Ben Juarez

CS 121

PS7

1. We want to determine the presence of non-trivial functional dependencies given simple E-R diagrams.
    a. There are no non-trivial functional dependencies since it is a many-to-many relationship.
    b. Many-to-one relationship between B & A, respectively ⇒ b ➔ a
    c. Many-to-one relationship between A & B, respectively ⇒ a ➔ b
    d. One-to-one relationship between A & B ⇒ b ➔ a, a ➔ b

---

2. Let a = α, B = **β**, y = ***γ***, d = **δ**
    a. Union rule: If a ➔ B holds & a ➔ y holds, then a ➔ By holds
        i. Augmentation rule: If a ➔ B, then aa ➔ aB
        ii. Augmentation rule: If a ➔ y, then aB ➔ yB
        iii. Transitivity rule: If aa ➔ aB & aB ➔ yB, then aa ➔ yB
        iv. If aa ➔ yB, then a ➔ yB due to equivalence of aa & a
        v. QED
    b. Decomposition rule: If a ➔ By holds, then a ➔ B holds & a ➔ y holds
        i. Reflexivity rule: By ➔ y, By ➔ B since B, y ⊆ By
        ii. Transitivity rule: If a ➔ By & By ➔ y, then a ➔ y
        iii. Transitivity rule: If a ➔ By & By ➔ B, then a ➔ B
        iv. QED
    c. Pseudotransivity rule: If a ➔ B holds & yB ➔ d holds, then ay ➔ d holds
        i. Augmentation rule: If a ➔ B, then ay ➔ By
        ii. Transitivity rule: If ay ➔ By & yB ➔ d, then ay ➔ d
        iii. QED

---

3. R = (A, B, C, D, E) & F = {A ➔ BC, CD ➔ E, B ➔ D, E ➔ A}
    a. We want to specify all candidate keys for R & show that each candidate key is a superkey for R. We know a candidate key is a minimal superkey, so for some attribute-set X, if X ➔ ABCDE then X is a superkey for R, implying that it is also a candidate key. Let us examine the following attribute-sets.
        i. A:
            1. Decomposition: If A ➔ BC, then A ➔ B & A ➔ C

2. Transitivity: If A ➔ B & B ➔ D, then A ➔ D
3. Union: If A ➔ C & A ➔ D, then A ➔ CD
4. Transitivity: If A ➔ CD & CD ➔ E, then A ➔ E
5. Reflexivity: A ➔ A since A ⊆ A
6. Union: With A ➔ BC (plus 1-5), then A ➔ ABCDE
7. A is a candidate key
8. QED

ii. E:

1. Transitivity: If E ➔ A & A ➔ ABCDE, then E ➔ ABCDE
2. E is a candidate key
3. QED

iii. CD:

1. Transitivity: If CD ➔ E & E ➔ ABCDE, then CD ➔ ABCDE
2. CD is a candidate key
3. QED

iv. BC:

1. Augmentation: If B ➔ D, then BC ➔ CD
2. Transitivity: If BC ➔ CD & CD ➔ E, then BC ➔ ABCDE
3. BC is a candidate key
4. QED

v. Therefore, we have proved that the candidate keys for R are A, E, CD, BC.

b. We want to describe all functional dependencies (trivial and non-trivial) that will appear in the closure F+ of F. Let us summarize them below using the previously proved candidate keys along with other appropriate combinations.

i. A ➔ ABCDE and all dependencies generated from Decomposition
ii. B ➔ D (given)
iii. B ➔ B (By Reflexivity)
iv. B ➔ BD (By Union)
v. BD ➔ BD (By Augmentation or Reflexivity)
vi. BD ➔ B (By Decomposition)
vii. BD ➔ D (By Decomposition)
viii. BC ➔ ABCDE and all dependencies generated from Decomposition
ix. CD ➔ ABCDE and all dependencies generated from Decomposition
x. C ➔ C (By Reflexivity)
xi. D ➔ D (By Reflexivity)
xii. E ➔ ABCDE and all dependencies generated from Decomposition
xiii. All valid combinations from: (candidate key) + (A/B/C/D/E) ➔ X

1. where X ⊆ R
2. where A/B/C/D/E implies attributes from the given R

4. Given R(A, B, C, D), we want to determine if A➔➔BC logically implies A➔➔B and A➔➔C through proof or counterexample (where ➔➔ indicates multivalued dependency). It does not appear that any of the inference rules could be utilized to prove that the claim is true. So, using the formal and pictorial definition of multivalued dependencies, let us provide a counterexample through tables. Let us first establish a simple table that satisfies A➔➔BC:

Table 1	**A**	**B**	**C**	**D**
t1	a	b	b	d
t2	a	c	c	e
t3	a	b	b	e
t4	a	c	c	d

By definition, we can see that Table 1 would indicate that A➔➔BC holds:

- t1[A] = t2[A] = t3[A] = t4[A]
- t1[BC] = t3[BC] AND t2[BC] = t4[BC]
- t1[AD] = t4[AD] AND t2[AD] = t3[AD] (where R - BC = AD)

Now let us try to modify Table 1 into Table 2 such that it satisfies A➔➔BC and A➔➔B:

Table 2	**A**	**B**	**C**	**D**
t1	a	b	c	d
t2	a	c	c	e
t3	a	b	c	e
t4	a	c	c	d

Now, we can see that Table 2 still satisfies A➔➔BC:

- t1[A] = t2[A] = t3[A] = t4[A]
- t1[BC] = t3[BC] AND t2[BC] = t4[BC]
- t1[AD] = t4[AD] AND t2[AD] = t3[AD] (where R - BC = AD)

Table 2 also satisfies A➔➔B:

- t1[A] = t2[A] = t3[A] = t4[A]
- t1[B] = t3[B] AND t2[B] = t4[B]
- t1[ACD] = t4[ACD] AND t2[ACD] = t3[ACD] (where R - B = ACD)

However, Table 2 does not satisfy A➔➔C, thus providing a counterexample where A➔➔BC and A➔➔B hold true while A➔➔C does not hold:

- t1[A] = t2[A] = t3[A] = t4[A]
- t1[C] = t3[C] AND t2[C] = t4[C]
- **t1[ABD] ≠ t4[ABD] AND t2[ABD] ≠ t3[ABD] (where R - BC = ABD)**

Therefore, by counterexample, the original claim is false.

---

5. Given R(A, B, C, D, E, G) and F = {A➔E, BC➔D, C➔A, AB➔D, D➔G, BC➔E, D➔E, BC➔A}
    a. We want to compute a canonical cover F_c of F. We shall compute F_c after each step.
        i. Union: If D➔G & D➔E, then D➔GE
            1. Now we have, F_c = {A➔E, BC➔D, C➔A, AB➔D, D➔GE, BC➔E, BC➔A}
        ii. Transitivity: If BC➔A & A➔E, then BC➔E
        iii. Therefore, BC➔E is extraneous
            1. Now we have, F_c = {A➔E, BC➔D, C➔A, AB➔D, D➔GE, BC➔A}
        iv. Augmentation: If C➔A, then BC➔AB
        v. Transitivity: If BC➔AB & AB➔D, then BC➔D
        vi. Therefore, BC➔D is extraneous
            1. Now we have, F_c = {A➔E, C➔A, AB➔D, D➔GE, BC➔A}
        vii. C➔A is logically implied by F (obvious)
        viii. Therefore, B is extraneous in BC➔A
            1. Now we have, F_c = {A➔E, C➔A, AB➔D, D➔GE}
        ix. Final computation: F_c = {A➔E, C➔A, AB➔D, D➔GE}
    b. Now, we want to find a candidate key for R and demonstrate that it is a superkey for R by computing its attribute-set closure. We then want to demonstrate that it is a candidate key through computation of the attribute-set closures of its proper subsets. We will first test if BC is a superkey.
        i. Let us compute the attribute-set closure for BC (BC+):
            1. Start with BC+ = BC
            2. BC➔D, so BC+ = BCD

3. BC➔D & D➔G, so BC+ = BCDG (Transitivity)
4. BC➔E, so BC+ = BCDEG
5. BC➔A, so BC+ = ABCDEG
6. Therefore, we have shown BC is a superkey

ii. Now, let us compute the attribute-set closures of its subsets (B+, C+)

1. Start with B+ = B
    a. B+ = B
2. Start with C+ = C
    a. C➔A, so C+ = AC
    b. C➔A & A➔E, so C+ = ACE (Transitivity)
3. BC is a candidate key since B+ and C+ is not R.

c. We want to decompose R into a BCNF schema and prove that each of the final relation-schemas is in BCNF. We know that all dependencies in F_c are not preserved by this BCNF decomposition. At this point, we have established that BC is a superkey for R. Following the process for BCNF decomposition, we arrive at the following by starting with A➔E:

i. R1 = (A, E) - In BCNF since A is a primary key

ii. R2 = (R - E) = (A, B, C, D, G)

1. We know C➔A
2. R2 = (C, A) - In BCNF since C is a primary key

iii. R3 = (R - A) = (B, C, D, G)

1. We know D➔G
2. R3 = (D, G) - In BCNF since D is a primary key

iv. R4 = (R - G) = (B, C, D) - In BCNF since BC is primary key (already know that BC is a candidate key)

v. So, we arrive at the final decomposition:

1. R1 = (A, E)
2. R2 = (C, A)
3. R3 = (D, G)
4. R4 = (B, C, D)

vi. Let us note that the following dependencies were not preserved:

1. AB ➔ D
2. D ➔ E (from D ➔ GE)

d. We want to find a second BCNF decomposition for R (and prove that each of the final relation-schemas is in BCNF). We know that all dependencies in F_c are not preserved by this BCNF decomposition. Following the same process for BCNF decomposition, let us start with a different dependency AB➔D:

i. R1 = (A, B, D) - In BCNF since AB is a primary key

ii. R2 = (R - D) = (A, B, C, E, G)

1. We know A➔E
2. R2 = (A, E) - In BCNF since A is a primary key

iii. R3 = (R - E) = (A, B, C, G)
    1. We know C➔A
    2. R3 = (C, A) - In BCNF since C is a primary key
iv. R4 = (R - A) = (B, C, G) - In BCNF since BC is a primary key (already know that BC is a candidate key)
    1. We know BC➔D & D➔G, so by Transitivity, we know BC➔G
v. So, we arrive at a slightly different decomposition:
    1. R1 = (A, B, D)
    2. R2 = (A, E)
    3. R3 = (C, A)
    4. R4 = (B, C, G)
vi. Let us note that the following dependencies were not fully preserved:
    1. D ➔ GE

e. We want to create a 3NF schema for R using the 3NF schema synthesis algorithm. Following the 3NF synthesis algorithm, we start with our previously defined F_c = {A➔E, C➔A, AB➔D, D➔GE}. Note we have established that BC is a candidate key. We can arrive at the following:
    i. We loop through each functional dependency in F_c:
        1. R1 = (A, E) - In 3NF since A is a primary key
        2. R2 = (C, A) - In 3NF since C is a primary key
        3. R3 = (A, B, D) - In 3NF since AB is a primary key
        4. R4 = (D, G, E) - In 3NF since DG is a primary key
    ii. At this point, no schema contains the candidate key BC, so let us add another relation-schema with the candidate key:
        1. R5 = (B, C) - In 3NF since BC is a candidate key
    iii. So, we arrive at the following 3NF schema for R:
        1. R1 = (A, E)
        2. R2 = (C, A)
        3. R3 = (A, B, D)
        4. R4 = (D, G, E)
        5. R5 = (B, C)

---

6. Given schema R(*course_id, section_id, dept, units, course_level, instructor_id, term, year, meet_time, room, num_students*). Given the following functional dependencies:

- *{course_id}* ➔ *{dept, units, course_level}*
- *{course_id, section_id, term, year}* ➔ *{meet_time, room, num_students, instructor_id}*
- *{room, meet_time, term, year}* ➔ *{instructor_id, course_id, section_id}*

    a. We want to find all of candidate keys of R and prove that each is a superkey (no need to prove that they are candidate keys ~ *thank you for this*)

i. Let R(*course_id, section_id, dept, units, course_level, instructor_id, term, year, meet_time, room, num_students*) = R(C, S, D, U, L, I T, Y, M, R, N)
    1. {C} ➔ {DUL}
    2. {CSTY} ➔ {MRNI}
    3. {RMTY} ➔ {ICS}

ii. Compute (C)+:
    1. (C)+: C
    2. C ➔ DUL
        a. C ⊆ C, so (C)+: CDUL
    3. No further updates since CSTY and RMTY are not subsets of (C)+. So, C is not a candidate key.

iii. Compute (CSTY)+:
    1. (CSTY)+ = CSTY
    2. CSTY ➔ MRNI
        a. CSTY ⊆ CSTY, so (CSTY)+ = MRNICSTY
    3. C ➔ DUL
        a. C ⊆ MRNICSTY, so (CSTY)+ = MRNICSTYDUL = CSDULITYMRN
    4. R ⊆ (CSTY)+, so CSTY = (course_id, section_id, term, year) is a candidate key

iv. Compute (RMTY)+:
    1. (RMTY)+ = RMTY
    2. RMTY ➔ ICS
        a. RMTY ⊆ RMTY, so (RMTY)+ = RMTYICS
    3. C ➔ DUL
        a. C ⊆ RMTYICS, so (RMTY)+ = RMTYICSDUL
    4. CSTY ➔ MRNI
        a. CSTY ⊆ RMTYICSDUL, so (RMTY)+ RMTYICSDULN = CSDULITYMRN
    5. R ⊆ (RMTY)+, so RMTY = (room, meet_time, term, year) is a candidate key

b. We want to identify the two canonical covers for the above set of functional dependencies. We also want to identify the canonical cover that is the most appropriate.

i. Let us compute F_c^1:
    1. We can clearly determine that instructor_id is extraneous which allows us to arrive at the following possibility:
        a. *{course_id}* ➔ *{dept, units, course_level}*
        b. *{course_id, section_id, term, year}* ➔ *{meet_time, room, num_students, instructor_id}*
        c. *{room, meet_time, term, year}* ➔ *{course_id, section_id}*

ii. Let us compute F_c^2:

1. Again, we can clearly determine that instructor_id is extraneous which allows us to arrive at the following possibility:
   a. *{course_id}* ➔ *{dept, units, course_level}*
   b. *{course_id, section_id, term, year}* ➔ *{meet_time, room, num_students}*
   c. *{room, meet_time, term, year}* ➔ *{instructor_id, course_id, section_id}*

iii. In regards to which canonical cover is the most appropriate, it appears that F_c^1 is a better choice over F_c^2 since it seems more useful to know the instructor_id from information regarding a specific course and section itself (as opposed to getting the instructor_id from information regarding class/section time and location). If someone wanted to find a certain instructor_id, they would be more likely to "search" up the class rather than "searching" up the room and time.

c. We want to suggest both a normal form and a schema decomposition that would be the best in this situation.

i. Let us provide a normal form (3NF) starting with F_c^1. Note that we simply turn each functional dependency into its own relation-schema:

1. R1 = (course_id, dept, units, course_level)
   a. Primary key: course_id
2. R2 = (course_id, section_id, term, year, meet_time, room, num_students, instructor_id)
   a. Primary key: (course_id, section_id, term, year)
   b. Foreign key: course_id
   c. Candidate key: (room, meet_time, term, year)
3. So, we arrive at the following:
   a. R1 = (course_id, dept, units, course_level)
      i. R1 = (C, D, U, L)
   b. R2 = (course_id, section_id, term, year, meet_time, room, num_students, instructor_id)
      i. R2 = (C, S, T, Y, M, R, N, I)

ii. Let us provide a schema decomposition (BCNF) using F and R(*course_id, section_id, dept, units, course_level, instructor_id, term, year, meet_time, room, num_students*):

1. Set R1 = (course_id, dept, units, course_level)
   a. Primary key: course_id
      i. In BCNF
2. R2 = (R - {dept, units, course_level}) = (course_id, section_id, instructor_id, term, year, meet_time, room, num_students) =

(course_id, section_id, term, year, meet_time, room, num_students, instructor_id)

a. Primary key: (course_id, section_id, term, year)
b. Foreign key: course_id
c. Candidate key: (room, meet_time, term, year)
    i. In BCNF

3. So, we arrive at the following decomposition:
    a. R1 = (course_id, dept, units, course_level)
        i. R1 = (C, D, U, L)
    b. R2 = (course_id, section_id, term, year, meet_time, room, num_students, instructor_id)
        i. R2 = (C, S, T, Y, M, R, N, I)

iii. We can clearly see that both processes produced the same schemas. We can determine that these would be best for actual use in a course management system because these schemas would efficiently, correctly, and completely represent the data considering the known dependencies. We should note that, relative to other databases, this database would seemingly have a low number of records which provides another support for these schemas. Also, enforcing the necessary constraints associated with this representation would be quite practical. For example, it would be reasonably practical to enforce that each unique course_id maps to a section_id, term, and year, while also enforcing that the unique combination (course_id, section_id, term, year) would map to the combination of (meet_time, room, num_students, instructor_id). Overall, these schema definitions are very logical and efficient and would be able to correctly enforce the appropriate dependencies.

---

7. Given the following database and dependencies:
    a. emails(email_id, send_date, from_addr, to_addr, subject, email_body, attachment_name, attachment_body)
    b. email_id ➔ send_date, from_addr, subject, email_body
    c. email_id ➔➔ to_addr
    d. email_id, attachment_name ➔ attachment_body

Let us perform 4NF decomposition on emails:

- Start with 7.b:
    - R1 = (<u>email_id</u>, send_date, from_addr, subject, email_body)
        - Primary key: email_id

            - In 4NF since email_id is clearly a superkey (7.b dependency)
    - R2 = (R - {send_date, from_addr, subject, email_body}) = (email_id, to_addr, attachment_name, attachment_body)
        - Choose email_id ➔➔ to_addr (equivalent to email_id ➔ to_addr)
        - R2 = (email_id, to_addr)
            - Primary key: (email_id, to_addr)
            - Foreign key: email_id
                - In 4NF since email_id ➔➔ to_addr is a trivial multivalued dependency (7.c dependency)
    - R3 = (R - to_addr) = (email_id, attachment_name, attachment_body)
        - Primary key: (email_id, attachment_name)
        - Foreign key: email_id
            - In 4NF since email_id is clearly a superkey (7.d dependency)
- With the following redefinitions of our relation-schemas, we arrive at the following:
    - R1 = received_emails(email_id, send_date, from_addr, subject, email_body)
    - R2 = email_recipients(email_id, to_addr)
    - R3 = attachments(email_id, attachment_name, attachment_body)