Problem 1:

a) A, BD, DE, DH

b) 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,

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

c) 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 } }

d) The BCNF schema from c) isn't dependency-preserving.

Because:

R_1 = { HE }, $F_1 = \Pi_{R1}(F)$={ H->E, E->H }

R_2 = { ABCDE }, $F_2 = \Pi_{R2}(F)$={ A->D, DB->A, DB->C, HD->B, A->E }

$F_1 \cup F_2 \neq F$

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

Problem2:

a) 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.

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), (custid, carid, rentdate, pickedupbid, returnbid) -> (rentalfee, citytax) }, { bid -> (bcity, bstate) } }

c) R1 CK: (custid, carid, rentdate, pickedupbid, returnbid);

R2 CK: bid

d) The above schema isn't in BCNF because of F1.

1. 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) }

2. 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) }
3. 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) } }

e) The BCNF schema from d) is dependency-preserving.

f) 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)
R1 CK: (custid, carid, rentdate, pickedupbid, returnbid);
R2 CK: 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, citytax) }, { (rentdate, pickedupbid, returnbid) -> citytax }, { bid -> (bcity, bstate) } }
e)
The BCNF schema from d) is dependency-preserving.

g) 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();
```

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