CSC343 Assignment 3

Mengning Yang(1002437552) Keyu Wang(1003540625)

1.(a) we get all the candidate key by the following algorithm:

Given a set F of functional dependencies that hold on R, we can find all candidate keys of R as follows:

1. Find all attributes that have not appeared on the RHS of any FD. Denote this set by A.
2. Denote the set of attributes that appear on the RHS of some FD, but not on the LHS of any FD by B.
3. Compute the closure set A^+, if A^+=R, then A is the only candidate key.
4. If A^+≠R, then for each attribute x in R-B, test whether A∪{x} is a candidate key. If not, try to add another attribute in R-B to A and test whether it is candidate key.
5. Repeat step 4, until all candidate keys have been found.

Left: step 1 in the algorithm above	M	Right: step 2 in the algorithm above
CE	ABD	F

CE must be a part of the key, and F must not be a part of the key.

C^+ = C

CE^+ = ABCDEF, CE^+ = all attributes in R

Therefore, CE is the key.

1.(b)

A → B as (a)

CD → A as (b)

CB → D as (c)

CE → D as (d)

AE → F as (e)

Step 1: Since all RHS are singleton. None of them can be split.

Step 2: For each FD, try to reduce the LHS:

(a) A^+ = A, this FD is not redundant, we cannot reduce the LHS of this FD.

(b) C^+ = C, D^+ = D. No singleton yields A, this FD is not redundant, we cannot reduce the LHS of this FD.

(c) C^+ = C, B^+ = B. No singleton yields D, this FD is not redundant, we cannot reduce the LHS of this FD.

(d) C^+ = C, E^+ = E. No singleton yields D, this FD is not redundant, we cannot reduce the LHS of this FD.

(e) A^+ = AB, E^+ = E. No singleton yields F, this FD is not redundant, we cannot reduce the LHS of this FD.

Let S denotes the set of FD'S

Step3. Try to eliminate each FD.
(a) A^{+}(S−a) = A. We need this FD.
(b) CD^{+}(S−b) = CD. We need this FD.
(c) CB^{+}(S−c) = CB. We need this this FD.
(d) CE^{+}(S−d) = CE. We need this this FD.
(e) AE^{+}(S−e) = B. We need this this FD.
Therefore, the given FDS form a minimal basis.

1.(c)
Relation:
For each FD X->Y in Minimal basis:
R1 (AB), R2(ACD), R3(BCD), R4(CDE), R5(AEF)
Since CDE is the superkey of L, no relation needs to be added.

1.(d)
They all are already in BCNF. Because these relations are formed from each FDs of the original relation, therefore R1 with FD A-> B, R2 with FD CD->A, R3 with FD CB->D, R4 with FD CE->D, R5 with FD AE->F. Each of these FD's LHS is the superkey of the relation. Therefore, every relation in Part (c) is in BCNF.

2. (a) Proof:
To prove the claim is true, we need to show that the decomposition fulfills three conditions.
Let relation R be decomposed into two relations R_1 and R_2.
i. $R_1 \cup R_2$ = R
ii. $R_1 \cap R_2$ ≠ null
iii. $[R_1 \cap R_2]^{+}$ -> R_1 or $[R_1 \cap R_2]^{+}$-> R_2
Suppose $[R_1 \cap R_2]^{+}$-> R_2, then a row of R_1 can combine with exactly one row of r2 in the natural join (since in R_2 a particular set of values for the attributes in R1 ∩ R2 defines a unique row).
Similarly, if $[R_1 \cap R_2]^{+}$ -> R_2, then a row of R_1 can combine with exactly one row of r2 in the natural join as well.
Hence, the claim is true.

2.(b)
Yes. By definition,
We say a relation R is in BCNF if for every nontrivial FD X->Y that holds in R, X is a superkey and X -> A satisfies 3NF if and only if X is a superkey or A is prime.
The difference between 3NF and BCNF is that BCNF is stricter than 3NF, 3NF is a subset of BCNF.
1.If R is in 3NF, it may or may not be in BCNF.
2. If R is in BCNF, it must be in 3NF.
Therefore, a relation and a set of FDs can be in both BCNF and 3NF at the same time when for every non-trivial FD x->Y that holds in R, X is a superkey.

3. Prove or disprove that:
(a) If A → B then B → C
The claim is false.

Since no information about relation between B and C are provided.
Assume A represent student number of a student, B represents gender of the student, C represents the cGPA of the student. We know that A -> B, however, B doesn't functionally determine C. Therefore, the claim is false.

(b) If AB → C then A → C and B → C
The claim is false.
Assume attribute A represent latitude, B represent longitude. The combination of A and B determines C, a point on the world map. We cannot get C from only A or B, therefore, by counter example, the claim is false.

4. Design and DDL
(a)Define a single relation for this domain.
JuiceBar(<u>city, order_id,</u> beverage_name, beverage_size, phone_number, manager, beverage_price, beverage_calories, beverage_count, loyalty_card_id, home_store, transaction_count, date)

For convenience, give each entity sets a letter for easy reference:
A: city
B: order_id
C: beverage_name
D: beverage_size
E: phone_number
F: Manager
G: beverage_price
H: beverage_calories
I: beverage_count
J: loyalty_card_id
K: home_store
L: transaction_count
M: date

(b)Write all of the functional dependencies
city -> phone_number, manager
order_id-> date, beverage_name, beverage_size, beverage_price, beverage_calories, Loyalty_card_id
beverage_name, beverage_size-> beverage_price, beverage_calories
Loyalty_card_id -> home_store, transction_num
city, beverage_name, beverage_size -> beverage_count

which corresponds to: A->EF
B->MCDGHJ
CD->GH
J->LK
ACD->I

(c)Provide a useful instance of you relation that shows all three types of anomalies.

city	order_id	beverage_name	beverage_size	phone_number	manager	beverage_price	beverage_calories	beverage_count	loyalty_card_id	home_store	transaction_count	date
Toronto	001	lime	large	12345	mark	6	700	64	999	Toronto	23	8/29
Toronto	002	kiwi	reg	12345	mark	5	500	33	999	Toronto	24	8/29
van	032	lime	large	85757	arnald	6	700	68	345	van	90	9/02
quebec	092	apple	large	38792	stanley	6	700	20	987	quebec	69	7/28
quebec	093	apple	large	38792	stanley	6	700	19	987	quebec	70	7/28

Anomalies which have presented:

1. Redundancy: Lots of duplicate information for the Toronto and Quebec tuples, they are repeating the same information for the home store, phone number and manager for every order.
2. Update: Changing the manager of Toronto store in one tuple requires updating all Toronto tuples.
3. Deletion: Removing the order with order_id 032 in Vancouver can remove the information for Vancouver store entirely.

(d) Decompose your relation so that it does not violate BCNF.
---Using A->EF we decompose our initial big relation into two new relations.
A+ = {AEF}, we have R1 = {AEF} and R2 = {ABCDGHIJKLM}

---Project the FDs onto R1= {AEF}
A+ = AEF, so we get A->EF, which is a superkey.
E+ = E, nothing
F+ = F, nothing
EF+ = nothing
No need to find the closure for A's supersets
So, this relation satisfies BCNF.

---Project the FDs onto R2 = {ABCDGHIJKLM}
A+ = AEF, we get nothing
B+ = BCDGHJKLM, we get B->CDGHJKLM (but not A), so this FD violates BCNF, we have to abort this relation and decompose it more.

---Decompose R2 using B->CDGHJKLM into R3 ={BCDGHJKLM} and R4 = {ABI}

---Project the FDs onto R3 ={BCDGHJKLM}
B+ = BCDGHJKLM, we get B->CDGHJKLM, which is a superkey.
C+, D+, G+, H+, we get nothing
J+ = JLK, we get J->LK, which does not cover all attributes, so this FD violates BCNF, we have to abort this relation and decompose it more.

---Decompose R3 using J->LK into R5 = {JLK} and R6={BMCDGHJ}

---Project the FDs onto R5 = {JLK}
J+ = JLK, we get J->LK, which is a superkey
L+, nothing
K+, nothing
No need to find the closure of J's supersets
JK+ = JKL, nothing
So, this relation satisfies BCNF.

---Project the FDs onto R6={BMCDGHJ}
B+ = BCDGHJKLM, we get B->CDGHJM, which is a superkey
M+, C+, D+, G+, H+, J+, we get nothing
CD+ = CDGH, we get CD->GH, which does not cover all attributes, so this FD violates BCNF, we have to abort this relation and decompose it more.

---Decompose R6 using CD->GH into R7 = {CDGH} and R8= {BCDJM}

---Project the FDs onto R7 = {CDGH}
C+, D+, G+, H+, we get nothing
CD+ = CDGH, we get CD->GH, which is a superkey
DG+, GH+, CH+, CG+, DH+, we get nothing
No need to find the closure of CD's supersets
DGH+, we get nothing
So, this relation satisfies BCNF.

--- Project the FDs onto R8= {BCDJM}
B+ = BCDGHJKLM, we get B->CDJM, which is a superkey
C+, D+, J+, M+, we get nothing
No need to find the closure of B's supersets
CD+, CJ+, CM+, DJ+, DM+, JM+, CDJ+, CDM+, DJM+, we all get nothing
So, this relation satisfies BCNF.

---Return to R4 and Project the FDs onto R4 = {ABI}
A+ = AEF, nothing
B+ = BCDGHJKLM, nothing
I+ = I, nothing
AB+, AI+, BI+, we all get nothing
So, this relation satisfies BCNF.

---Finally, we get
R1 = {AEF} with FD A->EF
R5 = {JLK} with FD J->LK
R7 = {CDGH} with FD CD->GH
R8 ={BCDJM} with FD B->CDJM
R4 = {AIB} with no FDs

Which corresponds to
Store(city, phone_number, manager)
Membership(loyalty_card_id, transaction_number, home_store)
Beverage(name, size, price, calories)
Order(order_id, beverage_name, beverage_size,loyalty_card_id,date)
Inventory(city, order_id, beverage_count)

(e)Explain how your new relations prevent the anomalies in part(c).
We decomposed the big relation by the algorithm of BCNF, which offers no anomalies (due to no redundancy).

(f)DDL (see fruites.ddl and fruits-demo.txt)
describe the decisions for any constraints you put in your DDL file.

relation store:
the city is the key because there is only one store per city.
the phone number for each store should be unique and not null.
the manager for each store should also be not null.

relation membership:
the id, aka loyalty_card_id is the primary key.
the transaction_number is an integer and should not be null, it at least is 0.
the homestore references the store(city), because city of the home store can only be the stores that the Fresh Juice business opened.

relation beverage:
the name and size of a beverage should not be null.
and the price and calories should at least be 0.
name and size of a beverage is the primary key, because there could be large size and regular size of a beverage, the calories and prices between large and regular beverage are different.

relation orders:
the id, aka the order_number is the primary key.
the beverage_name, beverage_size reference to beverage(name, size) together, because the combination of a beverage should only be what the stores have.
the date should not be null.
the loyalty_card_id references membership(id) because it can only be customer who actually is a member of the store.

relation beverage_inventory:
order_id INT references orders(id), because we can only have order ids that actually is an order.
the combination of city and order_id should be unique because there can't be two orders with the same order_id in a same store. The combination of city and order_id is the primary key of this relation.
We keep track of the inventory of each drink by storing the order_id of each order, the beverage_count is for the drink of the order_id that ordered in the 'city' store.