**Problem 1:**

1. A, BD, DE, DH
2. F

Convert to FD whose right is single-attribute-dependency format:

F’ = { A->D, DB->A, DB->C, HD->A, HD->B, HD->H, A->H, E->H, H->E }

Delete useless dependencies:

F” = { A->D, DB->A, DB->C, HD->A, HD->B, A->H, E->H, H->E }

So,

Fmin = { A->D, DB->A, DB->C, HD->A, HD->B, A->H, E->H, H->E }

1. F is 1NF.

The nonprimary attribute C and functional dependency DB->C construct a full functional dependency, so F is 2NF.

A->H, H->E construct a transitive functional dependency of primary attribute E, so F isn’t BCNF.

ρ = { {HE }, { ABCDE } }

1. The BCNF schema from c) isn’t dependency-preserving.

Because:

R1 = { HE }, F1 = ΠR1(F)={ H->E, E->H }

R2 = { ABCDE }, F2 = ΠR2(F)={ A->D, DB->A, DB->C, HD->B, A->E }

F1 ∪ F2 ≠ F

ρ = { { ADH }, { ABCD }, { ABDH }, { EH } }

**Problem2:**

1. Its non-primary attributes are partial-functional-dependency for its primary attributes, so it isn’t in 2NF.

It has the following disadvantages: data redundancy; delete exception; insert exception; update exception.

1. R = { { custid, custname, custcity, custstate, carid, carmodel, caryear, rentdate, rentalfee, citytax, pickedupbid, returnbid }, { bid, bcity, bstate } }

F= { { custid -> (custname, custcity, custstate), carid -> (carmodel, caryear), (custid, carid, rentdate, pickedupbid, returnbid) -> (rentalfee, citytax) }, { bid -> (bcity, bstate) } }

1. Canonical cover:{ (custid, carid, rentdate, pickedupbid, returnbid), bid}
2. The above schema isn’t in BCNF because of F1.
3. custid -> (custname, custcity, custstate) in F11 isn’t in BCNF.

R11 = { custid, custname, custcity, custstate }

F11 = { custid -> (custname, custcity, custstate) }

R12 = { custid, carid, carmodel, caryear, rentdate, rentalfee, citytax, pickedupbid, returnbid }

F12 = { carid -> (carmodel, caryear), (custid, carid, rentdate, pickedupbid, returnbid) -> (rentalfee, citytax) }

1. carid -> (carmodel, caryear) in F12 isn’t in BCNF.

R121 = { carid, carmodel, caryear }

F121 = { carid -> (carmodel, caryear) }

R122 = { custid, carid, rentdate, rentalfee, citytax, pickedupbid, returnbid }

F122 = { (custid, carid, rentdate, pickedupbid, returnbid) -> (rentalfee, citytax) }

1. so, the R set in BCNF is { { custid, custname, custcity, custstate }, { carid, carmodel, caryear }, { custid, carid, rentdate, pickedupbid, returnbid }, { bid, bcity, bstate } }.

F = { { custid -> (custname, custcity, custstate) }, { carid -> (carmodel, caryear) }, { (custid, carid, rentdate, pickedupbid, returnbid) -> (rentalfee, citytax) }, { bid -> (bcity, bstate) } }

1. The BCNF schema from d) is dependency-preserving.
2. b)

R = { { custid, custname, custcity, custstate, carid, carmodel, caryear, rentdate, rentalfee, citytax, pickedupbid, returnbid }, { bid, bcity, bstate } }

F= { { custid -> (custname, custcity, custstate), carid -> (carmodel, caryear), rentdate -> rentalfee, (rentdate, pickedupbid, returnbid) -> citytax }, { bid -> (bcity, bstate) } }

c) Canonical cover:{ (custid, carid, rentdate, pickedupbid, returnbid), bid}

d)

R = { { custid, custname, custcity, custstate }, { carid, carmodel, caryear }, { rentdate, rentalfee }, { rentdate, pickedupbid, returnbid, citytax }, { custid, carid, rentdate, pickedupbid, returnbid }, { bid, bcity, bstate } }

F = { { custid -> (custname, custcity, custstate) }, { carid -> (carmodel, caryear) }, { (custid, carid, rentdate, pickedupbid, returnbid) -> (rentalfee) }, { (rentdate, pickedupbid, returnbid) -> citytax }, { bid -> (bcity, bstate) } }

e)

The BCNF schema from d) is dependency-preserving.

1. I will add an intermediate relationship { tid, pickedupbid, returnbid } and functional dependency tid -> (pickedupbid, returnbid).

R = { { custid, custname, custcity, custstate }, { carid, carmodel, caryear }, { rentdate, rentalfee, citytax }, { custid, carid, rentdate, tid }, { tid, pickedupbid, returnbid }, { bid, bcity, bstate } }.

F = { { custid -> (custname, custcity, custstate) }, { carid -> (carmodel, caryear) }, { (custid, carid, rentdate, tid) -> (rentalfee, citytax) }, { tid -> (pickedupbid, returnbid) }, { bid -> (bcity, bstate) } }

**Problem3:**

1. select table\_name from information\_schema.tables where table\_schema='bakery' and table\_name like "c%";
2. select TABLE\_NAME,COLUMN\_NAME,CONSTRAINT\_NAME from INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE where CONSTRAINT\_SCHEMA ='bakery' AND REFERENCED\_TABLE\_NAME = 'orders' AND REFERENCED\_COLUMN\_NAME='oid';
3. SELECT table\_name, count(\*) FROM information\_schema.`COLUMNS` where table\_schema='bakery' GROUP BY table\_name;
4. DROP PROCEDURE IF EXISTS q4p;

DELIMITER $$

CREATE PROCEDURE `q4p`()

BEGIN

DECLARE stop\_flag int DEFAULT 0;

DECLARE table\_name VARCHAR(200);

DECLARE col\_name VARCHAR(200);

DECLARE temp\_stop\_flag\_table\_name int;

DECLARE temp\_stop\_flag\_col\_name int;

DECLARE has\_col\_flag int;

-- DECLARE query\_str VARCHAR(500);

declare cols\_cur cursor for select COLUMN\_NAME from information\_schema.COLUMNS where information\_schema.`COLUMNS`.TABLE\_SCHEMA = "bakery" AND information\_schema.`COLUMNS`.TABLE\_NAME = "orders";

declare tables\_cur cursor for select information\_schema.`TABLES`.TABLE\_NAME FROM information\_schema.`TABLES` WHERE information\_schema.`TABLES`.TABLE\_SCHEMA = 'bakery';

declare CONTINUE HANDLER FOR NOT FOUND SET stop\_flag=1;

DROP TEMPORARY TABLE if EXISTS table\_tmp;

CREATE TEMPORARY TABLE table\_tmp(col varchar(200), val varchar(200));

OPEN tables\_cur;

FETCH tables\_cur INTO table\_name;

WHILE stop\_flag<>1 DO

SET temp\_stop\_flag\_table\_name = stop\_flag;

OPEN cols\_cur;

FETCH cols\_cur INTO col\_name;

WHILE stop\_flag<>1 DO

set temp\_stop\_flag\_col\_name = stop\_flag;

SELECT COUNT(\*) FROM information\_schema.`COLUMNS` WHERE information\_schema.COLUMNS.TABLE\_SCHEMA="bakery" AND information\_schema.COLUMNS.TABLE\_NAME=table\_name AND information\_schema.COLUMNS.COLUMN\_NAME=col\_name INTO has\_col\_flag;

IF has\_col\_flag<>0 THEN

SET @query\_str = CONCAT("INSERT INTO table\_tmp SELECT '", col\_name, "', cast(", col\_name, " as char) FROM ", table\_name);

-- SELECT table\_name, col\_name;

PREPARE stmt FROM @query\_str;

EXECUTE stmt;

DEALLOCATE PREPARE stmt;

END IF;

set stop\_flag = temp\_stop\_flag\_col\_name;

FETCH cols\_cur INTO col\_name;

END WHILE;

CLOSE cols\_cur;

set stop\_flag = temp\_stop\_flag\_table\_name;

FETCH tables\_cur INTO table\_name;

END WHILE;

SELECT col, COUNT(DISTINCT val) FROM table\_tmp GROUP BY col;

DROP TEMPORARY TABLE IF EXISTS table\_tmp;

CLOSE tables\_cur;

END$$

DELIMITER ;

CALL q4p();

1. SELECT \* FROM customer WHERE customer.firstname in ("Cheese", "Cream", "Chocolate", "Sugar") OR customer.lastname in ("Cheese", "Cream", "Chocolate", "Sugar");