There are 14 questions and 15 pages in this quiz booklet. To receive credit for a question, answer it according to the instructions given. You can receive partial credit on questions. You have 80 minutes to answer the questions.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be harder than others. Attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can’t understand your answer, we can’t give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.
LAPTOPS MAY BE USED; NO PHONES OR INTERNET ALLOWED.

Do not write in the boxes below

<table>
<thead>
<tr>
<th>1-6 (xx/34)</th>
<th>7-11 (xx/24)</th>
<th>12-13 (xx/20)</th>
<th>14-15 (xx/22)</th>
<th>Total (xx/100)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Name:
I  Short Answer

1. [6 points]: Which of the following sets of transactions, if run concurrently, could experience the phantom (aka Halloween) problem, assuming that the database has no special provision to avoid phantoms and is using strict two-phase locking, and that the transactions shown are the only ones running at the time?

(For each set of transactions, circle “Yes” if the phantom problem could occur, or “No” otherwise.)

A. Transaction 1:
   avg = SELECT AVG(salary) FROM emp WHERE dept = 'eecs'
   SELECT name FROM emp WHERE salary > avg AND dept = 'eecs'

   Transaction 2:
   UPDATE emp SET salary = salary * 1.1

   Could exhibit the phantom problem: Yes  No

B. Transaction 1:
   avg = SELECT AVG(salary) FROM emp WHERE dept = 'eecs'
   SELECT name FROM emp WHERE salary > avg AND dept = 'eecs'

   Transaction 2:
   INSERT INTO emp(id,name,dept,salary) VALUES (1, 'joe', 'eecs', 50k)

   Could exhibit the phantom problem: Yes  No

C. Transaction 1:
   count = SELECT COUNT(*) FROM emp WHERE dept = 'eecs'

   Transaction 2:
   INSERT INTO emp(id,name,dept,salary) VALUES (1, 'joe', 'eecs', 50k)

   Could exhibit the phantom problem: Yes  No

Name:
2. [6 points]: Rewrite the following SQL queries into a flattened (non-nested) representation, or circle “no non-nested representation is possible”.

A. SELECT AVG(salary)
   FROM
   ( SELECT DISTINCT department, salary
     FROM emp
   )

   No non-nested representation is possible

   Flattened query:

B. SELECT ename,
   (SELECT dname FROM dept WHERE dept.did = emp.did)
   FROM emp

   No non-nested representation is possible

   Flattened query:
3. [4 points]: Consider a relation $S$ (id, title, length, record-date, sub-genre) storing several million rock songs. Suppose we want to run two queries with equal frequency:

A. Find songs recorded in the 1980s and 1990s that are longer than 8 minutes.

B. Find songs of the sub-genre “metal ballad”, recorded before 1960

The songs in the table span over a total of 60 years (up to today) and are uniformly distributed across years. There are only 10 distinct sub-genres, with songs also uniformly distributed by sub-genre. Songs are typically short, with many fewer songs as you move away from the mean length of 4 minutes, with only about 1% of the songs being longer than 8 minutes. New songs are rarely inserted into the database. Without indexes, these queries take about the same amount of time to run. The database only supports B+Tree indexes. Indexes may have composite keys consisting of multiple attributes concatenated together. If you can only create one index on this table, what index would you recommend creating to make these queries run as fast as possible? Should this index be clustered or unclustered? Justify your answer.

(Write your answer in the space below.)
II Schema Design Questions

Suppose you are designing a schema to record information about recipes. Your database needs to record the following information:

- For each recipe, its title, author (e.g., “grandma”), and a rating from 1–5.
- For each recipe, the ingredients it uses, and the quantity of each ingredient (expressed as a string, e.g. “1 cup”). Recipes use many ingredients, and ingredients are used by many recipes.
- For each ingredient, its name, and a category (e.g., “spice”, “dairy”, etc).

4. [8 points]:
Draw an entity relationship diagram for this database. Please draw entities as squares, attributes as ovals, and denote relationships as diamonds between pairs of entities, with a label of the form “1:1”, “N:1” or “1:N”, where “1:N” indicates that a single entity on the left side has a relationship with many entities on the right, but each entity on the right has a relationship with only one entity on the left. Give each entity, relationship, and attribute a name. Recipe authors, ingredient quantities, and ingredient categories should be represented as string attributes, not separate entities.

(_draw your diagram in the space below.)
5. [6 points]:
Use your ER diagram to determine a relational schema for this database. For each table, denote its schema as:

tablename1 (field1-name, ... , fieldn-name)

Underline the field or fields that comprise the primary key of each table.

(Write your answer in the space below.)

6. [4 points]:
Is the schema you developed redundancy free? Does it have any insertion or deletion anomalies? Briefly (in one sentence) state why or why not?

(Circle your answer, and write the explanation below.)

Yes  No

Explanation:

Name:
III Query Planning Questions

Consider the following relational schema for storing data from a Facebook-like website that records information about users, friends, and posts.

user (id BIGINT PRIMARY KEY, fullname CHAR(22), email CHAR(16), gender TINYINT, birthday DATE)
// = 8 + 22 + 16 + 1 + 3 = 50 bytes record size

friendship (user1 BIGINT REFERENCES user.id, user2 BIGINT REFERENCES user.id, since DATE, edgerank TINYINT)
// = 8 + 8 + 3 + 1 = 20 bytes record size

privacylevels (id SMALLINT PRIMARY KEY, name CHAR (8))
// = 2 + 8 = 10 bytes record size

posttypes (id SMALLINT PRIMARY KEY, name CHAR(7), graphrank TINYINT)
// = 2 + 7 + 1 = 10 bytes record size

post (id BIGINT, type SMALLINT REFERENCES posttypes.id, fromuser BIGINT REFERENCES user.id, touser BIGINT REFERENCES user.id, privacy SMALLINT REFERENCES privacylevels.id, time TIMESTAMP, content CHAR(68))
// = 8 + 2 + 8 + 8 + 2 + 4 + 68 = 100 bytes record size

Posts can be one of several types and the importance of a given posttype is captured by its graphrank, which you can assume to be an integer in the range 1–100. For instance, a post which is a status message may have a graphrank of 100 whereas an activity such as listening to music on Spotify might only have a graphrank of 25.

Similarly, edgerank is indicative of the strength of a connection between friends. Assume that edgerank can vary from 1–10, and that high edgeranks correspond to stronger connections. Note that connections are usually asymmetric, so that for every connection, two tuples exist in the friendship table, with potentially different edgerank in the two directions.

Below is a SQL query that fetches 500 days worth of news feeds from a specific user’s friends, subject to filters on post importance, connection strength and privacy.

```
SELECT user.fullname, posttypes.name, COUNT(*)
FROM user, friendship, posttypes, privacylevels, post
WHERE friendship.user1 = 1988009010
AND friendship.user2 = user.id
AND friendship.edgerank > 5
AND post.fromuser = friendship.user2
AND post.type = posttypes.id
AND posttypes.graphrank > 25
AND post.privacy = privacylevels.id
AND privacylevels.name != 'onlyme'
AND post.time > DATE_SUB(CURRENT_TIMESTAMP(), INTERVAL 500 DAY)
GROUP BY user.fullname, posttypes.name
```

Name:
7. [8 points]: Assuming that the relational query optimizer executing this query never considers plans requiring cross-products, list all possible join orders that the optimizer will consider for the 5 tables in this query. Restrict yourself to left-deep joins.

Given a plan “Table A joined with Table B (where A is the inner and B is the outer), joined with table C (as the inner)”, denote that as ((A B) C). Plans with the same join order but with swapped inner/outer relations do not need to be listed separately. You can use the shorthand U, F, PT, PL, and P for the user, friendship, posttype, privacylevel, and post tables, respectively.

(Write your answer in the space below.)
For the following problems, you are given that the number of distinct post types is 50, that there are 5 equally likely privacy levels and that the edgerank and graphrank values are uniformly distributed. Including all application activity, an average user posts three times a day. The user with id 1988009010 has 4000 friends, all with distinct names.

8. [4 points]: Estimate the fraction of tuples that would be initially selected from the friendship, privacylevels and posttypes tables if all of the non-join predicates are applied to them before any join processing begins.

(Write your answer in the space below.)

9. [6 points]: Suppose unclustered B+-tree indexes exist on the attributes user.id, friendship.user1, post.id, post.fromuser, and post.touser. If the optimizer only supports nested loops and index nested loops join methods, which of the possible join orders you listed in Problem 7 will execute the fastest? Indicate which tables are inner/outer and the join algorithm used for each join. Provide a brief justification or calculation.

(Write your answer in the space below.)
10. [6 points]: What is the cost, in terms of total pairs of tuples evaluated by the 5 joins, of the fastest join order you selected in Question 9? For purposes of this problem, one pair of tuples is evaluated whenever a tuple in the outer relation is compared to a tuple in the inner relation of a nested loops join, or when a lookup is done on an index in an index nested loops join. Show your calculations.

(Write your answer in the space below.)
IV Join Algorithms

For each of the equality joins below, we provide the sizes $|R|$ and $|S|$ of the two tables $R$ and $S$ being joined, the number of tuples $J$ that satisfy the join predicate, whether there is an unclustered B+Tree index on either table, and whether either heap file is ordered on some attribute. In all cases, assume that the size of the larger table is less than $M^2$ (where $M$ is the number of pages of memory), and that the smaller table and its indexes can fit into memory, but that the larger table cannot. Assume that the CPU time to compare two tuples, or do a lookup in an index or hash file is about $1/1,000$th the time to read a tuple from disk, and that there is no difference between random and sequential I/O. For each join, indicate the best join method, A–E, and provide a brief justification of your choice.

- A. Index nested loops join, with $R$ as the inner
- B. Index nested loops join, with $S$ as the inner
- C. Hybrid Hash Join
- D. Sort Merge Join
- E. Simple Nested Loops Join

11. [8 points]:
- $|R| = 100$ MB (100K tuples), $|S| = 10$ KB (100 tuples), $J = 1$ tuple, no indexes, both tables unsorted

Enter the best join algorithm A–E: 
1 sentence justification of your answer:

- $|R| = 100$ MB (100K tuples), $|S| = 10$ KB (100 tuples), $J = 1$ tuple, index on join attribute of $S$, both tables unsorted

Enter the best join algorithm A–E: 
1 sentence justification of your answer:

- $|R| = 100$ MB (100K tuples), $|S| = 10$ MB (10K tuples), $J = 100$K tuples, index on join attribute of $R$, both tables sorted on join attribute

Enter the best join algorithm A–E: 
1 sentence justification of your answer:

- $|R| = 100$ MB (100K tuples), $|S| = 10$ MB (10K tuples), $J = 100$K tuples, index on join attribute of $R$, both tables unsorted

Enter the best join algorithm A–E: 
1 sentence justification of your answer:

Name:
V Isolation

Consider a table LaLiga with schema `<name CHAR(32), goals INTEGER>` which stores the number of goals scored by each player in La Liga (a Spanish Soccer league). At the end of every month, you’re responsible for updating the table to reflect the goals scored by each player during that month. One of the star players in La Liga, Lionel Messi, had 26 goals before last month, and scored a brace (two goals in a game) and a hat trick (three goals in a game) last month. To update his goal total, you decide to run the following transaction.

```
BEGIN TRANSACTION
n = SELECT goals FROM LaLiga WHERE name = 'Lionel Messi'
n = n + 2
UPDATE LaLiga SET goals = n WHERE name = 'Lionel Messi'
n = SELECT goals FROM LaLiga WHERE name = 'Lionel Messi'
n = n + 3
UPDATE LaLiga SET goals = n WHERE name = 'Lionel Messi'
COMMIT
```

12. [12 points]: By accident you run this transaction twice, concurrently. For the following questions you should assume that SELECT and UPDATE statements execute atomically.

A. Suppose both runs of the transaction execute to completion under isolation level **Serializable**. What are the possible value(s) of goals for Messi?

B. Suppose both runs of the transaction execute to completion under isolation level **Read-committed**. What are the possible value(s) of goals for Messi?

Name:
C. Suppose both runs of the transaction execute to completion under isolation level \textit{Read-uncommitted}. What are the possible value(s) of goals for Messi?

D. Suppose one run of the transaction executes to completion under isolation level \textit{Read-uncommitted}, while the other one executes to completion under isolation level \textit{Read-committed}. What are the possible value(s) of goals for Messi?
VI Serializability

You are running a banking system which allows users to transfer money from their account to someone else’s account. Every account also has points associated with it: for every $100 in a user’s account, they get 1 point. Your banking system has two tables, as follows:

Accounts (name CHAR(32), balance INTEGER)
Points (name CHAR(32), points INTEGER)

Your system supports three different transactions: transfer(A, B) transfers money from A’s bank account to B’s, update-balance(A, n) sets the balance of A’s account to n, and update-points(A) recalculates the points A has from the user’s balance. These transactions are defined below:

\[\text{transfer}(A, B): \]
BEGIN TRANSACTION
L1: a = SELECT balance FROM Accounts WHERE name = A
L2: b = SELECT balance FROM Account WHERE name = B
L3: UPDATE Accounts SET balance = a+b WHERE name = B
L4: UPDATE Points SET points = (a+b)/100 WHERE name = B
L5: COMMIT

\[\text{update-balance}(A, n): \]
BEGIN TRANSACTION
L6: UPDATE Accounts SET balance = n WHERE name = A
L7: COMMIT

\[\text{update-points}(A): \]
BEGIN TRANSACTION
L8: UPDATE Points SET points = 0 WHERE name = A
L9: n = SELECT balance FROM Accounts WHERE name = A
L10: UPDATE Points SET points = n/100 WHERE name = A
L11: COMMIT

The Lx are labels for use in describing schedules; they are not part of the code.

You run three transactions concurrently (here [Tx] is the transaction id):

1. transfer(Alice, Bob) [T1]
2. update-balance(Alice, 1000) [T2]
3. update-points(Alice) [T3]

Name:
13. [10 points]:
If the initial balance of both accounts is $1,000, and both accounts initially have 10 points, what are all of the possible values for the accounts and balances after running T1, T2, and T3 concurrently using strict two phase locking?

(Write your answer in the space below.)

14. [12 points]: Now consider the following interleaving of statements.

T1   T2   T3
--   --   --
   L8
L1
L2
L3
   L6
   L7
L4
L5
   L9
   L10
   L11

A. Is this schedule serial-equivalent? If so, write down the equivalent serial order of the transactions.

B. Would this schedule be permitted by strict two phase locking with record level locks? If not, why not?

End of Quiz I

Name: