CS 186 Fall 2021 Introduction to Database Systems

'Final'

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 W186 Fall 2021 Final

`If you do not have special accommodations, you will have 170 minutes to complete the final.`

1.0.1 Contents:

- The final has 12 questions, each with multiple parts, and worth a total of 186 points.

1.0.2 Aids:

- You may use 3 pages (double sided) of handwritten notes as well as a calculator.
- You must work individually on this exam.
- Do not share this exam until solutions are released.

1.0.3 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.

1.0.4 Taking the exam remotely:

- Do NOT post on piazza if you want to ask a question, use the clarification form. We reserve the right to deduct points from the midterm if you use piazza during the exam.
- 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.

(a) What is your full name?

(b) What is your student ID number?

(c) SID of the person to your left (Put None if you are taking the exam remotely):

(d) SID of the person to your right (Put None if you are taking the exam remotely):

1. (4 points) Déjà Vu

You must select cars for a race. The following SQL tables contain route and car data:

```
CREATE TABLE routes (
  route_id INT UNIQUE,
  total_length INT
);

CREATE TABLE cars (
  model VARCHAR UNIQUE,
  max_length INT
);
```

The `routes` table contains all possible routes for the race, as well as their corresponding `total_length`. The `cars` table contains all possible cars for the race, as well as the `max_length` any car can travel in a race.

The `total_length` and `max_length` columns may contain `NULL` values.

Your goal is to create a query with one row for each car and route pair where the car's `max_length` is strictly lower than the `route`'s `total_length`.

Output constraints:

- If a `route` doesn't specify a `total_length`, assume that all cars can drive on it.
- If a `car` doesn't specify a `max_length`, assume it cannot drive on any route.

Do the following queries fulfill the goal and the output constraints?

(a) (1 pt)

```
SELECT model, route_id
FROM cars INNER JOIN routes
ON max_length < total_length
```

◯ True

● False

This query is incorrect because it ignores any rows in `routes` where `total_length` is `NULL`.

(b) (1 pt)

```
SELECT model, route_id
FROM cars RIGHT OUTER JOIN routes
ON max_length < total_length OR total_length is NULL
WHERE model is not NULL AND max_length is not NULL
```

● True

◯ False

(c) (1 pt)

```
SELECT model, route_id
FROM cars LEFT OUTER JOIN routes
ON max_length < total_length OR total_length is NULL
```

◯ True

● False

This query is also incorrect. It includes all cars whose `max_length` is `NULL`.

(d) (1 pt)

```
SELECT model, route_id
FROM cars LEFT OUTER JOIN routes
ON max_length < total_length OR max_length is NULL
```

◯ True

● False

This query is also incorrect. It also includes all cars whose `max_length` is `NULL`.

2. (14 points) All I Need– Adrenaline!

Aditya and Shreyas decide to street race. They ask you to make a route for the race. The following SQL table describes potential roads to use in the route:

```
CREATE TABLE roads (
  name VARCHAR UNIQUE,
  start_city VARCHAR,
  end_city VARCHAR
);
```

There are no NULL values in this table.

Your goal is to create a SQL query with all valid routes in the output.

A valid route must follow these conditions:

- The route must have 3 distinct roads.
- The roads should be connected in a loop. In other words, the first road should end where the second begins, the second should end where the third begins, and the third should end where the first begins.
- Any potential route should appear only once in the output. **Two routes that use the same 3 roads in a shifted order are considered duplicates, and only one of the routes should appear in the output**. For example, road1 -> road2 -> road3 is considered the same route as road2 -> road3 -> road1, and only one of these should appear in the output (assuming all other conditions are met).

Keep in mind a few conditions about roads:

- Each road only goes one way (from `start_city` to `end_city`). A road going from `start_city` to `end_city` will therefore only appear once in the `roads` table.
- Some roads have the same `start_city` and `end_city`. In other words, these roads loop out of and return to the same city.

(a) (3 pt) Does this query fulfill all route conditions? If it doesn't, pick an answer choice explaining why. There may be multiple answers.

```
SELECT prefix_routes.r1_name, prefix_routes.r2_name, r.name
FROM
    (SELECT r1.name AS r1_name, r2.name AS r2_name,
    r1.start_city AS start_city, r2.end_city AS end_city
    FROM roads as r1, roads as r2
    WHERE r1.end_city = r2.start_city) AS prefix_routes, roads AS r
WHERE prefix_routes.r1_name > r.name
    AND prefix_routes.r2_name > r.name
    AND r.start_city = prefix_routes.end_city
    AND r.end_city = prefix_routes.start_city;
```

☐ A. Incorrect: Output contains routes not made of 3 distinct roads

☐ B. Incorrect: Output contains routes not connected in a loop

☐ C. Incorrect: Output contains duplicate routes

■ D. Correct

(b) (3 pt) Does this query fulfill all route conditions? If it doesn't, pick an answer choice explaining why. There may be multiple answers.

```
SELECT r1.name, r2.name, r3.name
FROM roads as r1, roads as r2, roads as r3
WHERE r1.end_city = r2.start_city
    AND r2.end_city = r3.start_city
    AND r3.end_city = r1.start_city
```

- [x] A. Incorrect: Output contains routes not made of 3 distinct roads
- [ ] B. Incorrect: Output contains routes not connected in a loop
- [x] C. Incorrect: Output contains duplicate routes
- [ ] D. Correct

This query does not enforce any ordering on the city names in the route. As a result, there are two problems:

- Since there are roads with the same `begin_city` and `end_city`, this query will generate routes where these roads are repeated 3 times (e.g. if road1 goes from city1 to city1, then this query will output road1 -> road1 -> road1 as a route). Option A is correct.
- Since there's no ordering on the city names, the same route may appear in shifted order (e.g. if road1 -> road2 -> road3 is a valid route, then road2 -> road3 -> road1 will also be in the output). Option C is also correct.

(c) (3 pt) Replace the `WHERE` clause in the previous subpart with a `WHERE` clause that makes the previous subpart's query correct. Pick from the following options:

- [x] A. `WHERE r1.end_city = r2.start_city AND r2.end_city = r3.start_city AND r3.end_city = r1.start_city AND r1.start_city > r2.start_city AND r1.start_city > r3.start_city`
- [ ] B. `WHERE r1.end_city = r2.start_city AND r2.end_city = r3.start_city AND r3.end_city = r1.start_city AND r1.start_city > r2.start_city`
- [ ] C. `WHERE r1.end_city = r2.start_city AND r2.end_city = r3.start_city AND r3.end_city = r1.start_city AND r1.start_city != r2.start_city AND r1.start_city != r3.start_city`
- [ ] D. `WHERE r1.end_city = r2.end_city AND r2.end_city = r3.start_city`
- [ ] E. The query is incorrect and none of the above options fix it.
- [ ] F. The query is already correct.

Option A enforces an ordering where the first city is the earliest one lexicographically. This ensures that duplicate routes and routes with repeating roads don't show up in the output.

(d) **(2 pt)** Assume that you create a query that fulfills all the route conditions. Now you want to make sure that no roads in any valid routes have "snowy" weather. You store the results from your query in the `potential_routes` table, and you also access the `road_conditions` table which has data about road weather:

```
CREATE TABLE potential_routes (
  r1_name VARCHAR,
  r2_name VARCHAR,
  r3_name VARCHAR
);

CREATE TABLE road_conditions (
  name VARCHAR UNIQUE,
  weather VARCHAR
);
```

`r1_name`, `r2_name`, and `r3_name` refer to the first, second, and third road names in the route.

Does the following query have all routes that fulfill **all** route conditions (including that no roads should have "snowy" weather)? If not, please select all options that explain why not.

```
SELECT pr.r1_name, pr.r2_name, pr.r3_name
FROM potential_routes AS pr, road_conditions AS rc
WHERE (rc.name = pr.r1_name OR rc.name = pr.r2_name OR rc.name = pr.r3_name)
    AND rc.weather != 'snowy'
```

- ☐ A. Incorrect: Output contains routes not made of 3 distinct roads
- ☐ B. Incorrect: Output contains routes not connected in a loop
- ■ C. Incorrect: Output contains duplicate routes
- ■ D. Incorrect: Output contains routes with 'snowy' roads
- ☐ E. Correct query

The `OR` conditions in the `WHERE` clause mean that any route without 'snowy' roads will show up three times in the output (once for each road that's not 'snowy'). Additionally, any `potential_routes` with 'snowy' roads will show up once in the output for every road that's not 'snowy'.

(e) **(3 pt)** Pick all the following options that make the previous subpart's query correct. If the query is already correct, and there is an option that changes it but keeps it correct, pick both the option that changes it and the "Query is already correct" option. There may be multiple correct answers.

- ☐ A. Replace the `WHERE` clause with:`WHERE rc.name = pr.r1_name AND rc.name = pr.r2_name AND rc.name = pr.r3_name AND rc.weather != 'snowy'`
- ☐ B. Replace the `WHERE` clause with:`WHERE (rc.name != pr.r1_name OR rc.name != pr.r2_name OR rc.name != pr.r3_name) AND rc.weather = 'snowy'`
- ☐ C. Add the following to the end of the query:`GROUP BY rc.weather HAVING COUNT(*) = 3`
- ■ D. Add the following to the end of the query:`GROUP BY pr.r1_name, pr.r2_name, pr.r3_name HAVING COUNT(*) = 3`
- ☐ E. Query is incorrect, and none of the above options fix it
- ☐ F. Query is already correct

Option D groups together duplicate routes and only yields groups with three rows. The only groups that have three rows are the routes where none of the three roads are 'snowy'.

3. (16 points) Disks, Files, and Buffers

(a) T/F

i. (1 pt) I/O cost of performing a full scan on a sorted file is the same as scanning a heap file, assuming both are packed.

- [x] True
- [ ] False

ii. (1 pt) A page directory keeps track of the amount of free space on data pages.

- [x] True
- [ ] False

(b) Secret SanTA

The TAs are doing a secret santa gift exchange. They create the following table to keep track of people's interests for their wishlist:

Name (String)	ID (integer)	Favorite Color (String)	Chosen?
Aditi	123	Red	False
Dylan	124	Turquoise	False
Ada	143	Red	False
David	149	Green	False
Kaitlyn	153	Olive	False
Sharon	176	Gold	False
Sabrina	187	Purple	False
Kayli	198	Yellow	False
Shreyas	202	Orange	False

Each Name with the same initial is stored on the same page of a heap file. For example, all the records for a Name starting with 'A' will be on the same page in memory. Assume each letter gets a unique page in memory.

i. (4 pt) The secret santas are looking up information form this table to decide what to give the TAs leading to the following access pattern:

S, A, K, S, K, A, K, D, A, S, A, K, D, K, S

Note: These letters refer to pages of a table and not the individual records listed in the above table.

We have 3 pages in the buffer (B = 3). Using the Clock replacement policy, which 3 pages remain in the buffer pool (in alphabetical order)? How many cache hits do we have?

S, K, D remain in the buffer. 8 cache hits.

ii. (2 pt) Now, consider a different scenario where we have an Alternative 2 clustered Index and an Alternative 2 unclustered Index, both with height 3. Assume that 4 records fit on a page for the heap file.

Suppose a range scan returns 4 records. Assuming that all matches were on the same leaf of the index, how many IOs does this take for the unclustered Index in the worst case?

8

iii. (2 pt) Suppose a range scan returns 4 records. Assuming that all leaf records were on the same leaf of the index, how many IOs does this take for the clustered Index in the worst case?

8

iv. (3 pt) Some more TAs from other classes want to join the secret santa too. About 500 more people need to be added to the table. Every time a new person joins, a new record (with schema shown earlier) is inserted into the file.

Before the drawing of names, to optimize the above scenario, which options for page, file, and index should be used?

Page Format	File Layout	Index
(a) Fixed length (Packed)	(d) Heap File	(f) Clustered Index on Id
(b) Fixed length (Unpacked)	(e) Sorted File	(g) Unclustered Index on Id
(c) Slotted Page		(h) None

☐ a

☐ b

■ c

■ d

☐ e

☐ f

☐ g

■ h

C, d, h

Before the name drawing, only inserts are being performed. Since the records have a variable length, we need to use a slotted page. A heap file is the best file layout because many inserts are being done and heap files have a constant time insert while the cost to maintain the sorted file becomes progressively worse with more records. Lastly, an index will just add overhead to this insertion process so we will not need one.

v. (3 pt) At the end of the week, everyone will draw a name from the table and update the `Chosen?` Column to `True` for the person they selected. Everyone has to be assigned a name. Each TA will get a unique ID for the person they are gifting and need to look up the record in the table, update the Chosen? Column, and use the information listed to help figure out what to gift them. Each secret santa has an id for their person.

To optimize the above scenario (assume no insertions from this point on), which options for page, file, and index should be used considering we still need to do the insert process from before?

Page Format	File Layout	Index
(a) Fixed length (Packed)	(d) Heap File	(f) Clustered Index on Chosen?
(b) Fixed length (Unpacked)	(e) Sorted File	(g) Unclustered Index on Id
(c) Slotted Page		(h) None

☐ a

☐ b

■ c

■ d

☐ e

■ f

■ g

☐ h

C, d, g

We still maintain a heap file but add an index on Id which makes it easy to look up the record.

4. (21 points) B+ Trees

(a) (2 pt) Which of the following are true? Select all that apply. There may be more than one correct answer.

A. Bulk-loading a B+ tree with fill factor > 0.5 can result in a tree with more nodes than we would get by following the standard insertion algorithm taught in class.

B. Assuming records can fit on one page, looking up a single record using an alternative 2 index will always cost more I/Os than lookups that utilize an alternative 1 index

C. B+ trees can have composite primary keys of different data types (e.g. (Integer, String))

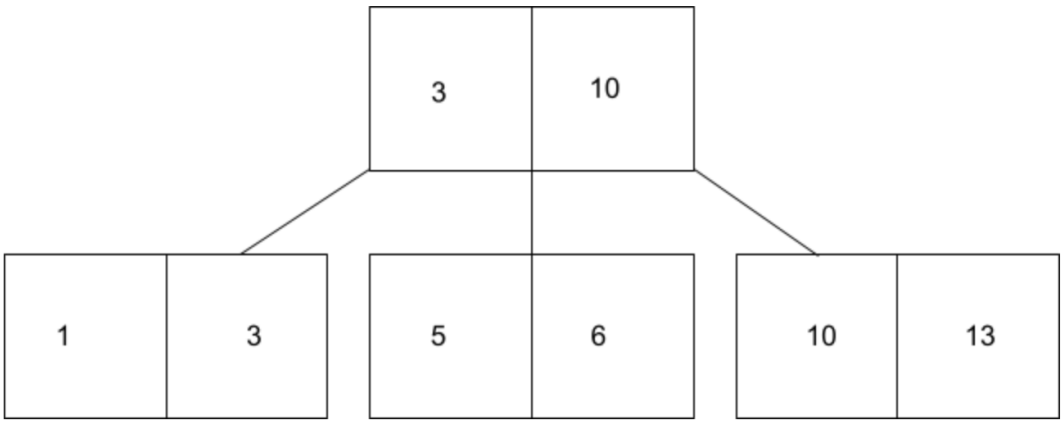
D. An alternative-3 index will always be at least as cost-efficient for lookups as an alternative-2 index in terms of I/Os.

E. None of the above.

- [x] A
- [ ] B
- [x] C
- [x] D
- [ ] E

(b) Tree Examination

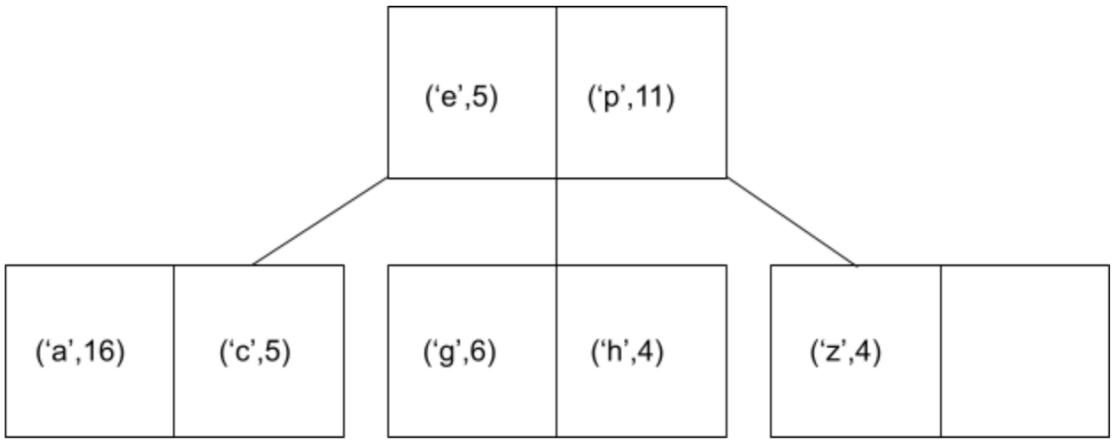
Under the rules taught in this course, decide if the following B+ Trees are possible **assuming no deletions and no rebalancing**. Remember that strings are sorted lexicographically 'a' being the lowest value to 'z' being the greatest.



i. (1 pt)

- ○ YES
- ● NO

No, the 3 cannot be on the left node.

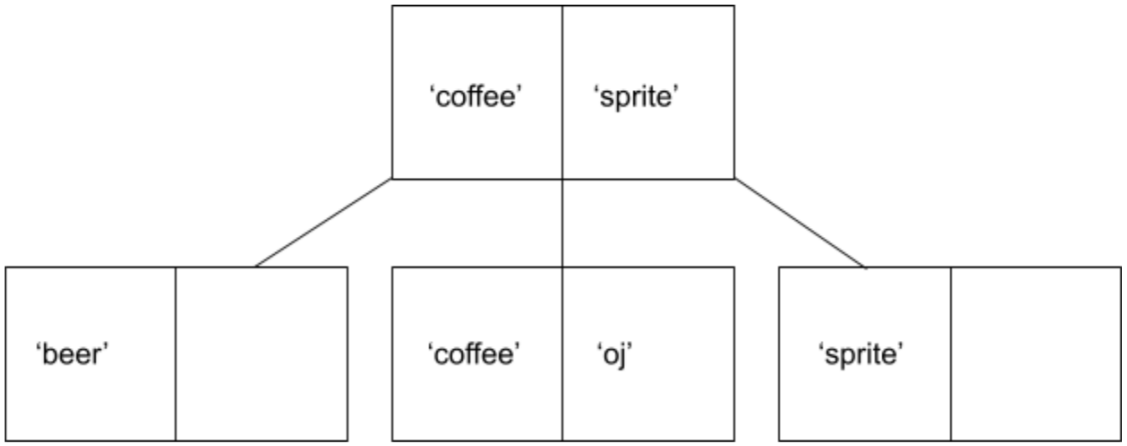


ii. (1 pt)

○ YES

● NO

No, the root node values aren't in the leaf nodes.

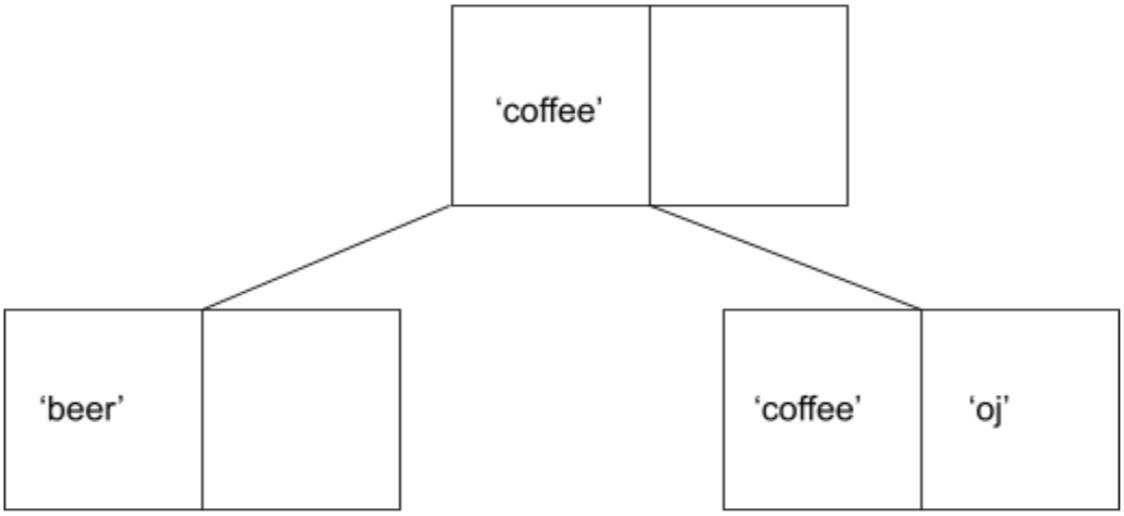


iii. (1 pt)

○ YES

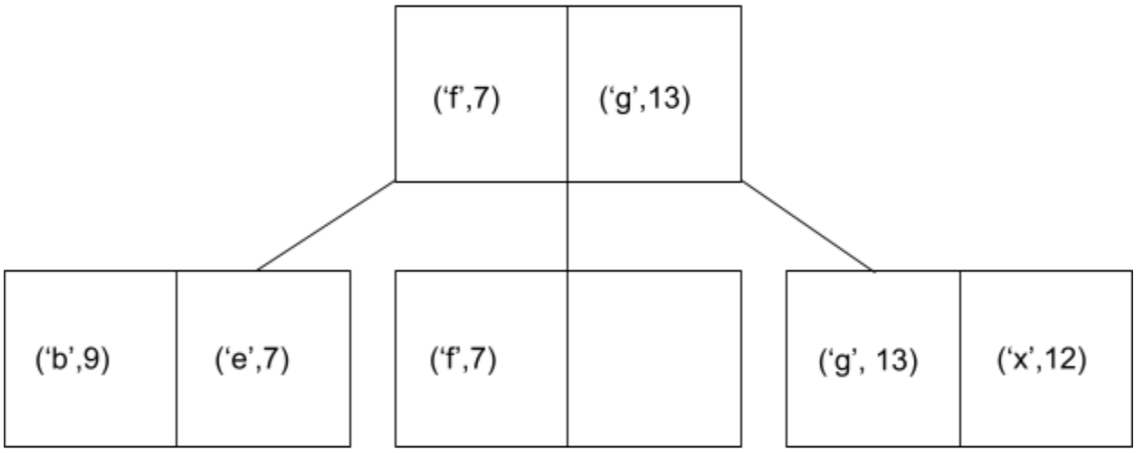
● NO

No ('oj' cannot end up in the center node). In a split, the right node would get 2 keys, and so 'oj' would have to end up in the rightmost node, not the center node.



iv. (1 pt)

- ● YES
- ○ NO

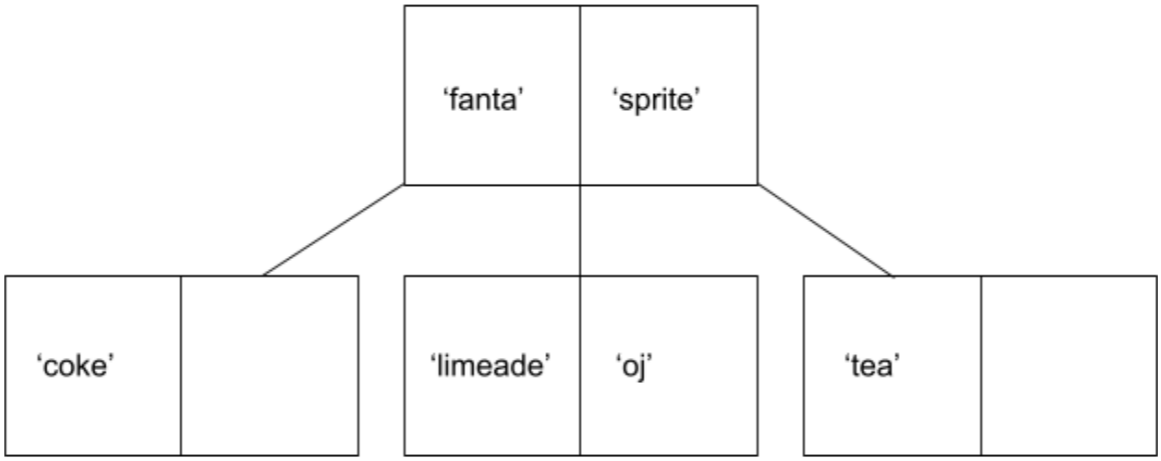


v. (1 pt)

- ● Yes
- ○ NO

Yes, this is possible – the left root key is in the middle node and the right root key is in the right node. Try the insertion pattern ('b', 9), ('f', 7), ('g', 13), ('x', 12), ('e', 7) to construct this tree.

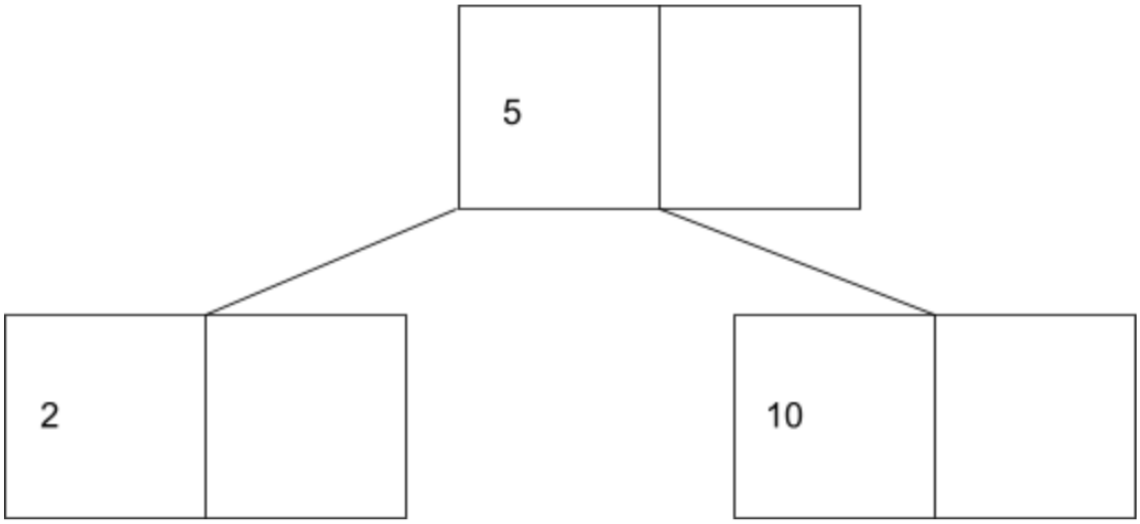
vi. **(2 pt)** Consider the following valid B+ tree which may have had any number of prior deletions and insertions. How many insertions will it take to change the height of the tree, at minimum?



1

1 – insert any key between 'limeade' and 'sprite' lexicographically.

vii. **(2 pt)** Consider the following valid B+ tree which may have had any number of prior deletions and insertions. What is the maximum number of insertions we can put in without changing the height of the tree?



4

4 – the number of keys we can hold in a tree of height 1 with order 1 is 6, and there are 2 entries here.

(c) (2 pt) Assume we have an alternative 2 B+ tree of height 4 and that the index is cached (in memory). Assume that all records can fit on a page. How many I/Os will it take to lookup and fetch a single record?

1

1 – the index is cached so we just need to go to disk to fetch the record needed.

(d) (2 pt) Assume we have an alternative 3 B+ tree of height 5. Assume that all records can fit on a page. How many I/Os will it take to lookup and fetch a single record?

7

7 – we need to read in the root node, then traverse 5 more nodes to get from root to leaf, and then we need to fetch the data page from disk.

(e) School Queries

A school decides to create a database to keep the academic records of all former and current students

```
CREATE TABLE students (
    sid INTEGER PRIMARY KEY,
    name TEXT,
    major TEXT,
    gpa FLOAT,
    is_enrolled BOOLEAN
);
```

This table is getting to be somewhat large, and so they build some indexes to speed up the queries they'll be performing.

Index A:

- Alternative 2 unclustered B+ Tree on column `gpa`
- Order 5 (d=5)
- Height 3

Index B:

- Alternative 3 clustered B+ Tree on column `major`
- Order 3 (d=3)
- Height 2

Assumptions:

- There are 80 records where gpa <= 2.0
- There are 20 records where gpa == 4.0
- There are 20 records where major = 'computer science'
- The root is NOT cached in memory.
- The buffer manager starts empty at the start of each query.
- The table contains a total of 100 data pages.
- **There also exists a page directory for the table, with 100 entries per page.**
- One data page can store up to 10 records.
- Assume the leaf nodes are full.

Answer the following questions. Please pay attention to whether we're looking for **best case (minimum I/O cost)** or **worst case**.

i. (2 pt) How many I/Os would it take, **in the worst case**, to execute the following query?

```
SELECT sid, major FROM students WHERE gpa <= 2.0 AND is_enrolled = 1
```

92

Use Index A. 3 + 8 + 80 = 91 I/Os. Because of the <= in the query, we will start from the leftmost node and after reading in 3 index pages, will read in 8 leaf nodes and the 80 matching records will incur 1 I/O each.

ii. (1 pt) Which index would we use for the above query?

- (●) A
- ( ) B
- ( ) NEITHER

iii. (2 pt) How many I/Os would it take, **in the best case**, to execute the following query?

```
SELECT name, gpa FROM students WHERE gpa == 4.0 AND major == 'computer science'
```

5

Use the alt. 3 index B on 'major'. 3 + 20/10 = 5 I/Os

iv. (1 pt) Which index would we use for the above query?

- ( ) A
- (●) B
- ( ) NEITHER

5. (16 points) Dead Week, Dead Computers

Dylan knocked over 6 computers in Soda by accident. These computers stored tables of students' **name and major** in each of 6 of UC Berkeley's colleges — each computer stored one table corresponding to a college. The tables were in **alphabetical order based on name**. The computers are destroyed but their disk drives are fine, so Dylan hooks them up to the only working computer left. The computer's operating system is unable to distinguish between each drive, so **all the data pages have been mixed up into one heap file. However, the records remain in the same order on each page.**

Here is information on each drive:

- Six drives: Optometry, Engineering, L&S, Haas, Law, Journalism
- Each drive has one file on it which represents the table, file sizes are:
  - Optometry: 70 pages
  - Engineering: 70 pages
  - L&S: 70 pages
  - Haas: 70 pages
  - Law: 70 pages
  - Journalism: 70 pages
- One page contains records for 10 students

(a) Restore Tables

The tables need to be restored before tomorrow so that they can be uploaded to the university website.

i. (4 pt) How can you use sorting and hashing to re-create the original six tables (before Dylan mixed them up)? Assume that you are able to access the major for a given name and that majors are unique to each college (i.e. the computer science major will only be in one college). Please be specific and clear about your approach.

Create hash algorithm that uses major to identify which college the person is in. Use the external hash algorithm to partition names/majors into six buckets. Then sort each bucket individually.

Manually graded

ii. (2 pt) If you stated you'd use a hash function in your answer to the previous question, would the function have to be uniform? Select N/A if you don't believe a hash function is necessary.

◯ Yes

● No

◯ N/A

The function itself isn't uniform – if we had an uneven number of pages across colleges, we wouldn't see uniform hashing.

(b) (10 points) Major Grouping

You would like to group students by major and decide to use hashing for this. There are 5 majors in each college. You have 31 buffer pages (B = 31). Assume all pages are full, and all majors are distinct.

i. (5 pt) What is the I/O cost to hash all of the students into partitions based on their majors? There should be one partition for each major. Assume you have access to any hash functions you need and students are evenly distributed across majors. **Show your work in the space below.**

1680

Note: The wording for this question was poor. It should've asked: what is the cost to hash all of the data pages? Additionally, since you can hash the majors into partitions such that each partition only has one major, you can skip the conquer phase – so we also gave full credit to 840 I/Os.

Total # pages - 420

Total # partitions - 30

420 / 30 = 30 partitions of 14 pages each

Divide Phase: 420 (Read) + 420 (Write)

Conquer Phase: 420 (Read) + 420 (Write)

1680 I/Os

ii. (5 pt) However, some students have two majors and thus need to be hashed into multiple buckets. Therefore, you have modified the original hashing algorithm to allow keys to hash to multiple buckets (your hash function will do this). Given that 300 students across all 6 colleges are doing double majors, what is the cost of hashing all the data pages? Assume 5 majors for each college, 31 buffer pages (B = 31), students are evenly distributed across majors (including second majors), and you have any hash functions you need. **Show your work in the space below.**

1770

Note: See note under previous question. For the same reasoning, 870 I/Os also received full credit here.

Total # pages - 420

Total # partitions - 30

300 students / 10 = 30 pages extra will be produced

420 / 30 = 30 partitions of 14 pages each, add 1 page to each partition for second major, so 15 pages per partition

Divide Phase:

420 (Read) + 450 (Write)

Conquer Phase: 450 (Read) + 450 (Write)

1770 I/Os

6. (19 points) True INNER JOIN False

Reminder: For $R \bowtie S$, the inner relation is S, and the outer relation is R.

(a) (1.5 pt) Block Nested Loop Join works on any arbitrary join conditions whereas Index Nested Loop Join and Grace Hash Join are only efficient for equijoins.

- [x] True
- [ ] False

True. BNLJ works on any arbitrary joins.

(b) (1.5 pt) If the inner relation for Index Nested Loop Join is sorted on the join key, the output will also be sorted on the join key.

- [ ] True
- [x] False

False. The ordering of the output depends on the **outer** relation instead of the inner relation for INLJ.

(c) (1.5 pt) Sort Merge Join can be sped up if there is an index built on either of the join relations.

- [x] True
- [ ] False

True. An index scan may be cheaper than sorting.

(d) (1.5 pt) The optimization for Sort Merge Join is only possible if the total number of runs from both relations in the last merge pass is less than or equal to B - 1.

- [ ] True
- [x] False

False. See notes for other optimization options. Points given to both answers.

(e) (1.5 pt) If the number of buffer pages increases, the cost of Block Nested Loop Joins always decreases.

- [ ] True
- [x] False

False. Consider the case when one of the relation can fit into B - 2 buffer pages - the cost does not change.

(f) (1.5 pt) In the build and probe phase of Grace Hash Join, if the probed hash table is smaller than B - 2 buffer pages, then we can reduce the join cost by increasing the number of input buffers with the extra pages.

- [ ] True
- [x] False

False. The cost will remain the same because we are making 1 pass through the data for both cases.

(g) (1.5 pt) For Sort Merge Join, we only reset the record iterator over the inner relation.

- [x] True
- [ ] False

True. We only advance the iterator over the outer relation.

(h) **(1.5 pt)** For Block Nested Loop Join, we only reset the record iterator over the inner relation.

- ○ True
- ● False

False. For each block of outer relation, we would need to scan all records for each record in the inner relation.

(i) **(1.5 pt)** In one partitioning pass of Grace Hash Join, we must use different hash functions to partition the two relations.

- ○ True
- ● False

False. We must use the same hash function to partition the two relations. We use different hash functions across different partition passes.

(j) **(1.5 pt)** For Index Nested Loop Join, the total number of times needed to traverse the index tree from the root to the leaf depends on the number of matching records in the inner relation.

- ○ True
- ● False

False. The number of traversals depends on the number of records in the outer relation.

(k) **(1 pt)** For Block Nested Loop Joins, if $[R] < [S]$, then we always have $cost(R \bowtie S) \leq cost(S \bowtie R)$.

- ○ True
- ● False

False. See the next question for a counterexample.

(l) **(3 pt)** If you answered false to the previous question, give an example specifying

i. number of buffer pages B
ii. $[R]$ and $[S]$
iii. $cost(R \bowtie S)$
iv. $cost(S \bowtie R)$

If you answered True to the previous question, give a justification.

i. $\mathbf{B = 5}$
ii. $\mathbf{[R] = 5,\ [S] = 6}$
iii. $\mathbf{cost(R \bowtie S) = 5 + 2 * 6 = 17}$
iv. $\mathbf{cost(S \bowtie R) = 6 + 2 * 5 = 16}$

7. (17 points) Query OptimAXEation

(a) Query OptimAXEer

Consider the following tables:

```
CREATE TABLE X {
    cal INTEGER,
    beat INTEGER,
    stanfurd INTEGER
}

CREATE TABLE Y {
    our FLOAT,
    axe FLOAT
}
```

Number of pages and records per page:

```
X: 100 pages, 10 records/page
Y: 40 pages, 100 records/page
```

Index:

```
X: Alt 1 index on X.cal with h = 2 and 100 leaf pages
Y: Alt 2 unclustered index on Y.our with h = 2 and 20 leaf pages
```

```
Table Stats: If the statistic does not specify a uniform distribution,
you may not make any assumptions.

X.cal: min = 1, max = 50, 20 unique values, uniform distribution
X.beat: No information available
X.stanfurd: min = 1, max = 20, 5 unique values, uniform distribution
Y.our: min = 0, max = 100, uniform distribution
Y.axe: min = 0, max = 50, uniform distribution
```

For each question, use selectivity estimation to estimate the **number of I/Os** produced by the following queries. **You may not make any assumptions about independence.**

i. (2 pt)

```
SELECT * from X
WHERE X.cal < 30;
```

60

The selectivity will end up being 29/50. In this case, we will go down to the leaf level using our index, and read 58% of the leaf nodes. This results in .6 * 90 = 58 IOs. We also need to take into account the height of the tree, which adds another 2 IOs, giving us a total of 60 IOs.

ii. (2 pt)

```
SELECT * FROM Y
WHERE Y.our > 25 AND Y.our < 100;
```

1517 OR 40

The selectivity here is (100-25)/100 = 0.75. In this case, we will be reading 15 leaf pages, and each leaf page can hold 100 records. Because this is an unclustered index, we incur 1 IO per tuple, giving us 100 * 15 IOs. we also need to take into account our height of the tree, which is 2. This gives us the total of 2 + 15 + 100*15 IOs, or 1517 IOs.

It is also possible to do a less costly full scan on the table, since it is 40 pages long. This results in an IO cost of 40. Note that this would not cost 30 IOs because we cannot guarantee the pages of Y have ordered records/are ordered themselves.

(b) (3 pt) Here is a query that we would like to execute. This question is independent from previous parts and **assume that you are given the appropriate tables**.

```
SELECT *
    FROM A INNER JOIN X ON A.a = X.a,
    E
    Order by A.b, E.b;
```

Which of the following query plans will be retained by the first pass of the Selinger Optimizer given the total estimated I/O cost and output order?

```
A: Full scan on A   I/Os: 2000  Order: None
B: Index scan on A  I/Os: 2700  Order: A.a
C: Index Scan on A  I/Os: 2300  Order: A.b
D: Full scan on X   I/Os: 1500  Order: None
E: Index scan on X  I/Os: 3000  Order: X.a
F: Index scan on X  I/Os: 1600  Order: X.b
G: Full scan on E   I/Os: 2000  Order: None
H: Index scan on E  I/Os: 2200  Order: E.b
```

- [x] A
- [x] B
- [x] C
- [x] D
- [x] E
- [ ] F
- [x] G
- [x] H

A is correct because it is the minimum cost operation. B and C are correct because they are interesting orders (you inner join on A.a later, and order by A.b later). D and E are correct because D is the minimum cost operation to go through table D, and E is correct because it is used downstream and is an interesting order. F is not correct because X.b is not used anywhere. G is the minimum cost IO for table E, and H is an interesting order because we order by E.b later in the query.

(c) (3 pt) Here is the query we want to execute again:

```
SELECT *
    FROM A INNER JOIN X ON A.a = X.a,
    E
    Order by A.b, E.b;
```

Which of the following query plans will be returned by the second pass of the Selinger Optimizer?

Assume the following query plans are independent of the information given about A, X and E earlier.

```
A: A BNLJ X          I/Os: 500   Order: None
B: A GHJ X           I/Os: 480   Order: None
C: A SMJ X           I/Os: 400   Order: A.b
D: X SMJ A           I/Os: 600   Order: X.a
E: X BNLJ E          I/Os: 200   Order: None
F: E BNLJ X          I/Os: 550   Order: None
G: A BNLJ E          I/Os: 210   Order: None
H: E GHJ A           I/Os: 400   Order: None
```

- ☐ A
- ■ B
- ☐ C
- ☐ D
- ☐ E
- ☐ F
- ☐ G
- ☐ H

A is incorrect because it is not the minimum cost join, and there is no interesting order. C is incorrect because we should be joining on A.a, not A.b. D is incorrect because it's not the minimum cost join and the order is not interesting (X.a is not used downstream). E, F, G, and H should not be considered in this pass because they are cross joins, and we consider cross joins as late as possible.

(d) (3 pt) Here is the query we want to execute again. Given the following choices, which are possible combinations for our joins? **Assume this question is independent from parts b and c.**

```
SELECT *
    FROM A INNER JOIN X ON A.a = X.a,
    E
    Order by A.b, E.b;
```

```
A: {A,X} Join E
B: {X,A} Join E
C: {A,E} Join X
D: {E,A} Join X
E: {X,E} Join A
F: {E,X} Join A
G: E Join {A,X}
H: E Join {X,A}
I: X Join {A,E}
J: X Join {E,A}
K: A Join {X,E}
L: A Join {E,X}
```

- [x] A
- [x] B
- [ ] C
- [ ] D
- [ ] E
- [ ] F
- [ ] G
- [ ] H
- [ ] I
- [ ] J
- [ ] K
- [ ] L

A and B are correct because they are left deep and consider cross joins as late as possible. C,D,E and F are incorrect because they do not join E at the end. G, H, I, J, K, L are not left deep, so we would not use those plans under Selinger.

(e) (4 pt) Select all statements regarding Selinger Optimizer which are true.

- ☐ A. A right deep plan is equivalent to a left deep plan as long as we are consistent with the order we join in.
- ■ B. One way that we can reduce the number of total IOs during query optimization is writing intermediate operations back to disk
- ☐ C. In pass N of the Selinger optimizer, we join N joined tables with another set of joined tables.
- ☐ D. We never deal with cross joins when dealing using the Selinger optimizer
- ☐ E. None of the above

A is not correct because the reason why we use left deep plans is so that we do not have to keep streaming our output in over and over as we join more tables - we want to stream in the new table. B was given credit for either selecting or not selecting. The reason for this is that we do not materialize to disk during Selinger, but it is indeed an optimization that can occur to reduce the total number of IOs. D is incorrect because we deal with cross joins, but we try to push it off as late as possible.

8. (20 points) Transactions and Concurrency

(a) Schedules

For the following three questions, determine if the provided schedules are serial, serializable, and/or conflict serializable.

i. (2 pt) Given the following schedule, select the appropriate boxes that describe the schedule:

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

- [x] A. Serial
- [x] B. View Serializable
- [x] C. Conflict serializable
- [ ] D. None of the above

This schedule is serial because each transaction runs from start to finish without any interleaving actions. Therefore, it follows that it is also conflict serializable and view serializable (See Discussion 8 Slide 12).

ii. (2 pt) Given the following schedule, select the appropriate boxes that describe the schedule:

	1	2	3	4	5	6	7	8
T1	R(B)				W(B)			
T2			R(B)				W(C)	

- [ ] A. Serial
- [x] B. View Serializable
- [x] C. Conflict serializable
- [ ] D. None of the above

This schedule is not serial because each transaction does not run from start to finish without any interleaving actions. This schedule is conflict serializable since it is conflict equivalent to a serial schedule. It follows that it is also view serializable (See Discussion 8 Slide 12).

iii. (2 pt) Given the following schedule, select the appropriate boxes that describe the schedule:

	1	2	3	4	5	6	7	8
T1	W(B)				W(B)			
T2			W(B)				W(C)	

- [ ] A. Serial
- [x] B. View Serializable
- [ ] C. Conflict serializable
- [ ] D. None of the above

This schedule is not serial because each transaction does not run from start to finish without any interleaving actions. This schedule is not conflict serializable since the conflict dependency graph

would contain a cycle. This schedule is view serializable since it is view equivalent to a serial schedule.

iv. (2 pt) Select all the statements that are always true.

- [x] A. Both Strict 2PL and 2PL enforce conflict serializability.
- [ ] B. The order of blind writes will affect whether a schedule is serializable.
- [ ] C. A schedule S is serializable if and only if it is conflict equivalent to a serial schedule.
- [x] D. Two schedules are conflict equivalent if every conflict is ordered the same way.
- [ ] E. None of the above

A. True; by definition. B. False; see Discussion 8 Slide 18. C. False; conflict serializable schedules are also serializable, but not the other way around. Some serializable schedules are not conflict serializable. D. True; this is the definition of conflict equivalent.

(b) Conflict Serializability

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

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

- [x] A. T1
- [ ] B. T2
- [x] C. T3
- [ ] D. T4
- [ ] E. None of the above

Order of conflicts: T1 -> T2 (due to T1's R(A) conflicting with T2's W(A)) T3 -> T2 (due to T3's R(C) conflicting with T2's W(C)) T2 -> T1 (due to T2's W(A) conflicting with T1's W(C)) T3 -> T4 (due to T3's R(B) conflicting with T4's W(B))

Therefore, T1 and T3 point to T2 in the conflict dependency graph.

ii. (1 pt) Is the schedule provided conflict serializable?

- ○ Yes
- ● No

No, there is a cycle in the conflict dependency graph (see explanation in previous part).

(c) Multigranularity Locking and Deadlock

Use the following schedule of lock requests to answer the following questions.

	1	2	3	4	5	6	7	8
T1	S(A)			S(B)				
T2			X(B)		X(C)		X(D)	
T3		S(D)				X(A)		
T4								S(C)

Assume that transactions will hold onto locks they obtain for the entirety of the schedule provided. If a transaction aborts, assume that it remains inactive for the remainder of the schedule. For this question, transactions are assigned priority by the time of their first access/operation.

i. (2 pt) What transactions are involved in deadlock(s) (if any) for this schedule?

- [x] A. T1
- [x] B. T2
- [x] C. T3
- [ ] D. T4
- [ ] E. None of the above

Waits T4 -> T2 (T4's S(C) needs to wait for T2's X(C)) T2 -> T3 (T2's X(D) needs to wait for T3's S(D)) T3 -> T1 (T3's X(A) needs to wait for T1's S(A)) T1 -> T2 (T1's S(B) needs to wait for T2's X(B))

Notice that T1, T2, and T3 are involved in deadlock (there is a cycle in the waits-for graph)

ii. (1 pt) Assuming no use of deadlock avoidance algorithms, do any of the transactions in this schedule complete?

- ○ A. Yes
- ● B. No

Not transaction is able to complete, since each transaction is waiting for another transaction in order to make progress.

iii. (3 pt) Mark all statements that are true.

- [x] A. Using a "wait-die" deadlock avoidance policy will avoid all scenarios of deadlock.
- [ ] B. Deadlock avoidance is always strictly a better option (fewer transactions will be aborted) when compared against deadlock detection.
- [ ] C. The waits-for and the conflict-dependency graph can be substituted for each other depending on the schedule provided.
- [ ] D. There is a scenario where a specific transaction will wait under both a wait-die policy and a wound-wait policy.
- [ ] E. 2PL is implemented to prevent the occurrence of cascading aborts.
- [ ] F. None of the above

A. True; wait-die policy is a deadlock avoidance policy, therefore when used any potential instance of deadlock will be avoided. B. False; deadlock avoidance may result in aborting many transactions, whereas by detecting deadlocks we will abort one of the transactions in the deadlock to allow others to continue. C. False, the waits-for and conflict dependency graph describe fundamentally different aspects of our schedule. Conflict dependency graph looks to find if a schedule is serializable, whereas the waits-for graph is used to detect deadlocks. D. False; this is not possible. If a transaction aborts in one policy, then it will wait in the other policy (in wait-die high priority transactions are aborted, whereas in wound-wait the same high priority transaction will wait). E. False; 2PL does not prevent cascading aborts because in 2PL there is no constraint that all locks get released together when the transaction completes.

iv. (3 pt) Mark all statements that are true.

- [ ] A. Multigranularity locking by itself is enough to enforce the consistency.
- [x] B. There exists a scenario where one transaction can hold IS lock and another transaction can hold an IX lock on the same resource.
- [ ] C. For promoting from an IS/IX lock to a SIX lock, it is not necessary to release the SIS descendants.
- [x] D. In Strict 2PL, after a transaction has been aborted and the rollback is complete, it is safe to release all locks held by the transaction.
- [ ] E. When performing a write at the tuple level, X locks must be acquired at the database, table, page, and the tuple level.
- [ ] F. None of the above

A. False; While multigranularity locking helps enforce consistency, it is not sufficient by itself to guarantee consistency. In notes/lecture, we explicitly covered how it adheres to the isolation property in ACID. B. True; IS and IX locks are compatible, as per the lock compatibility matrix. C. False; In the project, S/IS descendants are released when promoting to a SIX lock because having a S/IS lock on a descendant when a SIX lock is held higher is not allowed. D. True; by definition of Strict 2PL. E. False; IX locks will be acquired at the database, table and page level and a X lock will be acquired at a tuple level.

9. (17 points) Distributed Transactions

Professor Natacha Crooks and her favorite PhD students, Audrey and Neil, are writing a paper together. Before submitting, they must all agree on the finished draft. They decide to use **2PC** (do not assume presumed abort), with Professor Crooks as the coordinator, in order to achieve consensus.

- It takes 2 minutes for Audrey and Neil to receive Professor Crooks' message.
- It takes 1 minute for Professor Crooks to receive Audrey's response.
- It takes 10 minutes for Professor Crooks to receive Neil's response.
- Assume that everyone takes 1 minute to log and flush (15 secs to log and 45 secs to flush).

(a) (1 pt) In the best case scenario, how many minutes will Professor Crooks have to wait, starting from the beginning of the protocol, before she gets the VOTE back from **both** Audrey and Neil?

i. 2 min from Professor Crooks to Audrey & Neil + 1 min for each to log and flush "VOTE YES" + 10 min to receive Neil's response = 13 min. Note that Audrey's response arrives first as Professor Crooks waits for Neil.

(b) (1 pt) In the best case scenario, how many minutes does 2PC take, starting from the beginning of the protocol to after the "END_TRANSACTION" is flushed?

i. 13 from above + 1 min for Professor Crooks to log and flush "COMMIT" + 2 min from Professor Crooks to Audrey & Neil + 1 min for both to log and flush "COMMIT" + 10 min to receive Neil's response + 1 min for Professor Crooks to log and flush "END_TRANSACTION" = 13 + 1 + 2 + 1 + 10 + 1 = 28.

(c) Commit

Both Audrey and Neil approve of the draft and will vote to COMMIT.

i. (2 pt) Who is the **first person** to decide to COMMIT? Select the best answer.

- ○ A. Audrey
- ○ B. Neil
- ● C. Professor Crooks. She sees that everyone voted "YES", including her self, after receiving Audrey and Neil's votes.
- ○ D. Audrey and Neil
- ○ E. Professor Crooks, Audrey, and Neil

ii. (1 pt) How many minutes will it take, starting from the beginning of the protocol, for the **first person** to decide to COMMIT?

Remember that once someone decides to COMMIT, they cannot change their mind, even after recovery.

A. 13 min to receive votes + 1 min for Professor Crooks to log and flush. If she did not log and flush, she might decide COMMIT, crash, then during recovery decide ABORT.

iii. (2 pt) Who is the **last person** to decide to log a COMMIT record? Select the best answer.

- ○ A. Audrey
- ○ B. Neil
- ○ C. Professor Crooks
- ● D. Audrey and Neil. Both Audrey and Neil receive Professor Crook's "COMMIT" simultaneously and will decide to COMMIT.
- ○ E. Professor Crooks, Audrey, and Neil

iv. (1 pt) How many minutes will it take, starting from the beginning of the protocol, for the **last person** to decide to COMMIT?

A. As above, after 16 min, both Audrey and Neil receive "COMMIT". Both can decide COMMIT without logging and flushing, because on failure and recovery, they will see "VOTE YES" in their log and ask the coordinator (Professor Crooks), who will tell them COMMIT.

(d) Recovery

Recall that Professor Crooks is the coordinator.

Shadaj is very evil. He offers Professor Crooks a $1 million research grant from Facebuck, on the condition that she sabotages the paper. She might pretend to fail and take a long time to recover. Assume that both Audrey and Neil still vote to COMMIT.

i. (2 pt) Professor Crooks wants to fail such that either Audrey or Neil does not know, **based on their local logs**, whether the decision was COMMIT or ABORT. Select *all* scenarios in which this is true.

- [x] A. Professor Crooks fails before sending the COMMIT message. Neither Audrey nor Neil see what each other voted.
- [x] B. Professor Crooks fails after sending a COMMIT message to Audrey (but not to Neil). Neil does not see what Audrey voted.
- [ ] C. Professor Crooks fails after sending all COMMIT messages.
- [ ] D. Professor Crooks fails before flushing "END TRANSACTION".

ii. (2 pt) Audrey and Neil realize **they can communicate with each other** to find out whether the transaction committed. Select *all* scenarios in which one of them would still not know, without asking Professor Crooks, whether the decision was COMMIT or ABORT.

- [x] A. Professor Crooks fails before sending the COMMIT message. Professor Crooks can fail before logging & flushing COMMIT. Then, when she recovers, she will ABORT. Therefore Neil and Audrey must know that she has logged & flushed before deciding.
- [ ] B. Professor Crooks fails after sending a COMMIT message to Audrey (but not to Neil).
- [ ] C. Professor Crooks fails after sending all COMMIT messages.
- [ ] D. Professor Crooks fails before flushing "END TRANSACTION".

(e) Optimizations

Shreya likes to go fast. She wants to optimize 2PC to reduce the amount of logging necessary. She decides to use **presumed abort**.

i. (3 pt) We know that Neil does not have to log and flush if he: receives "ABORT" in phase 2, then fails and recovers. Assume that all others do not fail and all network messages have been delivered. Why does Neil not have to log and flush? Select *all* explanations that are correct.

Note: We do not know if Neil's "ACK" was ever sent.

- [x] A. During recovery, he can ask Audrey (ignoring normal protocol). She will tell him to ABORT.
- [x] B. During recovery, he can ask Professor Crooks. She will tell him to ABORT.
- [x] C. During recovery, he might not see anything in his log, so he'll decide to ABORT.

ii. (1 pt) Conor is impatient. He wants to know whether the paper was COMMITted as soon as possible. When is the **earliest** Shreya is allowed to tell him that the paper was **COMMIT**ted?

- ◯ A. After Neil *or* Audrey logs and flushes "YES".
- ◯ B. After Neil *and* Audrey log and flush "YES".
- ● C. After Professor Crooks logs and flushes "COMMIT". Similar reasoning to earlier question on Natacha failing before sending COMMIT.
- ◯ D. After Neil *or* Audrey logs and flushes "COMMIT".
- ◯ E. After Neil *and* Audrey log and flush "COMMIT".

iii. (1 pt) When is the **earliest** Shreya is allowed to tell him that the paper was **ABORT**ed?

- ● A. After Neil *or* Audrey vote "NO", before sending the vote. Using presumed abort, if Neil or Audrey fail, they will abort anyway, so they don't need to log & flush. Sending the vote is an indication to Professor Crooks that they have logged & flushed, so sending the vote is not necessary.
- ◯ B. After Neil *and* Audrey vote "NO" and Professor Crooks receives Audrey's vote.
- ◯ C. After Neil *and* Audrey vote "NO" and Professor Crooks receives *both* votes.
- ◯ D. After Neil *or* Audrey receives "ABORT" from Professor Crooks.
- ◯ E. After Neil *and* Audrey receives "ABORT" from Professor Crooks.

16. (19 points) Parallel Query Processing

(a) For the following 4 questions, suppose we have tables R and S, with [R] = 100 and [S] = 50. Each page is 4KB. We have 5 machines but currently both R and S are on machine 1. Each machine has 5 buffer pages.

i. (2 pt) What is the network cost if we want to perform parallel Grace Hash Join on R and S? Assume we have a perfect hash function.

4/5 * 150 * 4 = 480 KB

ii. (3 pt) What is the total number of disk I/Os needed across all machines for the Grace Hash Join from the previous question? Do not include the initial read cost for sending data through the network.

On each machine, there are 20 pages of R and 10 pages of S. For R, we read [R], divide it into 4 partitions of 5 pages each, and write the 4 partitions. For S, we read [S], divide it into 4 partitions of 3 pages, and write the 4 partitions. At this point, we can build the in-memory hash table, so we need to read the 4 partitions of each table. So on each machine, the I/O cost is 20 + 5(4) + 5(4) + 10 + 3(4) + 3(4) = 94. Across all the machines, we have 5 * 94 = 470 I/Os in total.

iii. (3 pt) Now assume that we have already range partitioned R and S across 5 machines such that each machine has the same amount of data for each table. What is the total number of disk I/Os needed across all machines to perform Sort Merge Join without any optimization?

Each machine has 20 pages of R and 10 pages of S. On each machine, we need 2(20)(1+$log_4(4)$) + 2(10)(1+$log_4(2)$) + 20 + 10 = 150 I/Os. For 5 machines, it will be 750 I/Os.

iv. (3 pt) What is the answer to the previous question if we apply optimizations, if possible?

We can't save 2([R]+[S]) because there are 4 runs of R and 2 runs of S in the pass before merging. We also can't save 2[R] because runs(R) + 1 <= 4 is not satisfied. We can save 2[S] because runs(S) + 1 <= 4. So the I/O cost becomes 750 – 5(2)(10) = 650 I/Os.

(b) **(2 pt)** Suppose we have a table R with [R] = 10,000 and it is hash partitioned equally across 5 machines. We want to join R with table S with [S] = 50. Currently 20 pages of S is on machine 1 and 30 pages of S is on machine 2. Each page is 4KB. What is the network cost if we perform a broadcast join?

20(4)(4) + 30(4)(4) = 800 KB

(c) For the rest of this question, we have a Students table with the following schema: Students: (sid, name, score, age) Consider the following query: SELECT avg(score) FROM Students WHERE age < 20; Assume that each disk I/O takes 5ms.

i. **(2 pt)** If Students is range partitioned on sid across 3 machines where m1 has 50 pages of data consisting of sids [0, 100), m2 has 40 pages of data consisting of sids [100, 200), and m3 has 20 pages of data consisting of sids [200, 300). How long (in ms) does it take to execute the query, excluding the time it takes to aggregate the results?

We need to look at all pages in parallel. So the time it takes will be the longest time out of all machines, which is 50 * 5 = 250ms.

ii. **(2 pt)** If Students is round robin partitioned across 3 machines. How many disk I/Os across all machines does it cost to execute the query?

With round robin partitioning, each machine has $ceil(110/3)$ **= 37 pages of data. We need to look at all pages, so we will need 37 * 3 = 111 I/Os in total.**

iii. **(2 pt)** If Students is range partitioned on age where m1 has 20 pages of data consisting of ages [0, 10), m2 has 30 pages of data consisting of ages [10, 20), and m3 has 60 pages of data consisting of ages [20, 100). How long (in ms) does it take to execute the query, excluding the time it takes to aggregate the results?

We only need to look at m1 and m2. So the time it takes will be the longest time out of m1 and m2, which is 30 * 5 = 150ms.

17. (22 points) Recovery

Assume for all questions below that we are running the ARIES recovery algorithm.

(a) (1 pt) Which database properties does recovery enforce? Select all that apply.

- [x] A. Atomicity
- [ ] B. Consistency
- [ ] C. Isolation
- [x] D. Durability

With recovery, we're able to ensure that all operations from transactions that have not yet committed are undone (atomicity - we want either all or no operations of a transaction to persist), and we're able to ensure that committed transactions persist through redoing their operations (durability).

(b) (1 pt) Which of the following are true about disk? Select all that apply.

- [x] A. Writing to disk is much slower than writing to memory.
- [x] B. Data written to disk is persistent even when the database crashes.
- [ ] C. Evicting a page from the buffer pool means we read it in from disk.
- [ ] D. Dirty pages are pages with changes that are reflected on disk but not in memory.

A - Writing to a disk is much much slower than writing to memory (it's why we measure performance and costs I/Os). B - disk is persistence while memory is not. C - evicting a page from the buffer means we write it to disk. D - dirty pages are pages with changes reflected in memory but not on disk.

(c) (1 pt) ARIES uses a no-force policy because... (select one answer.)

- ○ A. we write all dirty pages to disk upon committing so we can ensure that all changes persist to disk even if the database crashes.
- ● B. we reduce the amount of unnecessary writes to disk.
- ○ C. it decreases the latency of writing dirty pages to disk when committing.
- ○ D. we eliminate the need for checkpointing in the log.

(d) (1 pt) ARIES uses a steal policy because... (select one answer.)

- ● A. we increase performance since memory is not locked up by dirty pages.
- ○ B. we write all dirty pages to disk upon committing so we can ensure that all changes persist to disk even if the database crashes.
- ○ C. it is much easier to enforce atomicity since we keep dirty pages in memory until we commit
- ○ D. we reduce the amount of unnecessary writes to disk.

(e) **(1 pt)** Which of the following is true? Select one answer.

- ○ A. We need to write the log records to disk **before** writing the corresponding dirtied pages because we need to ensure that we can recover all log records that were present in memory before the crash occurs.
- ● B. We need to write the log records to disk **before** writing the corresponding dirtied pages because we need to ensure that we have enough information to undo changes that made it to disk if the database crashes.
- ○ C. We need to write the log records to disk **after** writing the corresponding dirtied pages because we need to ensure that we can recover all log records that were present in memory before the crash occurs.
- ○ D. We need to write the log records to disk **after** writing the corresponding dirtied pages because we need to ensure that we have enough information to undo changes that made it to disk if the database crashes.

ARIES uses write ahead logging, meaning we need to flush log records to disk before the corresponding pages. A - we won't recover all log records in memory before the crash, only the ones that were flushed to disk since disk is persistent. B - this is correct. If we flush the dirtied pages to disk and the database crashes before we're able to write the corresponding log record, then we won't have any information about what operation that modified the page. Therefore, we need to flush the log record beforehand.

(f) **(1 pt)** Suppose we want to design a new recovery algorithm where we don't log CLR records. Can we still properly recover? Select one answer.

- ● A. Yes, logging CLR records is actually optional; we can ensure atomicity and undo transactions without them and we only log them for convenience upon recovery.
- ○ B. No, we must log CLR records in order to identify that we want to undo a given transaction.
- ○ C. No, undoing operations modifies pages so we need to ensure that we can undo these changes if the database crashes.
- ○ D. No, undoing operations modifies pages so we need to ensure that we can redo these changes if the database crashes.

We'll still be able to recover properly. Suppose we don't log CLR records and we're currently in the process of aborting a transaction when the database crashes. During recovery, we'll still end up aborting and undoing all operations of that transaction.

(g) **(1 pt)** In project 5, we flush the log upon writing a commit. Do we need to flush on abort? Select one answer.

- ○ A. Yes, we need to ensure that we track all changes and have all the necessary log records to abort the transaction upon recovery.
- ○ B. Yes, without flushing the log, we will not be able to identify which transactions to abort and may commit the wrong transactions.
- ● C. No, upon recovery, we will abort all transactions that have not been committed so we don't need to flush the log.
- ○ D. No, we don't flush the log because we want to follow the no-force policy

(h) (1 pt) Why do we checkpoint? Select one answer.

- ○ A. We checkpoint occasionally for space efficiency reasons since we can delete all log records written before the checkpoint.
- ○ B. We checkpoint occasionally to ensure that we don't lose data regarding the state of the log since without this data, we wouldn't be able to recover properly.
- ● C. We checkpoint occasionally for performance reasons since during recovery, we can begin recovery at the latest checkpoint.

(i) (1 pt) Why do we need both a begin and end checkpoint record? Select one answer.

- ○ A. When checkpointing, we indicate that we save all log records between the begin and end checkpoints records.
- ○ B. Since we checkpoint in the background while transactions are running concurrently, we must lock the state of the tables so that we don't end up with an inconsistent state.
- ● C. We checkpoint in the background while transactions are running concurrently so the saved tables can be the state of tables at any point between these two records.

(j) (1 pt) What are the goals of the analysis phase of the ARIES recovery algorithm? Select all that apply.

- ☐ A. To determine which committed transactions to redo to ensure that committed transactions persist.
- ■ B. To reconstruct the state of the dirty page table and transaction table.
- ☐ C. To analyze the log records and data lost from the database crash.
- ■ D. To determine which transactions need to be undone to ensure atomicity.

A - we redo all operations regardless of whether the transaction is committed. B - during analysis, we update the dpt and transaction tables using the log. C - we are unable to access data lost during the crash. D - after analysis, the only transactions remaining in the transaction table are the ones that are aborting. These are the transactions we undo during the UNDO phase.

Suppose our database just crashed. We recovered the log below and **an empty transaction table and dirty page table**. Answer the following questions below about the analysis phase.

LSN	Description	prevLSN
10	T1 Updates P1	0
20	T2 Updates P3	0
30	T1 Updates P2	10
40	T1 COMMITS	30
50	T3 Updates P2	0
60	T2 Updates P1	20
70	T2 ABORTS	60
80	CLR: undo LSN 60 (undoNextLSN: 20)	70
90	T1 ENDS	40

(k) **i. (1 pt)** Is it possible that there were more log records present in memory before the database crash beyond the ones recovered above? Select one answer.

- ○ A. Yes, during a database crash, we lose a random portion of our log that we are unable to recover regardless of whether it was in memory or on disk.
- ● B. Yes, data that was not written to disk was lost during the database crash.
- ○ C. No, we use write-ahead logging to ensure that we always are able to recover the entire log.
- ○ D. No, once the log record has been written to memory, it will persist in the event of a database crash.

Log records that were not written to disk (log records present only in memory) would be lost during the database crash.

ii. (1 pt) Is T1 in the transaction table after analysis completes?

- ○ Yes
- ● No

T1 ends at LSN 90 and is removed from the transaction table.

iii. (1 pt) What is T1's lastLSN after analysis completes? If T1 is not present in the transaction table, write N/A.

N/A

iv. (1 pt) Is T2 in the transaction table after analysis completes?

- ● Yes
- ○ No

v. (1 pt) What is T2's lastLSN after analysis completes? If T2 is not present in the transaction table, write N/A.

80

T2 aborts at LSN 70. We leave aborting transactions in the transaction table. The latest operation by T2 was at LSN 80. 100 or 110 were also accepted as answers for those counting the ABORT record written at the end of analysis.

vi. (1 pt) Is T3 in the transaction table after analysis completes?

- ● Yes
- ○ No

vii. **(1 pt)** What is T3's lastLSN after analysis completes? If T3 is not present in the transaction table, write N/A.

50

T3 is still running so we change its status to aborting and leave it in the transaction table. The latest operation by 3 was at LSN 50. 100 or 110 were also accepted as answers for those counting the ABORT record written at the end of analysis.

viii. **(1 pt)** What is P1's recLSN after analysis completes?

10

The earliest operation to dirty P1 occurred at LSN 10.

ix. **(1 pt)** What is P2's recLSN after analysis completes?

30

x. **(1 pt)** What is P3's recLSN after analysis completes?

20

(l) Regardless of what you got for the previous part(s), suppose you reconstruct the following transaction table and dirty page table from the analysis phase. Suppose you have the following log below **which is different from the one in the previous part**. Answering the following questions below about the redo and undo phases.

LSN	Description	prevLSN
10	T1 Updates P3	0
20	T1 Updates P1	10
30	T3 Updates P2	0
40	T1 Updates P4	20
50	T3 COMMITS	30
60	T2 Updates P2	0
70	T1 ABORTS	40
80	CLR: T1 undo LSN 40 (undoNextLSN: 20)	70
90	CLR: T1 undo LSN 20 (undoNextLSN: 10)	80

Transaction table:

Transaction Number	lastLSN	Status
T1	90	ABORTING
T2	60	ABORTING

Dirty page table:

Page Number	recLSN
P1	20
P2	30
P3	10
P4	40

i. (1 pt) List the LSNs of the records that we redo during the REDO phase (in the order they are redone). Separate the LSNs with commas.

10, 20, 30, 40, 60, 80, 90

We redo all operations (except for COMMITS, ABORTS, and ENDS) starting at the lowest recLSN in the DPT.

ii. (1 pt) List the LSNs of the records that we undo during the UNDO phase (in the order they are undone). Separate the LSNs with commas.

60, 10

The priority queue starts with the lastLSNs of all transactions (90, 60). We repeatedly poll for the highest LSN in the pq. 90 is not undoable (it was already redone in the REDO phase) so we do nothing and add the undoNextLSN (10) to the queue. We get 60 next and undo it. Last, we get LSN 10 and undo it. Using this algorithm, we're successfully able to undo all operations of the aborting transactions.

18. (1 points) Atlas Data Lake

(a) (0.5 pt) What query language does Atlas Data Lake use to run queries?

- ○ A. SQL
- ○ B. SQLite
- ● C. MQL
- ○ D. GraphQL

(b) (0.5 pt) What do agents handle in the Atlas Data Lake architecture?

- ● A. Agents evaluate the query plan and distribute data to other agents if its too big
- ○ B. Agents handle connections, parse the query, and build the query plan
- ○ C. Agents learn from previous query plans and apply optimizations to speed up the current query plan based on a set of heuristics.
- ○ D. Agents handle reorganizing the layout of stored data including packing, sorting, hashing, and other optimizations to speed up queries.

No more questions.