COL 362/632: Minor – I

February 5, 2019

NAME:

DIVYANSHU SAXENA

ROLL NO:

2016CS10316

NOTE: Please write only in the space provided. We are using the tool Gradescope which automatically determines where your answers are written. Anything outside the boxes will be disregarded.

1 Part A

1. **(2 points)** What is normalization? Why is it required?

Normalization is the ~~reduct~~ reduction of a table into two or more tables (or relations), such that the decomposition is lossless. ~~and reduces~~

Normalization is done so as to reduce data redundancy in a relation.

2. **(1 point)** Why is SQL called a "declarative" language? Explain briefly.

SQL has a specific set of commands that are used to perform certain specific tasks. For example; "SELECT" command performs the select operation, "INSERT" command inserts tuples in a table, and so on.
Hence, the statements made in SQL are actually "declarations".

3. **(2 points)** Is bulk loading of tuples generally faster than inserting the tuples one by one? Explain why.

Yes. Bulk loading of tuples is faster than inserting them one by one.
This is because of the spatial locality of the database tuples and because of the insertion overheads.
These insertion overheads are one time costs for each insert operation. For bulk loading, these costs get divided over the entire set of tuples.

4. **(2 points)** Prove or disprove: Under bag semantics, $R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$.

In the statement, duplicates do not matter. Hence, it should be true.

5. **(1 point)** State one condition under which a hash-join algorithm *cannot* be used?

Join operation over same name attributes.

6. **(1 point)** State one difference between virtual views and materialized views.

Virtual views aren't stored physically on the disk while materialized views are stored physically on the disk.

7. **(1 point)** What does ACID (as in the properties guaranteed by a DBMS) stand for?

ACID properties stand for:-
(1) Atomicity
(2) Consistency
(3) Isolation
(4) Durability.

2 Part B

1. **(2 points)** We can convert weak entity sets into strong entity sets by simply adding the appropriate attributes. Then, why are weak entity sets needed?

Weak Entity Sets help us keep data redundancy in check. If we added the primary key of the strong entity set in the weak entity set then there shall be data redundancy.
For example, lets consider Institution-Department dependency. A "Department" is a weak entity set, because many institutions may have same department. If however, we convert "Department" into a strong entity set by adding the Institution ID (the primary key of "Institution") then there shall be many occurences of the tuples with "Institution Id" = 1. This is redundant.

2. **(2 points)** Using Armstrong's axioms, prove that if $A \rightarrow B$ and $BC \rightarrow D$ then $AC \rightarrow D$.

$A \rightarrow B$, $BC \rightarrow D$

$\{A, C\}^+$:

$\Rightarrow A \rightarrow B$ $\{A, B, C\}$

~~$\Rightarrow AC \rightarrow BC$ $\{A, B, C\}$ [From Armstrong's axiom, if $X \rightarrow Y$ then $AX \rightarrow AY$]~~

$\Rightarrow AC \rightarrow BC$ $\{A, B, C\}$ [if $X \rightarrow Y$ then $AX \rightarrow AY$]

$\Rightarrow BC \rightarrow D$ $\{A, B, C, D\}$. [BC is a subset of $\{A, B, C\}$]

Hence, $AC \rightarrow D$

3. Given the table: `bornIn (name, age, city)` which describes which city a person was born in and the person's current age, the following query was formulated. Answer the following questions.

```
select name, age
from bornIn as b
where age =
   (select max(age)
    from bornIn
    where b.city = city)
group by name,age having (age > 60)
```

(a) **(2 point)** State what the query returns.

The query returns the names and ages of the oldest person in each city, where the age of the oldest person is more than 60.

(b) **(3 points)** Rewrite the query *without* any subqueries. If it cannot be done, explain why not.

```
select name, ~~age~~ max(age)
from bornIn
where age > 60
group by ~~a~~ name, city;
```

(c) **(1 point)** Suppose the query were changed to the following:

```
select name, age
from bornIn as b
where age =
   (select age
    from bornIn
    where b.city = city)
group by name,age having (age > 60)
```

Would it still return the same result? Briefly explain.

No.
The above query shall not return the people who ~~have~~ are eldest (because it does not use the maximum age).

4. **(3 points)** Given a table of edges in a graph as follows: `edge (f int, t int)` where `f` is the source node and `t` is the destination node, consider the following query:

```
with recursive path (f, t) as (
  (select f, t from edge)
  union
  (select path.t, edge.f from path, edge
  where path.f = edge.t))
select * from path;
```

Show the output of the given query above on the following toy dataset (only the final output is required).

f	t
1	2
2	3
3	4
4	1

path :-

f	t
1	2
2	3
3	4
4	1
2	4
3	1
4	2
1	3

5. Two tables were created using the following definitions:

```
CREATE TABLE m (title varchar(30), actor varchar(30) NOT NULL,
CONSTRAINT mpkey PRIMARY KEY (title),
CONSTRAINT mfkey FOREIGN KEY (actor) REFERENCES a(name));

CREATE TABLE a (name varchar(30), movie varchar(30) NOT NULL,
CONSTRAINT apkey PRIMARY KEY (name),
CONSTRAINT afkey FOREIGN KEY (movie) REFERENCES m(title));
```

Suppose the following tuples need to be inserted into table "a":
(''Kristen Stewart'', ''Breaking Dawn'')
(''Tom Cruise'', ''MI-IV''),
and the following into table "m":
(''Breaking Dawn'', ''Kristen Stewart''),
(''MI-IV'', ''Tom Cruise'').

(a) **(1 points)** Briefly state the problem in inserting these tuples using just INSERT statements.

The foreign key constraints shall create a problem in inserting the above tuples.
When we insert ("Kristen Stewart", "Breaking Dawn") then the constraint ~~mfk~~ afkey requires that there should be some tuple in table m with title "Breaking Dawn", which it does not find. Hence, the insertion fails. Similar is the case for the next tuple. Similarly, the constraint mfkey creates problems for other

(b) **(4 points)** Give a sequence of SQL commands required to insert these tuples that run on PostgreSQL. Explain each command in one sentence (note that you are not allowed alter or drop the tables, but you can alter the constraints).

```
CREATE TRANSACTION BEGIN
INSERT VALUES ("Kristen Stewart", "Breaking Dawn")
INTO a;
INSERT VALUES ("Tom Cruise", "MI-IV") INTO a;
INSERT VALUES ("Breaking Dawn", "Kristen Stewart")
INTO m;
INSERT VALUES ("MI-IV", "Tom Cruise") INTO m;
COMMIT;
```

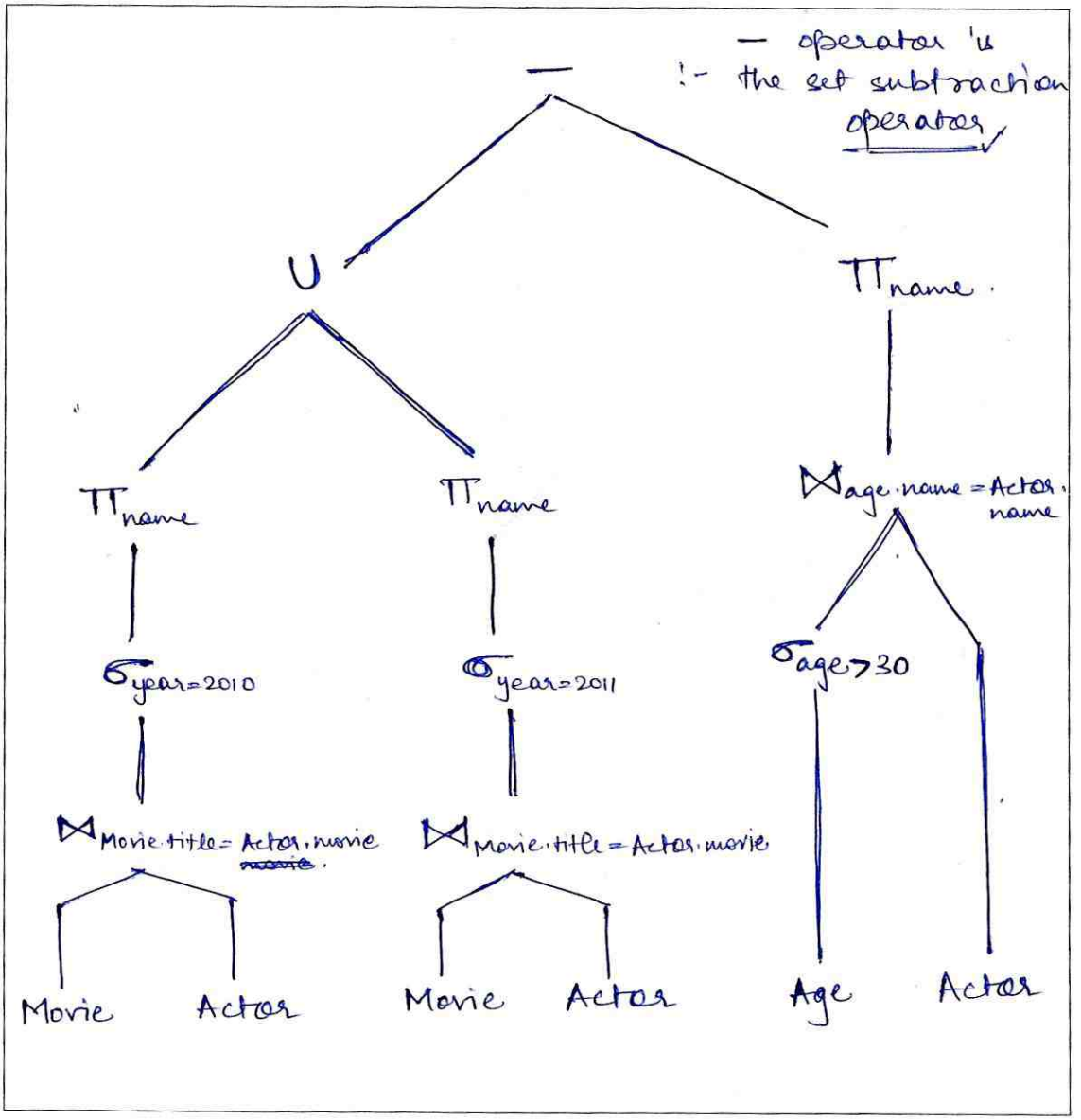
6. Given the following relations: Movie (<u>title</u>, year, boxoffice), Actor (<u>name, movie</u>), Age (<u>name</u>, age) which respectively describe movies by their title, year of production and total boxoffice returns, actors by their name and movies they have acted in, and names of people and their ages, answer the following questions.

```
(select name from Movie, Actor
 where Movie.title = Actor.movie
 and   Movie.year = 2010
union
 select name from Movie, Actor
 where Movie.title = Actor.movie
 and Movie.year = 2011)
except
 select Actor.name from Age, Actor
 where age > 30 and Actor.name = Age.name
```

(a) **(2 points)** What does the query return?

The query returns the name of all actors who have acted in a movie released in 2010 or in 2011 and are not aged more than 30.

(b) **(5 points)** Write the logical query plan for the query above. Follow the structure of the query as closely as possible.



7. **(5 points)** The `EXPLAIN` command was used to obtain the query plan of a query. The output was as follows:

```
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=14.51..14.53 rows=1 width=66)
   Group Key: object
   ->  Sort  (cost=14.51..14.52 rows=1 width=116)
         Sort Key: object
         ->  Seq Scan on yagofacts  (cost=0.00..14.50 rows=1 width=116)
               Filter: (((subject)::text <> (object)::text) AND ((predicate)::text = 'linksTo'::text))
(6 rows)
```

Write the SQL query that likely resulted in this query plan.

```
EXPLAIN ANALYZE SELECT *
FROM yagofacts
WHERE subject <> object
AND predicate = 'linksTo'
GROUP BY object
```

USE FOR ROUGH WORK