# CSC453 Homework 3

John William Lynch
CSC 453 : Homework 3
Due : Nov 2nd
Answers written below each bullet point provided:

## Part A: Functional Dependencies

### A-1 Transitive Dependency and Keys

You have a relation R(L,M,N,O,P,Q) and a set of functional dependencies F = {LNO→M, MN→LOP, N→O, OP→LN}.

- [2pt] Can we infer NP → LM from F ?
  - Yes, we can infer this by using the following logic of augmentation and transitivity. Although it is convoluted, (NP)+ contains {O, L, N, M} within its calls of references, therefore NP-> LM can be inferred from F.
- [3pt] Can we infer NQ → LO from F ?
  - We cannot infer this, as even though N, L, O, Q are all present within our relation R set, we see that within the item Q is never referenced. At no point do we see Q in a functional dependency with any other element, therefore we CANNOT infer that NQ -> LO from F, as none of the dependency rules accounts for an item to be referenced from null.

### A-2 Keys

(i) [5pt] Find **all** the candidate keys of the Relation R(ABCDE) with FD's:

D → C,
CE → A,
D → A,
and AE → D

- Remember that a candidate key is when X is a super key, however there is no subset Y of X that is a superkey, then we say that X is a candidate key:
- Using the above definition and provided schema, the candidate keys would be:
- From D, we can determine A and C, and then a combination of C/A alongside E (such as CE, or AE), can be used to determine A or D. However, notice that B is in our relation R, and is never stated, so we must also include it in our candidate keys. Again, if we reduce any of these keys further, we may remove B from the list, and therefore render it missing an attribute and not a superkey

- Candidate Keys:
    - B, E, A (BEA)+
    - B, E, C (BEC)+
    - B, E, D (BED)+
    - {BEA, BEC, BED} : B is not mentioned in the relations, so we must place it within the candidate keys

(ii) [5pt] Determine **all** the candidate and superkeys of the relation R(ABCDEF) with FD's:

AEF → C, BF → C, EF → D, and ACDE → F

- A super key is when X contains all attributes of a certain set:
- We notice that unlike the previous example, all elements in the relation are mentioned at least once. We can then use
- Super Keys:
- Again, we want to find the keys that contain all attributes in R
    - ABEF : The smallest key that will give us all elements. Because B isn't implied by any other attribute, we must imply it outselves; AEF implies C for our attributes, and EF implies D, thus cause **ABEF** to be the super key
- Candidate Keys:
    - We are given two candidate keys that will cover all elements:
    - ABEF : AEF covers C, and then EF covers D
    - ABECD: ACDE covers F, BF covers C

**A-3 Minimal Cover**

[5pt] Find a minimal cover for the following set F of functional dependencies.

A→BC
AB→D
C→AD
D→B

Show your working clearly. Points will be deducted if you do not show the extraneous attributes, and their elimination:

- D -> B is not covered by any other elements that we can reduce, we will note to keep D->B in our minimal cover.
- A->BC rewritten as :
- A->B
- A->C
- AB->D rewritten as :
- A->D
- B->D
- C->AD rewritten as:

- C->A
- C->D
- Further reduce: remove items that are already implied using logic. Need four items, A, B, C, D:
- **D->B** :
- B is now covered. We need to reach C using A: which is found using A->BC:
- **A->C**. Now A, B, covered.
- Need to make sure that C covers the remaining elements it hasn't reached.
- We also see that C covers AD, implied by C->AD. We have A->C already but this does not imply C->A, so we can add that one cover already created: C->AD
- Simply speaking ,removing one of the elements (AB->D) will give us the minimal cover
- **Final minimal cover: {A->C, C->AD, D->B}**

**A-4 Equivalence**

[15pt] Consider the following set of F.Ds. Determine if FD1 is equivalent to FD2 or to FD3:

To ensure that they are equivalent, we need to make sure that all F.Ds in FD1 can be derived from F.Ds in FD2 or FD3, and vice versa for FD2 and FD3

FD1:

{BC->D,ACD->B,CG->B,CG->D,AB->C,C->A,D->E,BE->C,D->G,CE->A,CE->G}

FD2:

{AB->C, C->A, BC->D ,CD->B,D->E,D->G,BE->C,CG->D}

**FD1 to FD2:**

-For FD1 to FD2: all elements match directly except CD-> B. If we view CD->B in FD1, it is not present. However, placing (CD)+ implies the following relation = {A,B,C,D,E,G}. This implies that CD can determine the following possibilities: CD->A, CD->B, etc. as ACD->B is present. Therefore FD2⊃FD1

**FD2 to FD1:**

- Some elements match, however ACD->B found in FD1 is not present in FD2. However, CD->B is present in FD2, and also AB->C is present. AB->C implies A->C, B->C. Looking for the closure (ACD)+ in FD2 presents us with the following variables {A, B, C, D, G}: due to this, we can see that ACD are all present in FD2, and we can then determine if ACD -> B.
- Because of this, FD1 ⊃ FD2

Because FD1 ⊃ FD2 and vice versa, then we satisfy the equivalence operation, and therefore Upon further review of FD2 and FD3, they both seem to be the same relational set but a few items are just rearranged.

FD3:
{AB->C,C->A, D->G, BE->C,CG->D,CE->G,BC->D,CD->B,D->E}
FD3 has an extra item, CE->G, which is also contained in FD1 at the end of the creation of the relation method. Using the same methods presented above, we can see that FD3 ⊃ FD1, as all elements in FD3 can be found right away in FD1, and also that the item that isn't found, CD->B, can be found by using (CD)+ in FD1. We will then do the same process as above from FD2. It does seem that both FD2 and FD3 can be considered equivalent to FD1

**Part B: Normalization**
**B-1. Dependency Preservation (7 points)**

For the relation R(w,x,y,z), consider the decomposition D consisting of $R_1$(w,y,z) and $R_2$(x,y), and the set of functional dependencies
F ={y→xz; yz→w; x→w}. Recall that the projection of set of functional dependences G on relation $R_x$ consists of every functional dependency in $(G)^+$ that contains only attributes from $R_x$.

**a.** Compute the projection of F on $R_1$.
**Projection** : {y->z, yz->w}
**b.** Compute the projection of F on $R_2$.
**Projection**: {y->x}
**c.** Does the decomposition D preserve the set of dependencies F? Why or why not?

No, the decomposition does not preserve the dependencies of F because there are a few elements that are missing from the given dependencies, such as x->w and y->xz

**B-2. Lossless Decomposition (8 points)**
Perform the test for the non-additive join property (lossless join) for the relation R($A_1$, $A_2$, $A_3$, $A_4$, $A_5$), and the decompositions $D_a$, $D_b$, $D_c$, $D_d$ and set of functional dependencies F given below. You can ignore attributes that are not mentioned in each particular subsection (e.g., you can ignore absence of $A_4$ in $D_d$, just test the join between $R_1$ and $R_2$):

$F = \{A_1 \rightarrow A_4; A_4 \rightarrow A_5; A_3 \rightarrow A_4\}$

$D_a = \{R_1(A_1, A_2), R_2(A_3, A_4, A_5)\}$
$D_b = \{R_1(A_3, A_4), R_2(A_4, A_5)\}$
$D_c = \{R_1(A_1, A_5), R_2(A_4, A_5)\}$
$D_d = \{R_1(A_1, A_2, A_3), R_2(A_1, A_2, A_5)\}$

To check if the following are lossless or not, we must use a method to test the following to see if any elements are missing in the decompositions: I will place an X inside of the box to see if it exists in that current relation in the first pass of the relation. After I place the X, I will then place an A to show that F (the functional dependencies), help fill out that row in order to make the decomposition have the non-additive join property:

Based upon notes/algorithm, once a row is completed then it is proven that we have a non-additive join, as it will yield our original relation:

**a.** Does the decomposition $D_a$ have the non-additive join property? Explain why or why not.

| | A1 | A2 | A3 | A4 | A5 |
|---|---|---|---|---|---|
| R1 | X | X | | **A** | **A** |
| R2 | | | X | X | X |

After adding in the X's, we see that A1 exists in R1, and thus it implies the existence of A4; A4 in turn brings in A5. However, A3 implies A4, but nothing implies A3 for our top row, and A3 is given in the second row, however A1 and A2 cannot be implied in the bottom row.
Because of this, this relationship does NOT contain the non-additive join property.

**b.** Does the decomposition $D_b$ have the non-additive join property? Explain why or why not.

| | A1 | A2 | A3 | A4 | A5 |
|---|---|---|---|---|---|
| R1 | | | X | X | **A** |
| R2 | | | | X | X |

Although we do have A5 implied by A4, because we are not given A1, A2 and they cannot be implied by the given schema. Therefore, this relationship does NOT contain the non-additive join property.

**c.** Does the decomposition $D_c$ have the non-additive join property? Explain why or why not.
$F = \{A_1 \rightarrow A_4; A_4 \rightarrow A_5; A_3 \rightarrow A_4\}$

| | A1 | A2 | A3 | A4 | A5 |
|---|---|---|---|---|---|
| R1 | X | | | **A** | X |
| R2 | | | | X | X |

Similar to the previous problem, this does contain A1, however A1 implies A4, which is also given in the schema. A5 is provided by the current relation. Because of this, A2 and A3 cannot be reached/implied. Therefore, this relationship does NOT contain the non-additive join property.

**d.** Does the decomposition $D_d$ have the non-additive join property? Explain why or why not.

| | A1 | A2 | A3 | A4 | A5 |
|---|---|---|---|---|---|
| R1 | X | X | X | **A** | A |
| R2 | X | **X** | | A | X |

Given A1 in row2, we see that we can imply A4 inside of Row 1, which in turn allows us to imply A5 in row1 as well, which is also present in row2. Due to this, we have a full row, and we DO have a completely lossless function.

**B-3. Normalization (15 points)**

Consider the universal relation

EMPLOYEE(ID, First, Last, Team, Dept, Salary)

with the following set F of functional dependencies:

ID → First
ID → Last
First, Last → ID
Last → Team
ID → Dept
ID → Salary
Salary → Dept

**a.** Identify candidate keys of EMPLOYEE.

-ID acts as a reference for four of the six items in our relation, it will acts as a candidate key:

-The only time that Team is referenced is during Last-> Team, therefore Last needs to be apart of our keys too

-To see ID, we need to use the First, Last name combo, so our keys will be:

Candidate keys: {ID, (First, Last)}

**b.** Construct a decomposition of EMPLOYEE into relations in 3NF that preserves dependencies. Show full working. How can you be sure that your decomposition is dependency-preserving?

Using the above candidate keys that will act in our new decomposition, we see that First, Last are now our prime attributes for creating the candidate key. Also, ID is apart of the prime attributes for the candidate key:

**Relation 1:**

EMPLOYEE(ID, First, Last, Team, Dept, Salary) decomposed into:

R1 = EMPLOYEE(ID, First, Last, Team, Dept, Salary)
With ID -> Team, Dept, Salary, First, Last

R2 = EMPLOYEE(First, Last, ID)
With First,Last -> ID

We now have all of our six attributes. For my second relation, it may seem odd that I have all of the super keys referencing each other, however this allows decomposition to be lossless, or dependency preserving to our original functional dependency.
This means that if someone attempts to do the first relation, they will be able to have a full functional relation that satisfies all items as ID helps reference all elements.

However, if someone was to do the second element, with just the first and last name, it may seem we are missing references to all of the other information. However, because First, Last name infer ID, we then have access to the inferences that IDs make, allowing us to create another relation that satisfies the previously defined functional dependencies.

**c.** Are all of the relations in your decomposition in BCNF? Either explain why they are, or identify one that is not and explain why it is not. (Note that for a relation to be in BCNF, the determinants of all functional dependencies in the relation must be superkeys *of that relation* – not superkeys of the original universal relation.)

Yes, because if we review the original candidate keys, we see that ID and First,Last compose the candidate keys, and in turn the super-key of our relation. Decomposing them into two separate relations where ID and First,Last are separate provide us with furthering the relation into 3NF. ID and First,Last are still the super keys in those

scenarios. When we look at the created dependencies in my decompositions, we see ID -> () and First,Last -> () where Id and First,Last are my super keys. Based on the BCNF criteria, we need X->A where X is a super key. In my scenarios ID is a super key, and First,Last is also a super key, therefore satisfying the requirement of BCNF

**B-4. 3NF (15 points)**
Which of the following relations is in Third normal form (3NF)? Give sufficient reasoning if not in 3NF.

**To do:**

(a) R(ABCD) F = {ACD → B; AC → D; D → C; AC → B}
This relation is in 3NF form
(b) R(ABCD) F = {AB → C; BCD → A; D → A; B → C}
No, this relation is not in 3NF form, as 3NF states that there should not be any determination of an element by transitive association, meaning the items should be directly there and we should not have to attempt to go looking for elements to match up. The statement AB -> C shows AB as a non superkey, and that C is not apart of the candidate keys.
(c) R(ABCD) F = {AB → C; ABD → C; ABC → D; AC → D}
No, this relation is not in 3NF form. AC-> D does contain transitive information and AC is not apart of the candidate keys that help determine the remaining items of the relation.
(d) R(ABCD) F = {C → B; A → B; CD → A; BCD → A}
Candidate keys: C determines B, CD determines A, A determines B, Combo of BCD determines A as well: using this, we see that all we need is the first LHS of the first two items: being C and CD:
Candidate Keys : {CD}
We also need to check if the RHS contain candidate keys: C->B, will throw an error, alongside the remaining functional dependencies.
Due to this, this Relation is Not in 3NF form

**B-5. (BCNF) (15 points)**
Which of the following relations is in BCNF? Give sufficient reasoning if not in BCNF.

For BCNF, we need to check that for every Functional Dependency where X->A, X is a super key. Therefore, we need to check candidate keys and find the super-key

(a) R(ABCD) F = {BC → A; AD → C; CD → B; BD → C}
Finding the candidate keys: BC -> A, CD ->B.

-This relation is NOT in BCNF form because we can get our super-key by using only two of the four above combinations. Because super-keys are supposed to be minimal, we must leave at least ONE of the above relations out, and once we check to see if it is BCNF, we will find that X is NOT a super-key, which in turn won't satisfy the BCNF requirement

(b) R(ABCD) F = {BD → C; AB → D; AC → B; BD → A}

Candidate Keys: BD referenced twice, then AB, AC need to be called so:
Super keys = {AB, AC, BD}

-using the above as super key as that is the most minimal set we can acquire, we then see that each satisfies the property of X being a super key when X->A.

Therefore, this relation is in BCNF

(c) R(ABCD) F = {A → C; B → A; A → D; AD → C}

A minimal cover would be as follows: A contains two of the above, and B contains an A, we can derive AD from using A as A->C and A->D can do so. Because of this our cover will be = {A, B}

-However, using the BCNF method, we know that we satisfy the condition for A and B, however once we hit AD the condition fails.

Therefore, the above relation is NOT in ABCD form.