NYU Tandon School of Engineering **Nov 25, 2018**
Computer Science and Engineering
CS6083, Fall 2018

<u>Problem Set #3 Sample Solution</u>

Problem 2:
(a) Candidate key: {A} or {C} or {D} or {H}

(b) Canonical Cover: Fc = {A → CE, C → BD, H → AI, D → H}.
or Fc ={A → C, C → BD, H → AEI, D → H}.

(c) This is BCNF because the left part of every functional dependency in Fc is superkey. And there is no trivial left.

(d) The result from c) part is dependency preserving. Every function relation can be checked in this BCNF schema.

Problem 3:
(a) This is not a good design for following reasons:
1) Current schema does not reflect a lot of functional dependencies. For instance, as assumptions in the question, {aid → aname} holds, but since the {aid} is not a candidate key, we will end up repeating this information in several tuples.
2) Inserting values for the separate entities in one table like Actor or movie will lead to storing NULL and duplicate values at several places. This will increase the storage and our data will be inconsistent.
3) We have to access this one table for querying even minimal data, this will be time consuming as everything is under this table.
4) Lastly, this schema does not adhere to the normalization rules. As everything is under one table, it will make maintenance and querying from the database difficult and it will be hard to see relation between each attribute. it's better to divide this big table to many smaller ones.

(b)
{aid} → {aname}
{mid} → {mtitle}
{aid, mid, rolename} → {hours}
{mid, rolename} → {payph}

(c) Candidate Keys: {aid, mid, rolename}

(d) same as (b)

{aid} → {aname}
{mid} → {mtitle}
{aid, mid, rolename} → {hours}
{mid, rolename} → {payph}

(e) No, it is not in BCNF. Because for schema to be BCNF for each non-trivial dependency A->B, A should be a superkey, which is not the case here. Convert into BCNF form:

Step 1: relation ActorMovie can be decomposed into ACTOR and R2 according to the violation by dependency {aid} → {aname}, we can get relation:
ACTOR = {aid, aname}
R2 = {aid, mid, mtitle, rolename, hours, payph}

Step 2: According to the violation by dependency {mid} → {mtitle}, we can get relation:
MOVIE = {mid, mtitle}
R3 = {aid, mid, rolename, hours, payph}

Step 3: According to the violation by dependency {mid, rolename} → {payph}, we can get relation:
PAY = {mid, rolename, payph}
R4 = {aid, mid, rolename, hours}

Thus, the BCNF form is:
ACTOR (aid, aname)
MOVIE (mid, mtitle)
PAY (mid, rolename, payph)
ACT (aid, mid, rolename, hours)

(f) Yes, it's dependency-preserving. Because each functional dependency in the canonical cover can be checked in BCNF.

(g)

Functional dependency:
{aid} → {aname}
{mid} → {mtitle}

{aid, mid, rolename} → {hours}
{mid, rolename} → {payph}
{aname, mid}→{payph}

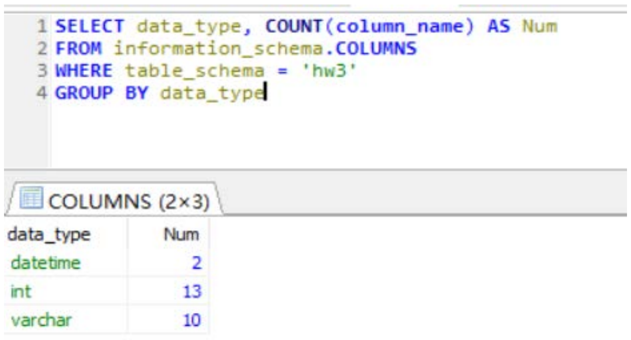
Candidate Keys: {aid, mid, rolename}

Canonical Cover:
{aid} → {aname}
{mid} → {mtitle}
{aid, mid, rolename} → {hours}
{mid, rolename} → {payph}
{aname, mid}→{payph}

No, it’s not in BCNF form, convert it into BCNF:
ACTOR (aid, aname)
MOVIE (mid, mtitle)
PAY (mid, rolename, payph)
ACT (aid, mid, rolename, hours)

No, the schema in section(e) is not dependency preserving because the functional dependency {aname, mid}→{payph} cannot be checked in the BCNF form. Thus, the 3NF form is:
ACTOR (aid, aname)
MOVIE (mid, mtitle)
ROLEPAY(mid, rolename, payph)
ACT (aid, mid, rolename, hours)
ACTORNAMEPAY (aname, mid, payph)

Problem 4:
(a)
```
SELECT data_type, COUNT(column_name) AS Num
FROM information_schema.COLUMNS
WHERE table_schema = 'hw3'
GROUP BY data_type
```



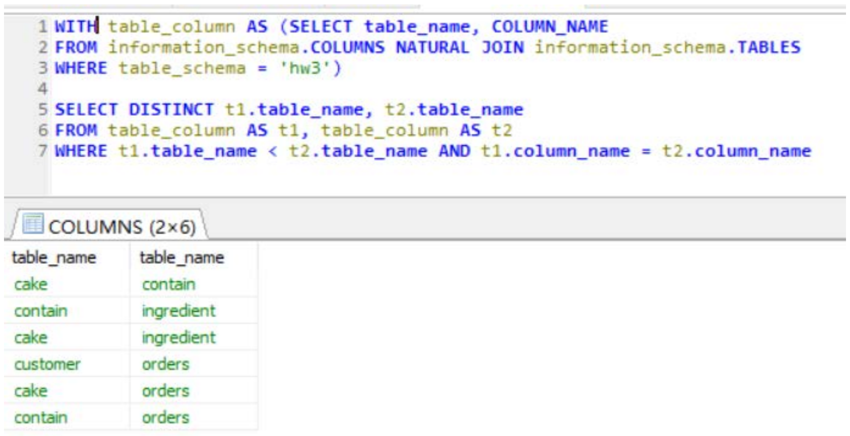
```
SELECT data_type, COUNT(column_name) AS Num
FROM information_schema.COLUMNS
WHERE table_schema = 'hw3'
GROUP BY data_type
```

COLUMNS (2×3)

data_type	Num
datetime	2
int	13
varchar	10

(b)
WITH table_column AS (SELECT table_name, COLUMN_NAME
FROM information_schema.COLUMNS NATURAL JOIN information_schema.TABLES
WHERE table_schema = 'hw3')

SELECT DISTINCT t1.table_name, t2.table_name
FROM table_column AS t1, table_column AS t2
WHERE t1.table_name < t2.table_name AND t1.column_name = t2.column_name



```
WITH table_column AS (SELECT table_name, COLUMN_NAME
FROM information_schema.COLUMNS NATURAL JOIN information_schema.TABLES
WHERE table_schema = 'hw3')

SELECT DISTINCT t1.table_name, t2.table_name
FROM table_column AS t1, table_column AS t2
WHERE t1.table_name < t2.table_name AND t1.column_name = t2.column_name
```

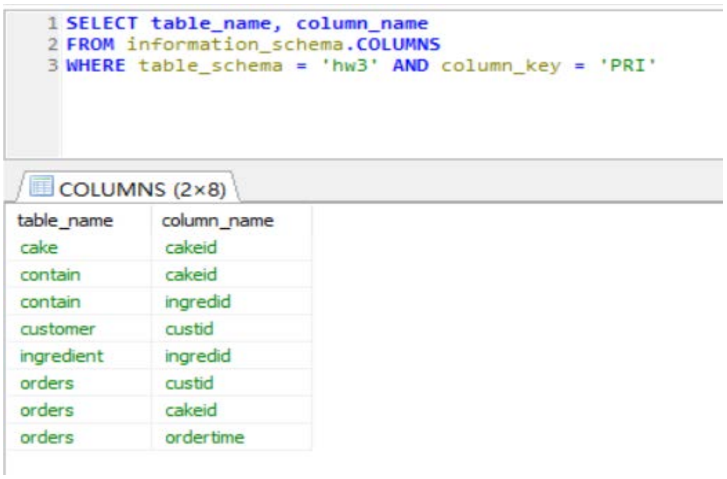
COLUMNS (2×6)

table_name	table_name
cake	contain
contain	ingredient
cake	ingredient
customer	orders
cake	orders
contain	orders

(c)
SELECT COUNT(*)
FROM information_schema.views
WHERE table_schema = 'hw3'

(d)
SELECT custID
FROM hw3.Customer, information_schema.columns I
WHERE table_schema = 'hw3' AND Customer.lastname = I.table_name

(e)
SELECT table_name, column_name
FROM information_schema.COLUMNS
WHERE table_schema = 'hw3' AND column_key = 'PRI'

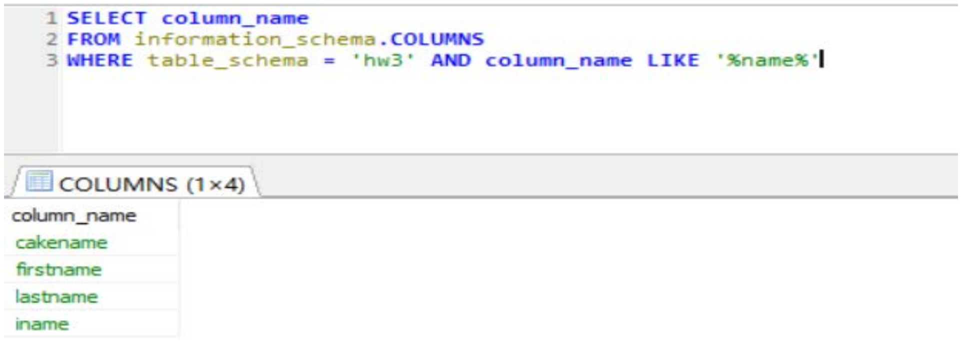


```
SELECT table_name, column_name
FROM information_schema.COLUMNS
WHERE table_schema = 'hw3' AND column_key = 'PRI'
```

COLUMNS (2×8)

table_name	column_name
cake	cakeid
contain	cakeid
contain	ingredid
customer	custid
ingredient	ingredid
orders	custid
orders	cakeid
orders	ordertime

(f)
SELECT column_name
FROM information_schema.COLUMNS
WHERE table_schema = 'hw3' AND column_name LIKE '%name%'



```
SELECT column_name
FROM information_schema.COLUMNS
WHERE table_schema = 'hw3' AND column_name LIKE '%name%'
```

COLUMNS (1×4)

column_name
cakename
firstname
lastname
iname