Question 1. [9 marks]

Consider the following schema:

```
create table Department(
   dID integer primary key,
   name text not null,
   division integer);



create table Employee(
   eID integer primary key,
   name text not null,
   salary float,
   department integer references Department(dID));



-- Employee "manager" manages employee "junior".
create table Manages(
   manager integer references Employee(eID),
   junior integer references Employee(eID),
   primary key(manager, junior));



-- This employee sold this amount on this day.
create table Sales(
   eID integer references Employee(eID),
   day date,
   amount integer);
```

Part (a) [4 marks]

Imagine what would happen if someone were removed from the Employee table. Suppose that we want the following to occur automatically:

- Any tuple(s) that in the Manages table that record who manages this person should remain in the Manages table, but their `junior` column should become null.
- Any tuple(s) that in the Manages table that record who this person manages should be removed from the Manages table.

Revise the DDL above to implement this reaction policy.

Solution:

```
create table Department(
   dID integer primary key,
   name text not null,
   division integer);

create table Employee(
   eID integer primary key,
   name text not null,
   salary float,
   department integer references Department(dID));

-- Employee "manager" manages employee "junior".
create table Manages(
   manager integer references Employee(eID) on delete cascade,
   junior integer references Employee(eID) on delete set null,
   primary key(manager, junior));

-- This employee sold this amount on this day.
create table Sales(
   eID integer references Employee(eID),
   day date,
   amount integer);
```

Part (b) [3 marks]

Consider this query:

```
SELECT ________________________
FROM Employee natural join Sales
GROUP BY day, eID;
```

Which of the following can go in the blank space shown? Circle either "Can" or "Can't" for each.

`eID`	Can	Can't
`count(day)`	Can	Can't
`average(amount)`	Can	Can't
`name`	Can	Can't
`max(salary)`	Can	Can't
`department`	Can	Can't

Solution: The commented-out terms below are illegal. Note that three of the attributes that were not grouped by actually can go in the select clause without aggregation. Because this was not covered in class, we accepted either answer.

```
SELECT
eID,
count(eID),
count(day),
day,
min(amount),
-- amount,
count(name),
name,  -- does work, but we accepted either answer
max(salary),
salary,  -- does work, but we accepted either answer
count(department)
department  -- does work, but we accepted either answer
FROM Employee natural join Sales
GROUP BY day, eID;
```

Part (c) [1 mark]

Write a SQL statement to increase the salary of every employee in the department with `dID` 212 by 10,000.

Solution:

```
update employee
set salary = salary + 10000
where department = 212;
```

Part (d) [1 mark]

According to the schema, must every employee have a manager? (No mark without an explanation.)

Circle one: Yes [✓ No]

Explain your answer briefly:

Solution: No, nothing requires it. Every manager and every junior must reference eIDs that actually occur in the Employees table, but nothing says an eID in the employees table must occur in the manages table as a junior (or as a manager).

Question 2. [12 marks]

This question also uses the schema from question 1. It is summarized here in a very abbreviated fashion. Underscores indicate a primary key.

```
Department(_dID_, name, division)

Manages(_manager, junior_)
   manager references Employee(eID)
   junior references Employee(eID)

Employee(_eID_, name, salary, department)
   department references Department(dID)

Sales(eID, day, amount)
   eID references Employee(eID)
```

1. Write a query to produce output structured like this:

```
 managername | managersalary | juniorname | juniorsalary
-------------+---------------+------------+--------------
```

Include in it a row for every pair of employees A and B such that A manages B, A's salary is more than twice as high as B's salary, B's total sales are more than 100,000, and B doesn't manage anyone.

Solution:

```
create view bigsellers as
select eid, sum(amount)
from sales
group by eid
having sum(amount) > 100000;

select e1.name as managername, e1.salary as managersalary,
        e2.name as juniorname, e2.salary as juniorsalary
from employee e1, employee e2, manages m
where e1.eid = m.manager and e2.eid = m.junior and
e1.salary > e2.salary * 2 and
e2.eid in (select eid from bigsellers) and
e2.eid not in (select manager from manages);
```

2. Let's say that employee x is a "level 1 report" to employee y if y manages x, employee x is a "level 2 report" to employee y if y manages someone who manages x, and so on. Write a query to produce output structured like this:

```
 level | totalsales
-------+------------
```

It should have three rows: one for the total sales of all employees who are level 1 reports to "Marissa Mayer", one for the total sales of her level 2 reports, and one for the total sales of her level 3 reports. If she has no reports at a given level, `totalsales` for that level should be 0.

Solution: The last part (including a `totalsales` of 0 for levels where Marissa Mayer has no reports) is very difficult. Full marks were given for getting everything else correct.

```
create view l1 as
select junior as eid, 1 as level
from manages, employee
where manager = eid and name = 'Marissa Mayer';

create view l2 as
select m2.junior as eid, 2 as level
from manages m1, manages m2, employee
where m1.junior = m2.manager and
m1.manager = eid and name = 'Marissa Mayer';

create view l3 as
select m3.junior as eid, 3 as level
from manages m1, manages m2, manages m3, employee
where m1.junior = m2.manager and m2.junior = m3.manager and
m1.manager = eid and name = 'Marissa Mayer';

create view mmpeople as
(select * from l1) union (select * from l2) union (select * from l3);

create view mmsums as
select level, sum(amount)
from mmpeople natural left outer join sales
group by level;

-- There are no entries in mmpeople for the levels where she has no reports.
-- so mmsums also won't include levels where she has no reports.
-- No kind of outer join will fix that.
-- So we need to identify any missing levels and add in rows for them with
-- zero for the sum.  But we don't have the possible level values (1, 2, and 3)
-- recorded in any table, so we'll have to create that.  You can't insert values
-- into a view, but we can create a table.

create table possiblelevels(level integer);
insert into possiblelevels values (1), (2), (3);
```

```
-- Now we can bring in any missing levels.

(  select * from mmsums )
          union
( select level, 0 as sum
  from  possiblelevels
  where level not in (select level from mmsums)
);
```

Question 3. [6 marks]

The following Java program connects to a database conforming to the schema from question 1, and then reports the employee IDs of everyone who reports to Marissa Mayer, everyone who reports to Bill Gates, and everyone who reports to Sheryl Sandberg. Complete the program. Assume all appropriate imports have been done. IMPORTANT: There is an API at the back of the test. Marking of Java syntax will not be strict.

```
class Company {
    public static void main(String args[]) throws IOException {
        String[] names = {"Marissa Mayer", "Bill Gates", "Sheryl Sandberg"};
        Connection conn;




        try {
            Class.forName("org.postgresql.Driver");
            // Assume a Connection to the database has been made and conn stores it.





            for (String who : names) {
                // Print the eID of everyone directly managed by who.









            }
            conn.close();
        } catch (Exception e) { System.err.println("uh oh!" + e.getMessage()); }
    }
}
```

```
import java.io.IOException;
import java.sql.*;

class Company {

    public static void main(String args[]) throws IOException {
        String[] names = {"Marissa Mayer", "Bill Gates", "Sheryl Sandberg"};
        String url;
        Connection conn;
        ResultSet rs;
        String queryString;

        try {
            Class.forName("org.postgresql.Driver");
            url = "jdbc:postgresql://localhost:5432/csc343h-dianeh";
            conn = DriverManager.getConnection(url, "dianeh", "");

            queryString = "select junior from Employee, Manages "
                    + "where manager = eID and name = ?";
            PreparedStatement ps = conn.prepareStatement(queryString);

            for (String who : names) {
                // Print the eID of everyone directly managed by who.

                // Insert that string into the PreparedStatement and execute it.
                ps.setString(1, who);
                rs = ps.executeQuery();

                // Iterate through the result set and report on each tuple.
                while (rs.next()) {
                    int eid = rs.getInt("junior");
                    System.out.println(who + " manages " + eid);
                }
                rs.close();
            }
            conn.close();

        } catch (Exception e) {
            System.err.println("uh oh!" + e.getMessage());
        }
    }
}
```