CS W186 Spring 2019 Final

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

Contents:

- You should receive one *double-sided answer sheet* and a 35-page *exam packet*.
- The midterm has *9 questions*, each with multiple parts.
- The midterm is worth a total of *125.5 points*.

Taking the exam:

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

- Use the blank spaces in your exam for scratch paper.

Aids:

- You are allowed **two handwritten** 8.5" × 11" double-sided pages of notes.
- The **only** electronic devices allowed are basic scientific calculators with simple numeric readout. No graphing calculators, tablets, cellphones, smartwatches, laptops, etc.

Grading Notes:

- All IOs must be written as integers. There is no such thing as 1.04 IOs – that is actually 2 IOs.
- 1 KB = 1024 bytes. We will be using powers of 2, not powers of 10
- Unsimplified answers, like those left in log format where simplification to integers is possible, will receive a point penalty.

1 SQL and Relational Algebra (11.5 points)

Consider the following schema for bike riders between cities:

```
CREATE TABLE Locations (
    lid INTEGER PRIMARY KEY,
    city_name VARCHAR
);

CREATE TABLE Riders (
    rid INTEGER PRIMARY KEY,
    name VARCHAR,
    home INTEGER REFERENCES locations (lid)
);

CREATE TABLE Bikes (
    bid INTEGER PRIMARY KEY,
    owner INTEGER REFERENCES riders(rid));
);

CREATE TABLE Rides (
    rider INTEGER REFERENCES riders(rid),
    bike INTEGER REFERENCES bikes(bid),
    src INTEGER REFERENCES locations(lid),
    dest INTEGER REFERENCES locations(lid));
);
```

1. (2 points) Select all of the following queries which return the `rid` of `Rider` with the most bikes. Assume all `Riders` have a unique number of bikes.

   A. `SELECT owner FROM bikes GROUP BY owner ORDER BY COUNT(*) DESC LIMIT 1;`

   B.
   ```
   SELECT owner FROM bikes GROUP BY owner HAVING COUNT(*) >= ALL
       (SELECT COUNT(*) FROM bikes GROUP BY owner);
   ```

   C.
   ```
   SELECT owner FROM bikes GROUP BY owner HAVING COUNT(*) = MAX(bikes);
   ```

> **Solution: A, B**
>
> A, B are correct
>
> C is incorrect syntax. MAX(bikes) does not make sense in this context.

2. (2 points) Select the `bid` of all `Bikes` that have never been ridden.

A.
```
SELECT bid FROM bikes b1 WHERE NOT EXISTS
    (SELECT owner FROM bikes b2 WHERE b2.bid = b1.bid);
```

B.
```
SELECT bid FROM bikes WHERE NOT EXISTS
    (SELECT bike FROM rides WHERE bike = bid);
```

C.
```
SELECT bid FROM bikes WHERE bid NOT IN
    (SELECT bike FROM rides, bikes as b2 WHERE bike = b2.bid);
```

Solution: B, C

A finds all of the bikes with no owners which is not the question we are trying to answer.

3. (2 points) Select the `name` of the rider and the `city_name` of the `src` and `dest` locations of all of their journeys for all rides. If a rider has not ridden a bike the output should be:

```
<name, null, null>
```

A.
```
SELECT tmp.name, s.city_name AS src, d.city_name AS dst FROM
locations s, locations d,
(riders r LEFT OUTER JOIN rides ON r.rid = rides.rider) as tmp
WHERE  s.lid = tmp.src AND d.lid = tmp.dest;
```

B.
```
SELECT r.name, s.city_name AS src, d.city_name AS dst FROM riders r
LEFT OUTER JOIN rides ON r.rid = rides.rider
INNER JOIN locations s on s.lid = rides.src
INNER JOIN locations d on d.lid = rides.dest;
```

C.
```
SELECT r.name, s.city_name AS src, d.city_name AS dst FROM rides
RIGHT OUTER JOIN riders r ON r.rid = rides.rider
INNER JOIN locations s ON s.lid = rides.src
INNER JOIN locations d ON d.lid = rides.dest;
```

Solution: None of these are correct, because the subsequent inner joins/where clauses will filter out the null rows from the outer join.

For the following questions fill in the blanks in the query below such that the query returns true if more people bike from home than any other location and false otherwise. If exactly as many people bike from home as from anywhere else then either true or false may be returned.

```
-- selects true for all rides where the rider left from home and false otherwise
WITH is_from_home (is_from_home) AS
    (SELECT _ _ (4) _ _ from riders, rides WHERE rid=rider),

-- counts the total number of rides from home and total number of rides not from home
count_rides (is_from_home, num_rides) AS
    (SELECT _ _ (5)_ _   FROM is_from_home GROUP BY _ _(6) _ _ )

SELECT is_from_home FROM count_rides
 WHERE  _ _(7) _ _   _ _ (8) _ _
     (SELECT num_rides FROM count_rides) LIMIT 1;
```

4. (0.5 points) Fill Blank 4

   A. src = home

   B. dst = home

   C. src = dest

> **Solution: A**
> A is true because we want all rides coming from home to be true

5. (0.5 points) Fill Blank 5

   A. COUNT(*)

   B. is_from_home

   C. is_from_home, COUNT(*)

> **Solution: C**
> We need both the count of the number of rides as well as the boolean value is from home so we can keep track of what kind of ride each count pertains to.

6. (0.5 points) Fill Blank 6

   A. bike

   B. owner

   C. is_from_home

> **Solution: C**
> The count must be grouped on whether or not the ride originated from home

7. (0.5 points) Fill Blank 7

   A. `is_from_home`

   B. `num_rides`

   C. `COUNT(*)`

> **Solution: B**
>
> We are searching for the max number of num rides . We cannot use COUNT(*) here because aggregates are not allowed in the WHERE clause.

8. (0.5 points) Fill Blank 8

   A. `>= ALL`

   B. `<= ALL`

   C. `IN`

> **Solution: A**
>
> We are looking for the maximum value of num rides so we want this to be greater than or equal to all other values of num rides

Relational Algebra

9. (1.5 points) Find the rid of all riders that dont own a bike

   A. $\pi_{\mathbf{rid}}(Riders) - \pi_{\mathbf{owner}}(Bikes)$

   B. $\pi_{\text{rider}}(Rides) - \pi_{\text{owner}}(Bikes)$

   C. $\pi_{\text{rid}}(Riders) - \pi_{\text{owner}}(Bikes \bowtie_{\text{bid = bike}} Rides)$

> **Solution: A**
>
> B finds which riders that have gone on rides do not own a bike, but there may be riders who haven't gone on rides who also don't own a bike. C has the same problem as B

10. (1.5 points) Select the rid of all of the riders whose home is Berkeley and that have ridden their bikes from home

    A. $\pi_{\text{rid}}(\sigma_{city\_name=\text{'Berkeley'}}(Rider \bowtie_{\text{home = lid}} (\pi_{city\_name} Locations) \bowtie_{\text{home = src}} Rides))$

    B. $\pi_{\text{rid}}(Rider \bowtie_{\text{home = lid}} (\sigma_{city\_name=\text{'Berkeley'}} Locations) \bowtie_{\text{home = src}} Rides)$

    C. $\pi_{\text{rid}}(\sigma_{city\_name=\text{'Berkeley'}}(Rider \bowtie_{\text{home = lid}} Locations \bowtie_{\text{home = src}} Rides))$

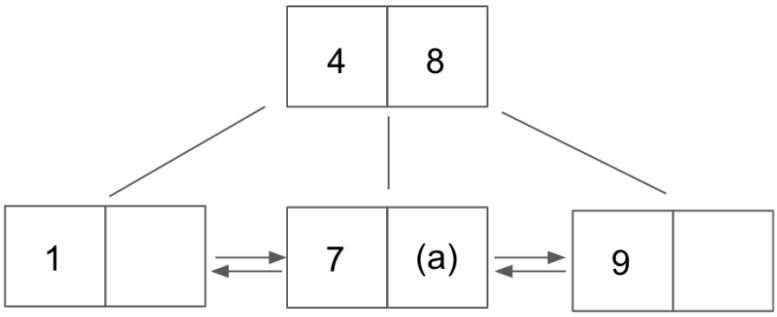
> **Solution: None**
>
> A projects the name in locations too early which means that the join will not have access to the lid column of Locations and will fail.
>
> B and C are wrong because the joins don't filter on rid.

2 B+ Trees (12 points)

1. (2 points) Which of the following statements are true?
   - A. Alternative 1 B+ trees can be clustered or unclustered.
   - **B. B+ trees with variable length keys can have a different number of entries in each node.**
   - C. Unclustered indices are more expensive to maintain than clustered indices.
   - D. When splitting a leaf node, we push up the split key rather than copying up.

> **Solution:** B
>
> A is false because Alternative 1 B+ trees must always be clustered as the data is in the leaves and the leaves are in sorted order according to the index
>
> C is false because clustered indices require you to maintain items in sorted order which can be very costly
>
> D is false because you copy up the split key when splitting a leaf node

Assume we have the following B+ tree with d = 1:



2. (2 points) Whats the minimum number of inserts needed to increase the height of the tree? Assume the slot for (a) is empty and that the keys at the leaf level are just 1, 7, and 9.

> **Solution:** 2. A possible insert pattern would be 2 and 3.

3. (2 points) What is the maximum number of inserts that can be performed without increasing the height of the tree?

> **Solution:** 3. The maximum number of entries in a height 1 tree is 2d (2d + 1) = 2 (1) (2 + 1) = 6. This means we if the number of entries > 6, the tree will increase in height. This means we can add in at most 3 more entries.

4. (1 point) If 10 was in position (a), would this still be a valid B+ tree?

   A. True

   B. False

Solution: False. 10 must be in the node to the right of 8

Assume you have a table:

```
CREATE TABLE Pets (
    petID INTEGER PRIMARY KEY,
    petName VARCHAR
);
```

Assume that the **Pets** table has **81 data pages**. and that **petID** is the **primary key of Pets**. Assume that each petName has 3 petIDs associated with it (3 pets have the same name). Additionally, we have the following indices on the tree:

- Index 1: Alternative 1 B+ tree of height 2 on petID
- Index 2: Unclustered Alternative 2 B+ tree of height 1 on <petName, petID>
- Index 3: Clustered Alternative 3 B+ tree of height 1 on <petID, petName>

5. (1.5 points) Index scans using which of the following indices (if any) will cost fewer I/Os than a file scan for the following query?

```
SELECT * FROM Pets WHERE petName =  'Miki'
```

   A. Index 1
   B. Index 2
   C. Index 3

Solution: The only index that is sorted first by petName is index 2. Index 1 and Index 3 are both first sorted on petID, meaning that it gives us no additional information about where petName = 'Miki'could potentially be. Consequently, Index 1 and index 3 will cost the same as a file scan but index 2 will be more efficient.

6. (1.5 points) Index scans using which of the following indices (if any) will cost the least I/Os for the following query?

```
SELECT * FROM Pets WHERE petID = 50 AND petName = 'Miki'
```

If multiples indices are tied for the least number of I/Os, select all.

   A. Index 1
   B. Index 2
   C. Index 3

Solution: petID is the primary key of the Pets table. This means that there will only be one tuple that matches this query and the tuple is defined by the petID. Performing this query using index 1 will result in 3 I/Os (traverse the tree) and performing this query using index 3 will result in 3 I/Os as well (2 I/Os to traverse the tree + follow the pointer to the entry). Index 2 will cost more I/Os as we will have to traverse the tree to find petName = Miki and then read in 3 petID tuples which could all be on different pages. This would have an IO cost of 2 + 3 (cost to traverse the tree + cost to get each of the tuples tuple) = 5. The answer is therefore Index 1 and Index 3

7. (2 points) What is the minimum fanout for an Alternative 1 B+ tree of height 2 on Pets? Assume that each leaf page can hold as many records as one data page from Pets.

Solution: The maximum number of leaves a tree of height h can hold is $(2d+1)^h$. h = 2 so we know that $(2d+1)^2 >= 81$ as we have 81 pages in Pets. The smallest d can be is 4. This means that the maximum fan out is $2(4)+1 = 9$

3 Sorting and Hashing (21 points)

For this question, we consider the following relation:

```
CREATE TABLE Avengers (
    ID INTEGER PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Species VARCHAR(50) NOT NULL,
    Weapon VARCHAR(50)
);
```

- `ID` is an integer ranging from `10,000,000-50,000,000`.
- Assume sorting or hashing on the `ID` key unless otherwise mentioned.
- Unless otherwise stated, assume that all hash functions used partition data evenly.

1. (2 points) If `Avengers` is a table with 100,000 pages and we have 100 pages in memory, how many I/Os would it take to sort the table?

   Solution: $k = 1 + \lceil \log_{99} \lceil \frac{100000}{100} \rceil \rceil$, so $k = 3$. So, the I/O cost is $2 * k * N = 600,000$ I/Os.

2. (2 points) If `Avengers` is a table with 100 pages and we have 100 pages in memory, how many I/Os would it take to sort the table?

   Solution: 200, as we only need one pass over the data due to $B \geq N$.

3. (2 points) Assuming `Avengers` has 1,000 pages and we have B buffer pages, what is the minimum value of B needed to complete sorting within 2 passes? Recall that the definition of a pass from lecture is a read and write over the data.

   Solution: B = 33, as $B(B-1) \geq 1,000$, and the lowest value of B that satisfies this is 33.

4. (2 points) Assuming we have 50 pages in memory, what is the maximum table size, in pages, for `Avengers` , that we can sort in 2 passes?

   Solution: 2450; $B(B-1) = 50(49) = 2450$.

5. (2 points) Assuming we have 10 pages in memory, what is the maximum table size for `Avengers`, in pages, that we can sort in 3 passes?

   Solution: 810; B(B-1)(B-1) = 10(9)(9) = 810.

6. (2 points) How many I/Os would it take to hash `Avengers`, assuming the table had 100,000 pages and our buffer had $B = 100$?

Solution: 623,611

1. Read in 100,000 pages from disk.
2. Hash the 100,000 pages into 99 partitions, each of size $\lceil 100,000/99 \rceil = 1011$ pages. This is 100,089 pages in total.
3. Write out 100,089 pages to disk.
4. Read in 100,089 pages to disk.
5. Hash the 100,089 pages (which are divided into 99 partitions) into $99^2 = 9801$ partitions, each of size $\lceil 1011/99 \rceil = 11$ pages. This is 107,811 pages in total.
6. Write out 107,811 pages to disk.
7. Read in 107,811 pages to disk.
8. Build in-memory hash table.
9. Write 107,811 pages to disk.

Note: originally the answer was 600,000, which is the same cost as sorting using the formula, but starting in Fall 2019, using the updated hashing method, the answer should be as above.

7. (2 points) How many I/Os would it take to hash `Avengers`, assuming the table had 100 pages and our buffer had $B = 100$?

Solution: 200 I/Os; only one pass is needed over the data, as $N = 100$ and $B \geq N$.

8. (2 points) What is the maximum possible size of `Avengers` if we had $B = 50$ pages in memory and wanted to hash the table within two passes?

Solution: 50 * 49 = 2450 pages; $B(B-1) = 50 * 49 = 2450$

9. (4 points) Now assume a non-uniform hash function, and assume `Avengers` is a table of 350 pages. Assume we have $B = 51$ pages in memory and get partitions of size (50, 100, 100, 40, 40, 10, 10) after the first pass. If we use perfect hash functions from pass 2 onwards, how many I/Os would it take to hash the `Avengers` table in total? **Include the cost of pass 1 in your calculations.**

Solution: Pass 1 takes 700 I/Os; it is just 2*350 = 700. Now, for pass 2 onwards, we are using perfect hash functions, so it helps our analysis to treat these things as small individual tables. Partition 1 will take 2 * 50 = 100 I/Os to completely hash. Partitions 2 and 3 will **each** take 2 * 2 * 100 = 400 I/Os to completely hash. Partitions 4 and 5 will **each** take 2 * 40 = 80 I/Os to completely hash, and partition 6 and 7 will each take 2 * 10 = 20 I/Os to completely hash. This gives us 700 + 100 + 400 + 400 + 80 + 80 + 20 + 20 = **1800 I/Os** to hash it in total.

10. (1 point) True or False: sorting and hashing will always have the same I/O cost

Solution: False

4 Joins and Parallelism (13 points)

For questions 1–4, we are trying to join the `Students` table with the `Classes` table. The `Students` table has 200 pages and the `Classes` table has 300 pages. Assume we have 12 pages of RAM. If you have to make a decision (e.g. inner/outer relation, what table to build the hash table out of), make the decision that will minimize the I/O cost. Assume pages are 1KB for both tables.

1. (2 points) How many I/Os will it take to perform a Block Nested Loops Join between `Students` and `Classes`?

Solution: Calculating the cost with [S] on the outside, we have
$[S] + \lceil \frac{[S]}{(B-2)} \rceil [C]$
$200 + (200/10)(300) =$**6200 I/Os**
With [S] as the inner relation, we have
$[C] + \lceil \frac{[C]}{(B-2)} \rceil ([S]) =$
$300 + (300/10)(200) =$ **6300 I/Os**. Taking the minimum of the costs, we get **6200** I/Os.

2. (2 points) How many partitioning passes are required to perform a Grace Hash Join? Assume we have a perfect hash function.

Solution: We must partition until one of the tables has partitions that are all $\leq$ B-2 big. Each partitioning pass divides the size of each partition by B-1 (because it uses B-1 output buffers to create B-1 partitions).

First pass $\lceil \frac{200}{11} \rceil = 19$. Too big still
Second pass 20 / 11 $\approx$ 2. Now its less than B-2, so the partitions fit.

2 partitioning passes

3. (2 points) Now we want to add 4 machines to our database. Currently all of the data is on the original machine. How much network cost do we expect there to be on average if we range partition both tables across the 5 total machines on the `cid` column? Assume the `cid` column is distributed uniformly across the same ranges for both tables and that we pick perfect ranges.

Solution: Each row has a 1/5 chance of going to any specific machine. That means that 1/5 of the time a row will stay on the original machine (and not cause any network cost) and 4/5 of the time it will have to move to another machine and incur a network cost. Therefore 4/5 of the total number of pages will incur a network cost and thus our total network cost is:

4/5 * (200 + 300) * 1KB = **400KB**

4. (3 points) Assume that the range partitioning in question 3 has completed. How many I/Os will **the original machine** have to perform for a Sort Merge Join (without the optimization in the final merge pass) in the average case?

Solution: The Students table has 40 pages on any machine and the Classes table has 60 pages. Remember our SMJ cost formula: $SortCost(S) + SortCost(C) + [S] + [C]$

It takes 2 passes to sort each table. The first sorting pass leaves us with < 11 sorted runs for both tables, so we only require 1 merging pass.
$SC(S) = 2 * 40 * 2 = 160$
$SC(C) = 2 * 60 * 2 = 240$

160 + 240 + 40 + 60 = **500 I/Os**

For questions 5-8, assume we have a `Products` table with 90 pages hash partitioned across 3 identical machines on the `pid` column. We have no indices built on any machine.

5. (1 point) Assume we used perfect hash functions to partition the table and that the `pid` column is uniformly distributed. How many I/Os will it take in total (over all machines) to run the following query:

```
SELECT * FROM products WHERE pid = 1 and price > 100;
```

Solution: Because of the perfect hash functions and uniform data, each machine has 30 pages. Only the machine that has `pid=1` needs to do a search, so it only takes **30 I/Os**

6. (1 point) Now, do not assume anything about the distribution of the data or the quality of the hash function used to partition the data. How many total I/Os will it take to run the same query as in question 5 in the worst case?

Solution: All 90 pages could be on one machine and `pid=1` could hash the machine. Therefore that machine has to do **90 I/Os**

7. (1 point) Instead we have the following query:

```
SELECT * FROM products where price > 100;
```

Assume that the hash function we used to partition the data is perfect again and that the `pid` column is uniformly distributed. How many I/Os will it take to the run the query?

Solution: We have to ask each machine to do a search because the data could be on any machine. Therefore every page must be read, so we will do **90 I/Os**.

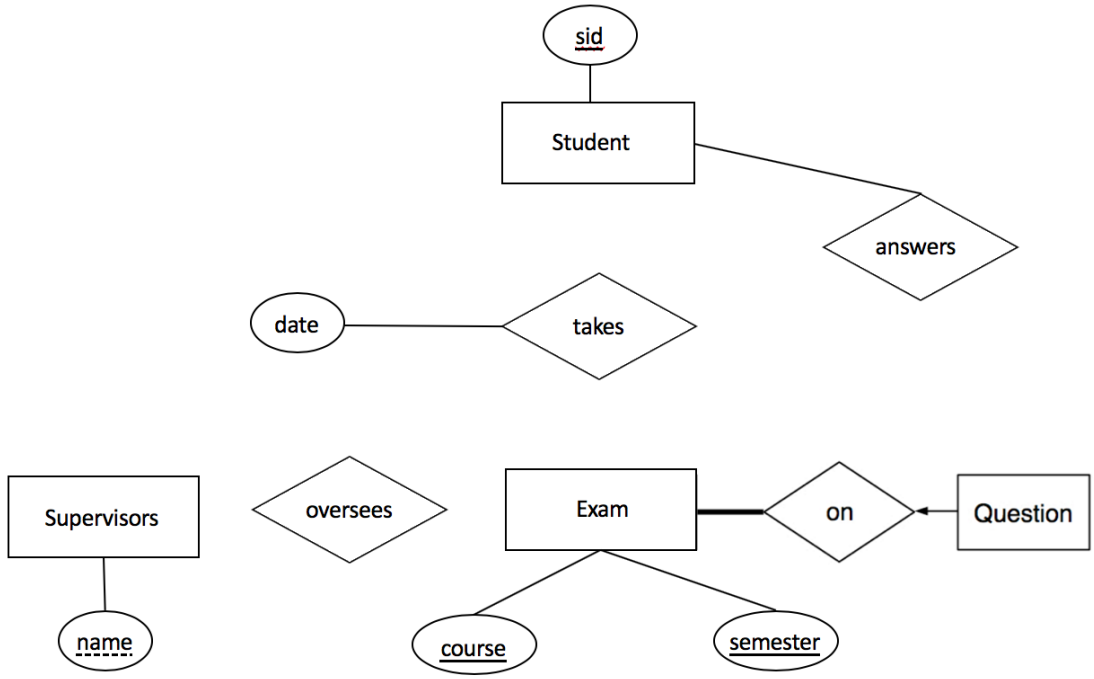
8. (1 point) Out of questions 6 and 7, which scenario will take less time to run?
   A. 6
   B. 7
   C. Same
   D. Indeterminate

Solution: We have to do the same number of I/Os for both questions. Question 6, however, amounts to 1 machine doing a full scan of 90 pages, whereas question 7 lets 3 machines run simultaneously on 30 pages of the data so question **7** finishes first.

5 ER Diagrams and FDs (14 points)

As president of Stanford University, you are tasked with designing a new final exam scheduling system to make it easier on students. Using the following assumptions, fill in the ER diagram we give you.

- A student, uniquely identified by his or her SID, takes maximum one exam on one date.
- A student may take any number of exams, and every exam is taken by at least one student.
- An exam is uniquely identified by the combination of a course and a semester.
- Every exam has at least one supervisor. A supervisor oversees exactly one exam.
- There is at least one question on every exam, and a question appears on at most one exam.
- A question on an exam may be answered by any number of students, and a student may answer any number of questions on an exam.



1. (1 point) What type of edge should be drawn between the **Supervisors** entity and the **oversees** relationship set?

   A. Thin Line

   B. Bold Line

   C. Thin Arrow

   D. Bold Arrow

> **Solution:** Each supervisor must oversees exactly one exam. This means that a supervisor must have a key constraint and participation constraint on their relationship with **oversees**

2. (1 point) What type of edge should be drawn between the **Exam** entity and the **oversees** relationship set?

   A. Thin Line

   B. Bold Line

   C. Thin Arrow

   D. Bold Arrow

Solution: At lease one supervisor oversees the exam. This is a participation constraint.

3. (1 point) What type of edge should be drawn between the **Student** entity and the **takes** relationship set?

   A. Thin Line

   B. Bold Line

   C. Thin Arrow

   D. Bold Arrow

Solution: Each student may take 0 or 1 or more exam on one day.

4. (1 point) What type of edge should be drawn between the **Exam** entity and the **takes** relationship set?

   A. Thin Line

   B. Bold Line

   C. Thin Arrow

   D. Bold Arrow

Solution: Student takes at least one exam in total. It's participation constraint.

5. (1 point) What type of edge should be drawn between the **Questions** entity and the **answers** relationship set?

   A. Thin Line

   B. Bold Line

   C. Thin Arrow

   D. Bold Arrow

Solution: Each **student** may answers 0, 1 or more **questions**.

6. (4 points) Consider the attribute set R = ABCDEF and the functional dependency set F = {BE → C, B → F, D → F, AEF → B, A → E}. Which of the following are candidate keys of R? Mark all that apply

   A. ACD

   B. AD

   C. FC

   D. BF

Solution: In computing attribute closure of a key K, we repeatedly process the set of functional dependencies F and add on attributes to the closure of K until there is nothing more that can be added. For ACD, we see that this leads first to ACDEF on the first processing of F and ABCDEF upon the second time we go through the loop. This means ACD is a superkey; however, for it to be a candidate key, we should check and make sure none of its subsets are superkeys for the table. Processing AD, however, gets us ADEF on the first iteration of the loop, ABDEF on the second, and ABCDEF on the third.

FC and BF are not superkeys (and therefore not candidate keys): both are their own attribute closures.

7. (3 points) Given Attribute Set R = ABCDEFGH and functional dependencies set F = {CE → GH, F → G, B → CEF, H → G}. Which of the following relations are included in the final decomposition when decomposing R into BCNF in the order of functional dependencies set F?

   A. BCEF

   B. ACEF

   C. ABD

   D. GDH

   E. GH

   F. CEH

Solution:

CE → GH violates BCNF, decompose into ABCDEF CEGH.

F → G No relation contains FG, skip.

B → CEF violates BCNF, decompose into ABD, BCEF, and CEGH

H → G violates BCNF, decompose into ABD, BCEF, CEH, GH.

Final relations are ABD, BCEF, CEH, GH.

8. (2 points) True or False: The decomposition of Attribute set R = ABCDEF, given the functional dependency set F = {B → D, E → F, D → E, D → B, F → BD}, into ABDE, BCDF is lossless.

Solution: False, it is lossy. ABDE ∩ BCDF = BD, BD → BDEF, which is not a superset of either ABDE or BCDF.

6 Query Optimization (10 points)

Consider the following schema for a sports league:

```
CREATE TABLE Teams(
    tid INTEGER PRIMARY KEY,
    tname VARCHAR(50),
    division VARCHAR(10),
    rank INTEGER
);
CREATE TABLE Players(
    pid INTEGER PRIMARY KEY,
    pname VARCHAR(50),
    tid INTEGER REFERENCES Teams(tid),
    age INTEGER,
    salary FLOAT
);
CREATE TABLE Games(
    home_tid INTEGER REFERENCES Teams(tid),
    away_tid INTEGER REFERENCES Teams(tid),
    game_date DATE,
    home_points INTEGER,
    away_points INTEGER,
    PRIMARY KEY (home_tid, away_tid, game_date)
);
```

Assume that:

- There are 30 teams in the league (so `Teams.tid` has 30 possible values)
- The following histogram shows the distribution of `Games.home_points`:

< 90	90-99	100-109	≥ 110
5%	35%	40%	20%

Consider the following query:

```
SELECT T.tname
  FROM Teams T, Games G
 WHERE T.tid = G.home_tid
   AND T.rank > 20
   AND G.home_points > 107;
```

1. (1 point) What is the selectivity for the predicate `G.home_points > 107`?

   Solution: 0.28

2. (1 point) What is the selectivity for the predicate `T.tid = G.home_tid`?

   Solution: 1/30

3. (1 point) True or False: We can apply `T.rank > 20` on `T` before we join `T` with `G`.

Solution: True

4. (1 point) True or False: We can apply `T.rank > 20` on `T`, apply `G.home_points > 107` on `G`, and then only keep the columns `T.tid` and `G.home_tid` before we join `T` with `G`.

Solution: False, also need to keep `T.tname` since that is in the final output.

Consider the following query:

```
SELECT T.tname, P.pname, G.game_date
  FROM Teams T, Players P, Games G
 WHERE P.tid = T.tid                                  (predicate 1)
   AND T.tid = G.home_tid                             (predicate 2)
   AND (G.home_points > 110 OR G.away_points < 90)    (predicate 3)
   AND P.age > 35                                     (predicate 4)
   AND P.salary > 15,000,000                          (predicate 5)
   AND T.division = 'PACIFIC'                         (predicate 6)
ORDER BY G.game_date ASC;
```

5. (1 point) True or False: The first pass of the Selinger optimizer will only consider the predicates with a single column (predicates 4, 5, 6); the predicates with two columns (1, 2, 3) will only be considered starting from the second pass.

Solution: False. Predicate 3's two columns are from the same table so it is also considered in pass 1.

6. (3 points) For each row in the following table, mark whether the join will be retained after pass 2. Assume that the optimizer only uses the predicates as given, and does not infer any other relationship between the joined tables.

	Left Tables	Right Table	I/O Cost	Sorted On
(A)	T	G	10,000	—
(B)	T	G	14,000	`T.tid`
(C)	G	T	10,500	`G.home_points`
(D)	G	T	12,000	`G.game_date`
(E)	G	T	11,000	`G.home_tid`
(F)	P	G	16,000	—
(G)	P	G	22,000	`P.tid`
(H)	G	P	19,000	`G.game_date`
(I)	G	P	18,500	`G.home_points`
(J)	P	T	9,500	—
(K)	P	T	9,000	`P.tid`
(L)	T	P	9,250	`T.tid`

Solution: A, B, D, K, L

7. (2 points) For each statement regarding pass 3 (the final pass), mark either True or False.

   A. The overall lowest cost join within the considered plan space will always be retained at the end of pass 3.

   B. The lowest cost join (within the considered plan space) sorted on `T.tid` will always be retained at the end of pass 3.

   C. Within the considered plan space, the lowest cost join sorted on `G.game_date` will always be retained at the end of pass 3.

   D. Within the considered plan space, the lowest cost join sorted on `G.home_tid` might be retained at the end of pass 3.

7 Transaction and Concurrency (19 points)

1. (7 points) For each assertion, fill in the corresponding bubble True or False.

   A. For a set of n transactions, the number of different serial schedules is exponential in n (i.e. there exists a constant c such that the number of schedules is $O(c^n)$).

   B. In an ACID-preserving DBMS that only allows serial schedules, the database must be in a consistent state throughout the execution of a transaction.

   C. A topological sort of the dependency graph of a conflict serializable schedule produces an ordering of an equivalent serial schedule.

   D. View serializability is not commonly enforced in practice for performance reasons.

   E. Two phase locking is a concurrency-control protocol that prevents deadlock.

   F. Two phase locking (including strict 2PL) is a concurrency-control protocol that is necessary for enforcing conflict serializibility.

   G. Consider a resource A where transactions T1 and T2 have shared locks on A, and T3 is in the waiting queue because it requested an X lock on A. If another transaction T4 tries to `acquire` an IS lock on A, the lock manager (as implemented in HW 5) grants it because the requested lock type is compatible with the existing S locks.

Solution: C, D

A is False since the number of schedules is $n!$.

B is False because the database only needs to be in a consistent state at the end of a transaction.

E is False since deadlock is possible under 2PL.

F is False because there are other ways of enforcing conflict serializibility such as timestamp-ordering.

G is False since we do not skip the queue during `acquire`.

We consider a transaction T_j to be *dependent* on another transaction T_i if T_j reads a value written by T_i before T_i has committed. This is because if T_i needs to be aborted, then so must T_j (the issue of *cascading aborts*).

We consider a schedule to be *recoverable* if for all pairs of transactions T_i, T_j such that transaction T_i is *dependent* on transaction T_j, T_j commits before T_i commits (if T_i commits).

Notice that if T_i commits before T_j, and the system crashes before T_j can commit, then we must abort T_j, and therefore we have to abort T_i. This violates ACID due to T_i having already committed, in which case the schedule would be *unrecoverable*.

2. (1 point) Mark True if the following schedules are recoverable, False otherwise.

A.

T1	**R(C)**	**W(C)**					**R(D)**
T2			**R(C)**	**W(C)**			
T3					**W(D)**	**Commit**	

B.

T1	R(C)	W(C)						R(D)
T2			R(C)	W(E)	Commit			
T3						W(D)	Commit	

Solution: A

B is False because T2 depends on T1 for the R(C) operation, but T2 commits before T1.

3. (4 points) Consider the notion of recoverable schedules from the previous question. For each assertion, mark either True or False.

   A. A conflict serializable schedule can be unrecoverable.

   B. 2PL always produces a recoverable schedule.

   C. Strict 2PL always produces a recoverable schedule.

   D. A casecadeless schedule is always recoverable (a schedule is *cascadeless* if it cannot suffer from cascading aborts).

Solution: A, C, D

A is True, see schedule B of the previous part.

B is False because consider a pair of WR operations by transactions T_i, T_j respectively. Under 2PL, after the write, T_i can release the lock without committing, and T_j can perform the read and commit before T_i commits.

C is True because for any pair of WR operations, strict 2PL would require T_i to commit before releasing the lock, thereby ensuring that T_j commits after T_i.

D is True, since if there are no cascading aborts, then there are no dependent transactions.

For the next two parts, consider the following schedule:

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

4. (1 point) We write an edge as ordered pair (T_i, T_j) if the edge goes from T_i to T_j. What is the set of edges of the dependency graph of the schedule above?

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

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

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

   D. $\{(T_2, T_1), (T_1, T_3), (T_3, T_2), (T_1, T_2)\}$

   E. $\{(T_2, T_1), (T_1, T_3), (T_3, T_2)\}$

5. (2 points) For each assertion, fill in the corresponding bubble True or False.

   A. The schedule is conflict serializable.

   B. Removing all of T_1's operations would produce a conflict serializable schedule.

   C. Removing all of T_3's operations would produce a conflict serializable schedule.

   D. Moving the R(A), W(A) operations of T_2 to the end of the schedule would produce a conflict serializable schedule.

Solution: B, D

A is False since there is a cycle in the dependency graph.

C is False because there is still a cycle with T_1 and T_2.

For the next three parts, we will now consider a locking protocol called strict 2PL with lock conversions. Here, in addition to being able to `acquire` and `release` locks, transactions are also able to `promote` a lock to a lock type with more permissions. As in the original strict 2PL scheme, transactions must release locks only after they have committed or aborted.

Consider a multigranularity scheme where for each operation in a schedule, the corresponding transaction requests the locks (using either `acquire` or `promote`) with **minimal privilege** that enable it to perform the operation.

Consider a database with a table Z. Z has pages A, B, C. Page A has tuples $A_1, \ldots, A_{100}$, page B has tuples $B_1, \ldots, B_{100}$, and page C has tuples $C_1, \ldots, C_{100}$.

Consider the following schedule:

	1	2	3	4	5	6	7	8
T1	R(A_5)	W(B_3)			W(A_5)			R(B)
T2			R(A_5)			W(B_1)	Commit	
T3				W(A_5)				

6. (1 point) What is the ordering of transaction numbers of the waiting queue for A_5 after timestep 5? Write your answer as a ordered list of integers.

   For example, if T7 will be processed and then T5, mark 7 on the first line, 5 on the second line, and N/A on the third line. Mark N/A for all lines that you do not need (if the queue is empty, mark N/A for all three lines on the answer sheet).

   Solution: (1, 3)

   A `promote` places the request at the front of the queue.

7. (2 points) After T2 commits at timestep 7 (and the lock manager processes any pending requests), what locks does T1 hold?

   Write your answer in ascending order of acquisition time. For lock upgrades, follow the convention of acquisition time as in HW 5.

   Please fill in the boxes in the following order:

Lock #1	Lock #2
Lock #3	Lock #4
Lock #5	Lock #6

   If T1 does not hold any locks, mark the checkbox at the bottom indicating so. You may not need all the spaces provided—please leave spaces that you do not need blank.

   Solution: IX(Z), IX(A), X(A_5), IX(B), X(B_3)

   As in HW 5, promotes do not change the acquisition time.

   Partial credit was given for the correct locks but incorrect order. No partial credit was given if the previous part was incorrect.

8. (1 point) After timestep 8, what lock types does T1 hold? Check all that apply, or check the $\emptyset$ option if T1 does not hold any locks.

Solution: IX, X, SIX

After timestep 8, the IX(B) gets promoted to SIX(B).

No partial credit was given if the previous part was incorrect.

8 Recovery (15 points)

Someone tripped over a cable and pulled the power cable to your database server, so your database crashed. When you boot it back up, you find the following log records and transaction and dirty page tables from the last checkpoint. Assume that the final disk flush is the update to P2 on LSN 30.

LSN	Record	prevLSN
10	UPDATE: T1 writes P2	null
20	UPDATE: T1 writes P3	10
30	UPDATE: T2 writes P2	null
40	Begin Checkpoint	-
50	UPDATE: T3 writes P2	null
60	End Checkpoint	-
70	UPDATE: T4 writes P4	null
80	T4 ABORT	70
90	CLR: UNDO T4 LSN 70	80
100	UPDATE: T2 writes P1	30
110	UPDATE: T3 writes P3	50
120	T1 COMMIT	20
130	UPDATE: T2 writes P4	100
140	T1 END	120

Transaction	lastLSN	status
T1	20	R
T2	30	R

PageID	recLSN
P3	20

1. (1.5 points) During the Analysis phase, which of the following log records are read?

   A. LSN 20

   B. LSN 50

   C. LSN 80

   Solution:

   A is False because you start reading from LSNs from the begin checkpoint record, which is at LSN 40

   B is True because LSN 50 is greater than the begin checkpoint record at LSN 40

   C is True for the same reason as B

2. (6 points) After the analysis phase, what is the state of the Transaction Table and the Dirty Page Table?

   - For the status column, write down R if the transaction is running, C is the transaction is committing, and A if the transaction is aborting.
   - If a transaction is not in the transaction table, put an X in the lastLSN and status boxes for that transaction
   - If a page is not in the dirty page table, put an X in the recLSN box for that page

Solution:

TID	Status	lastLSN
T1	X	X
T2	R	130
T3	R	110
T4	A	90

PID	recLSN
P1	100
P2	50
P3	20
P4	70

3. (3 points) Which of the following log records will be redone?

   A. LSN 10

   B. LSN 20

   C. LSN 30

   D. LSN 40

   E. LSN 80

   F. LSN 90

Solution:

A is False because you start from earliest recLSN in the DPT, which is 20

C is False because the recLSN for P2 (50) is greater than the LSN (30)

D is False because you don't need to redo checkpoint records

E is False because you don't need to redo abort records

F is True because you still redo CLR records

Answer the following 3 questions about the UNDO phase.

4. (0.5 points) What is the LSN of the first record to be undone?

> **Solution:** 130
> You start undoing from the maximum lastLSN in your transaction table

5. (0.5 points) What is the LSN of the second record to be undone?

> **Solution:** 110
> toUndo = lastLSNs of all Xacts in the Xact Table
> You undo the max number in the toUndo set
> So, toUndo starts off as 130, 110, 90
> After you undo 130 from the previous question, toUndo now looks like 100, 110, 90.
> Thus, the second record to undo is 110.

6. (0.5 points) Which transaction is the first to be completely undone?
   A. T1
   B. T2
   C. T3
   D. T4

> **Solution:** You undo all the transactions in the transaction table, so you should only initially consider the answers T2, T3, and T4. You undo these three transactions with the max LSN left to undo from each of these three transactions. T4's first record (so last record to undo) has the highest LSN between T2 and T3's first record, so T4 will be completely aborted and undone first.

7. (3 points) Mark the following statements as True or False.
   A. A No Steal policy with Force requires redo logging to achieve atomicity
   B. It is possible for the flushedLSN to be less than an on-disk pageLSN
   C. It is possible for a page's in-memory pageLSN to be less than that page's on-disk pageLSN
   D. If we flush the pages in the DPT table to disk upon checkpointing, the redo phase only needs to redo LSNs greater than or equal to the nearest checkpoint
   E. If we flush the pages in the DPT table to disk upon checkpointing, the undo phase only needs to undo LSNs greater than or equal to the nearest checkpoint
   F. CLR records can be redone, but they can never be undone

Solution:

A is False because a No steal policy with Force doesn't require any kind of logging to achieve atomicity.

B is False because the on-disk pageLSN must have been flushed to disk. The flushedLSN keeps track of the most recent LSN flushed to disk. Thus, the flushedLSN has to be greater than or equal to the on-disk pageLSN.

C is False because the pageLSN is the LSN of last record modifying a page. Thus, the in-memory pageLSN should always be greater than or equal to the on-disk pageLSN, since the in-memory pageLSN contains more recent information about which record was the last one to modify a page.

D is True because you only need to redo LSNs starting from the earliest recLSN in the DPT. Since you flushed the DPT table to disk at the checkpoint, you only need to redo LSNs that are greater than or equal to the nearest checkpoint.

E is False because a transaction that made some updates before the checkpoint and hasn't finished committing or aborting at the time of the crash must be aborted/rolled back. Thus, all the transactions actions must be undone (which includes the updates before the checkpoint).

F is True because you when you build back the state of the database at the time of the crash, you might have to end up redoing some CLRs. However, once you have undone an action and written a CLR for it, you never need to undo the undo (aka undo the CLR).

9 Distributed Transactions (10 points)

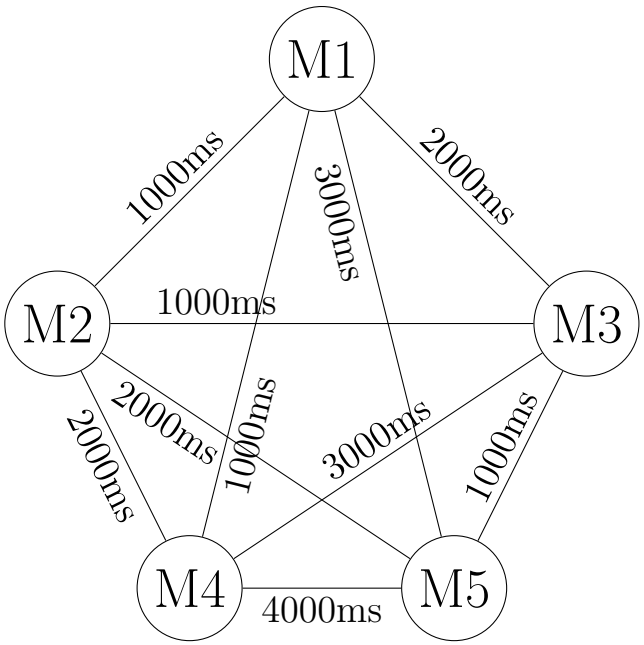
Our database runs on 5 machines and uses Two-Phase Commit.

Suppose our machines are connected as shown in the graph. For example, sending a message between Machine 2 and Machine 4 takes 2000 milliseconds in either direction.

Assume that everything is instantaneous except for the time spent sending messages between two machines (e.g. processing time, recovery), unless otherwise specified, and that we are using presumed abort. Assume that all machines vote yes.

We consider the following events (all times are measured relative to the start of the transaction):

- At 0ms: The transaction begins the 2PC protocol.
- At 3500ms: Machine 2 goes down.
- At 4500ms: Machine 1 goes down.
- At 5500ms: Machine 1 comes back up.
- At 6500ms: Machine 2 comes back up.
- At 7500ms: Machine 3 goes down.
- At 8500ms: Machine 2 goes down.
- At 9500ms: Machine 1 and Machine 4 go down.
- At 14000ms: Machine 5 goes down.
- At 20000ms: Machine 2 and Machine 4 come back up.
- At 25000ms: all remaining machines come back up.



Assume that our 2PC implementation is configured to use very high timeouts, such that no node will ever think another node has crashed, throughout any events discussed in this problem.

Suppose that Machine 1 is the coordinator.

1. (0.5 points) What is the first type of message that Machine 2 sends?

   A. VOTE YES B. PREPARE C. COMMIT D. ABORT E. None of these

   Solution: Machine 2 starts by voting.

2. (1 point) When does Machine 2 send this message (relative to the transaction starting)?

   Solution: At 1000ms, when it receives the PREPARE message from Machine 1.

3. (0.5 points) What is the second type of message that Machine 1 sends?

   A. VOTE YES B. PREPARE C. COMMIT **D. ABORT** E. None of these

   Solution: The round trip time between Machine 1 and Machine 5 is 6000ms, but Machine 1 goes down at 4500ms. Since we are using presumed abort, Machine 1, upon coming back up, aborts the transaction and sends out ABORT messages.

4. (1 point) When does Machine 1 send this message (relative to the transaction starting)?

   Solution: At 5500ms, when it comes back up.

Now suppose that Machine 4 is the coordinator.

5. (0.5 points) What is the second type of message that Machine 4 sends?

   A. VOTE YES B. PREPARE **C. COMMIT** D. ABORT E. None of these

   Solution: All machines are up when the PREPARE message reaches them, and Machine 4 is up to receive all of the VOTE YES messages (Machine 4 only goes down at 9500ms, but receives the last VOTE YES from Machine 5 at 8000ms). Therefore, Machine 4 continues by sending a COMMIT message.

6. (1.5 points) What is the final state of the transaction?

A. Committed B. Aborted

Solution: Each machine is able to send a VOTE YES to Machine 4 before crashing (if at all), and Machine 4 receives all VOTE YES messages and sends out COMMIT messages before crashing, so the transaction commits. That all the machines crash before the ACK messages are sent is immaterial.

7. (4 points) When does Machine 4 write the END record to its log (relative to the transaction starting)?

Solution: 34000ms.

When Machine 4 sends the COMMIT message, Machine 3 is already down. Machine 2 goes down before the message reaches it, and Machine 1 and 5 are the only machines to receive the message. Their ACK messages do not reach Machine 4, since it is down.

Machine 2 and 4 come up at 20000ms, so Machine 2 sends an ACK at 22000ms and Machine 4 receives it at 24000ms.

Machine 1, 3, and 5 come up at 25000ms. Machine 1 and 5 already received (and logged) the commit, so they both send ACK to Machine 4 (received at 26000ms and 29000ms respectively). Machine 3 did not receive the commit, so it sends an inquiry to Machine 4, who sends a response at 28000ms. Machine 3 sends an ACK at 31000ms, and the ACK is received at 34000ms. This is the last ACK, so Machine 4 writes END to its log.

Partial credit was given to the answers of:

- 33000ms (assuming a COMMIT was sent out at 25000ms instead of participants making inquiries).
- 37000ms (not accounting for Machine 5 having received the commit message before crashing).

---

8. (1 point) True or False. In a system utilizing Two-Phase Commit, if all participants vote YES, the transaction must commit.

Solution: False, the coordinator may crash before writing commit to the log.

10 Scratch Work (0 points)