CS 186
Fall 2020

# Introduction to Database Systems

Final Exam

**INSTRUCTIONS**

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

○ You must choose either this option

○ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

☐ You could select this choice.

☐ You could select this one too!

**You may start your exam now. Your exam is due at <DEADLINE> Pacific Time.** Go to the next page to begin.

**Preliminaries**

# 1 CS 186 Fall 2020 Final Exam (Online)

`Do not share this exam until solutions are released.`

#### 1.0.1 Contents:

- The final exam has 8 questions, each with multiple parts, and worth a total of 186 points.

#### 1.0.2 Taking the exam:

- You have 170 minutes to complete the midterm.
- You may print this exam to work on it.
- For each question, submit only your final answer on examtool.
- For numerical answers, do not input any suffixes (i.e. if your answer is 5 I/Os, only input 5 and not 5 I/Os or 5 IOs)
- Make sure to save your answers in examtool at the end of the exam, although the website should autosave your answers as well.

#### 1.0.3 Aids:

- You may use three pages (double sided) of handwritten notes as well as a calculator.
- You must work individually on this exam.

#### 1.0.4 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 half point penalty.

**(a)** What is your full name?

**(b)** What is your student ID number?

**(c)** What was your favorite tiktok / video shown in class?

**(d)** Out of our 27 lectures, how many did you attend live? Answer will not affect exam score in any way : )

1. **(16 points) Final Potpourri**

Choose True or False for each of the statements below.

**(a)** **(1 pt)** Running write-heavy query workloads on replicated databases is more efficient than running on key-based partitioned ones.

○ True

[✓ False]

We need to write to all replicas in a replicated system, while only need to write to one of the partitions in a partitioned system. Hence the latter is more efficient.

**(b)** **(1 pt)** Objects in the same array can have different number of key-value pairs in them in a well-structured JSON document.

[✓ True]

○ False

JSON does not require all objects to have the same schema.

**(c)** **(1 pt)** In broadcast join, we replicate the larger relation to all servers while partition the smaller relation across the servers.

○ True

[✓ False]

We replicate the smaller relation to all servers in broadcast join.

**(d)** **(1 pt)** For a query with a single join, the Selinger optimizer will always return the most efficient execution plan based on the provided cost estimates.

[✓ True]

○ False

This is true as the optimizer would enumerate all possible ways to perform the single join.

**(e)** **(1 pt)** In Hadoop, all intermediates results generated by mappers are stored on the disk.

[✓ True]

○ False

Storing intermediates is one of the defining features of MapReduce.

**(f)** **(1 pt)** Calling `filter` on an RDD will run the selection operation immediately to completion when invoked.

○ True

[✓ False]

`filter` is a transformation, not an action.

**(g) (1 pt)** All data stored in a RDD must be in the form of a (key,value) pair.

○ True

● False

An RDD can store any object of the same type that doesn't necessarily only be a (key,value) pair.

**(h) (1 pt)** You can express natural joins in relational algebra using just cross-products, selections, and projections.

○ True

● False

You need renames in addition to the above.

**(i) (1 pt)** Fixed length attributes with null values can be treated identically to variable length attributes when determining record layout in a slotted page.

● True

○ False

Yes, because null values lead to fixed length attributes becoming variable length.

**(j) (1 pt)** An LRU buffer replacement strategy is better than MRU when supporting multiple sequential scans.

○ True

● False

LRU causes sequentual flooding.

**(k) (1 pt)** To keep a B+tree clustered, periodic modifications to the data structure is necessary even in a read-only workload.

○ True

● False

Not necessary since there are no inserts.

**(l) (1 pt)** In OLAP, the dimension attributes are usually the GROUP BY attributes.

● True

○ False

Dimension attributes are GROUP BY; measure attributes are aggregated.

**(m) (1 pt)** The record ID for a tuple can change for sorted files regardless of the page layout.

● True

○ False

In sorted files, records can be moved to a different page or different location on a page in order to maintain the sort. This would change the record ID since record ID consists of both the page number and the location of the record on the page.

**(n)** **(1 pt)** Using the record header to store only pointers to the start of each variable length field is sufficient for variable length records.

○ True

[✓ False]

When storing pointers to the start of each variable length field, it is not possible to know where the last variable length field ends without also storing the length.

**(o)** **(1 pt)** Inserting to a sorted file costs the same IOs as inserting to a heap file in the best case.

[✓ True]

○ False

In the best case, the record can be inserted into the first page read in both files.

**(p)** **(1 pt)** Inserting records one by one sorted by the index column into a B+ tree achieves the same result as bulkloading with a fill factor of 1/2.

○ True

[✓ False]

The splitting procedure and time is different for these two procedures which will result in different final trees structures.

**2. (28 points) Query Bop-timization**

For this question, let's look at tables R, S, and T with the following properties. Assume all of the column distributions are independent of each other.

| Schema | Known Table Stats |
|---|---|
| CREATE TABLE **R**(a INTEGER, b FLOAT, c INTEGER) | **R.c:** min = 1, max = 40, 20 unique values |
| CREATE TABLE **S**(a INTEGER, b FLOAT, c BOOLEAN NOT NULL) | None Known |
| CREATE TABLE **T**(a INTEGER, b FLOAT, c INTEGER) | **T.a:** 3 unique values with following distribution: {1: 50%, 2: 25%, 3: 25%} |

**(a) (4 points) Selectivity Estimation**

Express the selectivity for each of the following queries as a **simplified fraction**. Do not input spaces in your answer (eg. 2/3).

**i. (1 pt)** `SELECT * FROM S WHERE S.c = TRUE`

**1/2**

- We assume a uniform distribution so 1/2 of values should be True and 1/2 values should be False.

**ii. (1 pt)** `SELECT * FROM T WHERE T.a <= 2`

**3/4**

We are given the distribution of T.c so we can use that distribution to estimate the selectivity instead of assuming a uniform distribution.

**iii. (2 pt)** `SELECT * FROM R WHERE R.a <= 20 OR R.c=15` You may assume that `15` is one of the values that R.c can take.

**29/200**

We assume that all of the columns are independent of each other, so: Sel(R.a <= 20 or R.c=15) = Sel(R.a <= 20 ) + Sel(R.c=15) - Sel(R.a <= 20 ) * Sel(R.c=15)

We have no info on R.a so Sel(R.a <= 20) = 1/10. Sel(R.c=15) is 1 / (# unique values for R.c) = 1/20.

Total selectivity: 1/10 + 1/20 - 1/10*(1/20) = 29/200

**(b) (10 points) System R Query Optimizer - Pass 1**

For the next sections, we will optimize the following query using the System R (aka Selinger) query optimizer.

```
SELECT R.a, S.b, T.c
    FROM R INNER JOIN S ON R.a = S.a
        INNER JOIN T ON R.b = T.b
    WHERE R.c <= 20 AND T.c <= 20
    GROUP BY S.b
```

**i. (2 pt)** How many leaf nodes are there in a full, height 1, alternative 3, clustered index on R.c? Reminder that a full tree is one in which any additional unique key insertions would cause a change in height. Also, from previous info in this question, R.c has 20 unique values.

Hint: Try some possible fanouts/orders.

**5**

A fanout of 5 and order of 2 allows for 5 full leaf nodes to store 20 unique values and for the inner node to be full with 4 values.

**ii. (3 pt)** Let's assume that T.c has a height 1, alternative 3, clustered index. The index has 100 leaf nodes; each node contains 10 values and fits on exactly one page. Every value of T.c matches with 2 data pages worth of records. If we are given that sel(T.c<=20) = 1/2, what is the IO cost of performing an index scan on T with this index?

**1051**

It takes 1 I/O to read the inner node. Due to the selectivity of 1/2, we only scan through 50 of the 100 leaf pages which costs 50 I/Os. Because it is an alternative 3, clustered index, we use 1 I/O for every page of matches. Therefore, since each node has 10 values and each value has 2 pages of matches, we have a total I/O cost of: $1 + 50 + 50 * 10 * 2 = 1051$

**iii. (1 pt)** Assume we have the following query and I/O costs for accessing single tables:

```
SELECT R.a, S.b, T.c
    FROM R INNER JOIN S ON R.a = S.a
        INNER JOIN T ON R.b = T.b
    WHERE R.c <= 20 AND T.c <= 20
    GROUP BY S.b
```

| Option | Access Plan | I/O Cost |
|---|---|---|
| a | R: Full scan | A I/Os |
| b | R: Index scan on R.b | B I/Os |
| c | R: Index scan on R.c | C I/Os |
| d | S: Full scan | D I/Os |
| e | T: Full scan | E I/Os |
| f | T: Index scan on T.b | F I/Os |
| g | T: Index scan on T.c | G I/Os |

Give a bound on A that guarantees that option a **will not** be kept after Pass 1 of the optimizer.

○ B<A

○ C<A

● Min(B,C)<A

○ Max(B,C)<A

○ None of these options

Option a does not produce interesting orders so it will not be kept if it is not the minimum access plan for table R.

Update: Poor wording of question, so all answers received credit.

**iv. (1 pt)** Give a bound on B that guarantees that option b **will not** be kept after Pass 1 of the optimizer.

○ A<B

○ C<B

○ Min(A,C)<B

○ Max(A,C)<B

● None of these options

Option B produces an interesting order used in a join so it will be kept even if it has a higher I/O cost than both of the other plans.

**v. (1 pt)** Give a bound on C that guarantees that option c **will not** be kept after Pass 1 of the optimizer.

○ A<C

○ B<C

● Min(A,B)<C

○ Max(A,B)<C

○ None of these options

The selection should be pushed down, so option c does not produce any interesting orders. It will be kept only if it is the minimum access plan for table R.

Update: Poor wording of question, so all answers received credit.

**vi. (1 pt)** Here is the query and table copied for convenience.

```
SELECT S.b
    FROM R INNER JOIN S ON R.a = S.a
        INNER JOIN T ON R.b = T.b
    WHERE R.c <= 20 AND T.c <= 20
    GROUP BY S.b
```

| Option | Access Plan | I/O Cost |
|---|---|---|
| a | R: Full scan | A I/Os |
| b | R: Index scan on R.b | B I/Os |
| c | R: Index scan on R.c | C I/Os |
| d | S: Full scan | D I/Os |
| e | T: Full scan | E I/Os |
| f | T: Index scan on T.b | F I/Os |
| g | T: Index scan on T.c | G I/Os |

Give a bound on E that guarantees that option e is the **only** access plan for table T kept after Pass 1 of the optimizer.

○ E<F

○ E<G

○ E<Min(F,G)

○ E<Max(F,G)

[✓ None of these options]

Option F will always be kept since it is an interesting order.

**vii.** **(1 pt)** Give a bound on F that guarantees that option f is the **only** access plan for table T kept after Pass 1 of the optimizer.

○ F<E

○ F<G

[✓ F<Min(E,G)]

○ F<Max(E,G)

○ None of these options

If option F is the cheapest access plan, it will be the only plan kept. Option G does not provide interesting orders since the selection should be pushed down.

**(c) (14 points) System R Query Optimizer - Pass 2/3**

```
SELECT S.b
    FROM R INNER JOIN S ON R.a = S.a
        INNER JOIN T ON R.b = T.b
    WHERE R.c <= 20 AND T.c <= 20
    GROUP BY S.b
```

Let's now look at passes 2 and 3 of the System R Query Optimizer. Assume that:

- We have **B** = **5** buffer pages.
- Before applying selectivity, R has 200 pages and S has 15 pages. You should not assume these tables are ordered in any way.
- We know sel(R.c <= 20) is 1/10
- R and S have no duplicates and have no indices.

We wish to calculate the cost of joining R and S with Sort Merge Join.

**i. (3 pt)** What is the estimated I/O cost of fully sorting R?

**260**

We know that after applying selectivity we will have 20 pages of R. We will use this value to compute how many passes we need.

$$passes = 1 + \lceil \log_{B-1} \lceil R/B \rceil \rceil$$

$$passes = 1 + \lceil \log_{5-1} \lceil 20/5 \rceil \rceil = 2$$

To compute the total cost, because we have applied the selectivity pushdown, we cannot use our typical formula of

$$Sort(X) = 2 * passes * [R]$$

Instead, we will need to manually consider how many IOs are needed. In Pass 0 (the sorting pass), all 200 pages will be read in, but only 20 pages will be written out into 4 runs of length 5 due to the selectivity. In Pass 1 (the runs merging pass), the 4 runs of length 5 will be read in and 1 run of length 20 will be written out. Thus the total cost is 200+20+20+20 = 260.

**ii. (3 pt)** What is the estimated I/O cost of fully sorting S?

**60**

$$passes = 1 + \lceil \log_{B-1} \lceil S/B \rceil \rceil = 2$$

$$Sort(X) = 2 * passes * [S] = 60$$

**iii. (3 pt)** What is the **total** estimated I/O cost of R SMJ S? Please make sure you provide the total cost of SMJ and remember to apply any possible optimizations.

**325**

SMJ = Sort(R) + Sort(S) + [R] + [S] - Optimizations.

After the first pass of sorting, R has 20/5=4 runs and S has 15/5=3 runs. The SMJ optimization where S's runs are not merged can be applied here saving 2*[S] IOs.

Therefore the total IO cost is: SMJ = Sort(R) + Sort(S) + [R] + [S] - 2[S] SMJ = 260+60+20+15-30 = 325 IOs

Partial credit was awarded to answers of 355 that did not use the optimization.

**iv. (1 pt)**

```
SELECT R.a, S.b, T.c
    FROM R INNER JOIN S ON R.a = S.a
        INNER JOIN T ON R.b = T.b
    WHERE R.c <= 20 AND T.c <= 20
    GROUP BY S.b
```

If we use R SMJ S for the query above, which columns will the join output be sorted on? Mark all that apply.

■ R.a

☐ R.b

☐ R.c

■ S.a

☐ S.b

☐ S.c

☐ T.a

☐ T.b

☐ T.c

☐ None of these options

The output of R SMJ T will be sorted on the "a" columns of R and S.

| Option | Access Plan | Sorted on | I/O Cost |
|---|---|---|---|

**v. (1 pt)**

```
SELECT R.a, S.b, T.c
    FROM R INNER JOIN S ON R.a = S.a
        INNER JOIN T ON R.b = T.b
    WHERE R.c <= 20 AND T.c <= 20
    GROUP BY S.b
```

Assume that we have the following I/O costs for joining 2 tables together:

| Option | Access Plan | Sorted on | I/O Cost |
|---|---|---|---|
| a | $R \bowtie S$ | N/A | A I/Os |
| b | $S \bowtie R$ | S.b | B I/Os |
| c | $S \bowtie T$ | N/A | C I/Os |
| d | $T \bowtie S$ | S.b | D I/Os |
| e | $R \bowtie T$ | N/A | E I/Os |
| f | $R \bowtie T$ | R.b | F I/Os |
| g | $T \bowtie R$ | T.c | G I/Os |

Give a bound that guarantees that option a **will not** be kept after pass 2 of the optimizer.

- [✓ B<A]
- Min(B,C,D,E,F,G)<A
- Max(B,C,D,E,F,G)<A
- None of these options

Option a is only compared against option b since those two plans join R and S. Option a does not produce any interesting orders so it will only be kept if it is the cheapest plan.

Update: Poor wording of question, so all answers received credit.

**vi. (1 pt)** Give a bound that guarantees that option b **will not** be kept after pass 2 of the optimizer.

- A<B
- Min(A,C,D,E,F,G)<B
- Max(A,C,D,E,F,G)<B
- [✓ None of these options]

Option a is only compared against option b since those two plans join R and S. Option b produces an interesting order as S.b is used in the downstream GROUP BY clause so it is always kept.

**vii. (1 pt)** Give a bound that guarantees that option d will be the **only** plan for joining S and T kept after pass 2 of the optimizer.

- D<C
- D<Min(A,C,D,E,F,G)
- D<Max(A,C,D,E,F,G)
- [✓ None of the above]

Option d is a cross join which is not considered by the optimizer.

**viii. (1 pt)** The Selinger/System R optimizer only considers Left Deep plans in order to reduce the query plan search space and improve the algorithm runtime.

[✓ True]

○ False

**3. (23 points) It's all transactional to me**

**(a) (5 points) On Schedule**

Assume we have the following schedule with four transactions acting on resources A, B, and C. All transactions commit some time after timestamp 8.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| T1 | | | | W(B) | | | | |
| T2 | | | W(A) | | | | R(B) | R(A) |
| T3 | R(A) | | | | R(C) | | | |
| T4 | | W(C) | | | | W(B) | | |

**i. (2 pt)** Which transactions point to T2 (if any) in the conflict serializability graph?

[✓ T1]

T2

[✓ T3]

[✓ T4]

None of the above

The graph has edges from T4, T3, and T1 to T2. It also has edges from T1 to T4 and T4 to T3.

**ii. (1 pt)** Is this schedule conflict serializable?

[✓ Yes]

No

The dependency graph does not contain a cycle.

**iii. (2 pt)** How many serial schedules are conflict-equivalent to this schedule?

**1**

Only T1->T4->T3->T2 is conflict-equivalent.

**(b) (10 points) Ulysses S. Granted Set**

The State of Wisconsin is using a database to store information about their recount for the presidential election. The database consists of 2 tables: Voters and Ballots. Voters consists of 2 pages labeled A and B while Ballots consists of 2 pages labeled C and D. Our database uses a multigranular locking system on these resources. Below is a table consisting of resource name, granted set, and waitlist. There are currently 4 transactions running. Waitlist requests are processed from left to right with the leftmost request to be granted next.

| Resource | Granted | Waitlist |
|---|---|---|
| Database | T1: IX, T2: IX, T3: IX, T4: IS | |
| Voters | T1: IX, T3: IS | T2: SIX |
| Ballots | T2: SIX, T3: IS, T4: IS | T1: S, T3: SIX |
| A | T1: S | |
| B | T1: X | |
| C | T2: X | |
| D | T3: S | |

**i. (4 pt)** Suppose T4 requests an S lock on pages A, B, C, and D. ***Assuming no other lock acquisitions have occurred besides those listed in the table (i.e. there are no intermediate IS or IX requests)***, which pages will immediately grant T4 the S lock?

☐ A

☐ B

☐ C

[✓ D]

☐ None of the above

T4 doesn't hold an IS lock on Voters so it won't be able to acquire an S lock on A and B. T4 will be placed in the waitlist for Page C since T2 already holds an X lock. T4 will be granted the S lock on D since the granted set consists entirely of S locks.

**ii. (2 pt)** Identify all transactions involved in deadlock (if any) for the lock table above.

[✓ T1]

[✓ T2]

☐ T3

☐ T4

☐ There is no deadlock

T2 is waiting for T1 so it can be granted the SIX lock on Voters but T1 waiting for T2 to give up its SIX lock on Ballots so it can acquire an S lock on it.

| Resource | Granted | Waitlist |
|---|---|---|

**iii. (1 pt)** Here is the above table copied for your convenience.

| Resource | Granted | Waitlist |
|---|---|---|
| Database | T1: IX, T2: IX, T3: IX, T4: IS | |
| Voters | T1: IX, T3: IS | T2: SIX |
| Ballots | T2: SIX, T3: IS, T4: IS | T1: S, T3: SIX |
| A | T1: S | |
| B | T1: X | |
| C | T2: X | |
| D | T3: S | |

Suppose that the lock table at the **next timestep** looks like the following:

| Resource | Granted | Waitlist |
|---|---|---|
| Database | T1: IX, T2: IX, T3: IX, T4: IS | |
| Voters | T1: IX, T3: IS | T2: SIX |
| Ballots | T2: SIX, T3: IS, T4: IS | T1: S, T3: SIX |
| A | T1: S | |
| B | | |
| C | T2: X | |
| D | T3: S | |

Which locking disciplines (if any) are ***guaranteed*** to be violated between the previous and current step?

[✓ 2 phase locking]

[✓ Strict 2 phase locking]

☐ None of the above

With Strict 2PL, all locks need to be unlocked at once. Since T2 released its X lock on Page B, it should have released all its other locks too. 2PL tells us that all lock requests must precede all unlock requests. Since T1 releases a lock while it is still in the waiting for more to be acquired, 2PL is also violated.

**iv. (1 pt)** Suppose that we take a look at the corresponding transaction schedule and discover that it does not satisfy 2 phase locking, let alone strict 2 phase locking. This means that the schedule is not conflict-serializable. True or false?

○ True

[✓ False]

While the reverse is true (strict 2PL guarantees us conflict-serializable schedules), this statement in general is false. For a general, easy-to-conceptualize example, we know that a schedule consisting solely of reads will be conflict-serializable regardless of lock/unlock ordering.

| Resource | Granted | Waitlist |
|---|---|---|

**v. (2 pt)** Let's refer to the original lock table again. Here it is reproduced for your convenience:

| Resource | Granted | Waitlist |
|---|---|---|
| Database | T1: IX, T2: IX, T3: IX, T4: IS | |
| Voters | T1: IX, T3: IS | T2: SIX |
| Ballots | T2: SIX, T3: IS, T4: IS | T1: S, T3: SIX |
| A | T1: S | |
| B | T1: X | |
| C | T2: X | |
| D | T3: S | |

Suppose T2 suddenly aborts. Which current waitlist requests (if any) are granted?

- ☐ T2: SIX
- [✓ T1: S]
- ☐ T3: SIX
- ☐ None of the above

Since we go in leftward order, T1: S is granted for Ballots since it was blocked by T2's SIX lock (which is now gone). The T3 SIX remains on the queue since it is incompatible with the S lock now held by T1. T2: SIX is not an answer since T2 is aborted.

**(c) (8 points) Dead(locked) Phantoms**

In this section, we will explore some design choices involving concurrency and transactions.

**i. (2 pt)** Course staff has been running into issues with the phantom problem in our database! As a refresher, the phantom problem occurs when one transaction reads different data in consecutive reads because a concurrent transaction has inserted/deleted a tuple in the meantime. For example, if T1 reads a list of products, then T2 inserts a new product, and then T1 re-reads, T1 will view a new product which is our "phantom" tuple. Can you suggest a strategy for preventing these phantom reads while still maintaining concurrency (**this means locking the entire database is not a valid answer**)?

To make your strategy concrete, assume the course staff database uses a single table known as `Students` which consists of two columns: `sid` (which is the primary key) and `grade`. The only two operations our transactions perform are reading grades from a range of `sids` and inserting a new student into the table. Assume no indices exist in the database.

Answers that mention predicate locking (e.g. locking on the range of records) will get full credit.

**ii. (2 pt)** The course staff database now seems to be constantly blocked by deadlock. A friend suggests a new policy: every transaction can only make one X lock request at a time. This means that every transaction can only write to one resource at a time. If a transaction needs to do more than one write, it will need to release its current X lock before it can request another one. She argues that this will eliminate deadlock. Is the friend correct? If yes, explain why this approach eliminates deadlock in 2-3 sentences. If no, specify the minimum number of locks a transaction must be either holding or waiting for to be involved in deadlock.

No. The minimum number of locks here is 2. Consider the case of two resources, A and B, and two transactions, T1 and T2. T1 holds an X lock on A while T2 is waiting for an S lock on A. T2 holds an X lock on B while T1 is waiting for an S lock on B. Clearly, there is deadlock here even though each transaction only has one X lock apiece.

**iii.** **(1 pt)** Course staff decides now that it wants to be able to use view serializable schedules in our database. One of the TAs suggests the following approach to check whether a schedule is view serializable. The first step is to create the conflict dependency graph as we learned in class. However, we want to also keep track of the last timestamp of a write for each resource. Now, we do a second pass over the schedule. For every conflict we see this time, we ***keep*** the edge from the existing conflict dependency graph if the conflict is between two writes that are not the last write to any resource; otherwise, we keep the edge in. If the resulting graph is cyclic, the schedule cannot be view serializable. Is the TA correct?

○ Yes

[✓ No]

Refer to the next solution.

**iv.** **(3 pt)** If you answered yes, explain in 2-3 sentences why a cycle in this new graph implies that the schedule is not view serializable. If you answered no, provide an example of a view serializable schedule whose modified conflict dependency graph is cyclic. In order to do this, you must separate the operations at each timestamp with commas. You are allowed a maximum of four timestamps, two transactions labeled T1 and T2, and at most two resources labeled A and B. As an example, if you believe the following schedule is view serializable but has a cycle:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| T1 | R(A) | | W(A) | |
| T2 | | R(B) | | W(B) |

Your answer would be: T1R(A),T2R(B),T1W(A),T2W(B).

No, it won't always be true. Consider the case of two transactions, T1 and T2, acting on one resource A. Let's say we have the following schedule:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| T1 | W(A) | | W(A) | |
| T2 | | W(A) | | W(A) |

With our algorithm, we have a cycle so we would assume the schedule is not view serializable. However, this schedule is actually view-equivalent to the serial schedule of T1 then T2. We can see that both schedules have the same initial and dependent reads (since there are no reads) and the same winning writes because in both T2 has the final write to resource A. Therefore, this algorithm does not always work.

The only correct answers here are: T1W(A),T2W(A),T1W(A),T2W(A); T2W(A),T1W(A),T2W(A),T1W(A); T1W(B),T2W(B),T1W(B),T2W(B); or T2W(B),T1W(B),T2W(B),T1W(B)

**4. (35 points) Recovering From 2020**

**(a) (4 points) True or False**

For the next 4 questions, indicate whether the statement is True or False.

**i. (1 pt)** Logging an UPDATE record using ARIES *always* involves updating the following state: `prevLSN`, `lastLSN`, `recLSN`, `pageLSN`, and `flushedLSN`.

○ True

[✓ False]

The `flushedLSN` is only updated when the log is flushed, which does not happen by default for an UPDATE log record. In addition, the `recLSN` is only updated if the modified page is not found in the Dirty Page Table.

**ii. (1 pt)** In ARIES, CLRs can be used to undo a CLR if the corresponding transaction aborts.

○ True

[✓ False]

CLRs are never undone. They can only be redone.

**iii. (1 pt)** Committing a transaction can cost more disk I/Os under UNDO logging than under ARIES.

[✓ True]

○ False

UNDO logging uses a Steal / Force policy, so committing a transaction involves flushing all corresponding data pages, while ARIES uses Steal / No-Force, so the data pages can be flushed at a later time.

**iv. (1 pt)** In ARIES, if a page P is in the Dirty Page Table, then the following are the tightest bounds we can come up with for the following terms respectively:

- $\text{pageLSN}_P(\text{memory}) \geq \text{recLSN}_P$
- $\text{pageLSN}_P(\text{disk}) \leq \text{pageLSN}_P(\text{memory})$

○ True

[✓ False]

The second inequality is false. We can come up with a tighter bound for $\text{pageLSN}_P(\text{disk})$ by changing $\leq$ to . If a page is in the Dirty Page Table, then the in-memory pageLSN of the page must be strictly greater than the on-disk pageLSN. Otherwise, the page would not be dirty if both versions were in sync.

**(b) (12 points) Running out the Clock**

The following log shows the operations the database has performed using ARIES. The buffer manager is constrained to **4** buffer frames and uses the **clock policy**.

Assume the following:

- Buffer frames are only used for data pages. The log tail is stored in a different part of memory with unlimited storage.
- The buffer manager accesses data pages in the exact order shown in the log and no other data pages are accessed.
- The buffer frames begin empty with the clock hand pointing at the first frame.
- Each page is pinned right before the access and unpinned right after the access.
- All on-disk pageLSNs are 0 initially.
- The WAL is not flushed for Aborts.

**Your answers should reflect the state of the database after all operations shown below have executed.**

***Hint:*** *Consider all of the actions taken when a data page is fetched or flushed under ARIES.*

| LSN | Record | prevLSN |
|---|---|---|
| 0 | Master | - |
| 10 | Update T1 writes P2 | null |
| 20 | Update T2 writes P3 | null |
| 30 | Update T3 writes P1 | null |
| 40 | Update T2 writes P4 | 20 |
| 50 | Update T3 writes P5 | 30 |
| 60 | Abort T2 | 40 |
| 70 | Update T1 writes P1 | 10 |
| 80 | Update T3 writes P2 | 50 |
| 90 | CLR T2 LSN 40, undoNextLSN: 20 | 60 |
| 100 | CLR T2 LSN 20, undoNextLSN: null | 90 |
| 110 | Update T3 writes P1 | 80 |
| 120 | End T2 | 100 |

**i. (2 pt)** What is the access pattern of the data pages recorded in the log? Express your answer using capital letters followed by numbers for the pages, and separate each access with a comma and no spaces.. For example, if you believe that according to the log, P1, P2, and P3 are accessed in that order, then you should write P1,P2,P3.

**P2,P3,P1,P4,P5,P1,P2,P4,P3,P1**

**ii. (1 pt)** What is the hit rate for the buffer manager? Write your answer as a simplified fraction.

**1/5**

We have the following hits (H) / misses (M) in order of page access: M, M, M, M, M, H, M, H, M, M.

**iii. (1 pt)** What frame does the clock hand point to after all of the above operations are complete?

[✓ Frame 1]

○ Frame 2

○ Frame 3

○ Frame 4

**iv. (2 pt)** What is the `recLSN` of page **P2**? If the page is not in the Dirty Page Table, answer **N/A**.

**80**

**v. (2 pt)** What is the `recLSN` of page **P4**? If the page is not in the Dirty Page Table, answer **N/A**.

**N/A**

**vi. (2 pt)** What is the `on-disk pageLSN` of page **P3**?

**20**

**vii. (2 pt)** What is the `flushedLSN`?

**90**

A WAL requirement is that before page $i$ is allowed to be flushed to disk, the following must be true: $\text{pageLSN}_i(\text{memory}) \leq \text{flushedLSN}$. The buffer manager evicts P2 at LSN 50 and the page is dirty with an in-memory pageLSN of 10, so the flushedLSN becomes 10. P3 is evicted at LSN 80 and the page is dirty with an in-memory pageLSN of 20, so the flushedLSN becomes 20. P1 is evicted at LSN 100 and the page is dirty with an in-memory pageLSN of 70, so the flushedLSN becomes 70. Finally, P4 is evicted at LSN and the page is dirty with an in-memory pageLSN of 90 (remember that CLRs update the page as well), so the final flushedLSN is 90.

**(c) (5 points) Analyze This**

Your database using ARIES crashes and you find the following log.

| LSN | Record | prevLSN |
|---|---|---|
| 0 | Master | - |
| 10 | Update T1 writes P1 | null |
| 20 | Update T1 writes P4 | 10 |
| 30 | Update T2 writes P2 | null |
| 40 | Begin Checkpoint | - |
| 50 | Update T3 writes P1 | null |
| 60 | Commit T1 | 20 |
| 70 | Update T3 writes P2 | 50 |
| 80 | End Checkpoint | - |
| 90 | Update T2 writes P4 | 30 |
| 100 | End T1 | 60 |
| 110 | Begin Checkpoint | - |
| 120 | Abort T3 | 70 |
| 130 | CLR LSN 70, undoNextLSN: 50 | 120 |
| | **CRASH** | |

The End Checkpoint record contains the following:

**Transaction Table**

| Xact ID | Status | lastLSN |
|---|---|---|
| T1 | Committing | 60 |
| T2 | Running | 30 |
| T3 | Running | 50 |

**Dirty Page Table**

| Page ID | recLSN |
|---|---|
| P1 | 50 |
| P4 | 20 |

Answer the following questions based on the state of the database after the end of the **entire** Analysis phase.

**i. (2 pt)** How many log records are written during the Analysis phase?

**1**

Only need an Abort log record for T2.

**ii. (1 pt)** What is the status of **T1**?

- ○ Running
- ○ Committing
- ○ Aborting
- [✓ Not in the Transaction Table]

**iii. (1 pt)** What is the status of **T3**?

- ○ Running
- ○ Committing
- [✓ Aborting]
- ○ Not in the Transaction Table

**iv. (1 pt)** What is the `recLSN` of page **P2**? If the page is not in the Dirty Page Table, answer **N/A**.

**70**

(d) **(5 points) RRR (and Redo) Week**

**This question is independent from the last question.** Consider the following **new** log for your database using ARIES. The database crashed and assume the **entire** Analysis Phase has been completed.

| LSN | Record | prevLSN |
|---|---|---|
| 0 | Master | - |
| 10 | Update T1 writes P2 | null |
| 20 | Update T2 writes P3 | null |
| 30 | Update T1 writes P1 | 10 |
| 40 | Update T3 writes P2 | null |
| 50 | Begin Checkpoint | - |
| 60 | Update T3 writes P1 | 40 |
| 70 | Commit T2 | 20 |
| 80 | End Checkpoint | - |
| 90 | Update T1 writes P4 | 30 |
| 100 | Update T3 writes P3 | 60 |
| 110 | Abort T1 | 90 |
| 120 | CLR T1 LSN 90, undoNextLSN: 30 | 110 |
| 130 | CLR T1 LSN 30, undoNextLSN: 10 | 120 |
| 140 | End T2 | 70 |
| 150 | Abort T3 | 100 |
| | **CRASH** | |

The Transaction Table and the Dirty Page Table after the entire Analysis Phase has completed are shown below:

**Transaction Table**

| Xact ID | Status | lastLSN |
|---|---|---|
| T1 | Aborting | 130 |
| T3 | Aborting | 150 |

**Dirty Page Table**

| Page ID | recLSN |
|---|---|
| P1 | 30 |
| P2 | 40 |
| P4 | 90 |

The following table indicates the on-disk `pageLSN` for each page before the Redo Phase has begun.

| Page ID | pageLSN |
|---|---|
| P1 | 0 |
| P2 | 10 |
| P3 | 100 |
| P4 | 0 |

**i. (1 pt)** What LSN does the Redo phase begin at?

**30**

Redo begins at the minimum recLSN in the DPT.

**ii. (4 pt)** Select the LSNs of all of the operations that are redone during the Redo phase.

- ☐ 0
- ☐ 10
- ☐ 20
- ■ 30
- ■ 40
- ☐ 50
- ■ 60
- ☐ 70
- ☐ 80
- ■ 90
- ☐ 100
- ☐ 110
- ■ 120
- ■ 130
- ☐ 140
- ☐ 150

**(e) (5 points) CTRL + Z**

Consider the same log, Transaction Table, and Dirty Page Table from the last problem, which have been copied below for convenience. Again, the database is using ARIES and assume after the crash that both the Analysis and Redo phases have been completed.

| LSN | Record | prevLSN |
|---|---|---|
| 0 | Master | - |
| 10 | Update T1 writes P2 | null |
| 20 | Update T2 writes P3 | null |
| 30 | Update T1 writes P1 | 10 |
| 40 | Update T3 writes P2 | null |
| 50 | Begin Checkpoint | - |
| 60 | Update T3 writes P1 | 40 |
| 70 | Commit T2 | 20 |
| 80 | End Checkpoint | - |
| 90 | Update T1 writes P4 | 30 |
| 100 | Update T3 writes P3 | 60 |
| 110 | Abort T1 | 90 |
| 120 | CLR T1 LSN 90, undoNextLSN: 30 | 110 |
| 130 | CLR T1 LSN 30, undoNextLSN: 10 | 120 |
| 140 | End T2 | 70 |
| 150 | Abort T3 | 100 |
| | **CRASH** | |

**Transaction Table**

| Xact ID | Status | lastLSN |
|---|---|---|
| T1 | Aborting | 130 |
| T3 | Aborting | 150 |

**Dirty Page Table**

| Page ID | recLSN |
|---|---|
| P1 | 30 |
| P2 | 40 |
| P4 | 90 |

**i. (2 pt)** How many log records are written during the Undo phase?

**6**

There are 3 CLRs for T3, and 1 CLR for T1. We do not need to log the CLRs that have already been written for T1 at LSN 120 and LSN 130. There is also an End log record for both T1 and T3.

**ii.** **(1 pt)** What is the maximum LSN of an operation for transaction **T1** that has a corresponding CLR written during the Undo phase? If no CLRs are written for transaction **T1**, answer **N/A**.

**10**

**iii.** **(1 pt)** What is the maximum LSN of an operation for transaction **T2** that has a corresponding CLR written during the Undo phase? If no CLRs are written for transaction **T2**, answer **N/A**.

**N/A**

**iv.** **(1 pt)** What is the maximum LSN of an operation for transaction **T3** that has a corresponding CLR written during the Undo phase? If no CLRs are written for transaction **T3**, answer **N/A**.

**100**

**(f) (4 points) Masking Mallory's High Crimes and Misdemeanors**

**i. (2 pt)** Your arch nemesis Mallory, known for her exploits, has found a way to infiltrate your database, which uses ARIES. Mallory has gained full control over all **on-disk pageLSNs**, and she can update them however and whenever she wants. In 2-4 sentences state which ACID property/properties (if any) Mallory can violate and explain how.

Mallory can prevent atomicity by increasing the on-disk pageLSNs, such that operations that were not persisted on disk are incorrectly skipped during the Redo phase. This breaks atomicity if only some of the changes are persisted to disk for a transaction. Mallory can also break durability with the same trick as before, since even if a transaction commits, the Redo phase may incorrectly skip its operation, which prevents the changes from being made persistent. Mallory can also break consistency because it is possible that a database integrity constraint is broken when a write for a transaction is skipped due to the increased on-disk pageLSNs. As an example, consider a bank account transfer.

**ii. (2 pt)** You successfully revoked Mallory's access of the on-disk pageLSNs, but you are now worried that Mallory can cause your database to *repeatedly* crash during the ARIES recovery process. In 2-4 sentences state 2 changes you can implement to reduce the effect of Mallory's attack on the latency of the recovery process and explain your answer.

Answers that are not compatible with ARIES or propose using a different logging mechanism besides ARIES will not receive credit. This includes answers such as switching to UNDO or REDO logging, enforcing a FORCE/NO-STEAL policy, etc.

Valid answers include the following: - Checkpoint more frequently - Avoid long running transactions - Periodically flush dirty pages in the background - Replicate or partition the database

**5. (19 points) Normalize Normalization**

Consider a relation R(A,B,C,D,E) with the following functional dependencies:

(a) A -> C
(b) BC -> D
(c) CD -> E
(d) E -> A

**(a)** **(4 pt)** Enumerate all of the (minimal) candidate keys in this relation. Separate them by commas, with no spaces in between. When listing keys, list them in alphabetically increasing order. (Thus, if your answers are ABC and DCB, list them as:ABC,BCD)

AB,BC,BE

**(b)** **(3 pt)** Which of the functional dependencies above are BCNF violations? List the corresponding FD numbers in increasing order. Thus, if your answer is A->C and CD->E, list them as: 1,3

1,3,4

**(c)** **(3 pt)** Using the BCNF decomposition algorithm described in class, and the first FD you listed in the previous question, provide the first step of your decomposition. Simply list the schemas of the two relations that result. When listing schemas, list them in alphabetically increasing order. Thus, if your answer is R(A, B, C) and S(A, C, E, D), then list: ABC,ACDE

ABDE,AC

**(d)** **(3 pt)** Using your answer from the previous question, continue decomposing the relations until you arrive at a BCNF decomposition. Simply list the schemas of the eventual relations. When listing schemas, list them in alphabetically increasing order. Thus, if your answer is R(A, B), S(C, D), and T(E, D, A), then list: AB,ADE,CD

AC,AE,BDE

**(e)** **(3 pt)** Are there any other BCNF decompositions? If not, simply answer "No". If yes, list the schemas of the eventual relations. When listing schemas, list them in alphabetically increasing order. Thus, if your answer is R(A, B), S(C, D), and T(E, D, A), then list: AB,ADE,CD

**Yes. AC,AE,BCD,DE using 3, then 1, then 4.**

**(f) (3 pt)** There is a single FD of the form X->Y where X and Y are both single attributes such that by adding that FD, there are no longer any violations in the original set. What is that FD? [Answer this as "X->Y", replacing X and Y each with one of A–E.]

**C->B or A->B**

**6. (32 points) Parallel Parking**

Suppose we have the following tables:

```
Cars(car_id, manufacturer, model, cost)
Parking(car_id, location, start, end)
```

We will abbreviate `Cars` as C and `Parking` as P. C has 450 pages of data while P has 150 pages of data.

**(a)** **(3 pt)** Both C and P have been round robin partitioned across three machines. In the worst case, how many KB of data will be sent across the network in a parallel grace hash join? Suppose that we have 5 buffer pages on each machine and that each page has 4 KB of data.

**2400**

In the worst case, we would have to send all pages of both tables across the network. The total size is (450 + 150) * 4 KB = 2400 KB.

**(b)** **(3 pt)** Now, suppose C and P have been equally range partitioned across three machines according to the same ranges on the `car_id` column. How many KB of data will be sent across the network in a parallel sort merge join? Again, suppose that we have 5 buffer pages on each machine and that each page has 4 KB of data.

**0**

All the data are in the correct machine, so there will be no network cost.

**(c)** **(3 pt)** Now, suppose C and P have been hash partitioned across three machines on `car_id`. We want to run the following query:

```
SELECT MAX(car_id) FROM Parking;
```

How many I/Os across all machines will it take to execute this query? Again, suppose that we have 5 buffer pages on each machine and that each page has 4 KB of data.

**150**

We will need to look at every page on each machine.

**(d)** **(3 pt)** As in the previous part, suppose C and P have been hash partitioned across three machines on `car_id`. We want to run the following query:

```
SELECT COUNT(DISTINCT car_id) FROM Parking;
```

How many KB of data will be sent across the network to execute this query in the worst case? Again, suppose that we have 5 buffer pages and that each page has 4 KB of data. Do not include the cost of aggregating the results for the query.

**0**

We can perform the query separately on each machine and aggregate the results afterwards.

**(e)** **(3 pt)** As in the previous part, suppose C and P have been hash partitioned across three machines (M1, M2, and M3) on `car_id`. Each of the three machines now has **30 buffer pages**, and now we want to perform a parallel grace hash join between C and P.

M1 has 80 pages of data from C and 70 pages of data from P.

M2 has 120 pages of data from C and 60 pages of data from P.

M3 has 250 pages of data from C and 20 pages of data from P.

For the remaining questions, assume that data is streamed from the network directly into memory during partitioning, so each machine does not incur any read I/Os when data is received from the network. Also, assume that the hash function used to partition in grace hash join is perfect.

What is the IO cost of a grace hash join on machine M1?

**348**

We need to do one pass of partitioning to get P to fit within B-2 buffers. Since we do not include the initial read and the final write, we must account for the write phase of partitioning and the read phase for matching. Partitioning 80 and 70 pages into 29 buckets would create 87 pages in both cases using a perfect hash function. Thus, the total cost is 2 * (87 + 87) = 348 I/Os.

**(f)** **(3 pt)** What is the IO cost of a grace hash join on machine M2?

**464**

We need to do one pass of partitioning to get P to fit within B-2 buffers. Since we do not include the initial read and the final write, we must account for the write phase of partitioning and the read phase for matching. Partitioning 120 and 60 pages into 29 buckets would create 145 pages and 87 pages using a perfect hash function. Thus, the total cost is 2 * (145 + 87) = 464 I/Os.

**(g)** **(3 pt)** What is the IO cost of a grace hash join on machine M3?

**0**

We do not need to do any partitioning because P fits within B - 2 buffers. Since we do not include the initial read and the final write, the overall I/O cost is 0.

**(h)** **(2 pt)** Suppose that each I/O takes 1 ms. How long does it take to do the entire parallel grace hash join? Assume that network costs are negligible, so do not include them in your calculation.

**464**

We need to wait for the slowest machine, which is M2.

(i) **(3 pt)** We gain access to three more machines (M1.1, M1.2, and M1.3) with 10 buffer pages each, but they can only receive input from M1. As data is streamed into M1, we decide to do a further range partition based on `car_id`. Assume that M1 immediately partitions and sends the data through the network as it is received, without writing it to disk.

M1.1 has 30 pages of data from C and 40 pages of data from P. M1.1 contains `car_id` values less than or equal to 50.

M1.2 has 30 pages of data from C and 20 pages of data from P. M1.2 contains `car_id` values greater than 50 and less than 100.

M1.3 has 20 pages of data from C and 10 pages of data from P. M1.3 contains `car_id` values greater than or equal to 100.

Instead of doing a hash join on M1, we decide to do a sort merge join on M1 using the three submachines. If each I/O takes 1 ms, how long does it take to do the parallel un-optimized sort merge join? Assume that data is streamed from the network directly into memory during partitioning, so each machine does not incur any read I/Os when data is received from the network.

**280**

The bottleneck is M1.1, so we only need to calculate the I/O cost from M1.1. We need two passes to sort C, and we need two passes to sort P. We need one extra pass to merge the two relations. Thus, the total cost is 4 * (30 + 40) = 280 since we do not include the initial read and the final write.

(j) **(2 pt)** If each I/O takes 1 ms, how long does it now take to do the entire parallel join? For this part only, you can use your answers to previous parts of the question, or you may assume that M2 takes 557.167 ms to complete its part of the join and that M3 takes 534.178 ms to complete its part of the join. Note that these are not the correct answers to previous parts of the question.

**464 (or 557.167 ms)**

We still need to wait for the slowest machine, which is M2.

(k) **(2 pt)** We then aggregate all results from M1.1, M1.2, M1.3, M2, and M3. Is the resulting join output correct? Explain your answer.

**Yes**

The tuples that route to M1, M2, and M3 are correct because matching tuples are hashed to the same machine. The tuples that route to M1.1, M1.2, and M1.3 are correct because matching tuples are range partitioned to the same machine.

(l) **(2 pt)** Is it guaranteed for the resulting join to have an interesting order? Explain your answer.

**No**

One part is sorted, and the other part is hashed. Hashing will not provide any interesting order.

**7. (17 points) Distributed (Vaccine) Transactions**

**(a) (1 pt)** Which strategy do we use to perform distributed deadlock detection?

- ○ Intersection of individual waits-for graphs
- [✓ ● Union of individual waits-for graphs]
- ○ Union of individual conflict dependency graphs
- ○ Intersection of the individual conflict dependency graphs
- ○ None of the above

Each machine has its own waits-for graph for the locks on that machine. We use the union of the individual waits-for graphs in order to determine global deadlock across the machines.

**(b) (1 pt)** Assuming no presumed abort, a participant does not have to ACK a phase 2 abort message from the coordinator.

- ○ True
- [✓ ● False]

Unlike presumed abort, participants need to send back an ACK for both commit messages from the coordinator and abort messages from the coordinator.

**(c) (4 points) Message Verification**

Say that we have three machines, named M1, M2, and M3, all working on the same transaction. M1 is designated the coordinator, and the other two machines are designated as participants. We pose three scenarios in the following questions. Assume the following: The scenarios are all independent of each other. The questions in this scenario are ordered in chronological order. We are under presumed abort. None of the machines crash, unless a machine is explicitly stated to have crashed in the question statement. Machines always receive messages, unless they are crashed at the time of the message being sent; in that case, assume that the machine did not receive the message. There is a network connection between M1 and M2, and a network connection between M1 and M3.

**i. Scenario 1**

For each set of potential actions, select the action(s) that would happen for the case where the transaction successfully commits. If none of the actions are correct, select "None of the above."

We start with M1 sending a PREPARE message to all participants.

**A. (1 pt)**

- [✓ ■ M2 sends a VOTE YES message to M1]
- ☐ M2 sends a VOTE NO message to M1
- ☐ None of the above

In order to commit, all of the participants need to vote yes.

**B. (1 pt)**

- [✓ ■ M3 sends a VOTE YES message to M1]
- ☐ M3 sends a VOTE NO message to M1
- ☐ None of the above

In order to commit, all of the participants need to vote yes.

**C. (1 pt)**

- [✓ M1 sends a COMMIT message to all participants]
- ☐ M1 sends an ABORT message to all participants
- ☐ None of the above

All of the participants voted yes, so the coordinator, M1, decides to commit, and sends a COMMIT message to all the participants.

**D. (1 pt)**

- [✓ M2 sends an ACK message to M1]
- [✓ M3 sends an ACK message to M1]
- ☐ None of the above

When the participants receive the COMMIT message, they must send an ACK back to the coordinator (M1).

**ii. (4 points) Scenario 2**

Now, consider the scenario where M2 votes yes while M3 votes no. For each set of potential actions, select the action(s) that would happen. If none of the actions are correct, select "None of the above."

We start with M1 sending a PREPARE message to all participants.

**A. (1 pt)**

- ■ M2 sends a VOTE YES message to M1
- ☐ M2 sends a VOTE NO message to M1
- ☐ None of the above

You are given that M2 votes yes.

**B. (1 pt)**

- ☐ M3 sends a VOTE YES message to M1
- ■ M3 sends a VOTE NO message to M1
- ☐ None of the above

You are given that M3 votes no.

**C. (1 pt)**

- ☐ M1 sends a COMMIT message to all participants
- ■ M1 sends an ABORT message to all participants
- ☐ None of the above

Because at least one of the participants voted no, the coordinator, M1, decides to abort, and sends an ABORT message to all participants.

**D. (1 pt)**

- ☐ M2 sends an ACK message to M1
- ☐ M3 sends an ACK message to M1
- ■ None of the above

Under presumed abort, when participants receive an ABORT message from the coordinator, they do not have to send an ACK back to the coordinator.

**iii. (7 points) Scenario 3**

Now, select all of the valid actions for the scenario where M1, M2, and M3 participate in a transaction that successfully commits. However, M3 crashes during the process. It is noted within the scenario when M3 crashes and when it later recovers. For each set of potential actions, select the action(s) that would happen. If none of the actions are correct, select "None of the above."

We start with M1 sending a PREPARE message to all participants.

**A. (1 pt)**

- ■ M2 logs a PREPARE record
- ☐ M2 logs an ABORT record
- ☐ None of the above

In order for the transaction to successfully commit, all participants must decide to commit, and they will log PREPARE records accordingly.

**B. (1 pt)**

- ■ M2 sends a VOTE YES message to M1
- ☐ M2 sends a VOTE NO message to M1
- ☐ None of the above

In order for the transaction to successfully commit, all participants must decide to commit, and they will send VOTE YES messages back to the coordinator.

**C. (1 pt)**

- ■ M3 logs a PREPARE record
- ☐ M3 logs an ABORT record
- ☐ None of the above

In order for the transaction to successfully commit, all participants must decide to commit, and they will log PREPARE records accordingly.

**D. (1 pt)**

- ■ M3 sends a VOTE YES message to M1
- ☐ M3 sends a VOTE NO message to M1
- ☐ None of the above

In order for the transaction to successfully commit, all participants must decide to commit, and they will send VOTE YES messages back to the coordinator.

**E. (1 pt)** M3 crashes at this point, immediately after the action(s) in the previous question. What happens next?

- ■ M1 sends a COMMIT message to all participants
- ☐ M1 sends an ABORT message to all participants
- ☐ None of the above

M1 has received VOTE YES messages from all of the participants, so it decides to COMMIT and sends the message to all the participants.

**F. (1 pt)** M3 recovers at this point, and M3 asks M1 what the final decision was. What happens next?

[✓ M1 sends a COMMIT message to M3]

☐ M1 sends an ABORT message to M3

☐ None of the above

The coordinator decided to COMMIT earlier, so it sends COMMIT the message to M3 again.

**G. (1 pt)**

[✓ M2 sends an ACK message to M1]

[✓ M3 sends an ACK message to M1]

☐ None of the above

When the participants receive the COMMIT message, they must send an ACK back to the coordinator (M1).

**8. (16 points) No Free Lunch, SQL**

**(a) (5 points) True or False**

**i. (1 pt)** Unlike the guarantee provided by ACID, databases modeled on BASE principles make no guarantees about consistency.

○ True

[✓ False]

One of the BASE principles is *eventual consistency*, which guarantees that eventually all replicas will become consistent once updates have propagated through the system.

**ii. (1 pt)** A lookup operation in Mongo's query language is most similar to a FULL OUTER JOIN in SQL.

○ True

[✓ False]

Mongo lookups are most similar to a LEFT OUTER JOIN (or RIGHT), since we guarantee that every document in the original group of documents is present even in the output, even if there are no matches in the collection the lookup is being performed on.

**iii. (1 pt)** Key-Value stores typically only allow string/integers as keys, but can allow any type of data to be stored as a value.

[✓ True]

○ False

**iv. (1 pt)** Modern RDBMSs are unable to support semi-structured data like XML or JSON.

○ True

[✓ False]

A few examples given in lecture were SQL Server, DB2 and Oracle for XML. Not directly mentioned, but MySQL, Postgres, and SQLite all support JSON either natively or through extensions.

**v. (1 pt)** One of the motivating forces behind developing NoSQL systems is the need to handle large Online Transaction Processing (OLTP) workloads.

[✓ True]

○ False

This is true. Web 2.0 applications involved increasing amounts of user generated data and updates (tweets, posts, likes, etc. . . ) that necessitated a high number of transactions executed by high numbers of users.

**(b) (9 points)**

You work for a large, online marketplace that handles thousands of purchases each day from millions of customers around the world. The company has the following collections in their Mongo database:

The `customers` collections contains documents representing every unique customer that has ever made a purchase through the company. Each document may have the following fields:

- `customerId`: a unique identifier for the customer. Always present
- `age`: the customer's age if it is known, otherwise the field is not present
- `city`: the customer's city of residence if it is known, otherwise the field is not present

The `purchases` collection contains documents representing individual purchases customers have made. Each has the following fields:

- `customerId`: identifier for the customer who made this purchase, corresponding to the field of the same name in `customers`. An index on this field is maintained for fast lookups. Always present
- `name`: The name of the product that was purchased. Always present
- `time`: integer representing when the purchase was made. Always present

You can assume that if a field does not exist in a document, a match on that field will filter out that document from the final result without erroring.

**i. (2 pt)** Consider the following query:

```
db.customers.aggregate(
    [
        {
            $match: {
                city: "Berkeley",
                age: {
                    $gte: 18,
                    $lt: 25
                }
            }
        }
    ]
)
```

Which of the following is the closest equivalent to the query above in SQL?

○ `SELECT * FROM customers WHERE city = 'Berkeley' OR (age >= 18 AND age < 25);`

○ `SELECT * FROM customers WHERE city = 'Berkeley' AND (age >= 18 OR age < 25);`

[✓ `SELECT * FROM customers WHERE city = 'Berkeley' AND age >= 18 AND age < 25;`]

○ `SELECT * FROM customers WHERE city = 'Berkeley' OR age >= 18 OR age < 25;`

The criteria within the match operator are applied conjunctively (AND'd all together)

**ii. (2 pt)** Consider the following query:

```
db.purchases.aggregate(
    {
        $match: {
            time: {
                $gte: 0,
                $lt: 1000
            }
        }
    },
    {
        $project: {
        _id: 0,
        newTime: "$time",
        name: 1
        }
    }
)
```

Which of the following is the closest equivalent to the query above in relational algebra?

○

$$\pi_{\text{newTime, name}}(\sigma_{\text{time}\geq 0\wedge \text{time}1000}(\text{purchases}))$$

○

$$\sigma_{\text{time}\geq 0\wedge \text{time}1000}(\pi_{\text{newTime, name}}(\text{purchases}))$$

[✓]

$$\pi_{\text{newTime, name}}(\rho_{\text{time}\rightarrow \text{newTime}}(\sigma_{\text{time}\geq 0\wedge \text{time}1000}(\text{purchases})))$$

○

$$\rho_{\text{time}\rightarrow \text{newTime}}(\pi_{\text{newTime, name}}(\sigma_{\text{time}\geq 0\wedge \text{time}1000}(\text{purchases})))$$

The project operator does a rename, so we must use

$$\rho$$

somewhere. Note that the option that projects before the rename attempts to project on the `newTime` field, which doesn't exist yet.

**iii. (3 pt)** Consider the following query for the next 3 questions.

```
db.customers.aggregate(
    [
        {
            $lookup: {
                from: "purchases",
                localField: "customerId",
                foreignField: "customerId",
                as: "customerPurchases"
            }
        },
        { $unwind: "$customerPurchases"},
        {
            $match: {
                city: "Berkeley",
                age: {
                    $gte: 18,
                    $lt: 25
                },
                "customerPurchases.time": {
                    $gte: 1584576000, // Timestamp for March 19th, 2020, 12AM
                    $lt: 1584662400 // Timestamp for March 20th, 2020, 12AM
                }
            }
        }
    ]
)
```

Describe each of the fields present in the documents produced by this query. If a field's value is a nested document, list (but don't describe) the fields of the nested document as well.

The query outputs documents with the following fields: - `_id`: is technically still present since we didn't explicitly project it out - `customerId`: identifier for a customer - `customerPurchases`: a nested document with the fields `time`, `name`, and `customerId` in it - `age` age of the customer who made the purchase - `city` customer's city of residence

Full credit if the student gets most of this correct (only forgets 3 fields or less) and acknowledges that customerPurchases is a nested document.

**iv. (2 pt)** You attempt to run the query but notice that it seems to be running slowly without producing any output. You run individual queries on the `customers` and `purchases` collections (no lookups) successfully and rule out connectivity/machine availability problems, and know with certainty there should be at least one document outputted by the query. You decide that the problem must be with the query itself. Explain why the query is running slowly.

Hint: MongoDB's query optimizer is relatively simple and can miss opportunities for optimizations like the ones we covered in class. For example, it only "pushes down selects" when they are preceded by a projection stage.

Mongo won't push down the $match. This means that to execute the query, every purchase ever made by any customer from any location is retrieved and unwound before the match stage is applied. Since the majority of these values won't be exactly what we're looking for (specific age, location and time), it will take a long time to before any output is available, and even longer to get the entire set of documents we want.

**v. (2 pt)** Describe a change you can make to the query to speed up the time it takes to execute without changing the set of documents that should be outputted. Be specific about which fields are involved in your changes and the location you place any new stages you introduce to the pipeline.

We need to push down the $match to before the $lookup. Note that we can't push down the entire match though, just the components related to the `age` and `city` stages specifically. The match on `time` relies on information in the `purchases` table that won't be available yet.

**No more questions.**