CSE 344 Final Examination

December 11, 2013, 8:30am - 10:20am

Name: _____

Question	Points	Score
1	50	
2	20	
3	15	
4	20	
5	40	
6	55	
Total:	200	

- This exam is open book and open notes but NO laptops or other portable devices.
- You have 1h:50 minutes; budget time carefully.
- Please read all questions carefully before answering them.
- Some questions are easier, others harder; if a question sounds hard, skip it and return later.
- Good luck!

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 my_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 the Relational Algebra

Name	Symbol
Selection	σ
Projection	Π
Join	$\bowtie$
Group By	γ
Set difference	$-$

SQL and Relational Languages

1. (50 points)

   An online picture sharing company uses a database with the following schema:

```
create table Usr (
    uid int primary key,
    uname text not null,
    city text not null);

create table Picture (
     pid int primary key,
     uid int not null references Usr(uid),
     size int not null,
     pdf text);
```

   Every user has a key (`uid`), a name (`uname`) and a `city`.

   Every picture has a key (`pid`), an author (`uid`) that is a foreign key to `Usr`, a `size`, and the `pdf` content (which is plain text).

Solution: Run this in postgres:

```
insert into Usr values(1,'Alice','Denver');
insert into Usr values(2,'Bob','Denver');
insert into Usr values(3,'Carol','Portland');
insert into Usr values(4,'David','Denver');
insert into Usr values(5,'Evan', 'Seattle');


insert into Picture values(10, 1, 1500000, 'abcd');
insert into Picture values(20, 1, 1500000, 'bcde');
insert into Picture values(30, 2, 1500000, 'cdef');
insert into Picture values(40, 1, 1500000, 'defg');
insert into Picture values(50, 2, 1500000, 'efgh');
insert into Picture values(60, 5,  500000, 'fghk');
insert into Picture values(70, 5, 4000000, 'ghkm');
```

```
Usr(uid, uname, city)
Picture(pid, uid, size, pdf)
```

(a) (9 points) Write a SQL query that retrieves all users who have only pictures of less than 1MB (`size < 1000000`). Your query should return the users' `uid` and `uname`

Solution:

```
select x.uid, x.uname
from usr x
where not exists (select *
                  from picture y
                  where y.size >= 1000000 and x.uid = y.uid);
```

```
Usr(uid, uname, city)
Picture(pid, uid, size, pdf)
```

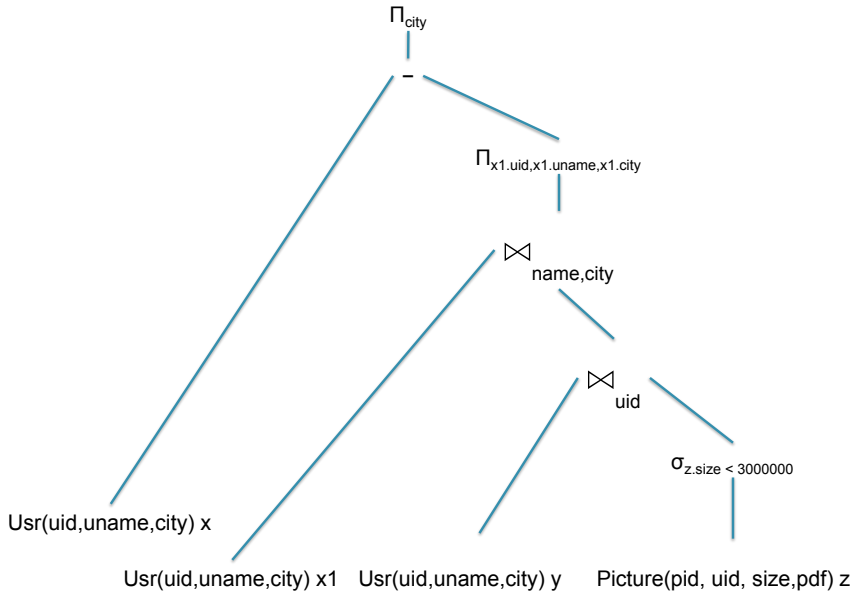
(b) (10 points) Write a Relational Algebra expression that is equivalent to the following SQL query:

```
select distinct x.city
from usr x
where not exists (select *
                  from usr y, picture z
                  where x.uname = y.uname
                    and x.city = y.city
                    and y.uid = z.uid
                    and z.size < 3000000);
```

Solution: It is useful to write the datalog program first:

```
T(uid, city) = Usr(uid, -, city), Picture(pid, uid, s, -), s < 3000000
Q(city)      = Usr(uid, -, city), not T(uid, city)
```

Note that we cannot compute the difference `Usr−T` because the type is incorrect. Hence, we first semi-join `T` with `Usr`, i.e. project back on the `Usr` attributes, then take the difference:



```
Usr(uid, uname, city)
Picture(pid, uid, size, pdf)
```

(c) Consider the following query:

```
select x.uid, x.uname,
         (select count(*)
          from Picture y
          where x.uid = y.uid and y.size > 1000000)
from Usr x
where x.city = 'Denver';
```

For each query below indicate if it is correct AND equivalent to the given query. You should answer 'yes' only of the query returns exactly the same answers.

i. (2 points) Is this query equivalent?

```
select x.uid, x.uname, count(*)
from Usr x, Picture y
where x.uid = y.uid  and y.size > 1000000 and x.city = 'Denver'
group by x.uid, x.uname;
```

i. **No**

Answer yes or no

ii. (2 points) Is this query equivalent?

```
select x.uid, x.uname, count(*)
from Usr x, Picture y
where x.uid = y.uid  and x.city = 'Denver'
group by x.uid, x.uname
having y.size > 1000000;
```

ii. **No**

Answer yes or no

iii. (2 points) Is this query equivalent?

```
select x.uid, x.uname, count(y.pid)
from Usr x left outer join Picture y on x.uid = y.uid and y.size > 1000000
group by x.uid, x.uname, x.city
having x.city = 'Denver';
```

iii. **Yes**

Answer yes or no

iv. (2 points) Is this query equivalent?

```
select x.uid, x.uname, count(*)
from Usr x left outer join Picture y on x.uid = y.uid and y.size > 1000000
group by x.uid, x.uname, x.city
having x.city = 'Denver';
```

iv. **No**

Answer yes or no

```
Usr(uid, uname, city)
Picture(pid, uid, size, pdf)
```

(d) Consider the following query:

```
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 > 1000000 and v.size < 3000000 and w.size = u.size;
```

For each query below indicate if it is correct AND equivalent to the given query:

i. (2 points) 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 > 1000000 and v.size < 3000000 and w.size > 1000000;
```

i. **Yes**

Answer yes or no

ii. (2 points) 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 > 1000000 and v.size < 3000000;
```

ii. **Yes**

Answer yes or no

iii. (2 points) 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 > 1000000 and u.size < 3000000 and u.size = w.size;
```

iii. **No**

Answer yes or no

iv. (2 points) 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 > 1000000 and v.size < 3000000 and w.size = v.size;
```

iv. **Yes**

Answer yes or no

Usr(uid, uname, city)	Artwork(wid, title, pdf)
Picture(pid, uid, size, pdf)	Author(aid,wid,aname)

(e) The company merges with a local adertising company in Denver, which has a database of artistic pictures. Each picture has a title, the pdf image, and may have several authors. All authors are from Denver.

```
create table Artwork (
   wid int primary key,
   title text,
   pdf text);

create table Author (
   aid int primary key,
   wid int not null references Artwork(wid),
   aname text not null);
```

i. (5 points) Some users and pictures occur in both databases. Write a SQL query that finds all common user,picture pairs. Two users are considered to be the same if their names and cities are the same; two pictures are considered to be the same if their pdf's are the same. Your query should return the user name (which is the same in both databases), its two keys in `Usr` and `Author`, and the two keys of the identical picture in `Picture` and `Artwork`.

Solution:

```
insert into Artwork values(101, 'Artwork by Alice and Fred', 'abcd');
insert into Artwork values(102, 'Artwork by Fred',           'xyzu');

insert into Author values(201, 101, 'Alice');
insert into Author values(202, 101, 'Fred');
insert into Author values(203, 102, 'Fred');

select distinct x.uid, v.aid, x.uname, y.pid, v.wid
from Usr x, Picture y,
     Artwork u, Author v
where x.uid = y.uid and u.wid = v.wid and x.city = 'Denver'
  and x.uname = v.aname and y.pdf = u.pdf;
```

Usr(uid, uname, city)	Artwork(wid, title, pdf)
Picture(pid, uid, size, pdf)	Author(aid,wid,aname)

ii. (10 points) The new company creates a new schema for the integrated data:

```
create table NewUsr (
    nuid int primary key,
    nuName text not null,
    city text not null);

create table NewPicture (
     npid int primary key,
     title text,
     size int,
     pdf text);

create table Authored (
     nuid int not null references newUsr(nuid),
     npid int not null references NewPicture(npid));
```

Write a sequence of SQL queries that insert all the data from the two old databases into the new database. You do not need to eliminate duplicates: that is, you will insert all the records from `Usr`, `Picture` into the new schema, then will insert all the records from `Artwork`, `Author` into the new schema, without worrying about duplicates. All keys in the `Usr`, `Picture` database are distinct from the keys in the `Artwork`, `Author` database.

Solution:

```
insert into NewUsr(nuid, nuName, city)
     select uid as nuid, uname as nuName, city from Usr;

insert into NewPicture(npid, title, size, pdf)
     select pid as npid, null as title, size, pdf from Picture;

insert into Authored(nuid, npid)
     select uid as nuid, pid as npid from Picture;

insert into NewUsr(nuid, nuName, city)
     select aid as nuid, aname as nuName, 'Denver' as city from Author;

insert into NewPicture(npid, title, size, pdf)
     select wid as npid, title, null as size, pdf from Artwork;

insert into Authored(nuid, npid)
     select aid as nuid, wid as npid from Author;
```

(This page is intentionally left blank.)

XML

2. (20 points)

Consider a relational database about publications, with the following schema:

```
Author(aid, name)       /* aid = key */
Authored(aid, pid)      /* aid = foreign key to Authored;
                           pid = foreign key to Paper */
Paper(pid, title, year) /* pid = key */
```

Solution:

```
create table author(aid int primary key, name text);
create table paper(pid int primary key, title text, year int);
create table authored(aid int references author, pid int references paper);

insert into author values(1, 'Alice');
insert into author values(2, 'Bob');
insert into author values(3, 'Carol');

insert into paper values(10, 'abc', 1995);
insert into paper values(20, 'def', 1995);
insert into paper values(30, 'ghk', 2010);

insert into authored values(1, 10);
insert into authored values(2, 10);
insert into authored values(2, 20);
insert into authored values(1, 30);
```

We export the entire data in XML, using the following DTD:

```
<!DOCTYPE bib [
        <!ELEMENT bib  (paper* )>
        <!ELEMENT paper   (pid, title,  year, author*)>
        <!ELEMENT author  (aid, name)>
        <!ELEMENT pid   (#PCDATA )>
        <!ELEMENT title (#PCDATA )>
        <!ELEMENT year  (#PCDATA )>
        <!ELEMENT aid   (#PCDATA )>
        <!ELEMENT name  (#PCDATA )>
]>
```

Solution:

```
<bib>
  <paper>
        <pid>10</pid>
        <title>abc</title>
        <year>1995</year>
        <author>  <aid>1</aid> <name>Alice</name> </author>
        <author>  <aid>2</aid> <name>Bob</name> </author>
  </paper>
  <paper>
        <pid>20</pid>
        <title>def</title>
        <year>1995</year>
        <author>  <aid>2</aid> <name>Bob</name> </author>
  </paper>
  <paper>
        <pid>30</pid>
        <title>ghk</title>
        <year>2010</year>
        <author>  <aid>1</aid>  <name>Alice</name> </author>
  </paper>
</bib>
```

(a) Answer the following questions:

i. (2 points) Are the `pid` elements unique in the XML document?

i. **yes**

Answer yes or no

ii. (2 points) Are the `aid` elements unique in the XML document?

ii. **no**

Answer yes or no

iii. (2 points) Could there be any `Papers` lost during export?

iii. **no**

Answer yes or no

iv. (2 points) Could there be any `Authors` lost during export?

iv. **yes**

Answer yes or no

```
<!DOCTYPE bib [
        <!ELEMENT bib  (paper* )>
        <!ELEMENT paper   (pid, title,  year, author*)>
        <!ELEMENT author  (aid, name)> ...]>
```

(b) (6 points) The following SQL query returns all papers written by Alice and Bob in 1995:

```
select x.title
from Paper x, Authored y, Author z,
              Authored u, Author v
where x.pid = y.pid and y.aid = z.aid
  and x.pid = u.pid and u.aid = z.aid
  and x.year = 1995
  and z.name = 'Alice'
  and v.name = 'Bob';
```

Write an equivalent query in XPath over the exported data.

Solution:

```
doc("file.xml")/bib/paper[year/text()=1995]
                         [author/name/text()='Alice']
                         [author/name/text()='Bob']/title
```

```
<!DOCTYPE bib [
        <!ELEMENT bib  (paper* )>
        <!ELEMENT paper   (pid, title,  year, author*)>
        <!ELEMENT author  (aid, name)> ...]>
```

(c) (6 points) The following SQL query returns all authors who published both in 1995 and in 2010:

```
select distinct w.name
from Paper x, Authored y, Paper u, Authored v, Author w
where x.pid = y.pid and y.aid = w.aid
  and u.pid = v.pid and v.aid = w.aid
  and x.year = 1995 and u.year = 2010;
```

Write an equivalent query in XQuery over the exported data.

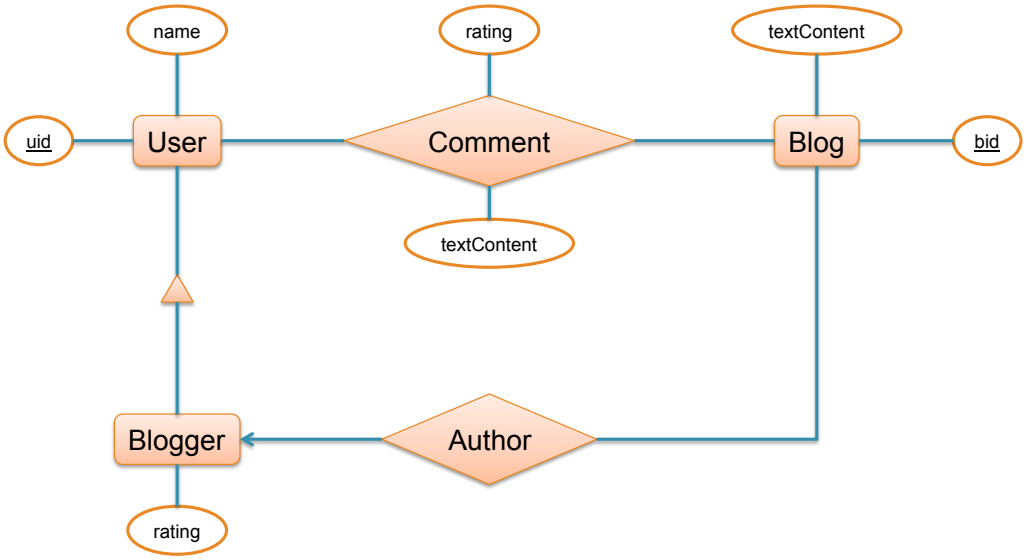
Solution:

```
let $a :=
   for $x in doc("file.xml")/bib/paper[year/text()=1995]/author,
        $u in doc("file.xml")/bib/paper[year/text()=2010]/author
   where $x/aid/text() = $u/aid/text()
   return $x/name/text()
for $x in distinct-values($a)
return <name> { $x } </name>
```

Database design

3. (15 points)

   (a) (5 points) Consider the following E/R diagram:



The schema describes a database of bloggers and blogs.

- Every blog is authored by exactly one blogger.
- Every blogger is a user.
- Users may comment on blogs.
- Each comment has an optional text content, and/or an optional rating.
- Each blogger also has a rating (which is not optional).

Write a sequence of SQL queries that create the tables for this E/R diagram; `uid`, `bid`, and `rating` are integers, all other attributes are `text`. You should turn in several CREATE TABLE statements, which include key, foreign key, and not null declarations.

(This page is intentionally left blank.)

Solution:

```
create table Usr(uid int primary key,
                 uname text not null);

create table Blgger(uid int primary key references Usr,
                    rating int not null);

create table Blog(bid int primary key,
                  textContent text,
                  blogger int references Usr not null);

create table Comment(uid int references Usr not null,
                     bid int references Blog not null,
                     rating int, textContent text);
```

(b) (5 points) Consider the following table:

A	B	C	D	E
0	0	0	0	0
1	1	1	1	1
2	0	2	2	2
3	1	0	3	3
4	0	1	0	4
5	0	2	1	5
6	1	0	2	0
7	0	1	3	1
8	1	2	0	2

Find all functional dependencies that hold on this table. You only need to write a minimal set of functional dependencies that logically imply all others.

Solution: We notice:

$$\begin{aligned} B &= (A \mod 2) \\ C &= (A \mod 3) \\ D &= (A \mod 4) \\ E &= (A \mod 6) \end{aligned}$$

Therefore: $A \to BCDE$, $D \to B$, $E \to BC$.

(c) (5 points) Using the functional dependencies you have identified at the previous question, decompose the relation in BCNF. Show your work, show the final result, and show the keys in the final result.

Solution:

1. $D+ = BD$. Split $ABCDE$ into BD and $ACDE$
2. $E+ = (B)C$. Split $ACDE$ into EC and ADE

Final answer: $R1(\underline{A}, D, E)$, $R2(\underline{E}, C)$, $R3(\underline{D}, B)$.

Views

4. (20 points)

(a) For each of the following statements indicate whether it is true or false:

i. (2 points) A *materialized* view is always up to date.

Answer Yes/No: i. **No**

ii. (2 points) A *virtual* view is always up to date.

Answer Yes/No: ii. **Yes**

iii. (2 points) By default, a view defined in SQL using the CREATE VIEW command is a *materialized view.*

Answer Yes/No: iii. **No**

iv. (2 points) An index is a *materialized* view.

Answer Yes/No: iv. **Yes**

(b) (12 points) Consider the following schema:

```
Sales(sid, product, month, category)   /* sid = primary key */
```

The relation records individual sales. Note that the same product may be sold in different months and/or under different categories.

You want to execute the following query, which computes the total sales of `Toy` products in all months other than `December`:

```
select product, count(*)
from Sales
where  category = 'Toy' and month != 'December'
group by product;
```

Unfortunately, you do not have access to the `Sales` table. Instead, the database administrator gave you access only to the following three views:

```
create view V1 as
  select product, count(*) as c1
  from Sales
  where category != 'Toy'  and month != 'December'
  group by product;

create view V2 as
  select month, count(*) as c2
  from Sales
  group by month;

create view V3 as
  select product, month, count(*) as c3
  from Sales
  group by product, month;
```

Rewrite the query computing total sales of `Toy` products in all months other than `December`, by referring only to the views `V1`, `V2`, `V3`, and without referring to the table `Sales`. If needed, create new views or use the `with` syntax (see the reference sheet at the beginning of this paper), but make sure you only refer to `V1,V2,V3` and not to `Sales`.

(This page is intensionally left blank)

Solution:

```
create table Sales(
    sid int primary key,
    product text,
    month text,
    category text);

insert into Sales values (1, 'aaa', 'June', 'Toy');
insert into Sales values (2, 'bbb', 'December', 'Toy');
insert into Sales values (3, 'ccc', 'June', 'Phone');
insert into Sales values (4, 'ddd', 'December', 'Phone');
insert into Sales values (5, 'ddd', 'December', 'Toy');
insert into Sales values (6, 'ddd', 'June', 'Toy');


create view  v4 as
  select product, sum(c3) as c4
  from v3
  where month != 'December'
  group by product;


select x.product, x.c4 - (case when y.c1 is null then 0 else y.c1 end)
from v4 x left outer join v1 y on x.product = y.product
where x.c4 - (case when y.c1 is null then 0 else y.c1 end) > 0;
```

Transactions

5. (40 points)

(a) Answer the following questions:

i. (3 points) Every serializable schedule is also conflict-serializable.

Answer Yes/No: i. **No**

ii. (3 points) Every conflict-serializable schedule is also serializable.

Answer Yes/No: ii. **Yes**

iii. (3 points) SQL Lite uses optimistic concurrency control.

Answer Yes/No: iii. **No**

iv. (3 points) Strict Two-Phase-Locking is guaranteed to produce a serializable schedule.

Answer Yes/No: iv. **Yes**

v. (3 points) Strict Two-Phase-Locking is guaranteed to produce a conflict-serializable schedule.

Answer Yes/No: v. **Yes**

vi. (3 points) Strict Two-Phase-Locking is guaranteed to avoid deadlocks.

Answer Yes/No: vi. **No**

(b) For each of the schedules below, indicate whether they are conflict-serializable. If you answer *yes*, then give the equivalent serial order of the transactions. Show your work.

i. (6 points) Is this schedule conflict-serializable? Show your work; if you answer 'yes', then indicate a serialization order.

```
R1(A), R1(B), W1(A), R2(B), W2(D), R3(C), R3(B), R3(D), W2(B), W1(C), W3(D)
```

ii. (6 points) Is this schedule conflict-serializable? Show your work; if you answer 'yes', then indicate a serialization order.

```
R1(A), R1(B), W1(A), R2(B), W2(A), R3(C), R3(B), R3(D), W2(B), W1(C), W3(D)
```

(c) A scheduler uses the strict two-phase locking protocol. In each of the cases below, indicate whether the scheduler may result in a deadlock. If you answer *yes*, then give an example of a shedule that results in deadlock.

i. (5 points) Can these transactions result in deadlock?

```
T1: W1(A), W1(C), CO1

T2: W2(B), W2(D), CO2

T3: W3(A), W3(B), CO3

T4: W4(D), W4(A), CO4
```

Answer 'yes' or 'no'. If you answer 'yes' then also indicate a schedule that results in deadlock:

Solution:
`W2(B),W3(A),W4(D),W2(D) -- now T2, T3, T4 are deadlocked.`
Transaction T1 is not needed.

ii. (5 points) Can these transactions result in deadlock?

```
T1: W1(A), W1(C), CO1

T2: W2(B), W2(D), CO2

T3: W3(A), W3(B), CO3

T4: W4(B), W4(C), CO4

T5: W5(C), W5(D), CO5
```

Answer 'yes' or 'no'. If you answer 'yes' then also indicate a schedule that results in deadlock:

Solution: No: all transactions acquire the elements in the order `A,B,C,D` and therefore they avoid deadlock.

Parallel Databases and MapReduce

6. (55 points)

   (a) A social network company stores the "likes" data in a very large file with a simple schema:

```
Likes(person1, person2)
```

   The following statistics are known about `Likes`:

Number of people in the social network	10^8
Number of records in `Likes`	10^{10}
Everybody likes somebody!	true
Size of the table `Likes`	1TB (10^{12})
HDFS chunk size	100MB (10^8)
Number of workers for the MapReduce jobs:	100

   We run a MapReduce program with the following functions:

```
function map(person1, person2)
   EmitIntermediary(person1, person2)

function reduce(person, list)
   Emit(person, length(list))
```

Number of people in the social network	10^8
Number of records in Likes	10^{10}
Everybody likes somebody!	true
Size of the table Likes	1TB (10^{12})
HDFS chunk size	100MB (10^8)
Number of workers for the MapReduce jobs:	100

```
function map(person1, person2)
   EmitIntermediary(person1, person2)

function reduce(person, list)
   Emit(person, length(list))
```

Answer the following questions approximatively. If the answer is totally dependent on the configuration then write 'System dependent'.

i. (5 points) How many map functions are there approximatively?

Write a number or say "system dependent": i. 10^{10}

ii. (5 points) How many map tasks are there approximatively?

Write a number or say "system dependent": ii. 10^4

iii. (5 points) How many reduce functions are there approximatively?

Write a number or say "system dependent": iii. 10^8

iv. (5 points) How many reduce tasks are there approximatively?

Write a number or say "system dependent": iv. **System dependent**

v. (5 points) How large is the size of the intermediate result approximatively?

Write a number or say "system dependent": v. **1TB (10^{12})**

vi. (5 points) How many records are in the final result approximatively?

Write a number or say "system dependent": vi. 10^8

(b) We modify the MapReduce program by adding a `combine` function:

```
function map(person1, person2)
   EmitIntermediary(person1, person2)

function combine(person, list)
   EmitIntermediary(person, length(list))

function reduce(person, list)
   Emit(person, sum(list))
```

For each of the following questions indicate whether they are true or false. The statements claim that certain parameters "decrease", without saying how much: you should respond "true" even if the decrease is tiny, but should respond "false" if there is no change.

i. (5 points) The addition of the `combine` function will cause the number of reduce functions to decrease.

i. **False**

True or false?

ii. (5 points) The addition of the `combine` function will cause the number of reduce tasks to decrease.

ii. **False**

True or false?

iii. (5 points) The addition of the `combine` function will cause the size of the intermediate result to decrease.

iii. **True**

True or false?

iv. (5 points) The addition of the `combine` function will cause the size of the final result to decrease.

iv. **False**

True or false?

v. (5 points) The `combine` function is executed *before* the data is reshuffled.

v. **True**

True or false?