CSE 414 Midterm Exam
Autumn 2019
October 30, 2019

- Please read all instructions (including these) carefully.
- Write your name and UW student number below.
- **This is a <u>closed-book exam</u>. You are allowed one page of note sheets that you can write on both sides.**
- No electronic devices are allowed, including **cell phones** used merely as watches. Silence your cell phones and place them in your bag.
- Solutions will be graded on correctness and ***clarity***. Each problem has a relatively simple and straightforward solution. Partial solutions will be graded for partial credit.
- Please write your answers in the boxed space provided on the exam, and clearly mark your solutions. You may use the blank sides as scratch paper. **Do not use any additional scratch paper.**
- *Relax. You are here to learn. Good luck!*

Relational algebra operators:

Union ∪ Difference — Selection σ Projection π Join ⋈
Rename ρ Duplicate elimination δ Grouping and aggregation ɣ Sorting τ

Database constraints:

```
Primary key, Foreign key, Unique, Not null
```

By writing your name below, you certify that you have not received any unpermitted aid for this exam, and that you will not disclose the contents of the exam to anyone in the class who has not taken it.

NAME: _____SOLUTIONS_____

Text in red indicates clarifications post-exam

STUDENT NUMBER: _____

Problem	Points	Problem	Points
1	10	4	9
2	32	5	12
3	12	**Total**	75

Problem 1: True/False (10 points total)

Select either True or False for each of the following questions.

a) SQL and Datalog are declarative query languages.

True False

b) The value of a Primary Key for a relation uniquely identifies a tuple within that relation.

True False

c) In the relational model, the field of a tuple can itself be a relation.

True **False**

d) When using the aggregate `COUNT(attr)`, tuples with NULL values in `attr` field will be added to the total count.

True **False**

e) In the relational model, the schema defines both the name and type of each attribute.

True False

f) In a single table's schema, a Primary Key cannot also be a Foreign Key.

True **False**

g) The SQL we have learned this quarter cannot express recursive queries.

True False

h) Expressing recursive queries in Datalog requires negation.

True **False**

i) The following Datalog rule is safe:

```
Q(a, b) :-  R(a, _), ! S(b, a)
```

True **False**

j) The following Datalog rule is safe:

```
Q(z) :-  R(a, b), S(b, c), T(c, z), ! R(a, z)
```

True False

Problem 2: SQL (32 points total)

We will work with the following schema for a database of movies in this exam.

ACTOR (pid, fname, lname, gender)
MOVIE (mid, name, year, revenue)
DIRECTORS (did, fname, lname)
CASTS (pid, mid, role)
MOVIE_DIRECTORS (did, mid)

A tuple in the `Casts` relation represents that a person from the `Actor` table with `pid`, starred (was cast) in a `Movie` with `mid`. `Movie_Directors` represents a person from `Directors` with `did`, who directed a `Movie` with `mid`.

ACTOR.pid**, MOVIE.**mid**, DIRECTOR.**did = primary keys of the corresponding tables
CASTS.pid = foreign key to **ACTOR**.pid
MOVIE_DIRECTORS.did = foreign key to **DIRECTORS**.did
CASTS.mid, **MOVIE_DIRECTORS**.mid= foreign keys to **MOVIE**.mid

You can assume none of the tables contain `NULL` values. You may use subqueries for these questions. When writing your queries you can assume that the system is case-insensitive.

a) (8 points) Write a query that returns all the roles played by actors named “Kevin Bacon” (first name `Kevin`, last name `Bacon`.) Your query should return the `role` field. Do not include duplicates roles.

```
SELECT DISTINCT c.role
FROM Actor a, Casts c
WHERE a.pid = c.pid AND
      a.fname = ‘Kevin’ AND
      a.lname = ‘Bacon’;
```

Schema repeated here for your reference:

ACTOR (pid, fname, lname, gender)
MOVIE (mid, name, year, revenue)
DIRECTORS (did, fname, lname)
CASTS (pid, mid, role)
MOVIE_DIRECTORS (did, mid)

b) (8 points) Write a query that returns the first name and last name of all actors who starred in both the movies `'In the Mood for Love'` and `'Chungking Express'.` Only return actors who were in both movies, not just one of the movies. Duplicate names are allowed.

```
SELECT a.fname, a.lname
FROM Actor a, Casts c1, Casts c2, Movie m1, Movie m2
WHERE a.pid = c1.pid AND c1.mid = m1.mid AND
      a.pid = c2.pid AND c2.mid = m2.mid AND
      m1.name = 'In the Mood for Love' AND
      m2.name = 'Chungking Express' ;
```

c) (8 points) Find the total number of movies made in each year, and return the results sorted by the number of movies, in descending order. Return the year numbers and count of movies made in that year.

```
SELECT m.year, COUNT(*)
FROM Movie m
GROUP BY m.year
ORDER BY COUNT(*) DESC ;
```

Schema repeated here for your reference:

ACTOR (pid, fname, lname, gender)
MOVIE (mid, name, year, revenue)
DIRECTORS (did, fname, lname)
CASTS (pid, mid, role)
MOVIE_DIRECTORS (did, mid)

d) (8 points) Write a query that returns all directors who only directed movies alone (i.e. who never directed a movie together with another director.) For any directors with no movies at all, include them in the output. Return the did field for each director.

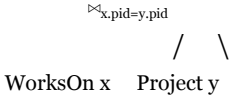
```
Solution #1:
 SELECT d1.did
 FROM Directors d1
 WHERE NOT EXISTS (SELECT *
   FROM Movie_Directors md1, Directors d2, Movie_Directors md2
   WHERE d1.did = md1.did AND d2.did = md2.did
     AND md1.mid = md2.mid AND d1.did != d2.did);

Solution #2:
 SELECT d.did
 FROM Director d
 WHERE d.did NOT IN (
    SELECT DISTINCT d1.did
    FROM Director d1, Director d2, MOVIE_DIRECTORS md1, MOVIE_DIRECTORS md2
    WHERE d1.did = md1.did AND d2.did = md2.did
      AND d1.did != d2.did AND md1.mid = md2.mid);

Solution #3:
 SELECT DISTINCT d.did
 FROM Directors d LEFT OUTER JOIN Movie_Directors md ON d.did = md.did
 WHERE md.mid IN (SELECT md2.mid FROM Movie_Directors md2
   GROUP BY md2.mid HAVING COUNT(*) = 1);

Solution #4:
 WITH Multi_Directed_Movies AS (
   SELECT mid FROM Movie_Directors
   GROUP BY mid HAVING COUNT(*) > 1)
 SELECT did FROM Directors
 EXCEPT
 SELECT did FROM Movie_Directors md, Multi_Directed_Movies mdm
 WHERE md.mid = mdm.mid

Solution #5:
 withOther(d1) :- Directors(d1,_,_), Movie_Directors(d1,m),
                  Movie_Directors(d2,m), d1 != d2.
 answer(d) :- Directors(d,_,_), !withOther(d).
```

Problem 3: Relational Algebra (12 points total)

We will use the same schema as problem 2, repeated here for your reference:

ACTOR (pid, fname, lname, gender)
MOVIE (mid, name, year, revenue)
DIRECTORS (did, fname, lname)
CASTS (pid, mid, role)
MOVIE_DIRECTORS (did, mid)

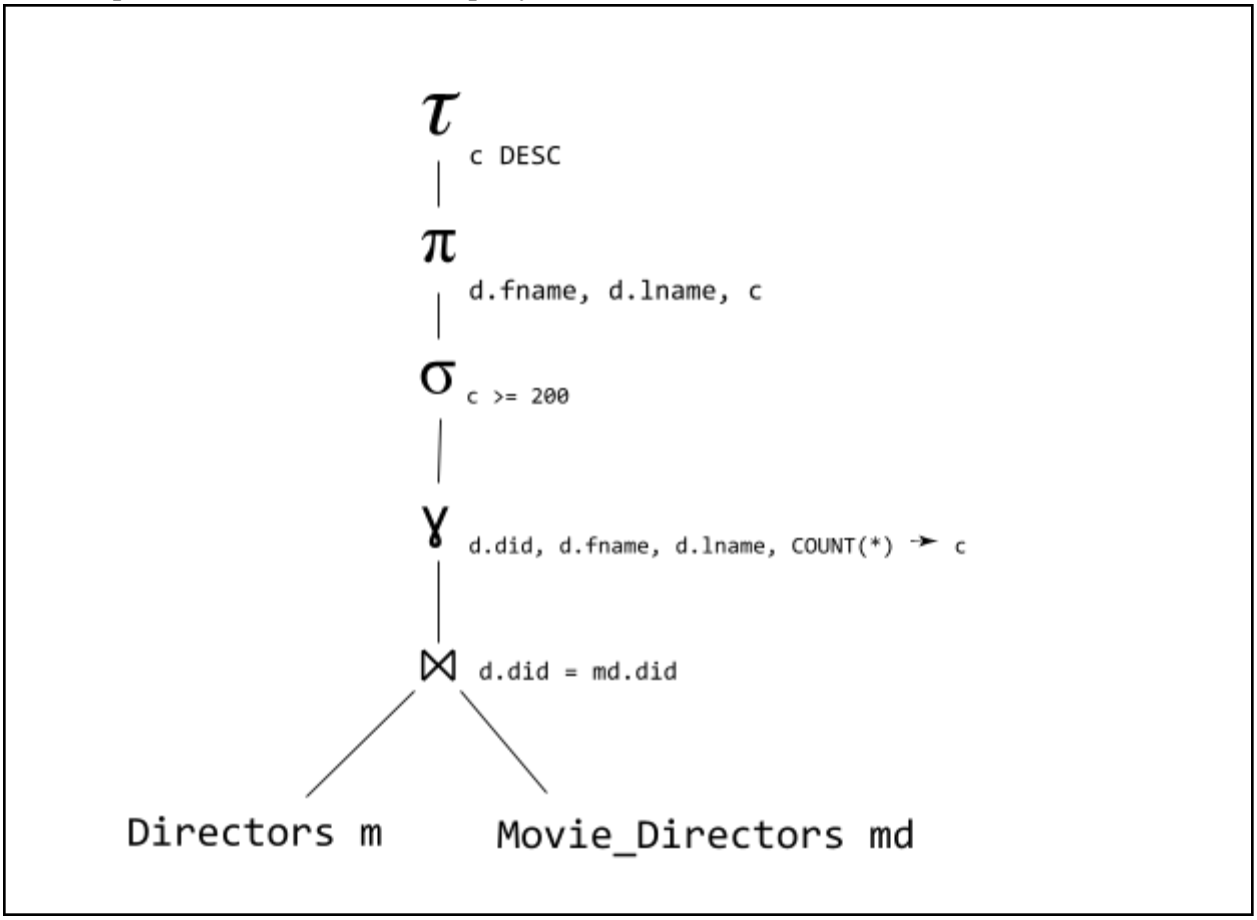
Note: you may use aliases in the query plan to avoid renaming, e.g.



a) (12 points) The following query lists all directors who directed at least 200 movies, in descending order of the number of movies they directed

```
SELECT d.fname, d.lname, COUNT(*) AS c
FROM Directors d, Movie_Directors md
WHERE d.did = md.did
GROUP BY d.did, d.fname, d.lname
HAVING COUNT(*) >= 200
ORDER BY c DESC;
```

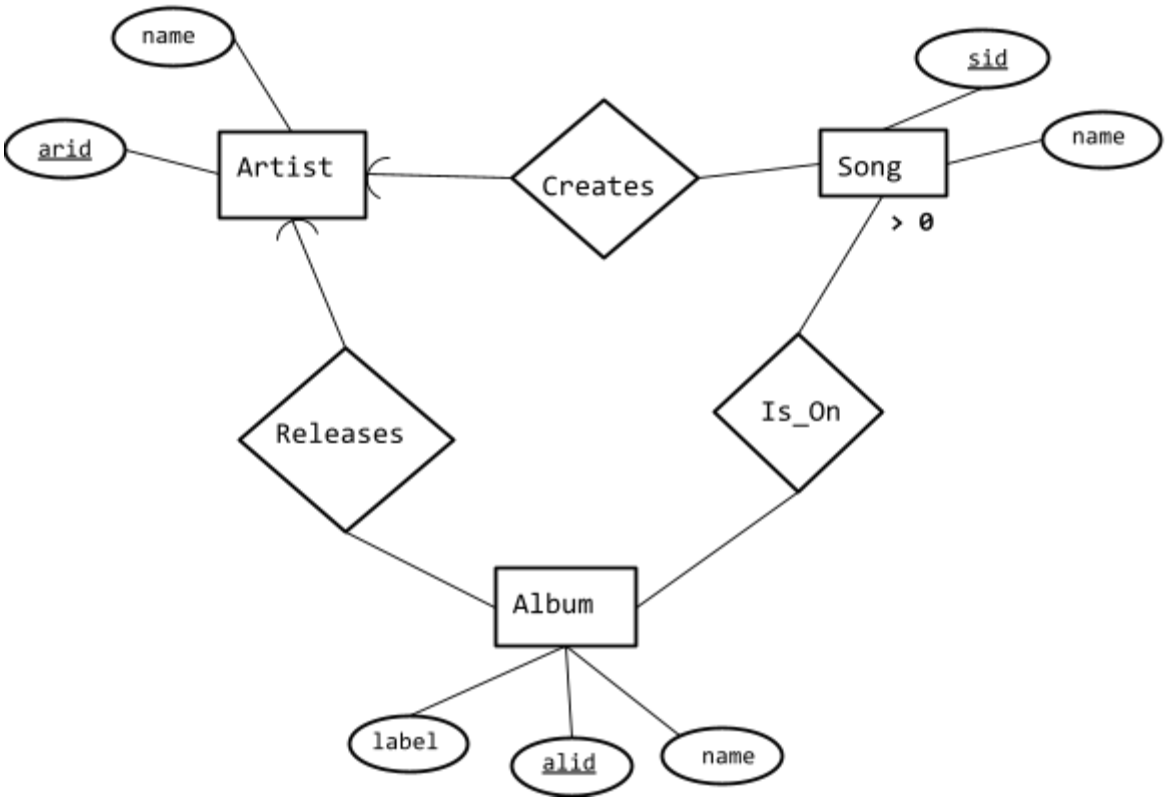
Write a Relational Algebra expression in the form of a logical query plan (you may draw a tree) that is equivalent to the above SQL query.



Problem 4: Entity Relationship Diagrams (9 points total)

For this problem consider a database of musical artists, albums, and songs. Artists can be individuals like Adele, or bands like The Beatles. We have three entities and relationships between them:

- A **Song** is created by a single **Artist**
- **Songs** are published on albums
- An **Album** is released by a single **Artist**



- **Is_On** is a many-to-many relationship. Albums can have multiple songs, and songs can be on multiple albums (The Essential Beatles, The Beatles' Greatest Hits, etc.) An Album must have at least one song.
- **Creates** is a many-to-exactly-one relationship from Song to Artist.
- Similarly, **Releases** is a many-to-exactly-one relationship from Album to Artist.

(a) Assume we are trying to **minimize the number of tables** we use to represent the above entities.

Fill in the create table statements for the **Song** and **Album** relations, making sure to preserve the constraints in the diagram above and trying to minimize the number of tables we will need in total. The underlined attributes above are the primary keys of the respective entity sets. The types of `arid`, `sid`, and `alid` are INT, and all other attributes are VARCHAR(100).

(8 points)

```
CREATE TABLE Song (

    sid INT PRIMARY KEY,
    name VARCHAR(100),
    arid INT NOT NULL,
    FOREIGN KEY arid REFERENCES Artist

);

CREATE TABLE Album (

    alid INT PRIMARY KEY,
    name VARCHAR(100),
    label VARCHAR(100),
    arid INT NOT NULL,
    FOREIGN KEY arid REFERENCES Artist



);
```

(b) Do we need a table for the "Is_On" relationship? Write a sentence explaining why/why not. (1 point)

```
Yes, we need a table for Is_On to store all the tuples in a many-to-many
relationship.
```

Problem 5: Functional Dependencies (12 points total)

Assume we have a relation R and its attributes in parentheses, and we know that it satisfies the following functional dependencies:

R (A, B, C, D, E)

A -> BC
D -> E
B -> E
CD -> A

For the functional dependencies below, you will decide if we can derive them from the information above. Circle YES if we can guarantee that R satisfies the functional dependency given the knowledge we have. If not, circle NO. (2 points each)

a) A -> A **YES** NO

b) A -> AD YES **NO**

c) AD -> A **YES** NO

d) CD -> ABCD **YES** NO

e) DE -> BE YES **NO**

f) C -> ABCDE YES **NO**