute a S lock for an IS lock (replace IS with a S).

Solution: Choice A is false because checking for view serializable is NP-Complete.
Choice B is false because you should run topological sort, not DFS. Consider a case where two nodes have only outgoing edges. DFS will never get to the second node.
Choice C is false because two phase locking guarantees conflict serializability, but does not prevent cascading aborts.
Choice D is true because this prevents redundancy.

Consider the following schedule for the next two questions:

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

2. (2 points) What is the set of edges of the dependency graph for the schedule? We write (T_i, T_j) if there exists a directed edge from T_i to T_j.

   A. $\{(T_1, T_4), (T_3, T_2), (T_4, T_2), (T_1, T_3)\}$

   B. $\{(T_2, T_4), (T_3, T_4), (T_4, T_1), (T_2, T_3), (T_1, T_4), (T_3, T_1)\}$

   C. $\{(T_4, T_2), (T_4, T_3), (T_1, T_4), (T_3, T_2), (T_4, T_1), (T_1, T_3)\}$

   D. None of the above

Solution: B. $\{(T_2, T_4), (T_3, T_4), (T_4, T_1), (T_2, T_3), (T_1, T_4), (T_3, T_1)\}$

3. (1 point) True or False: This schedule is conflict serializable.

Solution: False. There is a cycle between T_1 and T_4

After graduating from UC Berkeley, the number one public university, you start working at Congo Inc. with an extremely lucrative career managing databases for their Congo Web Services (CWS). The company tasks you with solving their concurrency problems. Good thing you took CS W186!

For the next two questions, consider a database X with a table Y. Y has pages A, B, and C. Page A has tuples A_1, ..., A_{10}, page B has tuples B_1, ..., B_{10}, and page C has tuples C_1, ..., C_{10}.

Your manager has given you the following schedule and tells you that there is currently too little concurrency because each transaction locks the entire database for every update. Having taken CS W186, you suggest using multigranularity locking (**with minimum privilege**)!

Transaction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
T1	R(A_1)				R(B)	W(B_2)		W(A_2)
T2		R(B_1)						
T3			R(C_1)					
T4				W(A_2)			W(B)	

4. (3 points) Using the new multigranularity scheme, list all the locks that T1 has after timestep 8 in the order of acquisition with the first lock acquired being on the top left and most recent lock being on the bottom right (following all time of acquisition rules from the homework). You may or may not need all the boxes. Leave unused boxes blank.

Lock 1	Lock 2	Lock 3	Lock 4
Lock 5	Lock 6	Lock 7	Lock 8

Solution: IX(X), IX(Y), IX(A), S(A_1), SIX(B), X(B_2)

After timestep 1, T1 will have acquired IS(X), IS(Y), IS(A), and S(A_1). After timestep 5, T1 will have acquired S(B). At timestep 6, T1 will change the S(B) into a SIX(B) (keeping the time of acquisition the same like on the homework) and acquire X(B_2). Finally, at timestep 8 T1 will successfully promote IS(X), IS(Y), IS(A) to IX locks, but will get blocked and placed on the wait queue for lock X(A_2)

5. (1 point) True or False: This schedule has deadlock.

Solution: True. T4 is waiting on T1 for page B at timestep 7, and at timestep 8 T1 waits on T4 for tuple A_2.

After solving the low concurrency issue, your manager praises your knowledge of databases and hands you a snippet of another schedule:

Transaction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
T1					W(A)			W(C)		
T2	R(A)					R(C)			W(A)	
T3			W(B)	W(A)						
T4		R(B)					R(C)			W(C)

Your manager tells you that the transactions seem to be `stuck`! You do some investigating and find out that the schedule causes the transactions to be in deadlock. Again, using your CS W186 knowledge, you decide to implement deadlock avoidance!

For the following four questions, assume that T_1 started first, T_2 started second, T_3 started third, and T_4 started last. If a transaction gets aborted, it will immediately release all locks it has acquired and the wait queue will be processed. Additionally, if a transaction gets placed on a waiting queue (blocked), locks that the transaction acquired up to that point are not released and the actions for that transaction **while it is blocked** do not happen (as if the action was never on the schedule). Additionally, assume promotes are implemented as described on homework 4.

6. (2 points) If you use the **wound-wait** approach, which transaction(s) will end up getting aborted? **Select all correct choices. If no transaction gets aborted, select None**

    A. T_1

    B. T_2

    C. T_3

    D. T_4

    E. None

    Solution: B, C, D. At timestep 3, T_3 causes T_4 to be aborted since it has higher priority (as described in lecture, priority is now - start time). T_3 ends up blocked at timestep 4 by T_2 for resource A. At timestep 5, T_1 causes T_2 to abort, which processes the queue and lets T_3 acquire the lock on resource A, but then is also aborted by T_1 since T_1 has higher priority.

7. (2 points) Which transaction(s) will end up blocked **by the end of timestep 10** if using wound-wait? **Select all correct choices. If no transaction gets blocked, select None**

    A. T_1

    B. T_2

    C. T_3

    D. T_4

    E. None

    Solution: E. The only transaction that gets blocked is T_3, but it gets processed and then aborted later. By the end of the schedule, no transaction is blocked.

8. (2 points) If you use the **wait-die** approach, which transaction(s) will end up getting aborted? **Select all correct choices. If no transaction gets aborted, select None**

    A. T_1

    B. T_2

    C. T_3

    D. T_4

    E. None

Solution: D. The very last W(C) will cause T_4 to get aborted since T_2 has higher priority and has a conflicting lock.

9. (2 points) Which transaction(s) will end up blocked **by the end of timestep 10** if using wait-die? **Select all correct choices. If no transaction gets blocked, select None**

   A. T_1

   B. T_2

   C. T_3

   D. T_4

   E. None

Solution: A. T_3 will be blocked at timestep 3 waiting for T_4 for resource B. T_1 will be blocked at timestep 5 waiting for T_2 for resource A. However, T_4 gets aborted at timestep 10 and releases its lock on B, allowing T_3 to be processed and unblocked.

11 Distributed Fires (15 points)

1. (1 point) Two-phase commit allows a distributed database to commit transactions as long as a majority of the servers are alive.

    A. True.

    B. False.

> **Solution:** False. Every server, not just the majority, needs to be alive to make progress (2PC needs unanimity).

2. (1 point) In a distributed database, if there is no lock dependency cycle at any individual node, then there cannot be a deadlock.

    A. True.

    B. False.

> **Solution:** False. There is a deadlock if there is a cycle in the union of lock dependencies across all nodes, not just at any indiviual node.

3. (1 point) In two-phase commit with two-phase locking, when do participants release their locks?

    A. After logging prepare.

    B. After logging commit.

> **Solution:** After logging commit. The transaction is not complete for the participant until they log a commit record.

The owner of a chain of boba shops was never any good at databases, and following an embarrassing incident where they couldn't figure out how to find their most popular drink, they've hired a Berkeley student (that's you!) to run their now-massive parallel distributed database.

You're talking to the owner one day, trying to explain the **two-phase commit protocol with presumed abort** that you just built for the database. The owner doesn't seem to understand why it works, though. They wonder **if the protocol would still work if various parts of the protocol were changed**.

For each of the following modifications, select all the problems that it might cause. **There might be zero, one, or multiple correct answers.**

4. (1 point) In phase 1, participants log prepare records *after* sending out the vote, instead of before.

    A. Transactions that commit might have operations that do not get committed.

    B. Transactions that abort might have operations that do not get aborted.

Solution: If a participant crashes after voting yes but before logging prepare, then it will abort the transaction upon recovery, whereas the coordinator may commit the transaction.

5. (1 point) In phase 1, participants log abort records *after* sending out the vote, instead of before.
   A. Transactions that commit might have operations that do not get committed.
   B. Transactions that abort might have operations that do not get aborted.

Solution: The abort record in presumed abort does not have to be flushed immediately, so it does not have to be logged immediately either.

6. (1 point) The coordinator does not flush the commit records that it writes.
   A. Transactions that commit might have operations that do not get committed.
   B. Transactions that abort might have operations that do not get aborted.

Solution: If the coordinator crashes before flushing the commit record, upon recovery, it may think the transaction has aborted, even though it told the user it committed. This would cause any inquiring participants to abort their operations instead of committing.

7. (1 point) In phase 2, the participant sends out acks before logging commit.
   A. Transactions that commit might have operations that do not get committed.
   B. Transactions that abort might have operations that do not get aborted.

Solution: If the participant crashes before logging commit, the coordinator may still receive all the acks sent, and then forget about the transaction. When the participant recovers, presumed abort will tell the participant to abort the transaction.

8. (1 point) In phase 1, a participant logs prepare, but accidentally votes `no` instead of `yes`.
   A. Transactions that commit might have operations that do not get committed.
   B. Transactions that abort might have operations that do not get aborted.

Solution: The transaction will abort, but neither of the two errors above will happen.

9. (1 point) In phase 1, the coordinator sends out prepare requests to the participants. Before hearing back, however, the coordinator gets cold feet and decides to abort the transaction. It immediately logs and flushes an abort record, and tells the user the transaction aborted.
   A. Transactions that commit might have operations that do not get committed.
   B. Transactions that abort might have operations that do not get aborted.

Solution: The transaction will abort, but neither of the two errors above will happen.

10. (1 point) A participant, after logging a prepare record, is too impatient to wait to hear the commit decision. It goes behind the coordinator's back and asks all the other participants what they voted. If they all voted yes, the participant logs a commit record and commits the transaction.

    A. Transactions that commit might have operations that do not get committed.

    B. Transactions that abort might have operations that do not get aborted.

Solution: If the coordinator crashes, presumed abort will cause the coordinator to abort the transaction upon recovery, even if all participants voted yes.

11. (1 point) A participant, after logging a prepare record, notices that the coordinator has crashed, so it aborts the transaction.

    A. Transactions that commit might have operations that do not get committed.

    B. Transactions that abort might have operations that do not get aborted.

Solution: The coordinator might have crashed after it logged a commit record, in which case the transaction has to commit during recovery.

12. (1 point) The coordinator, after logging a commit record, forgets to send phase 2 commit messages.

    A. Transactions that commit might have operations that do not get committed.

    B. Transactions that abort might have operations that do not get aborted.

Solution: The participants will eventually inquire about the status of the transaction, and the coordinator will tell them to commit.

13. (1 point) The coordinator is paranoid and instead of logging commit after phase 1, it waits for phase 2 to complete (receiving ACKS from all participants) before committing the transaction.

    A. Transactions that commit might have operations that do not get committed.

    B. Transactions that abort might have operations that do not get aborted.

Solution: This is as bad as the coordinator not flushing commit records (see above).

14. (1 point) After the coordinator logs a commit record, it logs an end record right away and forgets about the transaction.

    A. Transactions that commit might have operations that do not get committed.

    B. Transactions that abort might have operations that do not get aborted.

Solution: If a participant fails during phase 2 and the coordinator has already forgotten about the transaction, the participant will incorrectly abort the transaction.

15. (1 point) Why bother waiting for prepares? In phase 1, as soon as the coordinator receives a single YES vote, it logs a commit record right away and commits the transaction.

    A. Transactions that commit might have operations that do not get committed.

    B. Transactions that abort might have operations that do not get aborted.

Solution: Another participant might still be unable to complete the transaction.