

CSE344 Final Exam

Winter 2017

March 16, 2017

- Please read all instructions (including these) carefully.
- **This is a closed-book exam. You are allowed two pages of note sheets that you can write on both sides.**
- Write your name and UW student number below.
- No electronic devices are allowed, including **cell phones** used merely as watches. Silence your cell phones and place them in your bag.
- Solutions will be graded on correctness and **clarity**. Each problem has a relatively simple and straightforward solution. Partial solutions will be graded for partial credit.
- There are 17 pages in this exam, not including this one.
- There are 6 questions, each with multiple parts. If you get stuck on a question move on and come back to it later.
- You have 110 minutes to work on the exam.
- Please write your answers in the space provided on the exam, and clearly mark your solutions. You may use the blank pages as scratch paper. **Do not** use any additional scratch paper.
- Relax. You are here to learn. Good luck!

By writing your name below, you certify that you have not received any unpermitted aid for this exam, and that you will not disclose the contents of the exam to anyone in the class who has not taken it.

NAME: _____

SECTION: _____

STUDENT NUMBER: _____

Problem	Points	Problem	Points
1	30	4	23
2	30	5	40
3	40	6	37
Total		200	

Problem 1: Warm up (30 points total)

Select either True or False for each of the following questions. For each question you get 2 points for answering it correctly, -1 point for an incorrect answer, and 0 point for no answer. The minimum you will get for this entire problem is 0.

1. 2PL ensures conflict serializability and recoverability.
True False

2. For every query there always exists an index that can be used to speed it up.
True False

3. Predicate locking preserves ACID.
True False

4. BCNF is a lossless decomposition and it does not preserve all functional dependencies.
True False

5. MapReduce is designed for running transactional workloads efficiently.
True False

6. Sqlite will never result in deadlock due to running transactions.
True False

7. All queries in Datalog can be expressed in relational algebra.
True False

8. Subqueries that produce scalar values can be used in a WHERE clause.
True False

9. 4NF does not preserve multi-valued dependencies.
True False

10. A serial schedule is always conflict-serializable.
- True False
11. All relational algebra operators can be expressed in MapReduce.
- True False
12. Equijoin checks the equality of all common attributes between the two relations involved.
- True False
13. Using block partitioning ensures no data skew across servers.
- True False
14. A B+-tree index is designed to speed up range queries.
- True False
15. Executing the projection operator is faster in bag semantics than in set semantics.
- True False

Problem 2: Transactions

a) Assume $R(A)$ contains the following 4 integer tuples: [10, 20, 30, 40].
Given the following transactions:

T_1 : SELECT * FROM R WHERE A > 10

T_2 : UPDATE R SET A = A + 10 WHERE A > 20; COMMIT;

i). What locks will each transaction grab, assuming that we are executing under 2PL with tuple-level locking and shared read/exclusive write locks? Write S(10) for grabbing a shared lock on the tuple 10, X(10) for exclusive lock, and U(10) for unlocking all locks. (4 points)

T_1 :

T_2 :

ii) If we are not restricted to tuple-level locking and shared read/exclusive write locks, what is the minimum number of locks that each transaction needs to grab to execute the transactions while maintaining serializability? Each transaction is executed by a separate thread. If nonzero then describe clearly what kind of lock is used and briefly explain why. (4 points)

iii) Circle the strictest isolation level where it is possible for T_1 to return the following values. (i.e., it is impossible for T_1 to return the given results under any stricter mode). (2 points each, -1 for each wrong answer, 0 for no answer, minimum 0 points) (8 points)

i) T_1 returns: 20, 40, 50

Infeasible Read uncommitted Read committed Repeatable read Serializable

ii) T_1 returns: 40, 10, 20

Infeasible Read uncommitted Read committed Repeatable read Serializable

iii) T_1 returns: 20, 30, 50

Infeasible Read uncommitted Read committed Repeatable read Serializable

iv) T_1 returns: 20, 30, 40

Infeasible Read uncommitted Read committed Repeatable read Serializable

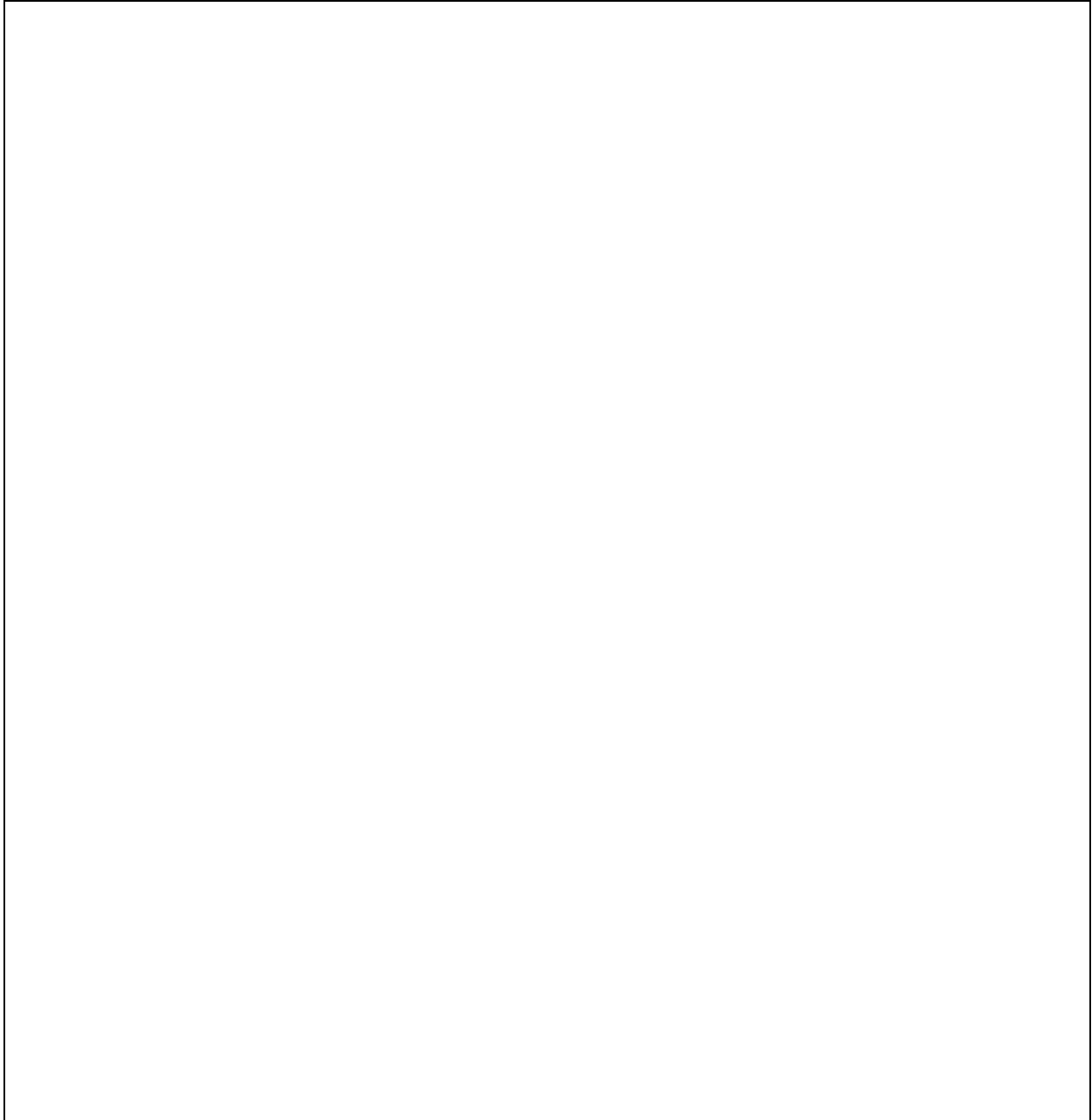
b) Given the following two transactions, devise a serializable but not conflict serializable schedule. Explain why it is a serializable schedule and why it is not conflict serializable. (8 pts)

T_1 : $R_1(C)$; $W_1(B)$; $W_1(A)$;

T_2 : $R_2(B)$; $W_2(A)$; $W_2(A)$;

c) Draw the precedence graph and conclude whether the schedule is conflict serializable. Write the equivalent serial schedule if it is conflict serializable. Otherwise write N/A. (6 points)

$R_1(A)$; $R_1(B)$; $R_1(C)$; $W_3(A)$; $R_2(A)$; $W_1(C)$; $W_2(C)$; $W_3(B)$; $R_3(C)$; $R_2(C)$; $W_3(B)$;



Problem 3: Writing Queries

Write the following queries using the schema below. Datalog and relational calculus queries are evaluated using set semantics, and SQL and relational algebra will be evaluated using bag semantics. All relational calculus queries should be domain independent and all Datalog queries should be safe. *While we are not asking for the most efficient solution, but we reserve the right to take off points if your solution is overly redundant or unnecessarily inefficient.*

Band(bid, name, genre)

Members(mid, yearJoined, name, position, bid) -- bid is foreign key to Band

Albums(aid, name, year, bid) -- bid is foreign key to Band

Tours(bid, year, location) -- bid is foreign key to Band

PlaysIn(aid, mid) -- mid played in album aid

a) Write a relational calculus query that returns the bid and name of all bands who has at least one member played in at least two different albums. (10 points)

b) Write a Datalog query that returns the names of all albums that are released before the band has performed in any tour. (10 points)

CSE 344 Final Exam Winter 2017

Schema repeated for your convenience.

```
Band(bid, name, genre)
Members(mid, yearJoined, name, position, bid) -- bid is foreign key to Band
Albums(aid, name, year, bid) -- bid is foreign key to Band
Tours(bid, year, location) -- bid is foreign key to Band
PlaysIn(aid, mid) -- mid played in album aid
```

c) Write a SQL query that returns the mid, name, and the number of times that each person has performed in Seattle, along with the band that the person is a member of. If a person has been member in multiple bands, then multiple entries should be returned, with each one showing the number of times that the person has performed in Seattle under that band. You can assume that each player will only perform after joining a band, and a player will join a given band only once or never. (10 points)

d) Write a SQL query that returns the album names of the bands where all its drummers joined between 1980 and 1990 (inclusive both ways). (10 points)

Problem 4: Conceptual Design

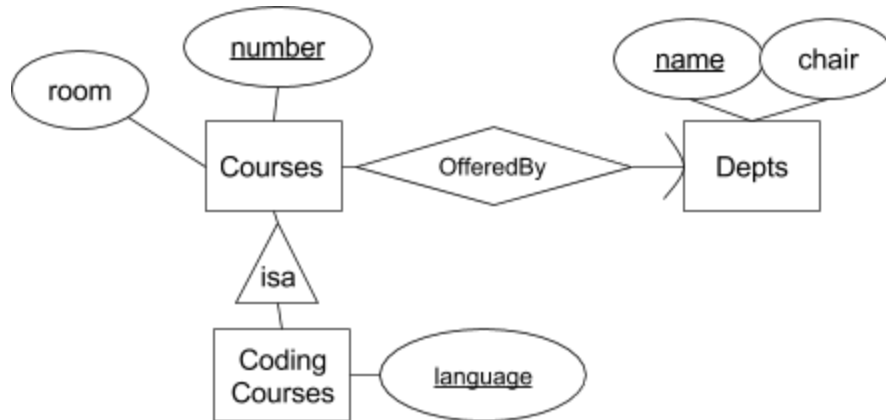
Given $R(A, B, C, D, E)$, and functional dependencies: $A \rightarrow C$, $BD \rightarrow A$, $D \rightarrow E$

a) Decompose R into BCNF. In each step, explain which functional dependency you used to decompose and explain why further decomposition is needed. Your answer should consist of a list of table names and attributes. Make sure you indicate the keys for each relation. (5 points)

c) (10 points) Convert the E/R diagram below to relations in BCNF form. Assume no values are NULL, and the arrow between OfferedBy and Depts is a round one. Include all keys and foreign keys. Use the following notation and explicitly state foreign key relationships. For instance:

R(a, b)

S(c, d) -- c is a foreign key to R



Problem 5: Parallel Query Processing

Given the following query and statistics:

```
SELECT *
FROM R, S
WHERE R.a = 10 and R.a = S.b
```

$T(R) = 100,000$	$V(R, a) = 300$	$\min(R.a) = 0$	$\max(R.a) = 1000$
$T(S) = 40,000$	$V(S, b) = 20$	$\min(S.b) = 0$	$\max(S.b) = 1000$

Assume there are no indexes on R or S. There are 4 nodes (N1, N2, N3, N4), and each node has enough main memory to hold its partition of R and S tuples.

We partition R and S using one of the following schemes:

i) block partitioned, with each node holding 25,000 tuples of R and 10,000 tuples of S.

ii) range partitioned in the following way:

N1: $0 \leq R.a < 250$ (same for S.b)	N2: $250 \leq R.a < 500$ (same for S.b)
N3: $500 \leq R.a < 750$ (same for S.b)	N4: $750 \leq R.a \leq 1000$ (same for S.b)

a) Under partition scheme i), how many tuples do you expect each partition to generate if the selection predicate was applied first? (4 points)

N1:	N3:
N2:	N4:

b) Instead of selection, suppose we evaluate the join predicate first. How many tuples do you expect each node to send to other nodes if we use broadcast join on R under partition i)? (4 pts)

N1:	N3:
N2:	N4:

c) Under partition scheme ii), how many tuples do you expect each node to generate if the selection predicate was applied first? (4 points)

N1:	N3:
N2:	N4:

d) Your roommate just invented the “distributed sort-merge join,” where each node applies the classical sort-merge join locally, and forward its S tuples to other nodes. After receiving the new S tuples from another node, each node applies sort-merge join locally again between the newly received S tuples and its existing R tuples. This repeats until the full join is computed. How many tuples will each node send out in total using this scheme if data was initially partitioned using scheme i)? (4 pts)

N1:	N3:
N2:	N4:

e) Is your roommate’s scheme any more efficient than the hash or broadcast join under either of the two partitioning schemes mentioned above? Explain why or why not. (5 points)

f) Your roommate comes up with yet another partitioning scheme: block partition R as in i), but replicate S entirely on all nodes. She finds that the query now runs faster even when compared to hash partitioning on R . a and $S.b$ (assume the nodes have enough memory to hold all tuples in either scheme). This is surprising because both schemes need to perform no network I/O for the join, and hash partitioning should read strictly less of S on every node, since S is partitioned instead of replicated. Provide one explanation for how this could occur. (5 points)

Problem 6: Short Questions

a) Consider two relations $R(A, B)$, $S(C, D)$, where all attributes are integers and cannot be NULL. For each identity below, indicate whether it is true or false. (4 points each, -2 for each wrong answer, 0 if no answer, minimum for problem 6 is 0) (12 points)

$$1) R \times S - R \bowtie_{B=C} S = R \bowtie_{B \neq C} S$$

True False

$$2) R - \Pi_{AB}(R \bowtie_{A=D} S) = \Pi_{AB}(R \bowtie_{A \neq D} S)$$

True False

$$3) \gamma_{C, \text{count}(*), \rightarrow F}(R \bowtie_{A=D} S) = \gamma_{C, \text{sum}(E), \rightarrow F}(S \bowtie_{A=D} (\gamma_{A, \text{count}(*), \rightarrow E}(R)))$$

True False

b) List one benefit of coarse grain locking over fine grain locking. (2 points)

c) Would the following Relational Algebra queries be considered domain independent? If no, which variables cause the dependence? (3 points each) (9 points)

i) $Q(a) = R(a)$

ii) $Q(a) = \forall b.(S(a, b))$

iii) $Q(a, b) = \exists c.T(a, b, c) \wedge \forall d.(T(a, b, d) \vee \exists e.S(d, e))$

d) Explain the difference between a heap file and a sequential file for storing data. (2 points)

e) Given the SQL query below:

```
SELECT distinct x.uid, x.username
FROM Usr x, Picture u, Picture v, Picture w
WHERE x.uid = u.uid AND x.uid = v.uid AND x.uid = w.uid
      AND u.size > 100 AND v.size < 300 AND w.size = u.size;
```

For each query below indicate if it is correct AND equivalent to the given query above:
(4 points each, -2 for each wrong answer, 0 if no answer, minimum for problem 6 is 0) (12 pts)

i) Is this query equivalent?

```
SELECT distinct x.uid, x.username
FROM Usr x, Picture u, Picture v
WHERE x.uid = u.uid AND x.uid = v.uid
      AND u.size > 100 AND v.size < 300;
```

True False

ii) Is this query equivalent?

```
SELECT distinct x.uid, x.username
FROM Usr x, Picture u, Picture v, Picture w
WHERE x.uid = u.uid AND x.uid = v.uid AND x.uid = w.uid
      AND u.size > 100 AND v.size < 300 AND w.size = v.size;
```

True False

iii) Is this query equivalent?

```
SELECT distinct x.uid, x.username
FROM Usr x, Picture u, Picture w
WHERE x.uid = u.uid AND x.uid = w.uid
      AND u.size > 100 AND u.size < 300 AND u.size = w.size;
```

True False

-- END OF EXAM --

-- Thank you for taking this class. Hope you learned a lot. --

-- Enjoy your spring break! --