

Problem 2

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

a) Derive all candidate keys for this schema.

- $E \rightarrow AGH$
- $EG \rightarrow ABCD$
- $C \rightarrow H$

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

b) Derive a canonical cover of the functional dependencies in F .

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

Canonical Cover: $\{E \rightarrow AGH, C \rightarrow H, A \rightarrow BC, BC \rightarrow D\}$

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

$\{C \rightarrow H\}$ 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:

$R_1 = (A, B, C, D, E, G, H)$

- $\{E \rightarrow AGH\}$

$R_1 = (AEGH)$ and $R_2 = (BCDE)$

- $\{BC \rightarrow D\}$

$R_1 = (AEGH)$, $R_2 = (BCD)$, $R_3 = (BCE)$

- $\{C \rightarrow H\}$ is lost on conversion.

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

No, as mentioned above, $\{C \rightarrow H\}$ is lost due to the $\{E \rightarrow AGH\}$ functional dependency. To convert to 3NF, we must add another table: $R_4 = (CH)$ to preserve $\{C \rightarrow H\}$

$R_1 = (AEGH)$, $R_2 = (BCD)$, $R_3 = (BCE)$, $R_4 = (CH)$

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.

- a) 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.

- b) 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.

- c) 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.

d) Derive a canonical cover of the functional dependencies in F.

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

$\{aid \rightarrow aname\}, \{mid \rightarrow mtitle\}, \{aid, mid, role \rightarrow payph, hours\}$

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

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

$R_1 = (aid, aname, mid, mtitle, role, hours, payph)$

- $\{Aid \rightarrow aname\}$ decomposes R_1

$R_1 = (aid, aname), R_2 = (aid, mid, mtitle, role, hours, payph)$

- $\{mid \rightarrow mtitle\}$ decomposes R_2

$R_2 = (mid, mtitle), R_3 = (aid, mid, role, hours, payph)$

- $\{aid, mid, role \rightarrow hours, payph\}$ which is preserved in R_3

The set would be $R_1 = (aid, aname), R_2 = (mid, mtitle),$ and $R_3 = (aid, mid, role, hours, payph)$

f) 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 \rightarrow role is preserved along with aid, mid \rightarrow role, hours, payph are saved in the third table.

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

g) 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 $\{mid, aname \rightarrow payph\}$ since actor name now determines the pay for the movie. The existing functional dependency $\{aid, mid, role \rightarrow payph, hours\}$ changes into $\{aid, mid, role \rightarrow hours\}$ 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:

$\{Aid \rightarrow aname\}$

$\{Mid \rightarrow mtitle\}$

$\{aid, mid, role \rightarrow hours\}$

$\{mid, aname \rightarrow payph\}$

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.

$R_1 = (\text{aid, aname, mid, mtitle, role, hours, payph})$

- {aid->aname}

$R_1 = (\text{aid, aname}), R_2 = (\text{aid, mid, mtitle, role, hours, payph})$

- {mid->mtitle}

$R_2 = (\text{mid, mtitle}), R_3 = (\text{aid, mid, role, hours, payph})$

- {aid, mid, role->hours} decomposes R_3

$R_3 = (\text{aid, mid, role, hours}), R_4 = (\text{aid, mid, payph})$

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

List of tables: $R_1 = (\text{aid, aname}), R_2 = (\text{mid, mtitle}), R_3 = (\text{aid, mid, role, hours}), R_4 = (\text{aid, mid, payph})$

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: $R_5 = (\text{mid, aname, payph})$

3NF: $R_1 = (\text{aid, aname}), R_2 = (\text{mid, mtitle}), R_3 = (\text{aid, mid, role, hours}), R_4 = (\text{aid, mid, payph}), R_5 = (\text{mid, aname, payph})$



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.

- a) 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';
```

```
3 • SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE
4 FROM information_schema.columns natural join information_schema.tables
5 WHERE table_schema = 'test';
```

Result Grid			
Filter Rows: <input type="text"/>			
Export:  Wrap Cell Content: 			
	TABLE_NAME	COLUMN_NAME	DATA_TYPE
▶	cake	cakeid	int
	cake	cakename	varchar
	cake	slices	int
	cake	status	varchar
	cake	price	int
	contain	cakeid	int
	contain	ingredid	int
	contain	qty	float
	customer	custid	int
	customer	custname	varchar
	customer	ccn	varchar
	customer	cphoneno	varchar
	customer	address	varchar
	customer	city	varchar
	customer	zip	varchar
	ingredient	ingredid	int
	ingredient	iname	varchar
	ingredient	price	int
	ingredient	available	tinyint
	orders	custid	int
	orders	cakeid	int
	orders	ordertime	datetime
	orders	pickuptime	datetime
	orders	pricepaid	int

- b) 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'
```

```
18
19 • SELECT TABLE_NAME, COLUMN_NAME
20 FROM INFORMATION_SCHEMA.TABLES natural join INFORMATION_SCHEMA.COLUMNS
21 WHERE COLUMN_NAME like '%ake%'
22 AND TABLE_SCHEMA = 'test'
23
```

TABLE_NAME	COLUMN_NAME
cake	cakeid
cake	cakename
contain	cakeid
orders	cakeid

- c) 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;
```

```
7 ✖ SELECT table_name, count(column_name) as row
8 FROM information_schema.columns natural join information_schema.tables
9 WHERE TABLE_SCHEMA = 'test'
10 GROUP BY TABLE_NAME
11 order by row desc;
12
```

table_name	row
customer	7
cake	5
orders	5
ingredient	4
contain	3

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

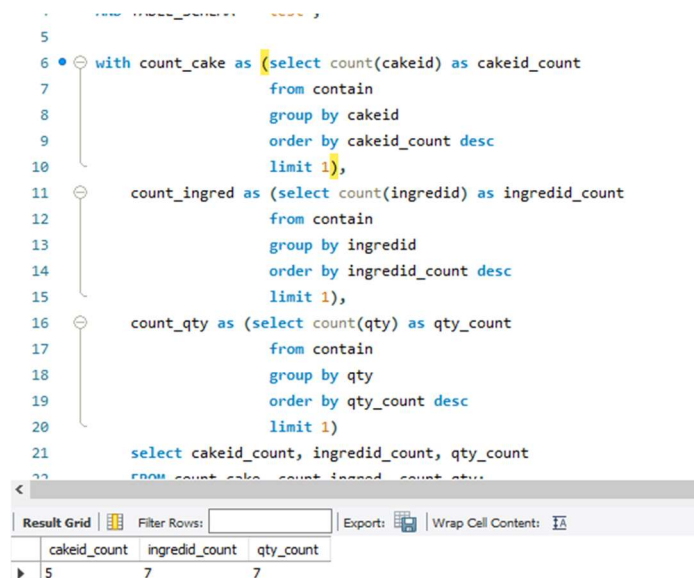
Example: If a column had a gender with 7 male and 4 female, then male would be the most common attribute

```
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;
```



e) 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'

```
18
19 • SELECT i.TABLE_NAME, COLUMN_NAME, i.CONSTRAINT_TYPE, k.REFERENCED_TABLE_NAME, k.REFERENCED_COLUMN_NAME
20 FROM information_schema.TABLE_CONSTRAINTS i , information_schema.KEY_COLUMN_USAGE k
21 where i.CONSTRAINT_NAME = k.CONSTRAINT_NAME
22 and i.CONSTRAINT_TYPE = 'FOREIGN KEY'
23 and k.REFERENCED_COLUMN_NAME = 'cakeid'
24 and k.REFERENCED_TABLE_NAME = 'cake'
25 ;
26
27
```

TABLE_NAME	COLUMN_NAME	CONSTRAINT_TYPE	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
contain	cakeid	FOREIGN KEY	cake	cakeid

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

```
17
18 • SELECT i.TABLE_NAME, COLUMN_NAME, i.CONSTRAINT_TYPE, k.REFERENCED_TABLE_NAME,k.REFERENCED_COLUMN_NAME
19 FROM information_schema.TABLE_CONSTRAINTS i , information_schema.KEY_COLUMN_USAGE k
20 where i.CONSTRAINT_NAME = k.CONSTRAINT_NAME
21 and i.CONSTRAINT_TYPE = 'FOREIGN KEY'
22 and k.REFERENCED_COLUMN_NAME = 'cakeid'
23 and k.REFERENCED_TABLE_NAME = 'cake'
24
```

TABLE_NAME	COLUMN_NAME	CONSTRAINT_TYPE	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
contain	cakeid	FOREIGN KEY	cake	cakeid
tesatrase	cakeid	FOREIGN KEY	cake	cakeid