Problem 2

Consider a relational schema R = {A, B, C, D, E, G, H}, satisfying the functional dependencies F = {E → G, E→ H, G → H, A → BC, BC → D, C → H, EG → A}.

1. Derive all candidate keys for this schema.

* E→AGH
* EG->ABCD
* C->H

{E} is the candidate key as it is the only key that is not dependent on another key.

1. Derive a canonical cover of the functional dependencies in F.

* Use the union rule to replace any dependencies in F
* Find a functional dependency a -> b with an extraneous attribute either in a or in b
* Because of {E->G}, G is extraneous in {EG->A} and can be simplified into {E->A}
* Because of {E->G} and {E->H}, {G->H} is redundant and can be removed
  + {E->G}, {E->A}, and {E->H} can be combined to form {E->AGH} under union rule.
* {C->H} remains untouched

**Canonical Cover: {E->AGH, C->H, A->BC, BC->D}**

1. Is the above schema in BCNF? Prove or disprove. If it is not in BCNF, convert it into BCNF.

{C->H} is a non-trivial dependency but C is not a superkey because it depends on A.

**No, this schema is not in BCNF form.**

To convert it into BCNF form, we would have to split the schema. For each functional dependency found in the canonical cover, we take off the right side from the table. The left side must remain in both tables for identification:

R1 = (A, B, C, D, E, G, H)

* {E->AGH}

R1 = (AEGH) and R2 = (BCDE)

* {BC->D}

R1 = (AEGH), R2 = (BCD), R3 = (BCE)

* {C->H} is lost on conversion.

1. Is the BCNF schema from (c) dependency-preserving? Prove or disprove. If not, convert into 3NF.

No, as mentioned above, {C->H} is lost due to the {E->AGH} functional dependency. To convert to 3NF, we must add another table: R4 = (CH) to preserve {C->H}

R1 = (AEGH), R2 = (BCD), R3 = (BCE), R4 = (CH)

**Problem 3**

Consider the following single-table database modeling the Actor-Movie database application  
discussed in class:

ActorMovie (aid, aname, mid, mtitle, role, hours, payph)

Actors are identified by a unique aid, and movies by a unique mid. An actor can play several roles in the same movie, but these roles must have different names (i.e., different values for the role attribute). Different actors playing the same role in the same movie must be paid the same per hour.

1. Explain why the above is not a good relational design. Name several reasons.

* Would have to make several tuples to identify an actor working for the same movie under multiple roles.
* If trying to find an actor’s basic info, would end up having to bring up an actor and the movie even if the movie details is not wanted.
* It would be repetitive to repeatedly state the actor’s name and the movie title when the actor id and movie title already imply what actor is working for what movie.
* If all actors in the same role are paid the same, then it is not necessary for role and payph to both be on the same table as the role implies the pay
* The table is not in 3NF nor BCNF form.

1. Identify the set F of non-trivial functional dependencies for this schema. (It is enough to identify any subset E such that the closures of E and F are the same.)

* {Aid->aname}
* {Mid->mtitle}
* {Aid, Mid, Role->payph, hours}
  + Aid, mid are included also included alongside role because the question states that two different actors must work for the same movie and have the same role.
  + Under this reasoning, the actor id, movie id are included to determine if two actors need to have the same payph and work the same hours.
    - Two different actors who have the same role but are in different movies will not have the same pay or work hours.
  + While not directly stated, it is being assumed that role also determines the number of hours being worked.

1. Derive all candidate keys for this table.

Candidate Keys: {aid, mid, role}

Aid, role, and Mid would be the candidate keys as these are independent entities that are not determined by any other keys in the table.

* Was stated in class that role will also serve as a candidate key on 4/25/2022.

1. Derive a canonical cover of the functional dependencies in F.

* {Aid->aname} and {Mid->mtitle} are atomic and cannot be simplified any further.
* There are no redundant relations involving {aid, mid, role->payph, hours}

{aid->aname}, {mid->mtitle}, {aid, mid, role-> payph, hours}

1. Is the above schema in BCNF? Prove or disprove. If it is not in BCNF, convert it into BCNF.

No, {aid->aname} and {mid->mtitle} use candidate keys aid and mid respectively but those two elements alone are not superkeys which violates BCNF.

R1=(aid, aname, mid, mtitle, role, hours, payph)

* {Aid->aname} decomposes R1

R1 = (aid, aname), R2 = (aid, mid, mtitle, role, hours, payph)

* {mid->mtitle} decomposes R2

R2 =(mid, mtitle), R3 = (aid, mid, role, hours, payph)

* {aid, mid, role -> hours, payph} which is preserved in R3

The set would be R1 = (aid, aname), R2 = (mid, mtitle), and R3 = (aid, mid, role, hours, payph)

1. Is the BCNF schema from e) dependency-preserving? Prove or disprove. If not, convert it into 3NF.

The BCNF schema is dependency preserving because mtitle and aname are preserved after the split, and aid, mid->role is preserved along with aid, mid->role, hours, payph are saved in the third table.

Mtitle and Aname were not dependent on {aid, mid->role, hours, payph} so no information is lost after splitting.

1. Suppose we also require that actors with same name must get the same pay per hour when acting in the same movie (even if they do not have the same role). How would this change your answers for parts b) through f)?

It creates an additional functional dependency {mid, aname -> payph} since actor name now determines the pay for the movie. The existing functional dependency {aid, mid, role-> payph, hours} changes into {aid, mid, role->hours} as role is no longer sufficient enough to determine pay.

* The candidate keys would not change as aid, mid, and role are still needed to determine the hours worked, and aname does not become a candidate key after the change.
  + {aid, mid, role}
* The set of functional dependencies after the additional condition:

{Aid->aname}  
{Mid->mtitle}  
{aid, mid, role->hours}  
{mid, aname -> payph}

Canonical cover:

* {aid->aname} and {mid->mtitle} remain the same due to being atomic.
* {aid, mid, role->hours} has no redundant relations.
* {mid, aname->payph} does not have a redundant relation

**Canonical Cover**: {aid->aname}, {mid->mtitle}, {aid, mid, role->hours}, {mid, aname->payph}

The set would not be in BCNF form for the same reasons as before: {aid->aname} is non-trivial but aid alone is not a superkey.

R1 = (aid, aname, mid, mtitle, role, hours, payph)

* {aid->aname}

R1 = (aid, aname), R2 = (aid, mid, mtitle, role, hours, payph)

* {mid->mtitle}

R2 = (mid, mtitle), R3 = (aid, mid, role, hours, payph)

* {aid, mid, role->hours} decomposes R3

R3 = (aid, mid, role, hours), R4 = (aid, mid, payph)

* {mid, aname->payph} is lost on conversion

List of tables: R1 = (aid, aname), R2 = (mid, mtitle), R3 = (aid, mid, role, hours), R4 = (aid, mid, payph)

**The set is not dependency-preserving** because {mid, aname->payph} is lost on conversion to BCNF. To convert to 3NF, must add an additional table: R5= (mid, aname, payph)

3NF: R1 = (aid, aname), R2 = (mid, mtitle), R3 = (aid, mid, role, hours), R4 = (aid, mid, payph), (mid, aname, payph)

Problem 4

In this problem, you have to explore the metadata querying facilities of your DBMS, in order to write metadata queries. For testing, you should use the same bakery schema and data as in Problem 1.

1. List all attributes in the schema and their type.

* The bakery schema used for this problem was named “test”

SELECT TABLE\_NAME, COLUMN\_NAME, DATA\_TYPE  
FROM information\_schema.columns natural join information\_schema.tables  
WHERE table\_schema = 'test';

Graphical user interface, application

Description automatically generated

1. List all tables that contain an attribute whose name contains the substring ``ake’’ (for example, ``cakename’’).

SELECT TABLE\_NAME, COLUMN\_NAME  
FROM INFORMATION\_SCHEMA.TABLES natural join INFORMATION\_SCHEMA.COLUMNS  
WHERE COLUMN\_NAME like '%ake%'  
AND TABLE\_SCHEMA = 'test'

Graphical user interface, text, application

Description automatically generated

1. List all tables sorted in decreasing order by the number of tuples they contain. For each table, output its name and the number of tuples in it.

SELECT table\_name, count(column\_name) as row   
FROM information\_schema.columns natural join information\_schema.tables  
WHERE TABLE\_SCHEMA = 'test'  
GROUP BY TABLE\_NAME  
order by row desc;

Graphical user interface, text, application

Description automatically generated

1. For each attribute in the Contain table, output its most common value.

Example: If a column had a gender with 7 male and 4 female, then male would be the most common attribute

with count\_cake as (select count(cakeid) as cakeid\_count  
 from contain  
 group by cakeid  
 order by cakeid\_count desc  
 limit 1),  
 count\_ingred as (select count(ingredid) as ingredid\_count  
 from contain  
 group by ingredid  
 order by ingredid\_count desc  
 limit 1),  
 count\_qty as (select count(qty) as qty\_count  
 from contain  
 group by qty  
 order by qty\_count desc  
 limit 1)  
 select cakeid\_count, ingredid\_count, qty\_count  
 FROM count\_cake, count\_ingred, count\_qty;

Graphical user interface, application

Description automatically generated

1. List all foreign keys referencing cakeid in Cake.

SELECT i.TABLE\_NAME, COLUMN\_NAME, i.CONSTRAINT\_TYPE, k.REFERENCED\_TABLE\_NAME,k.REFERENCED\_COLUMN\_NAME   
FROM information\_schema.TABLE\_CONSTRAINTS i , information\_schema.KEY\_COLUMN\_USAGE k  
where i.CONSTRAINT\_NAME = k.CONSTRAINT\_NAME  
and i.CONSTRAINT\_TYPE = 'FOREIGN KEY'  
and k.REFERENCED\_COLUMN\_NAME = 'cakeid'  
and k.REFERENCED\_TABLE\_NAME = 'cake'Graphical user interface, text, application, Word

Description automatically generated

To ensure that the statement checked all tables, an additional table was added referencing cake(cakeid) with the MySQL query repeated below.

Graphical user interface, text, application

Description automatically generated