CSE 414 Final Exam
Autumn 2019
December 12, 2019

- Please read all instructions (including these) carefully.
- **This is a closed-book exam. You are allowed a page 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.
- Each problem has a relatively simple and straightforward solution. Partial solutions will be graded for partial credit.
- There are 11 pages in this exam, including this one.
- 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.

Relational algebra operators:

Union ∪ Difference — Selection σ Projection π Join ⋈
Rename ρ Duplicate elimination δ Grouping and aggregation γ Sorting τ

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__

STUDENT NUMBER: _____

Text in red indicates clarifications post-exam

Problem	Points	Problem	Points
1	20	4	21
2	24	5	17
3	18		
Total		**100**	

Problem 1: Warm Up (20 points total)

Select either True or False for each of the following questions. For each question you get 2 points for answering it correctly. There is no penalty for an incorrect answer.

a) An example of a database constraint is the SQL keyword FOREIGN KEY.

True False

b) A functional dependency can hold on a particular relation and not be a key to that relation.

True False

c) In SQL, only one index on a table can be unclustered.

True **False**

d) In query optimization we consider many logical plans and many physical plans for a particular SQL query.

True False

e) In transactions, if a schedule is conflict serializable then it satisfies all ACID properties.

True **False**

f) Semi-structured data can be visualized in the form of a tree data structure.

True False

g) In the query plan of a parallel RDBMS, we always need to shuffle the data at least one time.

True **False**

h) In MapReduce, all intermediate results of a query are written to disk.

True False

i) A locking scheduler is one that uses multi-version concurrency control.

True **False**

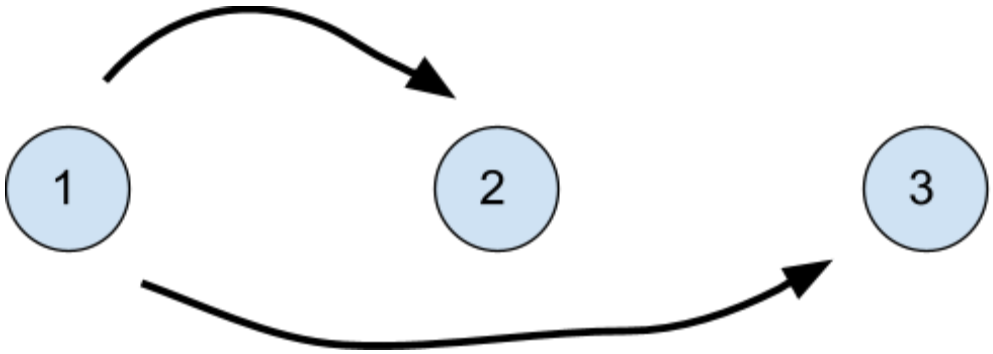
j) If a scheduler uses two-phase locking (2PL), deadlocks are not possible.

True **False**

Problem 2: Transactions (24 points total)

a) Draw the full precedence graph of the following schedule. Is this schedule conflict serializable? If so, write YES and an example order in which the transactions would have to run in the equivalent serial schedule. Otherwise write NO. (6 points)

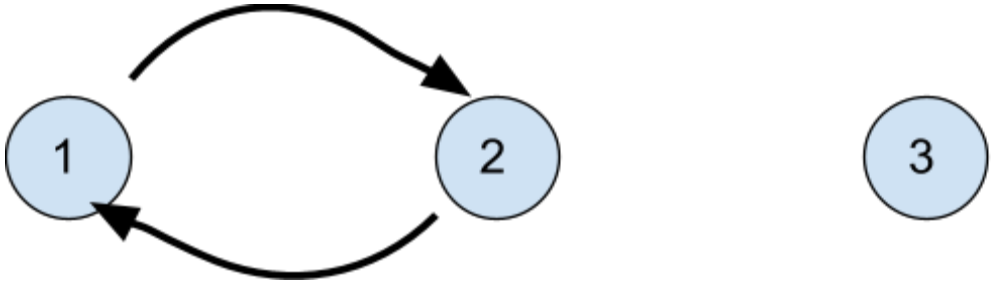
$R_1(A);\ W_1(A);\ R_2(A);\ R_1(B);\ R_3(B);\ W_3(B);\ W_2(A);\ R_3(B);$



Answer: **Yes**, it is conflict serializable. The order is **1 → 2 → 3** or **1 → 3 → 2**

b) Draw the full precedence graph of the following schedule. Is this schedule conflict serializable? If so, write YES and an example order in which the transactions would have to run in the equivalent serial schedule. Otherwise write NO. (6 points)

$R_2(B);\ R_1(B);\ R_1(A);\ W_2(B);\ R_3(A);\ W_1(B);$



Answer: **No**, it is not conflict serializable.

c) Read the schedule below and answer the following questions. A and B are each a field of a tuple in the database. You can assume writes to the database are only updates, not insert/deletes.

Time	Transaction 1	Transaction 2
1	Begin transaction	
2		Begin transaction
3		Lock(B)
4	Lock(A)	
5	Read(A)	
6		Read(B)
7	Write(A)	
8	Lock(B) **denied, waiting...**	
9		Write(B)
10		Commit, Unlock(B)
11	Lock(B) **granted**	
12	Write(B)	
13	Commit, Unlock(A), Unlock(B)	

This schedule follows the rules of two-phase locking. (3 points)

True False

This schedule follows the rules of strict two-phase locking. (3 points)

True False

This schedule is serializable. (3 points)

True False

This schedule has a deadlock. (3 points)

True **False**

Problem 3: BCNF Decomposition (18 points total)

Assume the following functional dependencies hold on the relation R:

R (A, B, C, D, E, F)

B -> AC
A -> BC
CE -> AD

a) Write the closures of the following attributes: (12 points)

$\{A\}^+$ = _{ABC}_____

$\{C\}^+$ = _{C}_____

$\{BE\}^+$ = _{ABCDE}_____

$\{AF\}^+$ = _{ABCF}_____

b) Decompose R into Boyce-Codd Normal Form with respect to the above functional dependencies, indicating the new relations and their attributes (for example R1(A, B...), R2(B, C...) (6 points)

Multiple answers possible depending on order of decomposition. E.g.

A -> ABC first:
R1(ABC) R2(ADEF)
(done)

C -> ABC first:
R1(ABC) R2(CDEF)
CE -> CDE second:
R1(ABC) R21(CDE) R22(CDF)
(done)

Problem 4: Indexing and Query Optimization (21 points total)

Assume we have relations R and S in a database, along with statistics on their attributes as shown below:

```
R(a integer,
  b float
  c integer)

S(d integer,
  e float)
```

R	S
T(R) = 1,000 V(R, a) = 100 V(R, b) = 1,000 V(R, c) = 10 minimum value of b: 0.0 maximum value of b: 100.0	T(S) = 5,000 V(S, d) = 500 V(S, e) = 5,000 minimum value of e: 50.0 maximum value of e: 150.0

`T(X)` is the number of tuples in a relation `X`.
`V(X, y)` is the number of distinct values for the attribute `y` in the relation `X`.
We assume the values of the attributes are uniformly distributed over their range.

For each of the queries below, estimate the number of tuples that will be returned in the output. (3 points each)

a)

```
SELECT *
FROM   R
WHERE  R.b > 25.0
```

X = (100.0- 25.0)/100.0 - 0.0) = .75

X * T(R) = 750

750 tuples

b)

```
SELECT *
FROM   S
WHERE  S.e > 70.0 AND S.e < 80.0
```

X = (80.0 - 70.0) / (100.0 - 0.0) = 0.1
X * T(S) = 500

500 tuples

c)

```
SELECT *
FROM   S
WHERE  S.d = 487
```

X = 1/V(S,d) = 1/500
X * T(S) = 10

10 tuples

d)

```
SELECT *
FROM   R
WHERE  R.a = 15 AND R.c = 82
```

X = X1 * X2 = 1/V(R,a) * 1/V(R, c) = 1/1000
X * T(S) = 1

1 tuples

e)

```
SELECT *
FROM   R, S
```

No join condition -> Cross product T(R) * T(S)

5,000,000 tuples

As the database designer, you know that the most common query that will be run on the system is:

```
SELECT R.a, S.c
FROM   R, S
WHERE  R.a = S.d AND
       R.c = X
```

where X is some integer defined by the user at runtime

a) Assume you have an unclustered index on R.c and no other indexes or sorting of the tables exists yet. Do you expect query performance to improve if you can change the unclustered index on R.c to a clustered index?
Explain why or why not. (4 points)

Yes, we would expect query performance to improve.
The number of distinct values of c in R is only 10, meaning the selectivity factor for that statement is only 1/10th. Unclustered indexes only become faster than sequential scans for selectivities less than 1/100th, so the clustered index could potentially speed up the query more than the unclustered index.

b) What is another attribute in either R or S that might speed up other possible queries if you created an unclustered index on it? Write the attribute name below. (2 points)
(Note, this would not be a clustered index)

S.d

By similar logic to the above, R.b, R.d, S.d, and S.e have high numbers of distinct values and are thus potentially useful as unclustered indexes and so any of them are acceptable. R.a is close to the threshold of 2%.

Problem 5: Parallel Databases and JSON(17 points total)

Say you are designing a parallel relational database to store data about films. You have three tables:

MOVIE (mid, name, year, revenue)
DIRECTORS (did, fname, lname)
MOVIE_DIRECTORS (did, mid)

`Movie_Directors.did` is a foreign key reference to `Directors.did`, and `Movie_Directors.mid` is a foreign key reference to `Movie.mid`. A tuple in `Movie_Directors` represents a director with `did` directing a movie with `mid`.

There is a large amount of data in all tables that would have to be spread between multiple machines to fit in the entire database. All tables have about the same number of tuples in them.

As the database designer, you know that the most common query that will be run on the system is:

```
SELECT d.fname, d.lname, COUNT(*)
FROM Directors d, Movie_Directors md
WHERE d.did = md.did AND d.fname = '[some name]' AND d.lname = '[some name]'
GROUP BY did
```

a) For each table, describe in a sentence or two how you would partition the table between machines if your goal is to maximize performance of the above query. To maximize performance, you should avoid as much shuffling of tuples as possible during query execution. (7 points)

`Directors`
We should **hash partition on d.did**, since that is the join condition of the query and group by attribute. If we do this, we can avoid shuffling the data entirely.

`Movie_Directors`
We should **hash partition on md.did**, since that is the join condition of the query and group by attribute. If we do this, we can avoid shuffling the data entirely.

`Movie`

Since movie isn't used in the query, you have a choice of how to partition it. Any of the schemes we learned are reasonable, as long as they split the data among multiple machines and not just one.

b) Now instead of the relational database above, you are working with a NoSQL document store version of a movies database. The general structure of the data is that each key is the unique movie identifier `mid`, and the value for that key is a JSON object with key-value pairs representing the information of the movie with that `mid`. For example, one entry is the following (key, value) pair for a movie with `mid=101`:

```
(101, {
      "name": "The Third Man",
      "year": 1949,
      "revenue": 105209,
      "cast": [
            {
                  "pid": 237
                  "fname": "Alida"
                  "lname": "Valli"
                  "role": "Anna Schmidt"
            },
            {
                  "pid": 937257
                  "fname": "Orson"
                  "lname": "Welles"
                  "role": "Harry Lime"
            },
            ...
            ]
      "director": {
                  "did": 23
                  "fname": "Carol"
                  "lname": "Reed"
            }
} )
```

Attributes in the JSON object can be referenced by their names. For example, in the above example of `key` 101 and `value` as the movie JSON object, `value.year` is the year field `1949`. The revenue attribute is the total amount of money made by the movie, in US dollars.

Answer the following questions about the NoSQL document store version of the Movies database.

a) What type is the value of **"`director`"** in the above object for the movie "The Third Man". (2 points)

It is the "object" type

b) Consider how you would represent a movie with multiple directors, like The Lego Movie. What type would you use for the value of "`director`" in that new movie object, and why? (4 points)

The most natural choice is to use an array for "director", and have each director as a separate object in that array. This makes it easy to do things like counting the number of directors in a later query.

b) Is there a benefit to using the same type for all entries with the same name (e.g. "director" above) in a semi-structured database? Explain in a sentence or two. (4 points)

Using the same type for director (e.g. using an array for the director entry of all movies, so that a single director is represented by an array of one element; a movie without director is represented by an empty array; etc.) creates a more uniform schema. This can help with data compression, integrity checking, and simplifying queries (less need for case statements, type conversions, etc.).

-- END OF EXAM --
-- Thank you for taking this class. We hope you learned a lot. --
-- Enjoy your winter break! --