Midterm Exam
CS 190 (CS 122D)
Winter 2020

Max. Points: 100
(Please read the instructions carefully)

Instructions:
- The total time for the exam is 80 minutes; be sure to budget your time accordingly.
- The exam is closed book and closed notes but “open cheat sheet”.
- Read each question first, in its entirety, and then carefully answer each part of the question.
- If you don’t understand something, ask one of the exam proctors for clarification.
- If you still find ambiguities in a question, note the interpretation you are taking.

<table>
<thead>
<tr>
<th>QUESTION</th>
<th>TOPIC</th>
<th>POINTS</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True/False</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>E-R</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>Key-Value Stores</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>SQL</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>NoSQL</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Cassandra</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>TOTAL</td>
<td>All</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
Question 1: Truth and Falsehood Are Still a Thing (30 pts)

(3 pts each) For each of the following questions, circle the appropriate response. You'll need to reference the following abbreviated SQL DDL for the query questions (PKEY = PRIMARY KEY, FKEY = FOREIGN KEY, REFS = REFERENCES). Non-PKEY fields may hold NULL values unless otherwise specified. As you answer the questions, you should assume that only these tables and indexes exist:

CREATE TABLE Dept (dno int PKEY, dname text NOT NULL, phone text);
CREATE TABLE Prof (pno int PKEY, pname text, salary int, rank text, deptno int FKEY REFERENCES Dept(dno));
CREATE INDEX ProfSalaryIndex ON Prof(salary);

a) In a parallel DBMS with local secondary indexes, the system may be able to execute the query `SELECT * FROM Prof WHERE salary = 9000` without involving all nodes in the cluster.

   TRUE  FALSE

b) In a parallel DBMS with global secondary indexes, the system may be able to execute the query `SELECT * FROM Prof WHERE salary = 9000` without involving all nodes in the cluster.

   TRUE  FALSE

c) In a parallel DBMS with global secondary indexes, the system may be able to execute the query `SELECT * FROM Dept WHERE dname = 'CS'` without involving all nodes in the cluster.

   TRUE  FALSE

d) A parallel DBMS is said to exhibit linear speedup if adding more nodes (e.g., tripling the cluster size) enables it to handle a proportionally bigger problem (e.g., querying over 900GB of data rather than 300GB) in the same amount of time (e.g., 2 minutes) as the original problem.

   TRUE  FALSE

e) The North American Van Lines photo in the parallel DBMS lecture (Lecture 2) was a picture of a 1 petabyte Teradata system on its way to Kmart in the mid 1980’s.

   TRUE  FALSE

f) For many years, relational database systems were the most (perhaps the only?) successful commercial application of parallel processing.

   TRUE  FALSE

g) The Hybrid Hash Join algorithm uses hashing to parallelize large joins over parallel DBMS clusters, but requires that the data sent to each node must fit in main memory for the local join.

   TRUE  FALSE

h) A disadvantage of JSON is that it is more complex than XML and has a larger specification.

   TRUE  FALSE

i) A multi-valued attribute in E-R must be translated to a relational side table to satisfy 1NF.

   TRUE  FALSE

j) A composite attribute in E-R must be translated to a relational side table to satisfy 1NF.

   TRUE  FALSE

SCORE: 30
**Question 2: To E-R is Human (15 pts)**

Consider the following E-R schema:

![E-R Diagram]

Consider how you could translate this E-R schema into a relational schema.

(a) (9 pts) Give a set of CREATE TABLE statements based on the delta tables approach to translating ISA hierarchies into relational schemas. To do so, feel free to borrow the abbreviated CREATE TABLE notation from Question 1. Be sure to specify all appropriate primary (PKEY) and foreign (FKEY) keys.

```
CREATE TABLE Person (ssno int PKEY, name text, age int);
```

```
CREATE TABLE Student (ssno int PKEY, major string, gpa float, FK ssno REFs Person(ssno));
```

```
CREATE TABLE Faculty (ssno int PKEY, rank string, area string, FK ssno REFs Person(ssno));
```

(b) (6 pts) A 21-year-old named Susanne with ssno 123 is a student majoring in Data Science with a GPA of 3.8. A 63-year-old faculty member named Mike with ssno 456 is a Professor whose area is Computer Science and a Music major with a GPA of 4.0. Draw a picture of your three tables containing the data for these two people.

| Person | | | |
|--------|--------|-------|
| ssno   | name   | age   |
| 123    | Susanne| 21    |
| 456    | Mike   | 63    |

| Student | | | |
|---------|--------|-------|
| ssno    | major  | gpa   |
| 123     | Data Science | 3.8 |
| 456     | Music   | 4.0   |

| Faculty | | | |
|---------|--------|-------|
| ssno    | rank   | area  |
| 456     | Professor | Computer Science |

**SCORE:** 15
Question 3: Consider Your Key Values (20 points)

Anticipating an eventual huge demand for CS122D, the instructor decided to use a key-value store with 3-way peer-to-peer replication and eventual consistency to store students' final exam scores. The key for the final exam bucket is student ID, and the value is the associated final exam grade on a scale of 0-100. Unfortunately, the midterm exam turned out to be a disaster, so all but three students dropped the class right after taking the midterm. Shortly before the final exam, the instructor inserted the three survivors – students 104, 37, and 222 – into the key-value store bucket with initial exam scores of 0.

The TAs (TA1 and TA2) have access to the key-value store as well. TA1 updated the students' exam scores at midnight on the last day of finals week, once the final exam grading was done. Student 104 got a 95, student 37 got a 100, and student 222 got a 90 on the final. TA1 did this by writing a script to read each student’s score record by id, add the final exam score to the current value associated with the id, and then write the updated score back. TA2 was asked to verify the entered scores, comparing what he sees in the key-value store to a written list of scores.

(a) (5 pts) TA1’s script runs using the peer-to-peer store’s quorum reads and writes. Suppose TA2 then writes a script to read the scores and that he runs his script also using quorum reads. How many possible combinations of scores are there that TA2’s script might see, if he runs it after receiving an “I'm done” text message from TA1? Circle your answer below and explain it (ever so) briefly.

Number of possible combinations (3 pts): 1 2 3 6 8

Brief explanation (2 pts): The use of quorums ensures strong consistency. Thus, TA2 will see TA1's writes - seeing the values 95, 100, and 90.

(b) (3 pts) When TA1’s script runs, at least how many copies of each score record will receive the data for each write before his script’s write operations are acknowledged? Circle your answer below.

0 1 2 3 > 3

(c) (3 pts) After TA1’s script runs, at least how many copies of each score record will eventually receive and persist the data for each write? Circle your answer below.

0 1 2 3 > 3

(d) (3 pts) Suppose that the instructor wants to check the TAs' work and runs a script like TA2's script at 9AM on the first day after finals week—but his script runs in read-one (eventually consistent) mode. How many possible combinations of scores might his script see?

1 2 3 8 > 8

(e) (3 pts) Suppose that the instructor instead runs his script runs in bounded staleness mode with a time period of 1 hour. How many possible combinations of scores might his script see in this case?

1 2 3 8 > 8

(f) (3 pts) In D. Abadi's IEEE Computer article, he argued that the CAP Theorem is too simplistic in its characterization of the tradeoffs routinely made in distributed data stores. He pointed out that some systems opt not to worry about consistency at all, even with a non-partitioned network – tolerating lost or even incorrect data values. He called his extended model, appropriately, the CRAP Theorem. ( )

TRUE FALSE

SCORE: 20
Question 4: YesSQL (10 points)

Consider once again the simple relational schema from Question 1, and assume the data is stored in PostgreSQL:

```
CREATE TABLE Dept (dno int PK, dname text NOT NULL, phone text);
CREATE TABLE Prof (pno int PK, pname text, salary int, rank text, deptno int FKEY REFS Dept(dno));
```

(a) (8 pts) Write a SQL query that outputs, for each rank, the number of professors and the average salary for faculty at that rank.

```
SELECT p.rank, COUNT(*) AS cnt, AVG(p.salary) AS avgsal
FROM Prof p
GROUP BY p.rank
```

(b) (2 pts) What will your query produce for the rank ‘Emeritus’ if one of the professors at that rank has a NULL salary? Answer briefly, by example, assuming that there are a total of three Emeritus professors and that the other two have salaries of 10000 and 20000, respectively.

```
<table>
<thead>
<tr>
<th>Rank</th>
<th>cnt</th>
<th>avgsal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emeritus</td>
<td>3</td>
<td>15000</td>
</tr>
</tbody>
</table>
```

Question 5: NoSQL (10 points)

Suppose UCI has a collection UCI.Prof in MongoDB that stores the faculty, and that it currently holds:

```
{"pno": 1, "pname": "Alex", "salary": 10000, "rank": "Emeritus", "deptno": 1}
{"pno": 2, "pname": "Ramesh", "salary": 20000, "rank": "Emeritus", "deptno": 2}
{"pno": 3, "pname": "Deb", "rank": "Emeritus", "deptno": 3}
{"pno": 4, "pname": "Jim", "salary": 40000, "rank": "Assistant", "deptno": 4}
{"pno": 5, "pname": "Susan", "salary": 50000, "rank": "Full", "deptno": 5}
{"pno": 6, "pname": "Vibha", "salary": 60000, "rank": "Full", "deptno": 6}
```

Write a pymongo `UCI.Prof.find( .... )` based Python code snippet that will return the equivalent of what the following PostgreSQL query would return:

```
SELECT pname, rank FROM Prof WHERE salary > 40000 ORDER BY pname ASC LIMIT 2;
```

```
UCI.Prof.find({'salary': {'$gt': 40000}, '_id': 0, 'pname': 1, 'rank': 1})
```

SCORE: _____
Question 6: So Many Systems, So Little Time (15 points)

UCI had the most undergraduate applicants of any UC campus this year – so maybe we should migrate all our data to Cassandra? Consider one last time the information we have about departments and professors:

CREATE TABLE Dept (dno int PKEY, dname text NOT NULL, phone text);
CREATE TABLE Prof (pno int PKEY, pname text, salary int, rank text,
                    deptno int FKEY REFS Dept(dno));

The Chancellor has deemed the following parameterized query to be very important:

```
SELECT d.dname, d.phone, p.pname, p.rank
FROM Dept d JOIN Prof p ON d.dno = p.deptno
WHERE d.dname = ? AND p.rank = ? ORDER BY p.pname;
```

(a) (10 pts) Write the Cassandra CREATE TABLE statement for a table that will be well-suited to answering this class of query very quickly for the Chancellor. (Pay special attention to your specification of its PRIMARY KEY!) Then write the CQL SELECT statement that the Chancellor can run to get his desired query results.

Table: (6)

```
CREATE TABLE DeptProf (dname text, dphone text, pname text, prank text)
PRIMARY KEY ((dname, prank), prname, pno); -- pno needed for key uniqueness
```

CQL Query: (4)

```
SELECT dname, dphone, prname, prank
FROM DeptProf
WHERE dname = ? AND prank = ? ORDER BY prname;
```

(b) (5 pts) Suppose that the Chancellor also wants to run a fairly similar query without the second (rank) parameter being specified, i.e.:

```
SELECT d.dname, d.phone, p.pname, p.rank
FROM Dept d JOIN Prof p ON d.dno = p.deptno
WHERE d.dname = ? ORDER BY p.pname;
```

Will this be feasible against your table from (a)? If so, show the (full!) CQL query that does it – and if not, explain why it’s not possible with your particular table design.

(3)

No, as the predicate doesn’t fully specify the partition key.
(2) If yes, with allowance added to the query, as that still won’t do it.

SCORE: _____