CENG 352 - Database Management Systems
2023-2
Written Homework 1

Anıl Eren Göçer
e2448397@ceng.metu.edu.tr

April 7, 2024

I. XML and XPath

i.

one
four
eight

ii.

one
two
three
four
five
six
seven
eight

iii.

<A>two< /A>
<B>three< /B>
<A>five< /A>
<B>six< /B>
<B>seven< /B>

iv.

two
three
five
six
seven

v.

four
eight

vi.

Empty

vii.

one
two

II. Database Design

2.1. Keys, Superkeys

a)

$\{AC \longrightarrow D, BE \longrightarrow A, AE \longrightarrow D, BE \longrightarrow A, BE \longrightarrow D, CE \longrightarrow D, ABC \longrightarrow D, ABC \longrightarrow E, ABE \longrightarrow D, ACD \longrightarrow E, ACE \longrightarrow D, BCD \longrightarrow A, BCD \longrightarrow E, BCE \longrightarrow A, BCE \longrightarrow D, BDE \longrightarrow A, ABCD \longrightarrow E, ABCE \longrightarrow D, BCDE \longrightarrow A\}$

b)

Here are all the keys of R:

{A, B, C}

{B, C, D}

{B, C, E}

c)

Here are all the superkeys for R that are not keys:

{A, B, C, D}

{A, B, C, E}

{B, C, D, E}

{A, B, C, D, E}

2.2. BCNF Decomposition

a)

Observe that C, I and J is **not** included any of the functional dependencies, neither on left hand side nor on right hand side of the functional dependencies.

Now, let's find the following closures:

$$\{C, I, J, A, E\}^{+} = \{C, I, J, A, E, B, F, G, H, D\} = \mathrm{R}$$

To calculate this closure, we first used the functional dependency $AE \longrightarrow BFGH$. Then, we used $\{B \longrightarrow ADE\}$

Another closure is found as follows:

$$\{C, I, J, B\}^{+} = \{C, I, J, B, A, D, E, F, G, H\} = \mathrm{R}$$

Since $\{C, I, J, A, E\}^{+}$ and $\{C, I, J, B\}^{+}$ include all the columns in R, $\{C, I, J, A, E\}$ and $\{C, I, J, B\}$ are the keys of R.

b)

Remember the condition for being in BCNF:

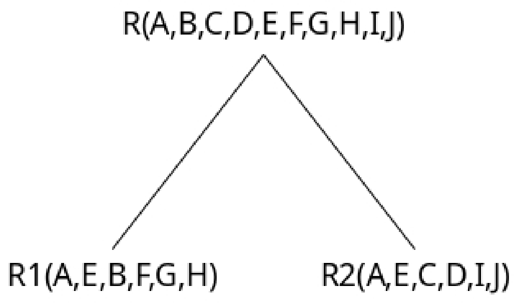
A relation R is in BCNF if:
If $A_1, ..., A_n \longrightarrow B$ is a non-trivial dependency in R, then $\{A_1, ..., A_n\}$ is a superkey of R.

Observe that $AE \longrightarrow BFGH$ violates this rule since AE is not a superkey. It is enough to find such example of functional dependencies. Therefore, R is not in BCNF.

c)

All of the functional dependencies violates BCNF rule. So, we can pick any of them.

Let's pick AE $\longrightarrow$ BFGH and decompose R(A,B,C,D,E,F,G,H,I,J) into R1 and R2.



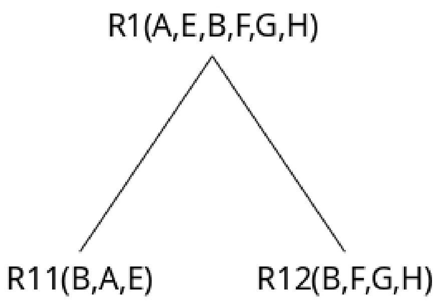
R2 does not have any functional dependency violating BCNF rule. Therefore, R2 is in BCNF and there is no need to decompose it further.

Now, let's take a closer look at R1. AE is the key of R1. Also, R1 has the following functional dependencies derived from functional dependencies of R:

$B \longrightarrow AE$
$F \longrightarrow G$
$G \longrightarrow F$

For example, the functional dependency $B \longrightarrow AE$ violates BCNF conditions since B is not a superkey.

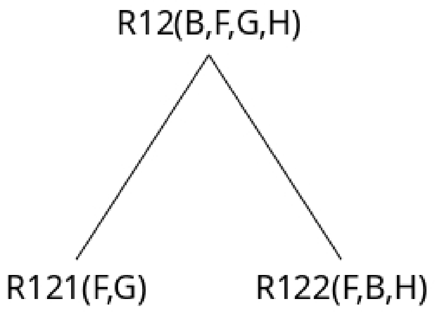
So, let's pick $B \longrightarrow AE$ and decompose R1 into R11 and R12.



R11 does not have any functional dependency violating BCNF rule, so R11 is in BCNF and there is no need to decompose it further.

Now, let's take a closer look at R12. It has some functional dependencies violating BCNF conditions. For example, one of them is $F \longrightarrow G$. F is not a superkey. Hence, it violates BCNF conditions.

Let's pick $F \longrightarrow G$ and decompose R12 into R121 and R122.

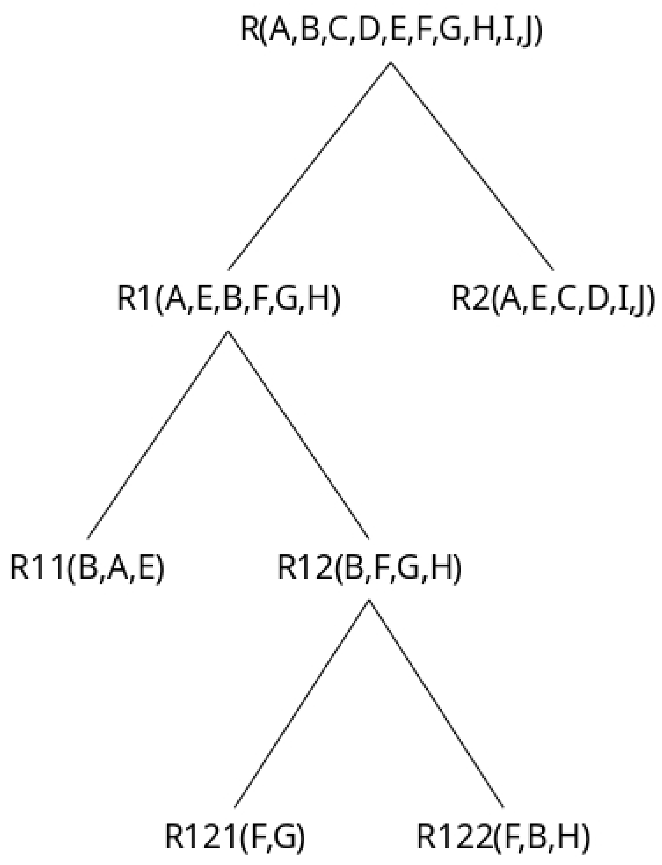


Now, both R121 and R122 are in BCNF because there is no functional dependency violating BCNF conditions in neither of them.

Overall, R is decomposed into followings:

R11(B,A,E)
R121(F,G)
R122(F,B,H)
R2(A,E,C,D,I,J)

To sum up, whole decomposition process can be summarized as follows:



d)

i) Let's find functional dependencies of resulting relations:

R11(B,A,E) has $F_1 = \{B \longrightarrow AE, AE \longrightarrow B\}$

R121(F,G) has $F_2 = \{F \longrightarrow G, G \longrightarrow F\}$

R122(F,B,G) has $F_3 = \{\}$

R2(A,E,C,D,I,J) has $F_4 = \{\}$

Since $AE \longrightarrow BFGH \notin F_1 \cup F_2 \cup F_4 \cup F_4$ as well as $B \longrightarrow ADE \notin F_1 \cup F_2 \cup F_4 \cup F_4$, the decomposition is **NOT** dependency-preserving.

ii) BCNF decomposition is always lossles. Therefore, my decomposition is lossless too.

We can confirm this by the fact that a decomposition is lossless if intersection of attributes of children is a key for either of them at each level of the decomposition tree, then the decomposition is lossless.

R121 and R122 has the intersection F and F is key for R121.
R11 and R12 has the intersection B and B is key for R11.
R1 and R2 has the intersection A,E and AE is key for R1.

Therefore, my decomposition is lossless.

2.3. 3NF Decomposition

a)

Step 1: Make RHS of each FD a single attribute.

We obtained the following functional dependencies:

$\{F \longrightarrow B, D \longrightarrow A, D \longrightarrow B, D \longrightarrow E, EF \longrightarrow D, C \longrightarrow E, B \longrightarrow A, B \longrightarrow E, BF \longrightarrow A, AE \longrightarrow F, CF \longrightarrow A, CF \longrightarrow D, CF \longrightarrow E\}$

Step 2: Eliminate redundant attributes from LHS.

We obtained the following functional dependencies:

$\{F \longrightarrow B, D \longrightarrow A, D \longrightarrow B, D \longrightarrow E, F \longrightarrow D, C \longrightarrow E, B \longrightarrow A, B \longrightarrow E, F \longrightarrow A, AE \longrightarrow F, F \longrightarrow A, F \longrightarrow D, F \longrightarrow E\}$

Step 3: Delete redundant functional dependencies.

We obtained the following functional dependencies:

$\{D \longrightarrow B, C \longrightarrow E, B \longrightarrow A, B \longrightarrow E, AE \longrightarrow F, F \longrightarrow D\}$

Therefore, following set is the **minimal cover**:

$\{D \longrightarrow B, C \longrightarrow E, B \longrightarrow A, B \longrightarrow E, AE \longrightarrow F, F \longrightarrow D\}$

b)

Step 1: We have computed minimal cover as follows:

$U = \{D \longrightarrow B, C \longrightarrow E, B \longrightarrow A, B \longrightarrow E, AE \longrightarrow F, F \longrightarrow D\}$

Step 2: Now, let's partition U:

$U_1 = \{D \longrightarrow B\}$
$U_2 = \{C \longrightarrow E\}$
$U_3 = \{B \longrightarrow A, B \longrightarrow E\}$
$U_4 = \{AE \longrightarrow F\}$
$U_5 = \{F \longrightarrow D\}$

Step 3: Now, let's form schemas $R_i = (X, FDs)$'s:

$R_1 = (DB; D \longrightarrow B)$
$R_2 = (CE; C \longrightarrow E)$
$R_3 = (ABE; B \longrightarrow A, B \longrightarrow E)$
$R_4 = (AEF; AE \longrightarrow F)$
$R_5 = (FD; F \longrightarrow D)$

Step 4: CA, CB, CD and CF are candidate keys for the relation R. However, none of the R_i has X part such that X is a superkey of R. Therefore, we have to add $R_O = (X_0,)$ where X_0 is a key of relation R. Let's take X_0 as CA:

$R_0 = (CA;)$
$R_1 = (DB; D \longrightarrow B)$
$R_2 = (CE; C \longrightarrow E)$
$R_3 = (ABE; B \longrightarrow A, B \longrightarrow E)$
$R_4 = (AEF; AE \longrightarrow F)$
$R_5 = (FD; F \longrightarrow D)$

As a result, we decomposed R into 3NF as follows:

$R_0(C, A)$
$R_1(D, B)$
$R_2(C, E)$
$R_3(A, B, E)$
$R_4(A, E, F)$
$R_5(F, D)$

c)

R_0 has key CA and it has no dependency violating BCNF conditions.

R_1 has key D and it has no dependency violating BCNF conditions because LHS of $D \longrightarrow B$ is a (super)key for R_1.

R_2 has key C and it has no dependency violating BCNF conditions because LHS of $C \longrightarrow E$ is a (super)key for R_2 .

R_3 has key B and it has no dependency violating BCNF conditions because LHS's of $B \longrightarrow A$ and $B \longrightarrow E$ are (super)keys for R_3.

R_4 has key AE and it has no dependency violating BCNF conditions because LHS of $AE \longrightarrow F$ is a (super)key for R_4.

R_5 has key F and it has no dependency violating BCNF conditions because LHS of $F \longrightarrow D$ is a (super)key for R_5 .

Hence, the decomposition is also in BCNF.

2.4. 4NF Decomposition

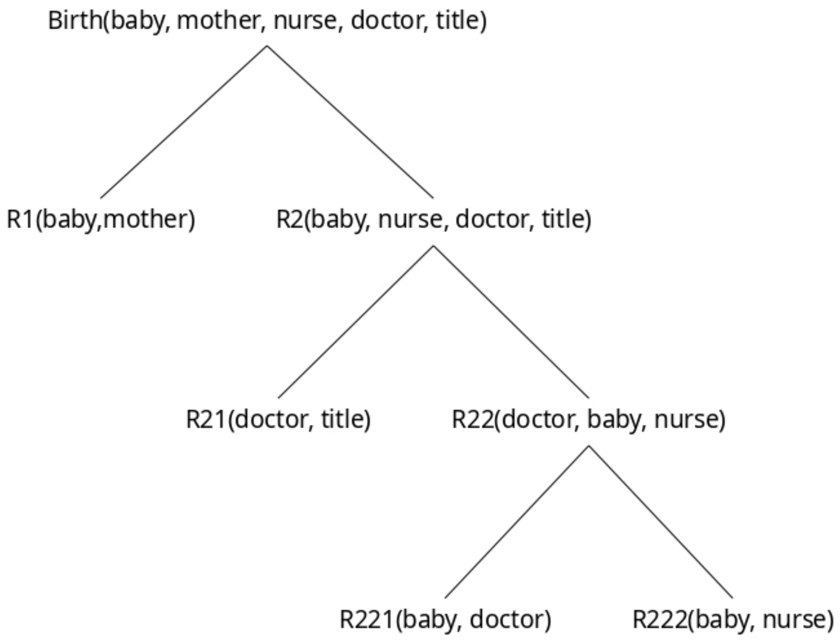
a)

Here are all functional and multi-valued dependencies ($->$ represents functional dependency and $->>$ represents multi-valued dependency):

doctor $->$ title
baby $->$ mother
mother $->>$ baby
baby $->>$ nurse
baby $->>$ doctor

Key of the relation is: {baby, nurse, doctor }

b)



Hence, we got the following relations as a result of decomposition:

R1(baby, mother)

R21(doctor, title)

R221(baby, doctor)

R222(baby, nurse)

III. Finding Functional Dependencies

I created the table and load its content as follows:

```
-- Create the table for question 3.
CREATE TABLE IF NOT EXISTS q3 (
    A INT,
    B VARCHAR(255),
    C VARCHAR(255),
    D REAL,
    E VARCHAR(255)
);

-- Load the csv file to the table.
\copy q3(A, B, C, D, E) FROM '/Users/eren/Desktop/CENG352_WHW1/dataset.csv' DELIMITER ','
    CSV HEADER;
```

a)

List of all FDs you identified and the corresponding SQL queries to discover them at the end of step 3.

Minimal Functional Dependencies Set and Successful Queries Corresponding to Them

Here, I will provide the smallest set including functional dependencies (like a minimal cover). They are the ones that cannot be derived from others. Since, there are lots of dependencies to be derived (which may cause me to put a lot of SQL code in this report), I have decided to present my solution in this way.

Here are these functional dependencies:

$A \longrightarrow E$
$B \longrightarrow D$
$D \longrightarrow B$
$AB \longrightarrow C$
$AD \longrightarrow C$

Here are the SQL queries that I used to find these dependencies:

```
-- Query for checking if A -> E is valid.
-- It returned true, so A -> E is a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.A) AS Result
    FROM (
        SELECT pairs.A AS A, COUNT(*) AS COUNT
        FROM (
            SELECT q.A AS A , q.E AS E
            FROM q3 q
            GROUP BY (q.A, q.E)
        ) AS pairs
        GROUP BY (pairs.A)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;
```

```
-- Query for checking if B -> D is valid.
-- It returned true, so B -> D is a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.B) AS Result
    FROM (
        SELECT pairs.B AS B, COUNT(*) AS COUNT
        FROM (
            SELECT q.B AS B , q.D AS D
            FROM q3 q
            GROUP BY (q.B, q.D)
        ) AS pairs
        GROUP BY (pairs.B)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;

-- Query for checking if D -> B is valid.
-- It returned true, so D -> B is a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.D) AS Result
    FROM (
        SELECT pairs.D AS D, COUNT(*) AS COUNT
        FROM (
            SELECT q.D AS D , q.B AS B
            FROM q3 q
            GROUP BY (q.D, q.B)
        ) AS pairs
        GROUP BY (pairs.D)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;

-- Query for checking if AB -> C is valid.
-- It returned true, so AB -> C is a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(*) AS Result
    FROM (
      SELECT A, B, COUNT(*)
      FROM (
            SELECT A, B, C
            FROM q3 q
          )
      GROUP BY (A,B)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;

-- Query for checking if AD -> C is valid.
-- It returned true, so AD -> C is a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(*) AS Result
    FROM (
      SELECT A, D, COUNT(*)
      FROM (
            SELECT A, D, C
            FROM q3 q
          )
      GROUP BY (A,D)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;
```

Some Extra Functional Dependencies and Successfull Queries Corresponding to Them

Here, I will provide some functional dependencies that can be derived from the functional dependency set presented in the previous chapter. Although there are many such examples, I will illustrate the most importants ones in which left hand side of the dependency is a key.

These are:

AB $\longrightarrow$ CDE
AD $\longrightarrow$ BCE

These imply $\{A, B\}^+ = \{A, B, C, D, E\}$ and $\{A, D\}^+ = \{A, B, C, D, E\}$. Hence, AB and AD are keys.

These findings can be verified by the following queries:

```
-- Query for checking if AB -> CDE is valid.
-- It returned true, so AB -> CDE is a functional dependency.
-- This also shows that AB is a key.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(*) AS Result
    FROM (
      SELECT q.A, q.B, COUNT(*)
      FROM q3 q
      GROUP BY (q.A, q.B)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;
```

```
-- Query for checking if AD -> BCE is valid.
-- It returned true, so AD -> BCE is a functional dependency.
-- This also shows that AD is a key.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(*) AS Result
    FROM (
      SELECT q.A, q.D, COUNT(*)
      FROM q3 q
      GROUP BY (q.A, q.D)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;
```

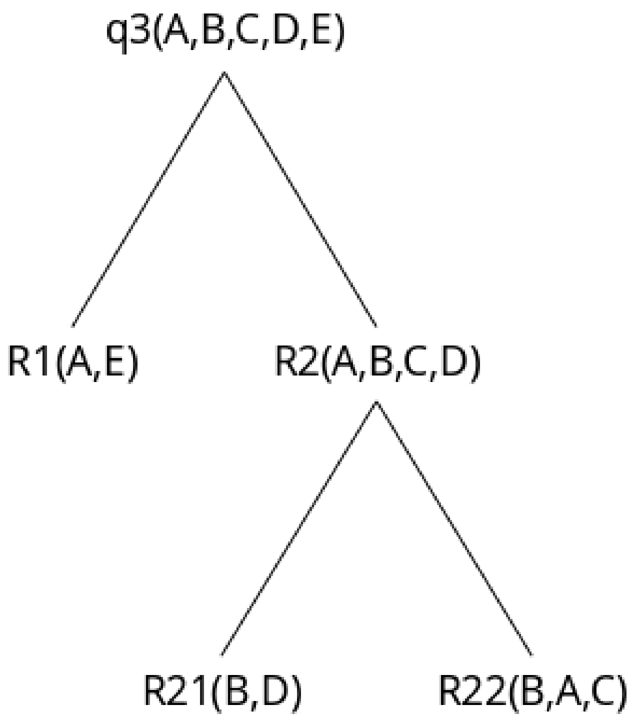
Unsuccessful Queries Verifying Some Other Attempts are Not Functional Dependencies

```
-- Query for checking if A -> B is valid.
-- It returned false, so A -> B is not a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.A) AS Result
    FROM (
        SELECT pairs.A AS A, COUNT(*) AS COUNT
        FROM (
            SELECT q.A AS A , q.B AS B
            FROM q3 q
            GROUP BY (q.A, q.B)
        ) AS pairs
        GROUP BY (pairs.A)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;


-- Query for checking if A -> C is valid.
-- It returned false, so A -> C is not a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.A) AS Result
    FROM (
        SELECT pairs.A AS A, COUNT(*) AS COUNT
        FROM (
            SELECT q.A AS A , q.C AS C
            FROM q3 q
            GROUP BY (q.A, q.C)
        ) AS pairs
        GROUP BY (pairs.A)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;

-- Query for checking if A -> D is valid.
-- It returned false, so A -> D is not a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.A) AS Result
    FROM (
        SELECT pairs.A AS A, COUNT(*) AS COUNT
        FROM (
            SELECT q.A AS A , q.D AS D
            FROM q3 q
            GROUP BY (q.A, q.D)
        ) AS pairs
        GROUP BY (pairs.A)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;

-- Query for checking if B -> C is valid.
-- It returned false, so B -> C is not a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.B) AS Result
    FROM (
        SELECT pairs.B AS B, COUNT(*) AS COUNT
        FROM (
            SELECT q.B AS B , q.C AS C
            FROM q3 q
            GROUP BY (q.B, q.C)
        ) AS pairs
        GROUP BY (pairs.B)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;

-- Query for checking if B -> E is valid.
-- It returned false, so B -> E is not a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.B) AS Result
    FROM (
        SELECT pairs.B AS B, COUNT(*) AS COUNT
        FROM (
            SELECT q.B AS B , q.E AS E
            FROM q3 q
            GROUP BY (q.B, q.E)
        ) AS pairs
        GROUP BY (pairs.B)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;

-- Query for checking if C -> D is valid.
-- It returned false, so C -> D is not a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.C) AS Result
    FROM (
        SELECT pairs.C AS C, COUNT(*) AS COUNT
        FROM (
            SELECT q.C AS C , q.D AS D
            FROM q3 q
            GROUP BY (q.C, q.D)
        ) AS pairs
        GROUP BY (pairs.C)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;

-- Query for checking if C -> E is valid.
-- It returned false, so C -> E is not a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.C) AS Result
    FROM (
        SELECT pairs.C AS C, COUNT(*) AS COUNT
        FROM (
            SELECT q.C AS C , q.E AS E
            FROM q3 q
            GROUP BY (q.C, q.E)
        ) AS pairs
        GROUP BY (pairs.C)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;

-- Query for checking if D -> E is valid.
-- It returned false, so D -> E is not a functional dependency.
SELECT
    CASE
        WHEN Result = 0 THEN 'true'
        ELSE 'false'
    END AS is_dependency
FROM (
    SELECT COUNT(pair_groups.D) AS Result
    FROM (
        SELECT pairs.D AS D, COUNT(*) AS COUNT
        FROM (
            SELECT q.D AS D , q.E AS E
            FROM q3 q
            GROUP BY (q.D, q.E)
        ) AS pairs
        GROUP BY (pairs.D)
    ) AS pair_groups
    WHERE pair_groups.COUNT != 1
) AS dependency_check;
```

b)

List of all SQL statements to create normalized tables.

Decomposition process can be summarized as follows:



At the end, we got following normalized tables:

R1(A,E)
R21(B,D)
R22(B,A,C)

Their corresponding SQL statements can be found as follows:

```
-- Normalized tables

-- This has functional dependency A -> E.
-- Since A is a key, it does not violate BCNF conditions.
CREATE TABLE IF NOT EXISTS R1(
    A INT,
    E VARCHAR(255),
    PRIMARY KEY (A)
);

-- This has functional dependency B -> D, D -> B.
-- Since B and D are (candidate)keys, they do not violate BCNF conditions.
-- I also made D UNIQUE because of the dependenct D -> B.
CREATE TABLE IF NOT EXISTS R21(
    B VARCHAR(255),
    D REAL UNIQUE,
    PRIMARY KEY (B)
);


-- This has functional dependency AB -> C.
-- Since AB is a (candidate)key, it does not violate BCNF conditions.
CREATE TABLE IF NOT EXISTS R22(
    B VARCHAR(255),
    A INT,
    C VARCHAR(255),
    PRIMARY KEY (A,B),
    FOREIGN KEY (A) REFERENCES R1(A),
    FOREIGN KEY (B) REFERENCES R21(B)
);
```

c)

List of all SQL statements that load the contents of the tables.

```
-- Content loading of normalized tables

INSERT INTO R1(A,E)
SELECT DISTINCT q.A, q.E
FROM q3 q;

INSERT INTO R21(B,D)
SELECT DISTINCT q.B, q.D
FROM q3 q;

INSERT INTO R22(B,A,C)
SELECT DISTINCT q.B, q.A, q.C
FROM q3 q;
```