Student Information

Name : Deniz Polat

ID : 2237790

1.1

a

Listing 1: XML

```
<X>
  <A>
    <A>one</A>
    <B>
      <B>two</B>
      <B>three</B>
    </B>
    <C>four</C>
  </A>
  <A>
    <B>
      <A>five</A>
      <A>six</A>
    </B>
    <C>seven</C>
  </A>
</X>
```

b

i four, seven

ii one, four, seven

iii seven

iv two, three

v two, three, five, six

vi two, three

1.2

```
{
    "Suppliers": [{
            "sid": "101",
            "sname": "Acme",
            "address": "123 Main",
            "parts": [{
                "pid": "92",
                "pname": "handle",
                "color": "Green",
                "cost": "5.21"
            }]
        },
        {
            "sid": "102",
            "sname": "Ace",
            "address": "456 Lake",
            "parts": [{
                    "pid": "92",
                    "pname": "handle",
                    "color": "Green",
                    "cost": "6.5"
                },
                {
                    "pid": "93",
                    "pname": "gasket",
                    "color": "Red",
                    "cost": "65.99"
                }]
        },
        {
            "sid": "103",
            "sname": "Figaro",
            "address": "678 First"
        }
    ]
}
```

Listing 1: JSON

This representation is redundant, because it duplicates some elements (for example (92, handle, Green, 5.21)). If we want to include parts that are not sold by any supplier, then we can go on the opposite direction and start from parts, list suppliers selling these parts and catalog prices that these suppliers sell this products. Yet, we will lose suppliers not selling any part this time (sid=103 for example). In order not to lose any data, we can create a json file such that we will start from suppliers, each containing a collection of parts they sell and the price which those parts are sold, and all possible parts that a supplier can sell. This design will be full of redundancies because Parts table's information will be kept in every Supplier once, which means lots of repetitions.

2.1

a

Here, I started with merging functional dependencies and finding nontrivial dependencies. As there are so many items, instead of finding the closure of every subset, I found subsets of left hand sides only.

$\{AuthorNo\}^+ = \{AuthorName, AuthorAddress, AuthorNo, AuthorEmail\}$
$\{AuthorEmail\}^+ = \{AuthorName, AuthorAddress, AuthorNo, AuthorEmail\}$
$\{PaperNo\}^+ = \{FirstAuthorNo, PaperTitle, PaperAbstract, PaperNo, PaperStatus\}$
$\{ReviewerNo\}^+ = \{ReviewerNo, ReviewerName, ReviewerEmail, ReviewerAddress\}$
$\{ReviewerEmail\}^+ = \{ReviewerNo, ReviewerName, ReviewerEmail, ReviewerAddress\}$
$\{ReviewerNo, PaperNo\}^+ = \{ReviewerNo, ReviewerName, ReviewerEmail, ReviewerAddress, PaperNo, FirstAuthorNo, PaperTitle, PaperAbstract, PaperStatus, Comments, ProgramComm, ReviewDate, Rating\}$

Now, we start applying algorithm by defining:

R(PaperNo, FirstAuthorNo, AuthorNo, AuthorName, AuthorEmail, AuthorAddress, PaperTitle, PaperAbstract, PaperStatus, ReviewerNo, ReviewerName,ReviewerEmail, Comments, ProgramComm,ReviewDate, Rating, ReviewerAddress) with functional dependencies:

- $AuthorNo \rightarrow AuthorName, AuthorAddress, AuthorEmail$
- $AuthorEmail \rightarrow AuthorName, AuthorAddress, AuthorNo$
- $PaperNo \rightarrow FirstAuthorNo, PaperTitle, PaperAbstract, PaperStatus$
- $ReviewerNo \rightarrow ReviewerName, ReviewerEmail, ReviewerAddress$
- $ReviewerEmail \rightarrow ReviewerNo, ReviewerName, ReviewerAddress$
- $ReviewerNo, PaperNo \rightarrow ReviewerName, ReviewerEmail, ReviewerAddress, FirstAuthorNo, PaperTitle, PaperAbstract, PaperStatus, Comments, ProgramComm, ReviewDate, Rating$

As R is not in BCNF, we decompose it into R_1 and R_2 as:
$R_1(AuthorNo, AuthorName, AuthorAddress, AuthorEmail)$
and
$R_2(AuthorNo, PaperNo, FirstAuthorNo, PaperTitle, PaperAbstract, PaperStatus, ReviewerNo, ReviewerName, ReviewerEmail, Commments, ProgramComm, ReviewDate, Rating, ReviewerAddress)$
as R_1 is in BCNF form (with superkey AuthorNo or AuthorEmail), we continue with investigating R_2. As R_2 is not in BCNF form (with a bad FD such as item 2), we split R_2 into R_3 and R_4 such as:
$R_3(PaperNo, FirstAuthorNo, PaperTitle, PaperAbstract, PaperStatus)$
and
$R_4(PaperNo, ReviewerNo, AuthorNo, ReviewerName, ReviewerEmail, Commments, ProgramComm, ReviewDate, Rating, ReviewerAddress)$

R_3 is in BCNF with key PaperNo, we investigate R_4.
We can easily see that R_4 is not in BCNF with a bad functional dependency item3, for instance. Therefore, as the next step, we split R_4 into R_5 and R_6 such as:
$R_5(ReviewerNo, ReviewerName, ReviewerEmail, ReviewerAddress)$
and
$R_6(ReviewerNo, PaperNo, AuthorNo, Comments, ProgramComm, ReviewDate, Rating)$

R_5 is in BCNF as the only FD relevant with R_5 are item4 and item5, and both have superkeys on LHS of the FDs. Therefore, we keep investigating R_6 and split it as it is not in BCNF such that:
$R_7(ReviewerNo, PaperNo, Comments, ProgramComments, ReviewDate, Rating)$
and
$R_8(ReviewerNo, PaperNo, AuthorNo)$

R_7 is in BCNF with key (ReviewerNo, PaperNo). When we investigate R_8, we see that there are no bad FD's in it (no FDs at all, actually). Therefore, R_8 is also in BCNF. As there are no relation left which is not in BCNF, the algorithm ends here. We can now create our tables with relations R_1, R_3, R_5, R_7 and R_8 as:

Author(AuthorNo, AuthorName, AuthorAddress, AuthorEmail)
(AuthorEmail unique)

Paper(PaperNo, FirstAuthorNo, PaperTitle, PaperAbstract, PaperStatus)
By looking at the name, I assume that:
FirstAuthorNo REFERENCES Author(AuthorNo)
but it is not an outcome of BCNF algorithm but just an assumption based on name.

Reviewer(ReviewerNo, ReviewerName, ReviewerEmail, ReviewerAddress)
(unique RevieverEmail)

Review(ReviewerNo, PaperNo, Comments, ProgramComments, ReviewDate, Rating)
ReviewerNo REFERENCES Reviewer(ReviewerNo)
PaperNo REFERENCES Paper(PaperNo)

WhoReviewedWhich(ReviewerNo, PaperNo, AuthorNo)
ReviewerNo REFERENCES Reviewer(ReviewerNo)
PaperNo REFERENCES Paper(PaperNo)
AuthorNo REFERENCES Author(AuthorNo).

b

With the nature of BCNF, any relation having BCNF is lossless (as intersection of 2 splitted relations is key of either of the clusters). Also, in my tables, the only common attributes between any 2 tables are keys. Thats why join of any tables in my database will be lossless.
Moreover, there is no FDs splitted in my new tables, every relationship exists untouched as it was in Conference table. None of the FD's are removed or lost. Therefore, my tables preserve dependency.

2.2

a

We first split RHS such that each FDs RHS will be one attribute:
AC $\rightarrow$ B
AC $\rightarrow$ G
AC $\rightarrow$ H
D $\rightarrow$ E
G $\rightarrow$ B
E $\rightarrow$ F
E $\rightarrow$ K
FD $\rightarrow$ K
ADF $\rightarrow$ C
H $\rightarrow$ B
H $\rightarrow$ G
H $\rightarrow$ H

As the second step, we remove items from LHS which are unnecessary (i.e. if FD can survive when we remove an item from LHS, then we do not need it). Then FDs become:

AC $\rightarrow$ B
AC $\rightarrow$ G
AC $\rightarrow$ H
D $\rightarrow$ E
G $\rightarrow$ B
E $\rightarrow$ F
E $\rightarrow$ K
D $\rightarrow$ K (*)
ADF $\rightarrow$ C
H $\rightarrow$ B
H $\rightarrow$ G
H $\rightarrow$ H

(*) Here, we can follow the steps D $\rightarrow$ E $\rightarrow$ K and obtain K without help of F, so we do not need F in this relation.

As the third step, we remove FDs which we do not need (i.e. if we do not lose any dependency when we remove one FD, then we do not need that FD so we could remove it from relation.). After this step, FDs become:
AC $\rightarrow$ H
D $\rightarrow$ E
G $\rightarrow$ B
E $\rightarrow$ F
E $\rightarrow$ K
ADF $\rightarrow$ C
H $\rightarrow$ G

Here are the explanations why we removed some FDs:
AC $\rightarrow$ B – Could be obtained as AC $\rightarrow$ H $\rightarrow$ G $\rightarrow$ B
AC $\rightarrow$ G – Could be obtained as AC $\rightarrow$ H $\rightarrow$ G
D $\rightarrow$ K – Could be obtained as D $\rightarrow$ E $\rightarrow$ K
H $\rightarrow$ B – Could be obtained as H $\rightarrow$ G $\rightarrow$ B
This becomes our minimal cover.

b

For decomposition, we first merge FDs having the same LHS from the minimal cover:
AC $\rightarrow$ H
D $\rightarrow$ E
G $\rightarrow$ B
E $\rightarrow$ FK
ADF $\rightarrow$ C
H $\rightarrow$ G

Then, we check whether any FDs include key of original relation. When we look at FDs and their closures, we see that there is no such LHS covering all attributes of original relation, so key of original relation does not occur in LHS of FDs. In order to preserve lossness, we need to add key of original relation to our tables. For that purpose, we find a key of original relation, which could be ACDH.
Now, we see that in 3NF decomposition, we have 7 different relations (tables) and their FDs:
$R_1(ACH)$, AC $\rightarrow$ H
$R_2(DE)$, D $\rightarrow$ E
$R_3(GB)$, G $\rightarrow$ B
$R_4(EFK)$, E $\rightarrow$ FK
$R_5(ACDF)$, ADF $\rightarrow$ C
$R_6(GH)$, H $\rightarrow$ G
$R_7(ACDH)$, {}

2.3

a

As sql queries are more than 1000 lines, I have attached them in the .zip file as FD-queries.sql. Yet, here I show the ones giving minimal cover:

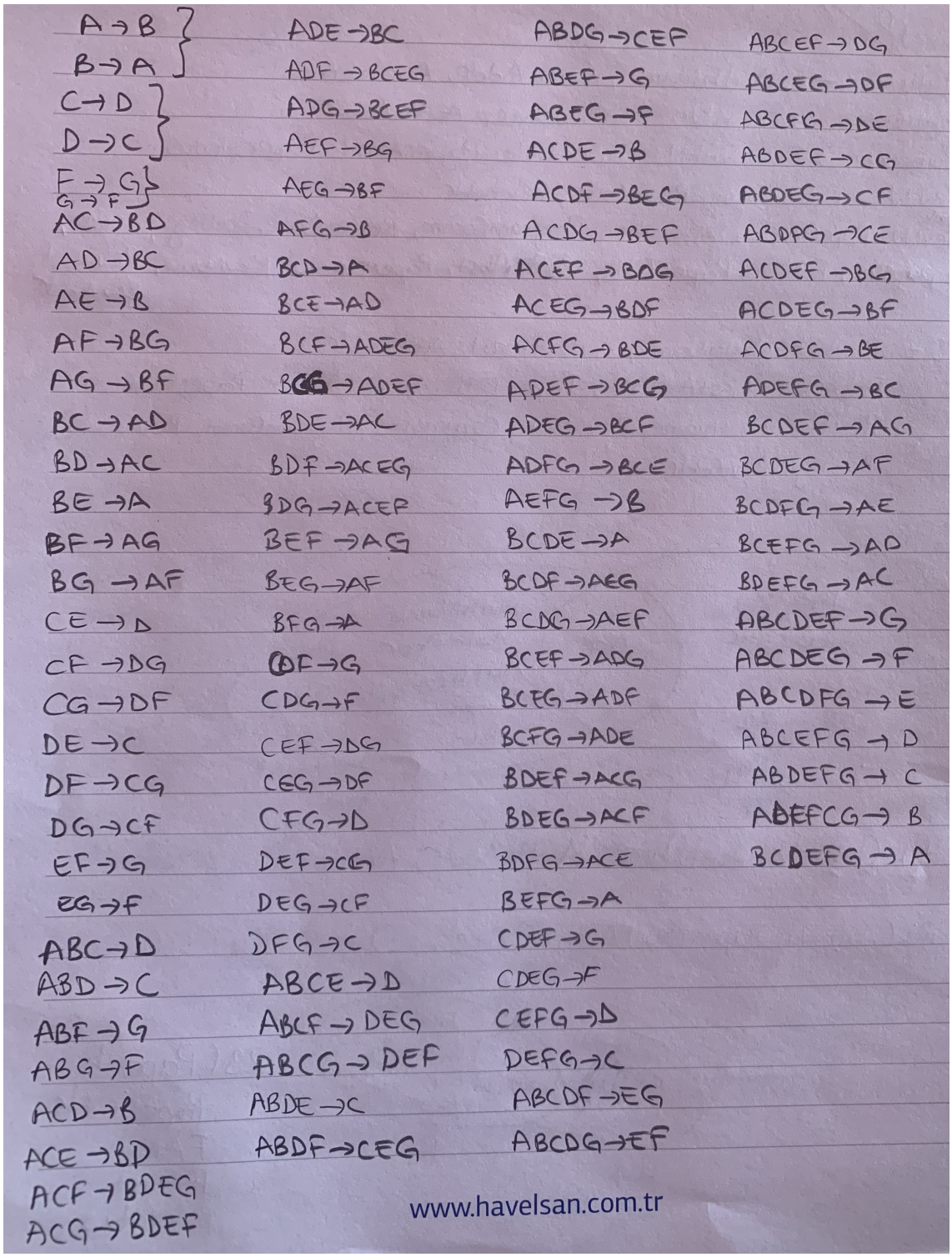
```
SELECT *
FROM Sample
WHERE NOT EXISTS (
    SELECT A
    FROM Sample
    GROUP BY A
    HAVING COUNT(DISTINCT(B)) != 1
    );
```

```
SELECT *
FROM Sample
WHERE NOT EXISTS (
    SELECT C
    FROM Sample
    GROUP BY C
    HAVING COUNT(DISTINCT(D)) != 1
    );
```

```
SELECT *
FROM Sample
WHERE NOT EXISTS (
    SELECT F
    FROM Sample
    GROUP BY F
    HAVING COUNT(DISTINCT(G)) != 1
    );
```

```
SELECT *
FROM Sample
WHERE NOT EXISTS (
    SELECT A,D,G
    FROM Sample
    GROUP BY A,D,G
    HAVING COUNT(DISTINCT(E)) != 1
    );
```

Here is the list of all nontrivial FDs.



non-trivial FDs

As I have done in 2.1.a, when minimal cover algorithm is applied, the FDs become:

A $\rightarrow B$
$B \rightarrow A$
$C \rightarrow D$
$D \rightarrow C$
$F \rightarrow G$
$G \rightarrow F$
$ADG \rightarrow E$

b

```
    CREATE TABLE AB(
    A VARCHAR PRIMARY KEY,
    B VARCHAR UNIQUE
);

CREATE TABLE CD(
    C INT PRIMARY KEY,
    D VARCHAR UNIQUE
);

CREATE TABLE FG(
    F INT PRIMARY KEY,
    G VARCHAR UNIQUE
);

CREATE TABLE ADGE(
    A VARCHAR REFERENCES AB(A),
    D VARCHAR REFERENCES CD(D),
    G VARCHAR REFERENCES FG(G),
    E INT,
    PRIMARY KEY(A,D,G)
);
```

c

```
    INSERT INTO AB
SELECT DISTINCT A,B FROM Sample;

INSERT INTO CD
SELECT DISTINCT C,D FROM Sample;

INSERT INTO FG
SELECT DISTINCT F,G FROM Sample;

INSERT INTO ADGE
SELECT DISTINCT A,D,G,E FROM Sample;
```

3

```
CREATE TABLE Customer (
CustNo VARCHAR PRIMARY KEY,
CustFirstName VARCHAR NOT NULL,
CustLastName VARCHAR NOT NULL,
CustCity VARCHAR NOT NULL,
CustState VARCHAR NOT NULL,
CustZip INT NOT NULL,
CustBal FLOAT NOT NULL
);
```

```
CREATE TABLE Employee (
EmpNo VARCHAR PRIMARY KEY,
EmpFirstName VARCHAR,
EmpLastName VARCHAR,
EmpPhone VARCHAR,
EmpEmail VARCHAR,
EmpStatus VARCHAR,
EmpSalary FLOAT,
supervisor VARCHAR DEFAULT '007' REFERENCES Employee(EmpNo) ON DELETE SET DEFAULT,
CHECK ( EmpEmail NOT LIKE '%' + Employee.EmpFirstName + '%' AND
        EmpEmail NOT LIKE '%' + Employee.EmpLastName + '%' )
);
```

```
CREATE TABLE Product (
ProdNo VARCHAR PRIMARY KEY,
ProdName VARCHAR,
ProdPrice FLOAT,
ProdShipDate DATE
);
```

```
CREATE TABLE "Order" (
OrdNo VARCHAR PRIMARY KEY,
CustNo VARCHAR NOT NULL,
EmpNo VARCHAR,
OrdDate DATE,
OrdName VARCHAR,
OrdCity VARCHAR,
OrdZip INT,
FOREIGN KEY(CustNo) REFERENCES Customer(CustNo) ON DELETE CASCADE,
FOREIGN KEY(EmpNo) REFERENCES Employee(EmpNo) ON DELETE SET NULL,
CHECK ( OrdName LIKE '%' + OrdCity + '%' )
);
```

```
CREATE TABLE Contains (
OrdNo VARCHAR,
ProdNo VARCHAR,
Qty VARCHAR,
FOREIGN KEY(OrdNo) REFERENCES "Order"(OrdNo) ON DELETE CASCADE,
FOREIGN KEY(ProdNo) REFERENCES Product(ProdNo) ON DELETE CASCADE,
PRIMARY KEY (OrdNo, ProdNo),
CHECK ( Qty >= 3 )
);
```

```
CREATE ASSERTION q3assertion CHECK (
NOT EXISTS(
SELECT *
FROM Contains
WHERE Qty < 30
)
);
```

```
CREATE TRIGGER q3trigger
    AFTER UPDATE OF EmpSalary ON Employee
    REFERENCING
        OLD ROW AS o
        NEW ROW AS n
    FOR EACH ROW
    WHEN (o.EmpSalary * 1.15 < n.EmpSalary)
    UPDATE Employee
    SET EmpStatus = 'Successful'
    WHERE EmpNo = o.EmpNo;
```