Homework #2

CS 34800

Nicholas Donahue

1.

I.

worker	wname	street	City
	P._x		

works	wname	cname	salary
	_x	G._y	_z

company	cname	city
	_y	

Conditions
_z > AVG._z

II.

worker	wname	street	City
	_y		

works	wname	cname	salary
	_y	G._x	

company	cname	city
	P.G._x	

Conditions
CNT._y = MAX.CNT._y

III.

works	wname	cname	salary
		“NY Corporation” G._y	_x _z

company	cname	city
	P._y	

Conditions
AVG._z > AVG._x

2.

i.
```
CREATE TABLE worker
        (wid integer,
        wname varchar(40),
        age integer,
        salary real CHECK (SALARY >= 5000),
        PRIMARY KEY (wid))
```

ii.
```
CREATE TABLE company
        (cid integer,
        budget real,
        managerid integer,
        PRIMARY KEY (cid),
        FOREIGN KEY (managerid) REFERENCES worker,
        CHECK((SELECT W.age
                FROM worker W, company C)
                WHERE W.wid = C.managerid) > 40)
```

iii.
```
CREATE ASSERTION ageCheck
        CHECK ((SELECT W.age
                FROM worker W, company C
                WHERE W.wid = C.managerid) > 40)
```

iv. A domain constraint like what we created in part (ii) works very similarly to the assertion in part (iii), however the domain constraint works on a query-by-query basis and checks values against the constraint before insertion to ensure that no invalid data is added. If an invalid query comes up, it simply gets denied. An assertion is just like this however it goes further to control the whole database object. An assertion can ‘lock-out’ all other queries from performing if a single query ends up contradicting the assertion. Assertions are also more resource expensive and complex than constraints, thus making it hard to implement into DBMS. An assertion will be better if there is extra resources and if the integrity is extremely crucial. This will ensure the most that there is nothing contradicting the assertion. However, if resources are limited and/or the DBMS **does not support** assertions, then of course a domain constraint will be better. Always be cautious when using assertions since they are more expensive than constraints.

3. R(A,B,C,D,E,F)

Left only: B, D
Right only: none
Neither: none   Both: A,C,E,F

Every key will contain 'B, D'.

BDA+ = BDA
BDC+ = BDCF
BDE+ = BDEA
BDF+ = BDFC

BDAC+ = BDACEF ✓
BDAE+ = BDAE
BDAF+ = BDAFCE ✓
BDCE+ = BDCEAF ✓
BDCF+ = BDCF
BDEF+ = BDEFCA ✓

Thus we have 4 candidate keys. They are:
1) {B,D,A,C}
2) {B,D,A,F}
3) {B,D,C,E}
4) {B,D,E,F}

4. R(A,B,C,D,E,F)   FDs: A -> D
B -> C
AB -> E
E -> F

Left only: A, B
Right only: C, D
Neither: none

{AB}+ = {A,B,C,D,E,F} ✓ Thus AB is a candidate key.

Decomposing using Synthesis algorithm + canonical cover set into 3NF…

R1(A, D)	w/ FD: A -> D
R2(B, C)	w/ FD: B -> C
R3(A, B, E)	w/ FD: AB -> E
R4(E, F)	w/ FD: E -> F

This is also in BCNF form. **Therefore, highest possible decomposition of R is into BCNF.**

5. R(A,B,C,D,E,F) FDs: A -> D
B -> C
AB -> E
E -> F

R1(A,B,E,F)

- Failed 3NF check
    - **This relation is in 1NF**

R2(A,B,C)

- Failed 3NF check
    - **This relation is in 1NF**

R3(A,D)

- Passed 3NF check
- Passed BCNF check
    - **This relation is in BCNF**

6. **Yes, this decomposition is dependency-preserving as all its functional dependencies are represented without loss.**

R1(K,L,M,N,O)	w/ FDs: KL -> M; K -> NO
R2(L,P,Q,R)	w/ FDs: L -> P; P -> QR
R3(N,S,T)	w/ FD: N -> ST

All FDs represented, none lost.

7. Using synthesis algorithm and canonical cover set to decompose into 3NF….

R(K,L,M,N,O,P,Q,R,S,T) FDs: KL -> M

LN -> OP
KN -> QR
K -> S
R -> T

Left only: K, L, N
Right only: M, O, P, Q, S, T

{KLN}+ = {KLMNOPQRST} ✓ thus KLN is the candidate key

Decompose each by key…

R1(K,L,M)	**w/ FD: KL -> M**	
R2(L,N,O,P)	**w/ FD: LN -> OP**	
R3(K,N,Q,R)	**w/ FD: KN -> QR**	
R4(K,S)	**w/ FD: K -> S**	
R5(R,T)	**w/ FD: R -> T**	
R6(K,L,N)		*** added to satisfy synth. algo. Since CK not in decomposed.**

The above is decomposed into 3NF.

8. R(A,B,C,D) FDs: ABC -> D
D -> A

i. Left only: B,C
Right only: none
Neither: none
Both sides: A, D * these must be added to 'left side' to get CKs.

{BC}+ = BC

{ABC}+ = {ABCD} ✓ thus ABC is a candidate key
{BCD}+ = {ABCD} ✓ thus BCD is a candidate key

Therefore there are 2 CKs for R. They are:

1) **ABC**
2) **BCD**

ii. -PASS on 3NF test (there are no non-prime attributes)
-FAIL on BCNF test (determinants do not equal CKs)

Therefore, this relation R is in 3NF

iii. We can try to decompose into 3NF using the synthesis algorithm and canonical cover set….

R1(A,D)	w/ FD: D -> A
R2(B,C,D)	w/ no FDs

HOWEVER this is <u>NOT lossless</u>! Relation loses FD 'ABC -> D'.

Therefore, this relation <u>can not</u> be decomposed into BCNF because the resulting decomposition ends with a loss of a functional dependency.