EECS 484 Sample Midterm Exam #2

Answer Book

1. **Answer all questions in this *Answer Book*.** The *Question Book* will not be graded.
2. Write your uniqname on each sheet of this *Answer Book* and your note sheet.
3. This is a closed-book exam. But you are allowed to bring notes on one double-sided 8.5x11 sheet of paper with you.
4. Please power down any electronic devices and place them in your backpack.
5. You have 120 minutes to complete this exam, and it has a total of 100 points.
6. If you see typos that are confusing, ask us to clarify. If a question is ambiguous and you don't have time to ask for clarifications, state the assumptions you made and answer the question.
7. For free-response questions, write your answer in the provided box.
8. If you need more space, find space on a different page in the answer book and answer it there. Clearly label and reference all work that you want to be graded in such a case.
9. Please sign the honor pledge. When you are done with the exam, please turn in the ***Answer Book***, the ***Question Book***, and your ***note sheet*** to a member of the teaching staff. You will be asked to show a photo ID.

Honor Code Pledge: I have neither given nor received unauthorized aid on this examination, nor have I concealed any violations of the Honor Code.

Your signature: ______________________

Your name: ______________________

Your uniqname: ______________________

Your exam room number: ______________________

Uniqname of person to your left (if none, write None): ______________________

Uniqname of person to your right (if none, write None): ______________________

(Page intentionally left blank. You can use it for rough work. You can also use your question book for rough work.)

Question 1: (15 points) ER diagrams

a.

i. (1 point) Circle either True or False:

True False X

ii. (1 point) Circle either True or False:

True False X

iii. (1 point) Circle either True or False:

True X False

iv. (1 point) Circle either True or False:

True False X

v. (1 point) Circle either True or False:

True X False

b.

i. (3 points) How many tables will be required to represent the ER diagram in the relational model?

Answer: 6

5 entities: Company, Job Vacancy, Employee, Candidate, and Manager. One additional table needed for the Post relationship.

(Interview is merged with Candidate. The entire Interview-Candidate is merged with Monitor. WorkFor is merged with Employee.)

ii. (7 points) Write the CREATE TABLE statements for the table that contains the Candidate attributes.

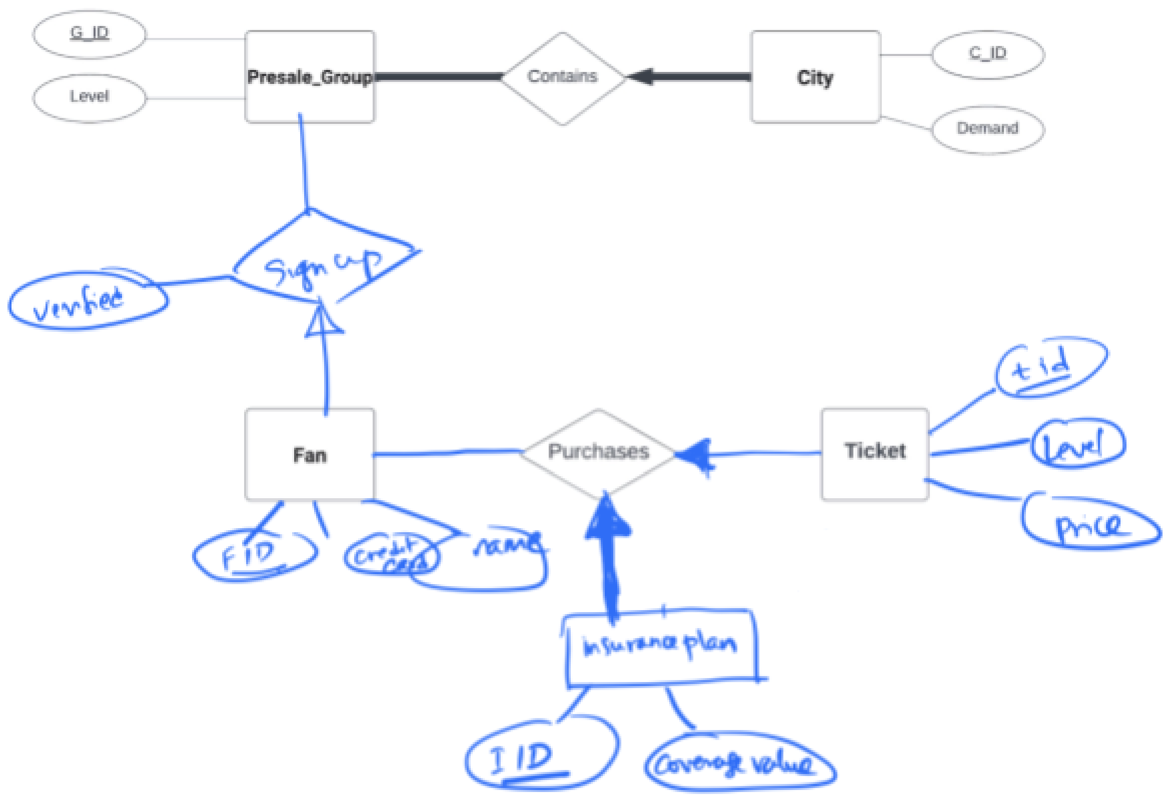
```
CREATE TABLE Candidate_Interview (
    cand_id INTEGER PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    e_id INTEGER NOT NULL,
    job_id INTEGER NOT NULL,
    m_id INTEGER,
    FOREIGN KEY (e_id) REFERENCES Employee,
    FOREIGN KEY (m_id) REFERENCES Manager,
    FOREIGN KEY (job_id) REFERENCES JobVacancy
    );
```

Elaboration on the answer above (not required from students):
We are folding the Candidate table with Interview relationship. So we need e_id and job_id to capture that. Those are set to NOT NULL because of solid arrow. But then the Interview Relationship is merged with Monitor relationship. So, we also need the manger's ID in there, assumed to be m_id in the manager table. Adding in UNIQUE for m_id would be incorrect. Same manager can monitor multiple interviews. Similarly, adding in UNIQUE for job_id or e_id also would be incorrect.

Your uniqname: _______________

Question 2: (8 points) ER Diagram Design

Fill in the missing constraints, attributes, and/or entities below.



Question 3: (23 points) Writing SQL queries

a. (3 points) Create a view called **StateUsers(user_id, state_name)** that stores what state a user currently lives in.

```
CREATE VIEW StateUsers AS
 SELECT U.user_id, C.state_name FROM
   Users U, Cities C WHERE
   U.current_city_id = C.city_id;
```

or

```
CREATE VIEW StateUsers (user_id, state_name) AS
 SELECT U.user_id, C.state_name FROM
   Users U, Cities C WHERE
   U.current_city_id = C.city_id;
```

b. (5 points) Create a view called **FriendCount(user_id, num_friends)** that stores the number of friends for each user. Some correct solution strategies:

<table>
<tr>
<td>

(Sol 1: Union ALL important with this strategy)
```
CREATE VIEW FriendCount AS
SELECT AllUsersInFriends.user_id, COUNT(*) AS num_friends
FROM (
  SELECT requester_id AS user_id
  FROM Friends
  UNION ALL
  SELECT accepter_id AS user_id
  FROM Friends
) AllUsersInFriends
GROUP BY AllUsersInFriends.user_id;
```

</td>
<td>

Sol 2: UNION suffices
```
Create View BiDirFriends(fr1, fr2) AS
  SELECT  accepter_id, requester_id
  FROM Friends  UNION
  SELECT request_id, accepter_id
  FROM Friends;

CREATE VIEW FriendCount AS SELECT b.fr1 AS user_id , COUNT(*) AS num_friends FROM BiDirFriends b GROUP BY b.fr1;
```

</td>
</tr>
<tr>
<td colspan="2">

Sol 3: Simplest solution perhaps
```
CREATE VIEW FriendCount AS
SELECT U.user_id, COUNT(*) AS num_friends FROM Users U, Friends F
WHERE U.user_id = F.requester_id OR U.user_id = F.accepter_id
GROUP BY U.user_id;
```

</td>
</tr>
</table>

Following is wrong: (Joining Friends with itself will overcount. Tested in sqlite with users being 3, 4, and 5 and Friends containing (3, 4) and (4,5) pairs.). (-2 points)

```
CREATE VIEW FriendCount AS
SELECT  U.user_id AS user_id, COUNT(*) AS num_friends
  FROM Users U, Friends F1, Friends F2
  WHERE
  U.user_id = F1.accepter_id OR U.user_id = F2.requester_id
  GROUP BY U.user_id;
```

Sol 4: *A much more complex solution exists that counts left column values separately from right column values and then adds them up by user, but requires FULL OUTER JOIN because some users may be missing in one column and we want to treat them as zeros.* COALESCE operator returns the first non-null argument in the list of arguments -- all appear to be needed. A good test of this is a friends table with (1, 2) and (2, 3).

```
CREATE VIEW V1 AS SELECT F.requester_id AS user_id, COUNT(*) AS num_friends FROM Friends F GROUP BY F.requester_id;
CREATE VIEW V2 AS SELECT F.accepter_id AS user_id, COUNT(*) AS num_friends FROM Friends F GROUP BY F.accepter_id;
-- Now add up the counts from the two tables for the same user_id. If user_id is missing, then treat count as 0.
CREATE VIEW FriendCount AS
SELECT COALESCE(V1.user_id, V2.user_id) AS user_id, COALESCE(V1.num_friends, 0) + COALESCE(V2.num_friends, 0) AS num_friends
FROM  V1 FULL OUTER JOIN V2 ON V1.user_id = V2.user_id;
```

c. (5 points) Write a SELECT query that returns a table with three columns: (**state_name**, **user_id**, **num_friends**). The table contains, for each state, the **user_id**s in that state that have the highest number of friends, and the number of friends they have.

```
-- Solution 1

SELECT S.state_name, S.user_id, FC.num_friends
FROM FriendCount FC, StateUsers S
WHERE
FC.user_id = S.user_id
AND FC.num_friends >=   (SELECT MAX(FC2.num_friends) FROM
                         FriendCount FC2, StateUsers S2
                         WHERE FC2.user_id = S2.user_id
                         AND S2.state_name = S.state_name)
ORDER BY S.state_name ASC, S.user_id DESC;

--  Solution 2
-- Find the max number of friends by state first.

CREATE VIEW STATEMAXFRIENDS AS
SELECT  S.state_name,  MAX(F.num_friends) AS MaxFriendCount
FROM StateUsers S, FriendCount F
WHERE S.user_id = F.user_id
GROUP BY S.state_name;

-- Now find the users who match the maxfriendcount in the above view by state..

SELECT S.state_name, S.user_id, F.num_friends
FROM StateMaxFriends SMF,  StateUsers S, FriendCount F
WHERE
SMF.state_name = S.state_name AND
S.user_id = F.user_id  AND
SMF.maxfriendcount = F.num_friends ORDER BY S.state_name ASC, S.user_id DESC;

-- Can drop unneeded views here.
```

2. (5 points) Create a view **Friends_of_Friends(f1, f2)** that contains the user ids of a pair of users such that the two users are not directly friends, but they share a common friend.

One Strategy:

(1) Find the pairs who are friends of friends
(2) Find the pairs of people who are directly friends of each other.

Subtract (2) from (1)
OK if duplicates are there or eliminated, since the problem didn't specify that.

It seems the easiest thing is to first create a table in which if (a,b) are friends, then (b,a) are also friends:

```
CREATE VIEW FRIENDS2 (f1, f2) AS
SELECT FR1.accepter_id, FR1.requester_id FROM Friends FR1
UNION
SELECT FR2.requester_id, FR2.accepter_id FROM Friends FR2;

CREATE VIEW Friends_of_friends (F1, F2) AS
SELECT  FS1.F1, FS2.F2
FROM Friends2 FS1, Friends2 FS2
WHERE FS1.F2 = FS2.F1 AND FS1.F1 < FS2.F2
EXCEPT
SELECT   *  FROM Friends2;
```

(MINUS is fine too instead of EXCEPT. Can drop FRIENDS2 now.)
(Friends specs don't say that the first attribute is < second attribute, so it is better to subtract Friends2 rather than Friends. The above answer does not return inverse pairs. If that is desired, then < can be replaced by <>.)

Another strategy (likely much less efficient since Non-friends table is likely to be huge):

(1) Compute non-friends (u, v) -- all pairs of users who are not in Friends2, such u <> v. (Can also do u < v).
(2) Then join non-friends with Friends2 FR1 X Friends2 FR2 to find those pairs (a, b) that are in non-friends such that (a, c) exists in FR1 and (b, c) exists in FR2.

3. (5 points) Write a query that displays all (state_name, city_name) pairs for cities belonging to a state such that city_name is shared with another state's city name. The result should have two columns (state_name, shared_city_name).

```
SELECT C1.state_name, C1.city_name
FROM Cities C1, Cities C2
WHERE
C1.city_name = C2.city_name AND
C1.state_name <> C2.state_name;
```

(OK to add DISTINCT)

a. (3 points) Return the first name and last name of users who have zero friends.

First compute ids of users who have friends:

$\rho(UsersWithFriends,\ (\pi_{accepterId}(Friends)\ \cup \pi_{requesterid}(Friends))$

$\pi_{firstname,lastname}((\pi_{userid}(\text{Users}) - \text{UsersWithFriends})\ \bowtie\ Users)$

b. (5 points) Return the names of all cities whose names are in all states (these are city names that occur in all states).

$(\pi_{cityname,,\ statename}(\text{Cities}) / \pi_{statename}(Cities))$

c. (5 points) Return the last names that are used with both of the first names 'Jim' and 'Pam'.

$\pi_{lastname}(\sigma_{firstname = 'Jim'}(Users)) \cap \pi_{lastname}(\sigma_{firstname = 'Pam'}(Users))$

Another solution:

p(Jims,$\sigma_{firstname = 'Jim'}(Users)$)

p(Pams, $\sigma_{firstname = 'Pam'}(Users)$)

$\pi_{lastname}$(Jims $\bowtie_{Jims.lastname=Pams.lastname} Pams$)

(Division can likely also be used by selecting the union of first names where they are equal to Jim and where there are equal to Pam as the right side. The left would be lastname, firstname pairs from the Users table.)

Question 5: (15 points) Functional Dependencies

a. (2 points) What are the key(s) (minimal) for the relation? List all keys if there are more than one.

A and D must be in the key, since they don't appear on the right. AD is a key.
Answers:
AD

b. (4 points) Which, if any, of the functional dependencies violate 3NF?

F -> E
C -> G
E -> B
C -> F

c.

i. (2 points) Give the resulting relations after the above BCNF decomposition.

The first FD that violates BCNF is F -> E

R1 (EF)
R2 (ABCDFG) -- F is the common attribute. E is removed from R2.

ii. (3 points) For the BCNF decomposition, give the dependencies that are preserved from the original list.

Preserved:
F -> E
C -> G
C -> F
AD -> CE (AD -> C is preserved. AD -> E is also preserved because AD -> C and C -> F -> E implies AD -> CE)
Regraded for AD -> CE, if you put it down, you got 1 point back

(Also, F -> B would be preserved, but that is not in the original list. OK to omit or include)

Not preserved:
E -> B

iii. (2 points) Give the keys for the two relations that result from the decomposition.

For R1: F is the key
For R2: AD still remains the key. That does not change.

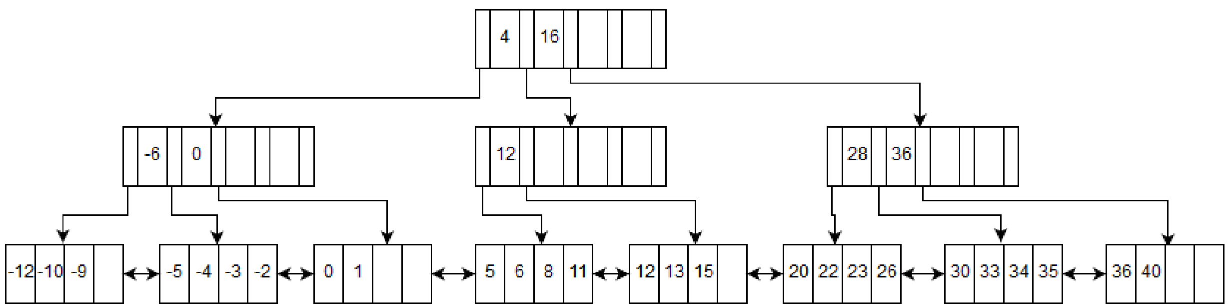
iv. (2 points) What is correct about the decomposition (enter the number or the text for the correct choice in the question book):

2.
(Lossless-join property satisfied, but not dependency-preserving.)

Question 6: (7 points) Tree Index

a. (4 points) How many page reads is this query expected to require? **Also, circle the nodes that are going to be read.**

7 reads.
-- Root node, the index node with 12 in it, and the rightmost 5 leaf nodes. (Circle these)



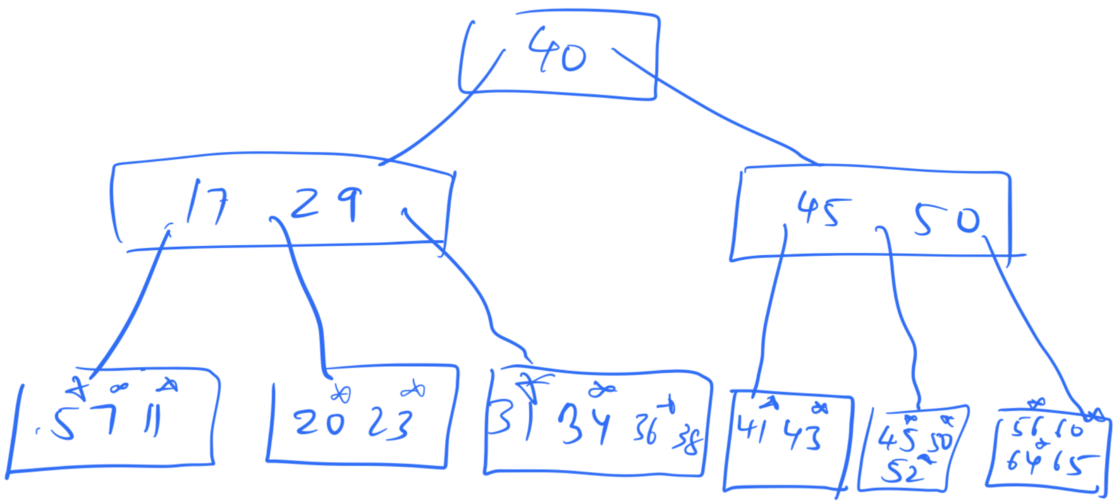
b. (3 points) How many page reads would this query have required if all attributes (i.e., SELECT * instead of SELECT rating in the above query) that satisfy the two rating constraints in the WHERE clause were being fetched?

7 for index page reads (including leaves) + 14 additional for the data pages

Total of 21.

Question 7: (20 points) B+-tree inserts and deletes

a. (10 points) Draw the final B+-tree after performing all of the following operations:
Insert 36*, 56*, 65*, and 45*.



50 should be 55 in the right index node.

b. (10 points) Start with the original B+-tree (i.e., no inserts done). Draw the final B+-tree after performing all these operations in order: Delete 7*, Delete 23*, and Delete 11*.