CSE 344 Midterm

Monday, February 6, 2012, 9:30-10:20

Name: _____

This is a closed book exam. You have 50'. Please write your answers in the space provided.

Question	Points	Score
1	40	
2	40	
3	20	
Total:	100	

1 SQL

1. (40 points)

   Consider a photo sharing Website, where users can post pictures, as well as comment and rate other user's pictures. The schema is:

   ```
   Users(uid, name)
   Comment(uid, pid, score, txt)
   Picture(pid, author, img)
   ```

   The database has the following constraints:

   - `Comment.uid` is a foreign key to `Users`
   - `Comment.pid` is a foreign key to `Picture`
   - `Picture.author` is a foreign key to `Users`
   - `Comment.score` is an integer number between 1 and 10
   - All attributes are `NOT NULL`

   (a) (10 points) Write a SQL query that returns all users who have given a score of 8 or higher to 50 pictures or more. For each user, your query should return the user ID and the name.

   Solution:

   ```
   select x.uid, x.name
   from Users x, Comment y
   where x.uid = y.uid and y.score >= 8
   group by x.uid, x.name
   having count(*) >= 50
   ```

```
Users(uid, name)
Comment(uid, pid, score, txt)
Picture(pid, author, img)
```

(b) (10 points) A picture is considered *highly rated* if it received at least one score of 10, from a user other than its author. A *cautious* user is a user who commented only on highly rated pictures. (A user who did not comment at all is also cautious.) Write a SQL query that finds all cautious users. Your query should return a list of `uid, name` pairs.

Solution:

```
select x.uid, x.name
from users x
where x.uid not in
   (select y.uid  -- these are the non-cautious users
    from comment y  -- check that y.pid is not highly rated
    where y.pid not in
       (select u.pid  -- the pictures that are highly rated
        from comment u, picture v
        where u.pid = v.pid and u.score = 10 and u.pid != v.author))
```

A negation can be avoided by using an aggregate, e.g. `having`(max(`score`) < 10.

```
Users(uid, name)
Comment(uid, pid, score, txt)
Picture(pid, author, img)
```

(c) (10 points) A hacker found a way to break into the system and ran the following commands:

```
insert into Comment(uid, pid, score, txt)
  select x.uid, y.pid, 0 as score, 'worst picture i ever saw' as txt
  from Users x, Picture y
  where x.uid = y.author;

update Picture
set author = (select x.uid
              from Comment x
              where x.pid = Picture.pid
              order by x.score desc
              limit 1);
```

That is, he inserted some fake comments, and messed up all the picture authors. Luckily, his two queries were logged by the system (that' why we know them) and no other user activities were performed on the database during or after the breakin. As usual, the hacker was quickly caught, arrested, and sent to jail, but he refused to cooperate in repairing the database. Your job is to repair the database. Assume that the database was in a consistent state right before the hacker's attack, and write SQL commands (one or more) that will restore the database to its original state. Your answer should consists of a sequence of INSERT/UPDATE/DELETE statements, and should not rely on any log or backup of the database. For example, you might turn in something like this (which are correct SQL statement, but are not the real answer):

```
delete from Comment
where exists (select * from Users x where x.uid = uid and x.name='Hacker Joe')

update Picture
set author = (select x.uid from Comment x
              where pid = x.pid and x.score = 10 limit 1)
```

Solution:

```
update Picture
set author =
 (select x.uid
  from Comment x
  where x.score = 0 and x.pid = Picture.pid
  limit 1);

delete from Comment
```

```
where score = 0;
```

```
Users(uid, name)
Comment(uid, pid, score, txt)
Picture(pid, author, img)
```

(d) (10 points) For each statement below, indicate whether it is true of false.

i. An index may help a select-from-where SQL query run faster, or may not affect its running time, but it can never make a query run slower.

i. **true**

True or false?

ii. An index may help an update (insert, delete, or update) SQL query run faster, or may not affect its running time, but it can never make a query run slower.

ii. **false**

True or false?

iii. Consider a selection operation $\sigma_{\texttt{price}>90 \wedge \texttt{price}<100}(\texttt{Product})$. Using an unclustered index on `price` will make the query at least as fast as scanning the entire table `Product`.

iii. **false**

True or false?

iv. Consider a selection operation $\sigma_{\texttt{price}>90 \wedge \texttt{price}<100}(\texttt{Product})$. Using a clustered index on `price` will make the query at least as fast as scanning the entire table `Product`.

iv. **true**

True or false?

v. A large table `Product(pid, name, price)` is queried intensively and is never updated. Then we should create three clustered indexes, on `Product(pid)`, `Product(name)`, and `Product(price)`.

v. **false**

True or false?

2 Relational Algebra and Relational Calculus

2. (40 points)

```
Users(uid, name)
Comment(uid, pid, score, txt)
Picture(pid, author, img)
```

Consider the same database schema as in the first question:

```
Users(uid, name)
Comment(uid, pid, score, txt)
Picture(pid, author, img)
```

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

```
select distinct u.uid
from Users u, Picture x, Comment y
where u.uid = x.author and x.pid = y.pid and y.score > 8
group by u.uid, x.pid
having count(*) > 10
```

Solution: $\Pi_{u.uid}(\sigma_{cnt>10}(\gamma_{u.uid,count(*)\rightarrow cnt}(\texttt{Users u} \bowtie_{u.uid=x.author} (\texttt{Picture x} \bowtie \sigma_{score>8}(\texttt{Comment y})))))$

```
Users(uid, name)
Comment(uid, pid, score, txt)
Picture(pid, author, img)
```

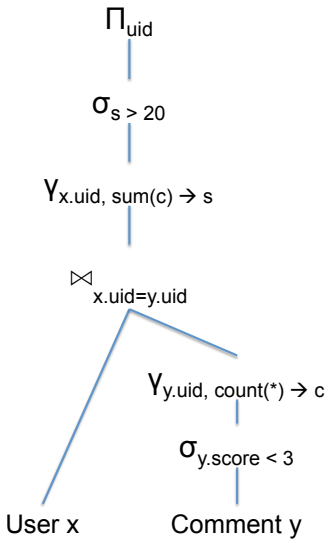
(b) (10 points) Write a Relational Algebra expression that is equivalent to the SQL query below:

```
select x.pid
from picture x
where not exists
   (select *
    from comment y
    where x.pid = y.pid and y.score <5)
```

Solution: $\Pi_{pid}(\texttt{Picture}) - \Pi_{pid}(\sigma_{score<5}(\texttt{Comment y}))$

```
Users(uid, name)
Comment(uid, pid, score, txt)
Picture(pid, author, img)
```

(c) (10 points) Consider the Relational Algebra expression below:



Write an equivalent SQL query *without* using any subqueries.

Solution:

```
select x.uid
from Users x, Comment y
where x.uid = y.uid and y.score < 3
group by x.uid
having count(*) > 20
```

```
Users(uid, name)
Comment(uid, pid, score, txt)
Picture(pid, author, img)
```

(d) (10 points) Consider the following query in the relational calculus:

$$Q(u,n) = \texttt{Users}(u,n) \wedge \forall p.\forall i.\,(\texttt{Picture}(p,u,i) \Rightarrow \forall v.\forall s.\forall t.\,(\texttt{Comment}(v,p,s,t) \Rightarrow s > 5))$$

Write an equivalent expression that does not use a universal quantifier.

Solution:

$$Q(u,n) = \texttt{Users}(u,n) \wedge \neg\exists p.\exists i.\,(\texttt{Picture}(p,u,i) \wedge \exists v.\exists s.\exists t.\,(\texttt{Comment}(v,p,s,t) \wedge s \leq 5))$$

3 XML and XPath

3. (20 points)

(a) (10 points) Consider the XML document below:

```
<a>
  <a>
    <a> 1 </a>
    <a> 2 </a>
  </a>
  <a>
    <a> 3 </a>
    <a> 4 </a>
  </a>
</a>
```

For each of the XPath queries below, indicate how many elements the expression returns:

i. `//a`

i. **7**

Answer

ii. `//a[a/text()=3]/a`

ii. **2**

Answer

iii. `//a[a/a/text()=3]/a/a`

iii. **4**

Answer

iv. `//a[a/text()=2][a/text()=3]`

iv. **0**

Answer

v. `//a[a/a/text()=2][a/a/text()=3]`

v. **1**

Answer

(b) (10 points) For each statement below indicate whether it is true of false.

i. XML databases are replacing relational databases in today's enterprise applications.

i. **False**

True or false?

ii. Compared to relational data, XML data is easier to exchange between applications or between users.

ii. **True**

True or false?

iii. The XPath expressions `/a/b[@c='5'][@d='2']` and `/a/b[@d='2'][@c='5']` are equivalent.

iii. **True**

True or false?

iv. The XPath expressions `/a/b[@c='5'][2]` and `/a/b[2][@c='5']` are equivalent.

iv. **False**

True or false?

v. The XPath expressions `/a/*//b` and `/a//*/b` are equivalent.

v. **True**

True or false?