CSE344 Final Exam
Winter 2017
March 16, 2017

- Please read all instructions (including these) carefully.
- **This is a <u>closed-book exam</u>. 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: ______Solutions_______________ SECTION: ____________

STUDENT NUMBER: ________________________________

Text in red is answer explanation or clarifications made during the exam.

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 <u>False</u>

2. For every query there always exists an index that can be used to speed it up.
   Can't speed up SELECT * FROM R -- requires full table scan regardless

True <u>False</u>

3. Predicate locking preserves ACID.

True <u>False</u>

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

<u>True</u> False

5. MapReduce is designed for running transactional workloads efficiently.

True <u>False</u>

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

<u>True</u> False

7. All queries in Datalog can be expressed in relational algebra.
   See 5.4.7 in textbook.

True <u>False</u>

8. Subqueries that produce scalar values can be used in a WHERE clause.
   See 6.3.1 in textbook.

<u>True</u> False

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

<u>True</u> False

10. A serial schedule is always conflict-serializable.

<u>True</u> False

11. All relational algebra operators can be expressed in MapReduce.

<u>True</u> False

12. Equijoin checks the equality of all common attributes between the two relations involved.

True <u>False</u>

13. Using block partitioning ensures no data skew across servers.

<u>True</u> False

14. A B+-tree index is designed to speed up range queries.

<u>True</u> False

15. Executing the projection operator is faster in bag semantics than in set semantics.

Requires removing duplicates in set semantics.

<u>True</u> False

Problem 2: Transactions

a) Assume `R(A)` contains the following 4 integer tuples: [10, 20, 30, 40].
Given the following transactions: Assume there are no indexes. Only write out the LOCKING related operations in the transactions. U(X) releases all locks on X, regardless of shared or exclusive.

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: `S(10), S(20), S(30), S(40), U(10), U(20), U(30), U(40)`

T_2: `S(10), S(20), S(30), S(40), X(30), X(40), U(10), U(20), U(30), U(40)`

We need to first grab the read (i.e., shared) locks of all elements in order to determine which ones to return / write, since there are no indexes in this case.

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)

Only 1 lock needed --- an exclusive table level lock on R.

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)
Write N/A if no such schedule exists.

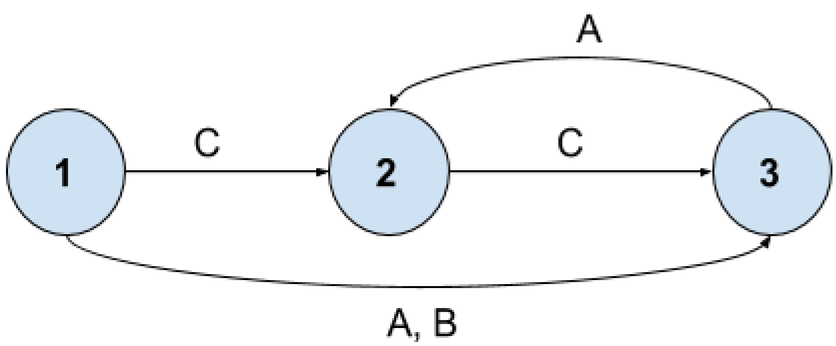
T_1: $R_1(C)$; $W_1(B)$; $W_1(A)$;
T_2: $R_2(B)$; $W_2(A)$; $W_2(A)$;

T_1: $R_1(C)$; $W_1(B)$; $W_1(A)$;
T_2: $R_2(B)$; $W_2(A)$; $W_2(A)$;

This is serializable to T1; T2, yet not conflict serializable due to the W-W conflict between $W_2(A)$;$W_1(A)$;

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)

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



N/A.

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.*

You can assume that anyone can be a member of only one band at any given time.

```
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
-- aid is foreign key to Albums, and mid is foreign key to Members
```

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)

The album in "at least two different albums" can be from different bands

```
Ans(b, n) = ∀g Band(b, n, g) ∧ ∃m, n1, y, p Members(m1, y, n1, p, b)
            ∧ ∃ a1, a2, n2, n3, y2, y3 Album(a1, n2, y2, b1)
            ∧ Album(a2, n3, y3, b2) ∧ PlaysIn(a1, m)
            ∧ PlaysIn(a2, m) ∧ a1 != a2;
```

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)

```
NonAns(aid, name) :- Albums(aid, name, year1, bid), Tours(bid, year2, _),
                     year1 >= year2
Ans(name) :- Albums(aid, name, _, _), not NonAns(aid, name)
OR
A(bid, year) :- Tours(bid, year, _), Tours(bid, year2, _), year > year2
FirstTour(bid, year) :- Tours(bid, year, _), not A(bid, year)
Ans(name) :- Albums(_, name, year, bid), FirstTour(bid, y2), year < y2
```

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) Return both the bid and name of the band in addition to other attributes.

```
SELECT m.mid, m.name, count(*) AS numPerformances, b.bid, b.name
FROM Members m, Band b, Tours t
WHERE m.bid = b.bid AND t.bid = b.bid
AND t.location = 'Seattle' AND m.yearJoined < t.year
GROUP BY m.mid, m.name, b.bid, b.name
```

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)
Assume each band has at least one drummer at any given time.

```
SELECT a.name
FROM Album a
WHERE NOT EXISTS (SELECT * FROM Members m
                  WHERE  m.bid = a.bid AND m.position = "drummer" AND
                         (m.yearJoined < 1980 OR m.yearJoined > 1990))
```

Problem 4: Conceptual Design

Given R(A, B, C, D, E), and functional dependencies: A → C, BD → A, D → 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)

One possible decomposition:

1. Use A → C:

Decompose R into R1(A, C) and T(A, B, D, E)

T violates BD → A and D → E, so we need to further decompose T

2. Use D → E:

Decompose T into R2(B, D, A) and R3(D, E)

Final relations: R1(<u>A</u>, C), R2(<u>B, D</u>, A), and R3(<u>D</u>, E)

Given R(A, B, C, D, E), and functional dependencies: A → C, BD → A, D → E

b) i) Suppose we decomposed R into R_1(B, C, D), R_2(A, B, D), and R_3(A, E) (this is not the answer to a) by the way...). Is this decomposition lossless? Verify this by showing the final tableau after applying the chase algorithm. Make sure you indicate which functional dependency (FD) did each row in the tableau come from. (6 points)

A	B	C	D	E	Which FD?
a1	b	c	d	e1	(initial tableau)
a	b	c1	d	e2	(initial tableau)
a	b1	c2	d1	e	(initial tableau)
a	b	c	d	e1	Apply BD → A
a	b	c1	d	e2	Apply BD → A
a	b1	c2	d1	e	Apply BD → A
a	b	c	d	e1	Apply A → C
a	b	c	d	e2	Apply A → C
a	b1	c	d1	e	Apply A → C

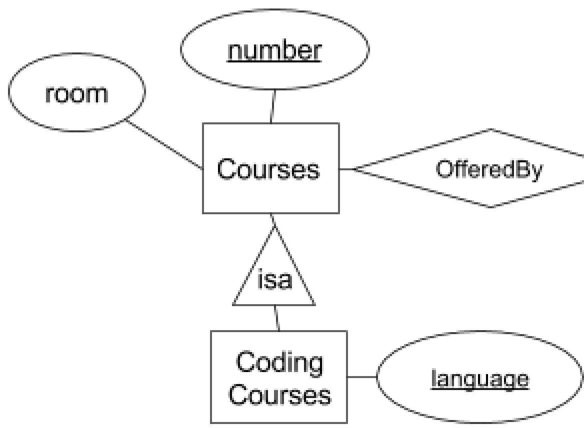
Final tableau shown in blue (D→ E does not induce any further changes)

ii) Decomposition is: <u>Lossy</u> Lossless

(circle one, 2 pts if correct, -1 if wrong, 0 if no answer, minimum for problem 4 is 0)

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
```




```
Courses(number, room, name) -- name is foreign key to Depts

CodingCourses(language, number) -- number is foreign key to Courses

Depts(name, chair)
```

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 <= R.a < 250	(same for S.b)	N2: 250 <= R.a < 500	(same for S.b)
N3: 500 <= R.a < 750	(same for S.b)	N4: 750 <= R.a <= 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: R: 25k*(1/300) = 83.3, S: 10k

N3: R: 25k*(1/300) =83.3, S: 10k

N2: R: 25k*(1/300) = 83.3, S: 10k

N4: R: 25k*(1/300) = 83.3, S: 10k

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)
"broadcast join on R" means R is broadcast to other nodes

N1: R: 25k to each node (75k if count total)

N3: R: 25k to each node (75k if count total)

N2: R: 25k to each node (75k if count total)

N4: R: 25k to each node (75k if count total)

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: R: 100k * (1/250) = 400
Due to range partitioning, N1 only gets values between 0 and 249. Since we are selecting R.a=10, we have 1/250 chance of picking a value that satisfies the selection predicate. Hence the expected number of tuples is 400.

N3: 0

N2: 0

N4: 0

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: 10k (30k if count duplicates)

N3: 10k (30k if count duplicates)

N2: 10k (30k if count duplicates)

N4: 10k (30k if count duplicates)

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)

The scheme from d) is the same as a broadcast join on S to all other nodes, so that will require 10k x 3 x 4 = 120k network I/O (counting duplicates).

Let's compare that with the two partitioning schemes and the two join algorithms mentioned:

- doing block partitioning and hash join requires shuffling all R and S tuples since the intial partitioning was done randomly. This costs 100k + 40k = 140k I/O, and scheme d) is better in this case.

- doing block partitioning and broadcast join on R requires sending 25k * 3 * 4 = 300k I/O, and scheme d) is better in this case.

- doing range partitioning and hash join requires 0 I/O in this case, since each node can just perform join locally. Hence this is better than scheme d).

- doing range partitioning and broadcast join on R requires sending either 4k * 3 * 4 = 48k I/O (if we assume that each partition gets 100k/250 = 4k tuples, which is an underestimate), or 300k I/O (if we assume that each partition gets 100k/4 = 25k tuples). Scheme d) is better in one of the cases.

We gave credit if students explained which case they assumed and described why the new scheme is better or worse.

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)

R.a is skewed. Using hash partitioning might end up hashing all R tuples to one node, and the executing local join on that node will make it become the bottleneck. On the other hand S.b being skewed has no effect since under the new scheme S is replicated entirely on all nodes.

g) Given temperature sensor data containing the sensor ID as the key and a list of temperature readings from that sensor as the value, write a MapReduce program to calculate the largest sensor id that found each temperature value in parallel. For instance, if the following sensors reported the following temperature readings:

Sensor ID	Readings
1	[78, 81, 79]
2	[80, 81, 83]
3	[76, 78, 81]

Your program should return: (76, 3), (78, 3), (79, 1), (80, 2), (81, 3), (83, 2). Assume that sensor ID is an integer and temperature readings are a list of integers. Do not worry too much about getting Java syntax correct, but make sure you clearly state what key-value pairs are emitted by map and how they are reduced. A sequential implementation will not receive credit. (10 points)

```
void map(int id, List<Integer> temps) {
    for (int temp : temps) {
        EmitIntermediate(temp, id);
    }
}

void reduce( int temp, List<Integer> ids ) {
    int maxId = -1;
    for (int id : ids) {
        if (id > maxId) {
            maxId = id;
        }
    }
    Emit(temp, maxId);
}
```

h) Explain briefly how your computation is parallelized. (4 points)

Each reducer processes a particular temperature value and all the sensors that generated that reading in parallel.

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

<u>True</u> False

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

True <u>False</u>

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

<u>True</u> False

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

Fewer deadlocks (incorrect to say no deadlocks since coarse grain locking doesn't necessarily mean table level locking), less locking overhead, fewer chances of thrashing.

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)

Yes

ii) Q(a) = ∀b.(S(a, b))

No, b

iii) Q(a, b) = ∃c.T(a, b, c) ∧ ∀d.(T(a, b, d) ∨ ∃e.S(d, e))

No, d

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

Heap file stores tuples in a random order while sequential file stores tuples in a prescribed order based on the chosen attribute.

e) Given the SQL query below:

```
SELECT distinct x.uid, x.uname
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.uname
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;
```

<u>True</u> False

ii) Is this query equivalent?

```
SELECT distinct x.uid, x.uname
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;
```

<u>True</u> False

iii) Is this query equivalent?

```
SELECT distinct x.uid, x.uname
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 <u>False</u>

-- END OF EXAM --
-- Thank you for taking this class. Hope you learned a lot. --
-- Enjoy your spring break! --