Assignment 2 - Solutions

CS44800 Staff

March 7, 2021

1 Functional Dependencies

Starting with \( A \rightarrow BC \),
we can conclude: \( A \rightarrow B \) and \( A \rightarrow C \).

Since \( A \rightarrow B \) and \( B \rightarrow D, A \rightarrow D \) (decomposition, transitive)
Since \( A \rightarrow CD \) and \( CD \rightarrow E, A \rightarrow E \) (union, decomposition, transitive)

Since \( A \rightarrow A \), we have \( A \rightarrow ABCDE \) (reflexive)
Since \( A \rightarrow ABCDE \) from the above steps (union)
Since \( E \rightarrow A \), \( E \rightarrow ABCDE \) (transitive)
Since \( CD \rightarrow E \), \( CD \rightarrow ABCDE \) (transitive)

Also, \( C \rightarrow C \), \( D \rightarrow D \), \( BD \rightarrow D \), etc.

Therefore, any functional dependency with \( A, E, BC, \) or \( CD \) on the left-hand side of the arrow is in \( F^+ \), no matter which other attributes appear in the FD. Allow \( * \) to represent any set of attributes in \( R \), then \( F^+ \) is \( BD \rightarrow B, BD \rightarrow D, C \rightarrow C, D \rightarrow D, BD \rightarrow BD, B \rightarrow D, B \rightarrow B B \rightarrow BD, \) and all FDs of the form \( A* \rightarrow \alpha, BC* \rightarrow \alpha, CD* \rightarrow \alpha, E* \rightarrow \alpha \) where \( \alpha \) is any subset of \( [A, B, C, D, E] \). The candidate keys are \( A, BC, CD, E \).

2 Functional Dependencies

The left side of each FD in \( F \) is unique. Also, none of the attributes in the left side or right side of any of the FDs is extraneous. Therefore the canonical over \( F_c \) is equal to \( F \)

3 Decomposition

3.1

A decomposition is lossless iff: \( R1 \cap R2 \rightarrow R1 \) or \( R1 \cap R2 \rightarrow R2 \). Since \( R1 \cap R2 = A, A \rightarrow ABC \), this is a lossless decomposition. One point is deducted if you only mention \( \Pi_{R1}(r) \bowtie \Pi_{R2}(r) = r \).

3.2

This is not a dependency-preserving decomposition because \( B \rightarrow D \) and \( CD \rightarrow E \) are not preserved in the decomposed tables. \( (F1 \cup F2)^+ \neq F^+ \)

3.3

Candidate keys for \( R \) is \( A, E, BC \) and \( CD \), thus \( B \rightarrow D \) violates BCNF and \( D \) is part of a superkey. \( R \) is in 3NF (FDs either have left-hand-side as superkey, or have right-hand-side as part of key).
Since $A \rightarrow ABCDE$, so $A$ is candidate key for $R1$. $R1$ is in BCNF (FDs have left-hand-side as superkey).

Since $A \rightarrow ABCDE, E \rightarrow ABCDE$, so $A$ and $E$ are both candidate keys for $R2$. $R2$ is in BCNF (FDs have left-hand-side as superkey).

Schema that is in BCNF is also in 3NF.
Points are deducted if you only mention $R1, R2$ are in 3NF.

3.4

One possible lossless decomposition is $(A, B, C, E)$ and $(B, D)$.

3.5

One possible lossless, dependency-preserving 3NF decomposition is $(A, B, C), (A, E), (C, D, E)$ and $(B, D)$.

4 **BCNF and Lossless Decomposition**

Trivial answer: the schema can be decomposed to schemas of only one attribute.

As long as the intersection of the decomposed schemas is not the key of any of the schema or the intersection is empty, the decomposition is not lossless. Let $R$ be a schema that is not in BCNF. Let $\alpha \rightarrow \beta$ be the FD that violates BCNF. We decompose $R$ into $(\alpha \cup \beta)$ and $(R - \beta - \alpha)$.

Points are deducted if the decomposition loses attributes or only an example is given or the algorithm is unclear.

5 **Multivalued Dependencies**

Counter Example:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>t2</td>
<td>a</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
</tr>
<tr>
<td>t3</td>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>t4</td>
<td>a</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
</tr>
</tbody>
</table>

6 **4NF**

books1(accessionno, isbn)
books2(isbn, author)
books3(isbn, title, publisher)
users1(userid, name, deptid)
users2(deptid, deptname)

Or

books1(accessionno, title)
books2(isbn, author)
books3(isbn, title, publisher)
users1(userid, name, deptid)
users2(deptid, deptname)

Notes: We need to decompose books2 (isbn, author, title, publisher) which is in 3NF since we have a multivalued dependency (isbn $\rightarrow$ author). Therefore, we get books2(isbn, author) and books3(isbn, title, publisher) which are in 4NF.
7 Open-Ended Design Problem

Given that this was an open-ended problem, there could be many different solutions. The following are two example solutions (Subsections 7.1 and 7.2).

7.1 Example Solution

7.1.1 List the attributes, and give a sample row in a “Universal Relation” view

Universal Relation

<table>
<thead>
<tr>
<th>Club</th>
<th>Officer Name</th>
<th>Member ID</th>
<th>Member Name</th>
<th>Attendee ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSWN</td>
<td>Vivian Carter</td>
<td>234555</td>
<td>Lucy Hernandez</td>
<td>234555</td>
</tr>
</tbody>
</table>

Universal Relation (Continued)

<table>
<thead>
<tr>
<th>Event</th>
<th>Date</th>
<th>Time</th>
<th>Building</th>
<th>Room</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puzzle Day</td>
<td>3/5/2021</td>
<td>6:00pm</td>
<td>LWSN</td>
<td>1142</td>
<td>100</td>
</tr>
</tbody>
</table>

7.1.2 List functional dependencies that should hold, and give brief description.

1. **ClubName** → **Description, OfficerName**
   - If we know the club, then we know the club’s description and officer (there is only 1 officer per club).

2. **Event** → **Date, Time, Building, Room, Club**
   - If we know the event, then we know the event’s date, time, room, and club hosting it.

3. **Room, Building** → **Capacity**
   - If we know the building and room, then we know the room’s capacity.

7.1.3 List any functional dependencies that seem obvious but should not hold, and explain.

1. **Room, Building** → **Event**
   - This functional dependency would not hold because a room can be used for many events.

2. **MemberID** → **ClubName**
   - This functional dependency would not hold because a student can join multiple clubs.

7.1.4 Give a lossless join, dependency-preserving decomposition based on your functional dependencies.

```
Clubs(ClubName, ClubDescription, OfficerName)
Members(MemberID, MemberName, ClubName)
Rooms(Building, Room, Capacity)
Events(Event, Date, Time, Building, Room, ClubName)
Attendees(AttendeeID, Event)
```
7.1.5 Give the SQL (including primary keys / unique constraints) to create the tables.

CREATE TABLE Clubs(
    ClubName VARCHAR(50),
    Description VARCHAR(100) NOT NULL,
    OfficerName VARCHAR(50) NOT NULL,
    PRIMARY KEY (ClubName)
);

CREATE TABLE Members(
    MemberID INTEGER,
    MemberName VARCHAR(50) NOT NULL,
    ClubName VARCHAR(50) NOT NULL,
    PRIMARY KEY (MemberID),
    FOREIGN KEY (ClubName) REFERENCES Clubs(ClubName)
);

CREATE TABLE Rooms(
    Building VARCHAR(20),
    Room VARCHAR(10),
    Capacity INTEGER NOT NULL,
    PRIMARY KEY (Building, Room)
);

CREATE TABLE Events(
    Event VARCHAR(100),
    EventDate DATE NOT NULL,
    EventTime TIME NOT NULL,
    Building VARCHAR(20) NOT NULL,
    Room VARCHAR(10) NOT NULL,
    PRIMARY KEY (Event),
    FOREIGN KEY (Building, Room) REFERENCES Rooms(Building, Room)
);

CREATE TABLE Attendees(
    AttendeeID INTEGER NOT NULL,
    Event VARCHAR(100) NOT NULL,
    CONSTRAINT AttendOnce UNIQUE (AttendeeID, Event),
    FOREIGN KEY (AttendeeID) REFERENCES Members(MemberID),
    FOREIGN KEY (Event) REFERENCES Events(Event)
);
7.2 Another Example Solution
(Courtesy of a student who wishes to remain anonymous)

7.2.1 List the attributes, and give a sample row in a “Universal Relation” view.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClubName</td>
<td>Name of the organization (unique)</td>
</tr>
<tr>
<td>Date</td>
<td>Event Date</td>
</tr>
<tr>
<td>Time</td>
<td>Start time of the event</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the event</td>
</tr>
<tr>
<td>Building</td>
<td>The building the event takes place in</td>
</tr>
<tr>
<td>Room</td>
<td>Room number of the event</td>
</tr>
<tr>
<td>Contact</td>
<td>The email address of the club representative in charge of the event</td>
</tr>
<tr>
<td>RSVPRequired</td>
<td>Whether or not attendees must RSVP</td>
</tr>
<tr>
<td>Capacity</td>
<td>The capacity of the room</td>
</tr>
<tr>
<td>Status</td>
<td>The event status. One of Approved, Pending, or Denied</td>
</tr>
<tr>
<td>President</td>
<td>The email address of the president for the club</td>
</tr>
<tr>
<td>Treasurer</td>
<td>The email address of the treasurer of the club</td>
</tr>
<tr>
<td>Advisor</td>
<td>The email address of the advisor of the club</td>
</tr>
<tr>
<td>Balance</td>
<td>The funds possessed by the organization</td>
</tr>
<tr>
<td>Table type</td>
<td>The type of tables in the room, could be tiered, trapezoid, etc.</td>
</tr>
<tr>
<td>Attendee First Name</td>
<td>First name of an event attendee</td>
</tr>
<tr>
<td>Attendee Last Name</td>
<td>Last name of an event attendee</td>
</tr>
<tr>
<td>Attendee Email Address</td>
<td>Email address of attendee. All attendees must be members</td>
</tr>
<tr>
<td>RSVP Date</td>
<td>The date the attendee RSVPed</td>
</tr>
<tr>
<td>Member First Name</td>
<td>Member first name</td>
</tr>
<tr>
<td>Member Last Name</td>
<td>Member last name</td>
</tr>
<tr>
<td>Member Email Address</td>
<td>Member email address</td>
</tr>
<tr>
<td>Member Join date</td>
<td>The date the person joined the club</td>
</tr>
<tr>
<td>Position</td>
<td>The position held by the member within the club</td>
</tr>
</tbody>
</table>
7.2.2 List functional dependencies that should hold, and give brief description.

1. ClubName, Date, Time → EventDescription, Building, Room, Contact, RSVPRequired, Status
   Each club can only have a single event at a given date and time.

2. Date, Time, Building, Room → ClubName, Contact, RSVPRequired, Status
   There can only be one event in a given room at a given time.

3. Building, Room → Capacity, TableType
   Any given room only has one capacity and table type.

4. ClubName → President, Treasurer, Advisor, Balance
   A given club can only have a single President, Treasurer, Advisor, and Balance.

5. Member EmailAddress → Member FirstName, Member LastName
   Email address uniquely identifies a person.

6. Attendee EmailAddress → Attendee FirstName, Attendee LastName
   Email address uniquely identifies a person.

7. Attendee EmailAddress, ClubName, Date, Time → RSVPDate
   An attendee can only RSVP once for a given event.

8. Member EmailAddress, ClubName → JoinDate, Position
   A given person can only join a club once and hold one position

7.2.3 List any functional dependencies that seem obvious but should not hold, and explain.

1. Member EmailAddress → ClubName, JoinDate, Position
   This does not hold because people can join more than one club.

2. Date, Time → EventDescription
   This does not hold because there can be multiple events at the same time, as long as they are associated with different clubs and take place in different locations.
7.2.4 Give a lossless join, dependency-preserving decomposition based on your functional dependencies.

The above attributes can be decomposed into the following relations:

- clubs(ClubName, PresidentEmail, TreasurerEmail, AdvisorEmail, Balance)
- rooms(Building, Room, Capacity, TableType)
- events(ClubName, Date, Time, Description, Building, Room, Contact, RSVPRequired, Status)
- people(EmailAddress, FirstName, LastName)
- members(EmailAddress, ClubName, JoinDate, Position)
- attendees(EmailAddress, ClubName, EventDate, EventTime, RSVPDate)

Most of the functional dependencies are enforced through primary key and foreign key relationships (see the below SQL). The rest can be easily checked using only one relation.

7.2.5 Give the SQL (including primary keys / unique constraints) to create the tables.

```sql
drop table if exists clubs;
create table clubs
(
    ClubName varchar(50),
    PresidentEmail varchar(50),
    TreasurerEmail varchar(50),
    AdvisorEmail varchar(50),
    Balance numeric(8, 2),
    primary key (ClubName),
    foreign key (PresidentEmail) references people(EmailAddress),
    foreign key (TreasurerEmail) references people(EmailAddress),
    foreign key (AdvisorEmail) references people(EmailAddress)
);

drop table if exists rooms;
create table rooms(
    Building varchar(8),
    Room int,
    Capacity int check (Capacity > 0),
    TableType varchar(80),
    primary key (Building, Room)
);

drop table if exists events;
create table events(
    ClubName varchar(50),
    Date date,
    Time int check (Time >= 0 and Time < 1440),
    Description text,
    Building varchar(8),
    Room int,
    Contact varchar(50),
    RSVPRequired bit,
    Status varchar(10) check (Status in ('APPROVED', 'DENIED', 'PENDING')),
    primary key (ClubName, Date, Time),
    foreign key (ClubName) references clubs(ClubName),
    ...
foreign key (Building, Room) references rooms(Building, Room),
foreign key (ClubName, Contact) references members(ClubName, EmailAddress)
);

drop table if exists people;
create table people(
    EmailAddress varchar(50),
    FirstName varchar(50),
    LastName varchar(50),
    primary key (EmailAddress)
);

drop table if exists members;
create table members(
    EmailAddress varchar(50),
    ClubName varchar(50),
    JoinDate date,
    Position varchar(50),
    primary key (EmailAddress, ClubName),
    foreign key (EmailAddress) references people(EmailAddress)
);

drop table if exists attendees;
create table attendees(
    ClubName varchar(50),
    EmailAddress varchar(50),
    EventDate date,
    EventTime int,
    RSVPDate date,
    primary key (ClubName, EmailAddress, EventDate, EventTime),
    foreign key (ClubName) references clubs(ClubName),
    foreign key (ClubName, EventDate, EventTime) references events(ClubName, Date, Time)
);