CSE 414 Final Exam

June 8, 2015

Name _____

Question 1	/ 18
Question 2	/ 4
Question 3	/ 6
Question 4	/ 6
Question 5	/ 20
Question 6	/ 15
Question 7	/ 15
Question 8	/ 12
Question 9	/ 12
Question 10	/ 8
Question 11	/ 12
Question 12	/ 12
Question 13	/ 10
Total	/ 150

The exam is closed book, closed notes, closed computers, closed electronics devices, closed phones of the smart or not-so-smart variety, closed telegraphs, closed gadgets of any other kind, open mind.

The exam lasts 110 min. Please budget your time so you get to all questions.

Please wait to turn the page until everyone has an exam and you are told to begin.

Relax. You are here to learn.

Reference Information

This information may be useful during the exam. Feel free to use it or not as you wish. You can remove this page from the exam if that is convenient.

Reference for SQL Syntax

Outer Joins

```
-- left outer join with two selections:
select *
from R left outer join S on R.x=55 and R.y=S.z and S.u=99
```

The UNION Operation

```
select R.k from R union select S.k from S
```

The CASE Statement

```
select R.name, (case when R.rating=1 then 'like it'
                when R.rating=0 then 'do not like it'
                when R.rating is null then 'do not know'
                else 'unknown' end)
                as a_rating
from R;
```

The WITH Statement

Note: with is not supported in sqlite, but it is supported SQL Server and in postgres.

```
with T as (select * from R where R.K>10)
select * from T where T.K<20
```

Reference for Relational Algebra

Name	Symbol
Selection	σ
Projection	π
Join	⋈
Group By	γ
Set Difference	–
Duplicate Elimination	δ
Renaming	ρ
Sort	τ

Question 1. (18 points, 6 each) SQL. Summer is a good time to enjoy ice cream, but there are so many different ice cream shops and flavors that it's hard to keep track of what's available. Fortunately we can use database technology to help.

We have built a small database to keep track of ice cream shops, flavors, and ingredients. Our database has three tables:

Store(sid, name, location)
Sells(sid, flavor)
Contains(flavor, ingredient)

Where:

- *Store* contains a list of stores and locations. Each store has a unique integer *id*, which is the key in the relation. Attributes *name* and *location* are strings.
  Example: (17, "Molly Moon", "Wallingford")
- *Sells* lists all the flavors sold by a particular store. An entry has a store number and flavor name, which is a string. Example: (17, "Maple Walnut")
- *Contains* lists the main ingredients found in each flavor. There often are multiple ingredients in a particular flavor such as ("Cherry Garcia", "cherries"), ("Cherry Garcia", "chocolate chips"). Both attributes are strings.

Write SQL queries that return the information requested below.

(a) (6 points) Write a SQL query that returns the name(s) of all stores in "Wallingford" that sell "Mint Chip" ice cream.

Store(sid, name, location)
Sells(sid, flavor)
Contains(flavor, ingredient)

Question 1. (cont) (b) (6 points) Write a SQL query that finds the most common ingredient used in different ice cream flavors and lists all of the flavors where it is used. For example, if the most common ingredient is "chocolate", then the list might include flavors like "chocolate", "chocolate chip", "chocolate chip cookie dough", and so forth. To simplify the question, you can assume that there will be a single ingredient that appears more often than any of the others, i.e., you do not need to worry about breaking ties. Also, just match strings for equality. You do not need to check for substrings, like finding "vanilla" in "vanilla bean" – those are different ingredients.

(c) (6 points) Some people have food allergies. Write a query that lists the names of all ice cream shops in "Ballard" that sell only flavors that do not contain the ingredient "nuts". (Again, just use string equality in the query – don't search for substrings.)

The next few questions concern the following XML file named vehicles.xml.

```
<Vehicles>
  <Car factory="Hyundai">
    <Model>Azera</Model>
    <HorsePower>200</HorsePower>
  </Car>
  <Car factory="Toyota">
    <Model>Camry</Model>
    <HorsePower>210</HorsePower>
  </Car>
  <Truck factory="Toyota">
    <Model>Tundra</Model>
    <HorsePower>220</HorsePower>
  </Truck>
  <Car factory="Hyundai">
    <Model>Elantra</Model>
    <HorsePower>190</HorsePower>
  </Car>
  <Car factory="Toyota">
    <Model>Prius</Model>
    <HorsePower>230</HorsePower>
  </Car>
</Vehicles>
```

Question 2. (4 points, 1 each) Below are several element declarations that could appear in a DTD related to the vehicles.xml file above. For each one, circle "OK" if the vehicles.xml file could be valid if this element declaration appears in the DTD. Circle "invalid" if the vehicles.xml file cannot be valid if this declaration is used to define Vehicles in the DTD.

(a) OK invalid <!ELEMENT Vehicles (Car*, Truck+, Car*)>

(b) OK invalid <!ELEMENT Vehicles (Car+, Truck*, Car)>

(c) OK invalid <!ELEMENT Vehicles ((Car|Truck)*)>

(d) OK invalid <!ELEMENT Vehicles (Car*, Truck*)>

(more questions about this XML file on the next page)

Question 3. (6 points, 2 each) For each of the following XPath expressions, give the number of items produced by the expression when it is applied to the vehicles.xml file on the previous page. Your answer only needs to be a number.

(a) /Vehicles/Car[@factory="Toyota"]/HorsePower

(b) /Vehicles/*[Horsepower>200]/Model

(c) //*[@factory="Toyota"]/@factory

Question 4. (6 points) Write an XQuery expression that returns a list of all the Cars (not Trucks) in the xml document and for each Car gives the manufacturer name and model name. The result should be a well-formed XML document, and the query should work on any XML file named "vehicles.xml" that is formatted like the one on the previous page, not just for the sample data. An example of the output format for one car is: <Car><Manufacturer>Toyota</Manufacturer><Model>Camry</Model></Car>.

Question 5. (20 points) Database design. Summer softball season is starting soon and we need to use a computer to keep track of everything involved in the Database Softball League (DSL). After asking lots of people we've come up with the following details about what needs to be in the data.

- The league has several *teams*. Each team has a unique team name (for example, "Grizzlies") and a mascot (for example, "Fuzzy Bear").
- There are many *players* in the league. Each player plays for exactly one team. A team has several players. A player has an id number (an integer) and a name.
- Each team has at least one *manager*. A manager may be someone who does not play, or may be a player on the team. Each manager manages exactly one team. As is true of players, managers have a id number (integer) and a name.
- There are many *games*. Each game involves two teams, is scheduled for a particular date, and is played on a specific field.
- Each *field* has a name (e.g., "field 1") and a location (a string describing where it is, such as "Woodland Park").

(a) (10 points) Draw an E/R diagram that captures this information.

(continued next page)

Question 5. (cont.) (b) (10 points) Give a series of SQL CREATE TABLE statements to create tables to hold the information in the E/R diagram from part (a) of this question. Your SQL statements should identify the key or keys of each table and include any appropriate constraints, including foreign key constraints. For full credit you should avoid creating unnecessary tables – just define what is needed to properly store the data identified in the E/R diagram from part (a).

Question 6. (15 points) Database design. Our friend Mr. Frumble is a competitive fisherman and he keeps database with information about who has caught particularly large fish. He has one table in his database with the schema Records(name, fish, river, year, weight). Here is the data:

Name	Fish	River	Year	Weight
Monica	Bass	Columbia	2002	12
Mike	Carp	Green	2013	50
Phil	Catfish	Columbia	1997	20
Adam	Pike	Skagit	1997	34
Clarence	Trout	Snake	2006	13
Adam	Salmon	Snoqualmie	1997	10
Tanya	Bass	Columbia	1961	9
Clarence	Pike	Skagit	2006	15

(a) (5 points) Identify all of the functional dependencies in this database. As in homework 6, this is a reverse engineering problem – we want to identify all of the functional dependencies that occur in the actual data in this particular table.

(b) (10 points) Is this database schema in BCNF? If so, justify your answer and identify the key(s). If not, decompose the database into tables that are in BCNF and identify the key(s) in the resulting tables. You should show the individual decomposition steps so we can follow your work.

Question 7. (15 points, 5 each) Serializibility. For each of the following schedules, draw the precedence (conflict) graph and decide if the schedule is conflict-serializable. If the schedule is conflict-serializable, give an equivalent serial schedule by listing the transactions in an order they could occur (do not write out all of the read-write operations, just give the order of the transactions, e.g., T1, T2, T3, if there is an equivalent serial schedule). If the schedule is not conflict-serializable, explain why not.

(a) R1(X) R2(Y) R3(X) R2(X) R3(Z) W1(Y) R3(Y) R1(Z) W2(Z)

(b) R1(X) R2(Y) R1(Y) W2(Y) W1(X) R3(Y) R2(X) W2(X) W3(Y)

(c) R1(A) R3(A) W3(B) R1(B) R2(A) W2(D) R3(D) W3(B) W1(D)

Question 8. (12 points) Locking. To guarantee conflict serializibility, the scheduler in a database system inserts lock ($L_i(X)$) and unlock ($U_i(X)$) operations in a schedule. One algorithm that we discussed was two-phase locking (2PL), where a transaction cannot perform any new lock operations after it has performed any unlock operations, but it is free to unlock database elements at any time after it has finished using them. Another algorithm that we examined was strict two-phase locking (Strict 2PL). This algorithm requires that a transaction may not unlock anything until the transaction is terminating, i.e., it cannot do any further lock/unlock, read, or write operations after it unlocks anything.

(a) (6 points) Give a brief example of a problem that can occur if we use simple 2PL that is prevented by using Strict 2PL.

(b) (6 points) Different database systems use different lock granularities. Some, like SQLite lock the entire database to guarantee serialization, others lock individual tuples, others lock larger units like a disk page or index, but not the entire database. What is the tradeoff? For each of the choices below, give one key advantage compared to the other. A brief sentence or phrase is enough for each answer.

(i) Give an advantage of using fine-grained locks like individual tuples, compared to coarse-grained ones like entire tables or databases:

(ii) Give an advantage of using course-grained locks compared to fine-grained ones:

Question 9. (12 points) Suppose we have two relations R(a,b) and S(b,c). The sizes of the relations are as follows:

T(R) = 50,000	T(S) = 10,000
B(R) = 1,000	B(S) = 2,000

There are no indexes for either relation. The available main memory on the machine is M = 505.

Now suppose we want to do the natural join R⋈S. We want to use the fastest algorithm that we can, given the available machine resources. Our database engine includes implementations of hash join and block nested loop join.

Which algorithm should we use and what will be the cost of using that algorithm? Give a brief justification for your choice of algorithm (hash join or block nested loop join). Then give the cost of your choice using formulas involving things like B(…) and T(…) and finally substitute in the actual numbers into the formulas and simplify to get the final cost estimate.

The next questions involve queries using data from the following table. This is a SQL table storing information about a collection of Things.

Things(number, name, value)

Attribute *number* is a unique integer value identifying each Thing and is the key of the relation. Attribute *name* is a string giving the name of the Thing, like 'widget' or 'donut'. Attribute *value* is a number giving the value of the Thing, like 12.95 or 0.49. There may be many Things in the collection with different numbers but the same name. Things with the same name might have the same value, or they might have different values. You may assume that there are no NULL values in the data, in case that makes a difference.

Here is a query that lists all of the different Thing names in the table and the average value of the Things with each particular name, but ignoring all things whose value is less than 1.00.

```
SELECT name, AVG(value)
FROM Things
WHERE value >= 1.0
GROUP BY name
```

Question 10. (8 points) Draw a relational algebra tree that is equivalent to the above SQL query.

Question 11. (12 points) We have a lot of Things, so we would like to execute our query using a 3-node shared-nothing parallel SQL database. The data is *block partitioned* across the three nodes, i.e., each node holds 1/3rd of the data and the data is not stored on the nodes in any particular order.

Here's the query again: SELECT name, AVG(value) FROM Things WHERE value>=1.0 GROUP BY name

Draw a relational algebra tree giving a query plan to execute this query on this 3-node cluster. Note: this question is about *parallel SQL*, **not** map-reduce or Pig.

Simplification: you do not need to draw the same tree 3 times. Just draw the tree for the middle machine, but be sure to clearly show where and when data is exchanged between machines and what attributes or values are used to decide how to shuffle it.

Math reminder: if we have two collections of numbers X and Y, the average value of all the numbers is (SUM(X)+SUM(Y)) / (COUNT(X)+COUNT(Y)). The value (AVG(X)+AVG(Y)) / 2.0 is not the same.

Machine 1 1/3 of Things	Machine 2 1/3 of Things	Machine 3 1/3 of Things

Question 12. (12 points) Map-Reduce. In the last two problems we examined the SQL query

SELECT name, AVG(value) FROM Things WHERE value>=1.0 GROUP BY name

We now have so much data that we want to use a map-reduce job to do the same analysis. Instead of having a SQL relation Things(number, name, value), the input to the map-reduce job is a sequence of key-value pairs. Each key-value pair has the structure (number, (name, value)), i.e., the key of each input key-value pair is a unique Thing number, and the value in each input key-value pair is a (name, value) tuple giving the corresponding Thing name and its value.

Describe a sequence of one or more map-reduce jobs (**not** Pig programs) that will compute the average value of each group of Things with the same name, ignoring all Things that have a value < 1.0. The final output should be a collection of key-value pairs (name, average_value).

You need to clearly describe the (key, value) pairs that are input to and output from each map and reduce phase of the map-reduce job(s) needed. Explain (concisely) how the output of each map or reduce stage is computed from its input. If it takes more than one map-reduce job to compute the final result you should show how the output of each job is used as input to subsequent ones. You should not give Pig code for the algorithm, and you do not need to worry about the precise notation used to reference fields in the data key-value pairs or tuples as long as your intent is clear.

There is additional space on the next page for your answer, if needed.

Question 12. (cont.) Additional space for your answer if needed.

Question 13. (10 points, 2 each) A few short questions to wrap up.

(a) A serializable schedule is always conflict serializable. True or false?

(b) For a typical relation R in a transaction database (think of credit card transactions as an example), it is normally the case that T(R) is significantly larger than B(R). True or false?

(c) If we specify SET TRANSACTION ISOLATION LEVEL SERIALIZABLE in a SQL transaction, we are guaranteed that the transaction will have the ACID properties. True or false?

(d) The time needed to read a block of data from a disk is approximately the following (circle the closest answer):

200 ns 20,000 ns (= 20 µsec) 200,000 ns (= 200 µsec) 20,000,000 ns (= 20 msec)

200,000,000 ns (= 200 msec = 0.2 sec) 2,000,000,000 nsec (= 2,000 msec = 2 sec.)

(e) NoSQL databases sometimes promise "eventual consistency", which means that updates may take some time to change all copies of data that has been duplicated and stored in different locations. Even so, a NoSQL database guarantees that all reads executed after a write will read the most recent value written. True or false?

Have a great summer and best wishes for the future!!

The CSE 414 staff