Explanation and Model Solution for Example Assignment: Decomposition and Normal Forms

COMPSCI 2DB3: Databases

Jelle Hellings Holly Koponen

Department of Computing and Software
McMaster University

Foreword

For Part 1, we have annotated the original assignment *description* to highlight all information of interest. Next, we provide the model solution to each question in this part. For Part 2, we provide only a detailed solution. As the solutions should be *explained* (readable), I have not included a rationale. I have added additional usefull information in remarks (that are not part of the solution).

Model Solution

Part 1: The analysis of a quick-event wizard for a local community

Rational:

As the first step, we analyze the description to find any information relevant for the dependencies holding on **event**. In this analysis, we use the following text annotations:

an entity	The highlighted piece of text resembled an entity-like object.
an attribute	The highlighted piece of text resembled an attribute of some entity-like object.
A RELATIONSHIP	The highlighted piece of text resembled a relationship between entity-like objects.
`a constraint`	The highlighted piece of text resembled a constraint on data.
~~not relevant~~	The highlighted piece of text is not relevant (e.g., context, application details, ...)

~~The local community leader came up with the idea of a *quick event wizard* via which users can quickly organize an event (e.g., invite users and order all necessary snacks, drinks, and other products). The community leader already contacted a consultant for an initial sketch of a table that can store all relevant data. The consultant came up with the following relational schema for that table:~~

event(id, user_id, date, inv_id, inv_confirmed, product, p_price, p_amount).

~~In this relational schema~~, each **event** has a `unique identifier` id.[1,2] Furthermore, the system keeps

[1] A quick look at the **event** table (and the following description) shows that the table stores information on *several* distinct entities and relationships. First, the table stores information on some conceptual entity **event**. Second, the table stores information on some conceptual entity **products**. Finally, the table stores relationships between **entity** and **users** (e.g., the relationship describing the organizer of an **event** and the relationship describing all invitees of an **event**) and between **entity** and **products**.

[2] The attribute *id* uniquely determines the attributes for the **event** entity.

track of the **user** that ORGANIZES the **event** (user_id) and the *date and time* of the event (date).[3] The system also keeps track of *all* INVITED **guests**.[4] In specific, the systems KEEPS TRACK, for each invited **guest** with identifier inv_id, whether that guest already confirmed its participation in the event (inv_confirmed).[5] Finally, the system keeps tracks of *all* **products**, e.g., snacks and drinks, that need to be ORDERED to organize the event (product),[6] the price of each of these products (p_price)[7] and the amount required of each product (p_amount).[8] ~~Next, an example of an instance of this relational schema:~~

id	user_id	date	inv_id	inv_confirmed	product	p_price	p_amount
1	1	Nov. 3, 3am	2	yes	chips	$2	4
1	1	Nov. 3, 3am	3	no	chips	$2	4
1	1	Nov. 3, 3am	2	yes	cola	$4	8
1	1	Nov. 3, 3am	3	no	cola	$4	8
2	1	Dec. 5, 7pm	2	no	chips	$2	2
2	1	Dec. 5, 7pm	2	no	cola	$4	1

~~The local community leader is not sure of the quality of this table, but has understood from the consultant that knowing the dependencies that hold on this table will help analyzing the quality of this table. Hence, the local community leader contacted you to determine all dependencies that hold on this table.~~[9]

Question

1. Provide a minimal cover of *all realistic* non-trivial functional dependencies that hold on the above relational schema. Argue, for each functional dependency, why this functional dependency hold.

   HINT: The local community leader only requires a *minimal cover*. Hence, there is no need for trivial functional dependencies and functional dependencies that can be derived from other functional dependencies.

   Solution:

   We have extracted the following functional dependencies from the description and verified that each of them hold on the example dataset:

[3] There is conceptually an many-to-exact-one relationship organizer between **event** and some **user** entity: each event is organized by exactly one user represented via the attribute *user_id*. Hence, we have "id ⟶ user_id". The attribute *date* is a proper attribute of the entity **event**. Hence, we have "id ⟶ date".

[4] There is conceptually a many-to-many relationship invited between **event** and some **user** entity: each event can have multiple invited guests and each guest can be invited for multiple events.

[5] The conceptual many-to-many relationship invited has an attribute *inv_id* that indicates whether the invited guest accepted its invitation: we have "X ⟶ inv_confirmed" with X the primary keys of **event** and some **user** entity. Hence, we have "id, inv_id ⟶ inv_confirmed".

[6] There is conceptually a many-to-many relationship ordered between **event** and **product**: for each event, one can order multiple products and each product can be ordered for multiple events.

[7] This is an attribute of the **product** entity (which has key *product*). Hence, we have "product ⟶ p_price".

[8] The conceptual many-to-many relationship ordered has an attribute *p_amount* that indicates the amount of a specific product that was ordered by a specific event: we have "X ⟶ p_amount" with X the primary keys of **event** and **product**. Hence, we have "id, product ⟶ p_amount".

[9] As mentioned before, the table **event** holds information on two distinct entities (**event** and **product**) and on two conceptual many-to-many relationships involving **event** (namely invited and ordered). Consequently, the invitees of an event are independent the products ordered for that event. We can express this *independence* via a *multivalued dependency* that states that the invitees and the ordered products are independent. Due to the Complementation inference rule, we can write this multivalued dependency as either "id, user_id, date ⟶⟶ inv_id, inv_confirmed" or as "id, user_id, date ⟶⟶ product, p_price, p_amount". As every multivalued dependency can be written as an equivalent join dependency, these multivalued dependencies are equivalent to the join dependency "$\bowtie\{X, Y\}$" with X = {id, user_id, date, inv_id, inv_confirmed} and Y = {id, user_id, date, product, p_price, p_amount}.

Functional Dependency	Reasoning
id ⟶ date	Each event (identified via *id*) has a unique time and date (stored in *date*).
id ⟶ user_id	Each event (identified via *id*) is related to exactly-one organizer (identified via *user_id*).
id, inv_id ⟶ inv_confirmed	For each user (identified via *inv_id*) invited for an event (identified via *id*), we keep track whether that user has accepted the invitation (stored in *inv_confirmed*).
product ⟶ p_price	Each product (identified via *product*) has a unique price (stored in *p_price*).
id, product ⟶ p_amount	For each product (identified via *product*) ordered for an event (identified via *id*), we keep track how many of that product was ordered (stored in *p_amount*).

2. Are there any other *non-trivial* dependencies that hold on this table? If so, provide an example of such a dependency and argue why this dependency holds.

   Hint: E.g., multivalued dependencies, inclusion dependencies, or join dependencies.

   Solution:
   The invitees of an event are independent the products ordered for that event. We can express this *independence* via either the multivalued dependency "id, user_id, date ⟶↠ inv_id, inv_confirmed", the multivalued dependency "id, user_id, date ⟶↠ product, p_price, p_amount", or the join dependency "$\bowtie\{X, Y\}$" with X = {id, user_id, date, inv_id, inv_confirmed} and Y = {id, user_id, date, product, p_price, p_amount}.

Part 2: Refinement of an order-table for a cinema chain

Our familiar local cinema chain owner has evaluated our initial design for the ticket sale and subscription system. Unfortunately, the cinema chain owner concluded that some details and functionality is missing. Hence, the cinema chain owner contacted another consultant to provide a basic design of a relational schema to store all relevant information. Now, the cinema chain owner wants a second opinion on this basic design from an expert. The consultant came up with the following relational schema to store all necessary information:

order(id, screening_time, product,
subscriber_id, sub_start, sub_duration,
film_id, film_length, film_score,
room_id, room_size, room_prop).

In this relational schema, the following order information will be stored:

i. The identifier *id* of the order, the start of the ordered screening (*screening_time*), and all *product*s ordered as part of the order (e.g., the base ticket, special seating, 3D glasses, and so on).

ii. The subscriber that placed the order *subscriber_id*. The system also keeps track of when the subscription of this subscriber started (*sub_start*) and how long the subscriber is subscribed (*sub_duration*).

iii. The film that is shown during the ordered screening (*film_id*) together with the length of the film (*film_length*, which is also the duration of the ordered screening). Furthermore, subscribers can review the film after visiting the screening (*film_score*) and each such score will have a value of *great*, *awful*, and *not-scored*.

iv. The room in which the ordered screening takes place (*room_id*) together with the size of the room (*room_size*) and all technical properties of the room (*room_prop*). Each room can have several technical properties, e.g., with possible values such as *3D*, *Dolby*, and *IMAX*.

Next, an example of an instance of this relational schema (we use shorthand notations for each attribute):

I	St	P	Si	Ss	Sd	Fi	Fl	Fs	Ri	Rs	Rp
1	Nov. 1, 1pm	ticket	1	Oct. 1	31	5	120	great	7	medium	3D
1	Nov. 1, 1pm	ticket	1	Oct. 1	31	5	120	great	7	medium	Dolby
1	Nov. 1, 1pm	3D	1	Oct. 1	31	5	120	great	7	medium	3D
1	Nov. 1, 1pm	3D	1	Oct. 1	31	5	120	great	7	medium	Dolby
2	Nov. 1, 1pm	ticket	2	Oct. 3	29	5	120	awful	7	medium	3D
2	Nov. 1, 1pm	ticket	2	Oct. 3	29	5	120	awful	7	medium	Dolby
2	Nov. 1, 1pm	3D	2	Oct. 3	29	5	120	awful	7	medium	3D
2	Nov. 1, 1pm	3D	2	Oct. 3	29	5	120	awful	7	medium	Dolby
3	Nov. 7, 2pm	ticket	2	Oct. 3	29	9	99	not-scored	3	large	IMAX
3	Nov. 7, 2pm	IMAX	2	Oct. 3	29	9	99	not-scored	3	large	IMAX
3	Nov. 7, 2pm	ticket	2	Oct. 3	29	9	99	not-scored	3	large	4D
3	Nov. 7, 2pm	IMAX	2	Oct. 3	29	9	99	not-scored	3	large	4D

According to the consultant, the set of attributes "*id, product, room_prop*" is a key, the following additional functional dependencies hold:

$$
\begin{aligned}
\text{id} &\longrightarrow \text{screening\_time, subscriber\_id, sub\_start, sub\_duration;}\\
\text{id} &\longrightarrow \text{film\_id, film\_length, film\_score, room\_id, room\_size;}\\
\text{subscriber\_id} &\longrightarrow \text{subscriber\_id, sub\_start, sub\_duration;}\\
\text{sub\_start} &\longrightarrow \text{sub\_duration;}\\
\text{sub\_duration} &\longrightarrow \text{sub\_start;}\\
\text{screening\_time, room\_id} &\longrightarrow \text{film\_id;}\\
\text{film\_id} &\longrightarrow \text{film\_length;}\\
\text{film\_id, subscriber\_id} &\longrightarrow \text{film\_score; and}\\
\text{room\_id} &\longrightarrow \text{room\_size.}
\end{aligned}
$$

Question

3. Is the relational schema **Order** in 3NF?

   If so, then explain why **Order** is in 3NF.

   Otherwise, decompose the schema using the 3NF Synthesis algorithm (Decompose-3NF) and document each step you make while applying the algorithm. Provide the functional dependencies that hold in each relational schema in your resulting decomposition (a minimal cover suffices). Explain whether this decomposition is lossless-join and whether it is dependency-preserving (with respect to the original functional dependencies). Finally, decompose the example dataset according to the relational schema obtained from the decomposition algorithm.

   Solution:

   The relational schema is not in 3NF. E.g., the functional dependency

   "subscriber_id $\longrightarrow$ subscriber_id, sub_start, sub_duration"

is a 3NF violation as this functional dependency is not trivial, the attribute "*subscriber_id*" is not a (super)key of the relational schema, and the attributes "*sub_start*" and "*sub_duration*" are not part of any key.

Next, we use the algorithm Decompose-3NF to put the relational schema into 3NF. To do so, we first determine a minimal cover of all the provided functional dependencies. We obtain:

$$\begin{aligned}\mathfrak{S}' = \{&\text{id} \longrightarrow \text{room\_id}, \text{id} \longrightarrow \text{screening\_time}, \text{id} \longrightarrow \text{subscriber\_id},\\ &\text{subscriber\_id} \longrightarrow \text{sub\_start},\\ &\text{sub\_duration} \longrightarrow \text{sub\_start},\\ &\text{sub\_start} \longrightarrow \text{sub\_duration},\\ &\text{screening\_time, room\_id} \longrightarrow \text{film\_id},\\ &\text{film\_id} \longrightarrow \text{film\_length},\\ &\text{film\_id, subscriber\_id} \longrightarrow \text{film\_score},\\ &\text{room\_id} \longrightarrow \text{room\_size}\}.\end{aligned}$$

The **for**-loop of Decompose-3NF will construct a relational schema for each $A \longrightarrow X$ in the minimal cover with attributes $A \cup B$, $B = \{Y \mid A \longrightarrow Y \in \mathfrak{S}'\}$. This will result in the following relational schemas:

- We get r_1(id, screening_time, subscriber_id, room_id) due to "id $\longrightarrow$ room_id", "id $\longrightarrow$ screening_time", and "id $\longrightarrow$ subscriber_id".
- We get r_2(subscriber_id, sub_start) due to "subscriber_id $\longrightarrow$ sub_start".
- We get r_3(sub_start, sub_duration) due to "sub_duration $\longrightarrow$ sub_start".
- We get r_4(sub_start, sub_duration) due to "sub_start $\longrightarrow$ sub_duration".
- We get r_5(screening_time, film_id, room_id) due to "screening_time, room_id $\longrightarrow$ film_id".
- We get r_6(film_id, film_length) due to "film_id $\longrightarrow$ film_length".
- We get r_7(subscriber_id, film_id, film_score) due to "film_id, subscriber_id $\longrightarrow$ film_score".
- We get r_8(room_id, room_size) due to "room_id $\longrightarrow$ room_size".

The **if**-condition of Decompose-3NF will construct a relational schema r_9(id, product, room_prop) for the key "*id, product, room_prop*". Finally, the **while**-loop will remove redundant relational schemas. In the above, only r_4 is redundant, as it is equivalent to r_3.

The following functional dependencies hold in each relational schema of the resulting decomposition:

Relation Scheme	Functional Dependencies
r_1(id, screening_time, subscriber_id, room_id)	"id $\longrightarrow$ screening_time, subscriber_id, room_id".
r_2(subscriber_id, sub_start)	"subscriber_id $\longrightarrow$ sub_start".
r_3(sub_start, sub_duration)	"sub_duration $\longrightarrow$ sub_start", "sub_start $\longrightarrow$ sub_duration".
r_5(screening_time, film_id, room_id)	"screening_time, room_id $\longrightarrow$ film_id".
r_6(film_id, film_length)	"film_id $\longrightarrow$ film_length".
r_7(subscriber_id, film_id, film_score)	"film_id, subscriber_id $\longrightarrow$ film_score".
r_8(room_id, room_size)	"room_id $\longrightarrow$ room_size".
r_9(id, product, room_prop)	

Remark. To find all functional dependencies that hold in a relational schema r, you can compute the closure of each set of attributes in that relational schema (with respect to the minimal cover $\mathfrak{S}'$), and only keep those functional dependencies that only use attributes of your relational schema r.

This decomposition is lossless-join and dependency-preserving (with respect to the original functional dependencies), as we have used the DECOMPOSE-3NF decomposition algorithm (which guarantees a lossless-join and dependency-preserving decomposition).

> **Remark.** We can also manually verify that the decomposition is dependency preserving: the set of functional dependencies $\mathfrak{S}''$ of our resulting relational schemas is equivalent to the earlier-computed minimal cover. To test whether a decomposition is dependency preserving in general, you check whether $\mathfrak{S}'^{+} = \mathfrak{S}''^{+}$.

Finally, we decompose the example dataset according to the relational schema obtained from the decomposition algorithm:

r_1	I	St	Si	Ri
	1	Nov. 1, 1pm	1	7
	2	Nov. 1, 1pm	2	7
	3	Nov. 7, 2pm	2	3

r_2	Si	Ss
	1	Oct. 1
	2	Oct. 3

r_3	Ss	Sd
	Oct. 1	31
	Oct. 3	29

r_5	St	Fi	Ri
	Nov. 1, 1pm	5	7
	Nov. 7, 2pm	9	3

r_6	Fi	Fl
	5	120
	9	99

r_7	Si	Fi	Fs
	1	5	great
	2	5	awful
	2	9	not-scored

r_8	Ri	Rs
	7	medium
	3	large

r_9	I	P	Rp
	1	ticket	3D
	1	ticket	Dolby
	1	3D	3D
	1	3D	Dolby
	2	ticket	3D
	2	ticket	Dolby
	2	3D	3D
	2	3D	Dolby
	3	ticket	IMAX
	3	IMAX	IMAX
	3	ticket	4D
	3	IMAX	4D

4. Is the relational schema **Order** in BCNF?

   If so, then explain why **Order** is in BCNF.

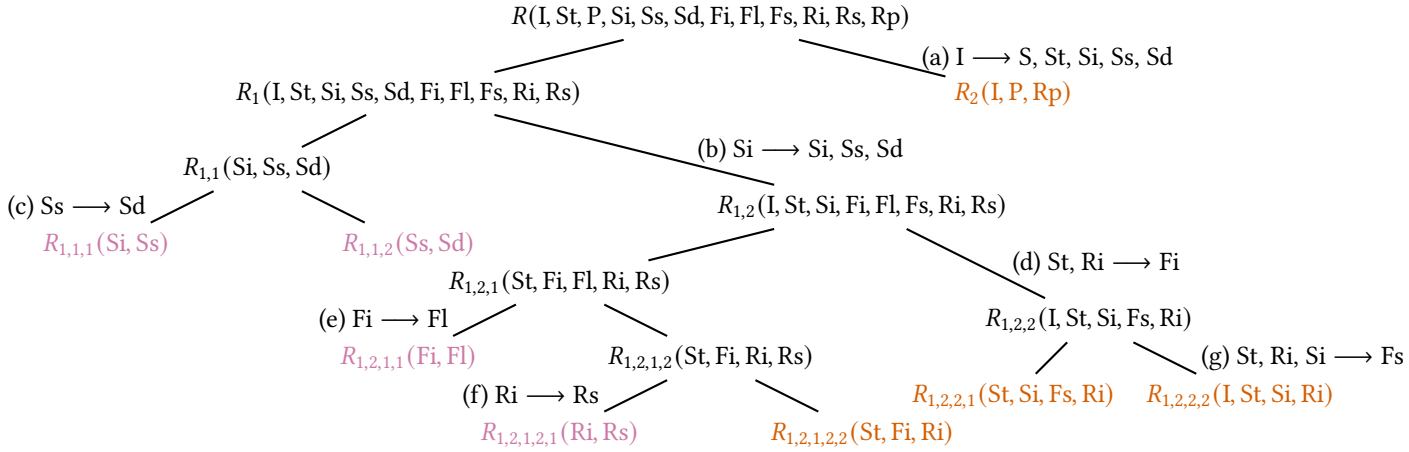
   Otherwise, decompose the schema using the BCNF Decomposition algorithm (DECOMPOSE-BCNF) and document each step you make while applying the algorithm. Provide the functional dependencies that hold in each relational schema in your resulting decomposition (a minimal cover suffices). Explain whether this decomposition is lossless-join and whether it is dependency-preserving (with respect to the original functional dependencies). Finally, decompose the example dataset according to the relational schema obtained from the decomposition algorithm.

   Solution:

   The relational schema is not in BCNF, as it is not in 3NF (alternatively, one can show that a BCNF

violation exists, e.g., the functional dependency "subscriber_id ⟶ subscriber_id, sub_start, sub_duration" is a BCNF violation as this functional dependency is not trivial and the attribute "*subscriber_id*" is not a (super)key of the relational schema).

Next, we use the algorithm DECOMPOSE-BCNF to put the relational schema into BCNF. In the below, we use $\rho(\dots)$ with $\rho(\dots)$ a relational schema to indicate that $\rho(\dots)$ is in BCNF as there are no functional dependencies in $\rho(\dots)$ that are BCNF violations and we use $\rho(\dots)$ with $\rho(\dots)$ a relational schema to indicate that $\rho(\dots)$ is in BCNF as it is a binary relationship. We perform the following steps:



(a) We begin with R(I, St, P, Si, Ss, Sd, Fi, Fl, Fs, Ri, Rs, Rp). The attribute "*id*" is not a key of R. Hence, all functional dependencies of the form id ⟶ X are BCNF violations. We split using

"id ⟶ screening_time, subscriber_id, sub_start, sub_duration"

and obtain R_1 (with all attributes in id^+ and R_2 with "*id*" and all remaining attributes of R:

$$R_1(\text{I, St, Si, Ss, Sd, Fi, Fl, Fs, Ri, Rs}); \qquad R_2(\text{I, P, Rp}).$$

The relational schema R_2 is in BCNF, as no non-trivial functional dependencies hold in R_2 (see also r_9 in the solution of Question 3).

(b) The relational schema R_1 is *not* in BCNF. E.g., the functional dependency

"subscriber_id ⟶ subscriber_id, sub_start, sub_duration"

is a BCNF violation. We split using this violation and obtain $R_{1,1}$ and $R_{1,2}$:

$$R_{1,1}(\text{Si, Ss, Sd}); \qquad R_{1,2}(\text{I, St, Si, Fi, Fl, Fs, Ri, Rs}).$$

(c) The relational schema $R_{1,1}$ is *not* in BCNF. E.g., the functional dependency

"sub_start ⟶ sub_duration"

is a BCNF violation. We split using this violation and obtain $R_{1,1,1}$ and $R_{1,1,2}$:

$$R_{1,1,1}(\text{Si, Ss}); \qquad R_{1,1,2}(\text{Ss, Sd}).$$

Both $R_{1,1,1}$ and $R_{1,1,2}$ are binary and, hence, are in BCNF.

(d) Next, we look at the relational schema $R_{1,2}$. This relational schema is also *not* in BCNF. E.g., the functional dependency

$$\text{“screening\_time, room\_id} \longrightarrow \text{film\_id”}$$

is a violation. We split using this violation and obtain $R_{1,2,1}$ and $R_{1,2,2}$:

$$R_{1,2,1}(\text{St, Fi, Fl, Ri, Rs}); \qquad R_{1,2,2}(\text{I, St, Si, Fs, Ri}).$$

(e) The relational schema $R_{1,2,1}$ is *not* in BCNF. E.g., the functional dependency

$$\text{“film\_id} \longrightarrow \text{film\_length”}$$

is a BCNF violation. We split using this violation and obtain $R_{1,2,1,1}$ and $R_{1,2,1,2}$:

$$R_{1,2,1,1}(\text{Fi, Fl}); \qquad R_{1,2,1,2}(\text{St, Fi, Ri, Rs}).$$

The relational schema $R_{1,2,1,1}$ is binary and, hence, is in BCNF.

(f) The relational schema $R_{1,2,1,2}$ is *not* in BCNF, however. E.g., the functional dependency

$$\text{“room\_id} \longrightarrow \text{room\_size”}$$

is a BCNF violation. We split using this violation and obtain $R_{1,2,1,2,1}$ and $R_{1,2,1,2,2}$:

$$R_{1,2,1,2,1}(\text{Ri, Rs}); \qquad R_{1,2,1,2,2}(\text{St, Fi, Ri}).$$

The relational schema $R_{1,2,1,2,1}$ is binary and, hence, is in BCNF. The relational schema $R_{1,2,1,2,2}$ is in BCNF as “screening_time, room_id $\longrightarrow$ film_id” is the only non-trivial functional dependency that holds in $R_{1,2,1,2,2}$ (see also r_5 in the solution of Question 3) and “screening_time, room_id” is a key of $R_{1,2,1,2,2}$.

(g) Next, we look at the relational schema $R_{1,2,2}$. The original set of functional dependencies does *not* have a violation for this schema. However, we can derive

$$\{\text{“screening\_time, room\_id} \longrightarrow \text{film\_id”}, \text{“film\_id, subscriber\_id} \longrightarrow \text{film\_score”}\} \models \text{“screening\_time, room\_id, subscriber\_id} \longrightarrow \text{film\_score”},$$

and “screening_time, room_id, subscriber_id $\longrightarrow$ film_score” is a BCNF violation. We split using this violation and obtain $R_{1,2,2,1}$ and $R_{1,2,2,2}$:

$$R_{1,2,2,1}(\text{St, Si, Fs, Ri}); \qquad R_{1,2,2,2}(\text{I, St, Si, Ri}).$$

(h) The relational schema $R_{1,2,2,1}$ is in BCNF as “screening_time, room_id, subscriber_id $\longrightarrow$ film_score” and “screening_time, room_id, subscriber_id” is the only non-trivial functional dependency that holds in $R_{1,2,2,1}$ and “screening_time, room_id, subscriber_id” is a key of $R_{1,2,2,1}$. Finally, the relational schema $R_{1,2,2,2}$ is in BCNF as “id $\longrightarrow$ screening_time, subscriber_id, room_id” is the only non-trivial functional dependency that holds in $R_{1,2,2,2}$ and “id” is a key of $R_{1,2,2,2}$.

The following functional dependencies hold in each relational schema of the resulting decomposition:

Relation Scheme	Functional Dependencies
R_2(id, product, room_prop)	
$R_{1,1,1}$(subscriber_id, sub_start)	"subscriber_id ⟶ sub_start".
$R_{1,1,2}$(sub_start, sub_duration)	"sub_duration ⟶ sub_start",
	"sub_start ⟶ sub_duration".
$R_{1,2,1,1}$(film_id, film_length)	"film_id ⟶ film_length".
$R_{1,2,1,2,1}$(room_id, room_size)	"room_id ⟶ room_size".
$R_{1,2,1,2,2}$(screening_time, film_id, room_id)	
	"screening_time, room_id ⟶ film_id".
$R_{1,2,2,1}$(screening_time, subscriber_id, film_score,room_id)	
	"screening_time, room_id, subscriber_id ⟶ film_score".
$R_{1,2,2,2}$(id, screening_time, subscriber_id, room_id)	
	"id ⟶ screening_time, subscriber_id, room_id".

This decomposition is lossless-join, as we have used the Decompose-BCNF decomposition algorithm (which guarantees a lossless-join decomposition). This decomposition *is not* dependency preserving: the functional dependency "film_id, subscriber_id ⟶ film_score" is not preserved as this functional dependency cannot be derived from the functional dependencies in the above table.

> **Remark.** The Decompose-BCNF decomposition algorithm *cannot* guarantee that the decomposition is dependency preserving, as there are cases in which the only lossless-join BCNF decompositions are not dependency preserving.

Finally, we decompose the example dataset according to the relational schema obtained from the decomposition algorithm:

R_2	I	P	Rp
	1	ticket	3D
	1	ticket	Dolby
	1	3D	3D
	1	3D	Dolby
	2	ticket	3D
	2	ticket	Dolby
	2	3D	3D
	2	3D	Dolby
	3	ticket	IMAX
	3	IMAX	IMAX
	3	ticket	4D
	3	IMAX	4D

$R_{1,1,1}$	Si	Ss
	1	Oct. 1
	2	Oct. 3

$R_{1,1,2}$	Ss	Sd
	Oct. 1	31
	Oct. 3	29

$R_{1,2,1,1}$	Fi	Fl
	5	120
	9	99

$R_{1,2,1,2,1}$	Ri	Rs
	7	medium
	3	large

$R_{1,2,1,2,2}$	St	Fi	Ri
	Nov. 1, 1pm	5	7
	Nov. 7, 2pm	9	3

$R_{1,2,2,1}$	St	Si	Fs	Ri
	Nov. 1, 1pm	1	great	7
	Nov. 1, 1pm	2	awful	7
	Nov. 7, 2pm	2	not-scored	3

$R_{1,2,2,2}$	I	St	Si	Ri
	1	Nov. 1, 1pm	1	7
	2	Nov. 1, 1pm	2	7
	3	Nov. 7, 2pm	2	3

5. According to the consultant, the following multivalued dependencies also hold:

$$\text{ID} \twoheadrightarrow \text{product}; \text{ and}$$
$$\text{ID} \twoheadrightarrow \text{room\_prop},$$

in which ID = {id, screening_time, subscriber_id, sub_start, sub_duration, film_id, film_length, film_score, room_id, room_size}. Is the relational schema **Order** in 4NF?

If so, then explain why **Order** is in 4NF.

Otherwise, decompose the schema using the 4NF Decomposition algorithm (Decompose-4NF) and document each step you make while applying the algorithm. Provide the functional dependencies that hold in each relational schema in your resulting decomposition (a minimal cover suffices). Explain whether this decomposition is lossless-join and whether it is dependency-preserving (with respect to the original functional dependencies). Finally, decompose the example dataset according to the relational schema obtained from the decomposition algorithm.

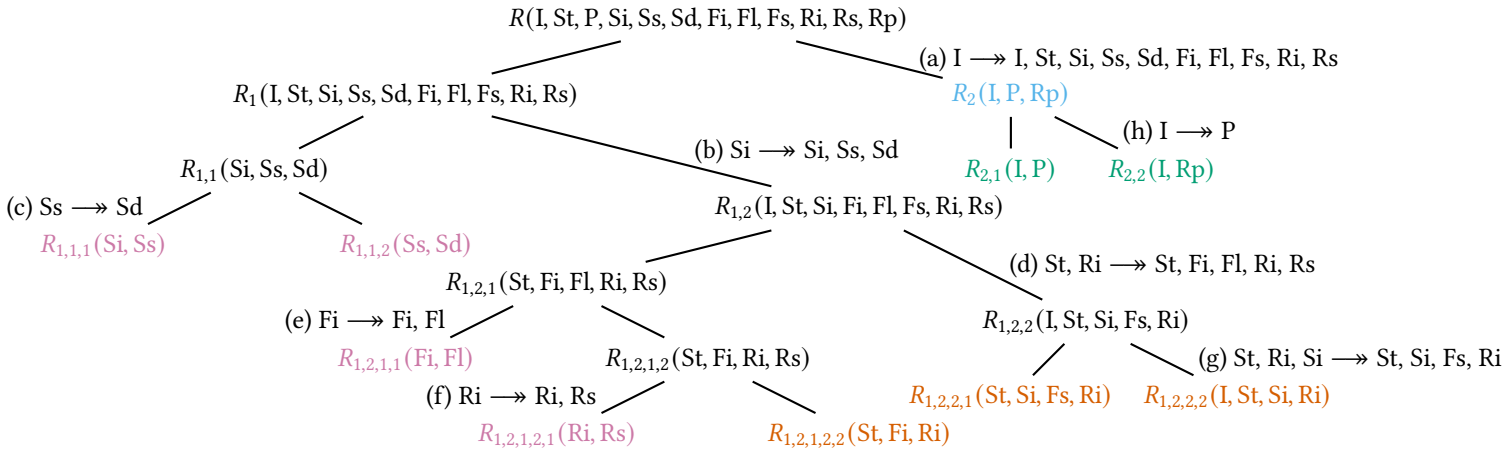
Solution:

The relational schema is not in 4NF, as it is not in BCNF (alternatively, one can show that a 4NF violation exists, e.g., due to Replication, the functional dependency "subscriber_id $\longrightarrow$ subscriber_id, sub_start,sub_duration" implies the multivalued dependency subscriber_id $\twoheadrightarrow$ subscriber_id, sub_start,sub_duration, and this multivalued dependency is a 4NF violation as this multivalued dependency is not trivial and the attribute "*subscriber_id*" is not a (super)key of the relational schema).

Next, we use the algorithm Decompose-4NF to put the relational schema into 4NF. We will use the BCNF decomposition as a starting point: we can do so, as every step with violation $\alpha \longrightarrow \beta$ in the algorithm Decompose-BCNF can be translated to a valid step in the algorithm Decompose-4NF. Namely, in the algorithm Decompose-4NF, we use the violation $\alpha \twoheadrightarrow \alpha^+$, in which α^+ is the attribute closure of α with respect to only the functional dependencies. Note that $\alpha \longrightarrow \alpha^+$ implies $\alpha \twoheadrightarrow \alpha^+$ due to Replication.

In the below, we use $\rho(\dots)$ with $\rho(\dots)$ a relational schema to indicate that $\rho(\dots)$ was an end-point in the BCNF decomposition but is *not* in 4NF, we use $\rho(\dots)$ with $\rho(\dots)$ a relational schema to indicate that $\rho(\dots)$ is in 4NF as there are no multivalued dependencies in $\rho(\dots)$ that are 4NF violations and we use $\rho(\dots)$ with $\rho(\dots)$ a relational schema to indicate that $\rho(\dots)$ is in 4NF as it is a binary relationship.

Hence, we start by applying all splitting steps we can re-use from the BCNF decomposition:



(a) The multivalued dependency

"id ⟶» id, screening_time, subscriber_id, sub_start, sub_duration, film_id, film_length, room_id, room_size",

which we obtain by applying Replication on id ⟶ id$^+$, is a 4NF violation. We split using this violation and obtain R_1 and R_2:

$$R_1(\text{I, St, Si, Ss, Sd, Fi, Fl, Fs, Ri, Rs}); \qquad R_2(\text{I, P, Rp}).$$

(b) The multivalued dependency

"subscriber_id ⟶» subscriber_id, sub_start, sub_duration",

which we obtain by applying Replication on subscriber_id ⟶ subscriber_id$^+$, is a 4NF violation. We split using this violation and obtain $R_{1,1}$ and $R_{1,2}$:

$$R_{1,1}(\text{Si, Ss, Sd}); \qquad R_{1,2}(\text{I, St, Si, Fi, Fl, Fs, Ri, Rs}).$$

(c) The multivalued dependency

"sub_start ⟶» sub_start, sub_duration",

which we obtain by applying Replication on sub_start ⟶ sub_start$^+$, is a 4NF violation. We split using this violation and obtain $R_{1,1,1}$ and $R_{1,1,2}$:

$$R_{1,1,1}(\text{Si, Ss}); \qquad R_{1,1,2}(\text{Ss, Sd}).$$

Both $R_{1,1,1}$ and $R_{1,1,2}$ are binary and, hence, are in 4NF.

(d) The multivalued dependency

"screening_time, room_id ⟶» $screening\_time, film\_id, film\_length, room\_id, room\_size$",

which we obtain by applying Replication on screening_time, room_id ⟶ (screening_time, room_id)$^+$, is a 4NF violation. We split using this violation and obtain $R_{1,2,1}$ and $R_{1,2,2}$:

$$R_{1,2,1}(\text{St, Fi, Fl, Ri, Rs}); \qquad R_{1,2,2}(\text{I, St, Si, Fs, Ri}).$$

(e) The multivalued dependency

$$\text{“film\_id} \longrightarrow\!\!\!\!\!\rightarrow \text{film\_id, film\_length”},$$

which we obtain by applying Replication on $\text{film\_id} \longrightarrow \text{film\_id})^+$, is a 4NF violation. We split using this violation and obtain $R_{1,2,1,1}$ and $R_{1,2,1,2}$:

$$R_{1,2,1,1}(\text{Fi, Fl}); \qquad R_{1,2,1,2}(\text{St, Fi, Ri, Rs}).$$

The relational schema $R_{1,2,1,1}$ is binary and, hence, is in 4NF.

(f) The multivalued dependency

$$\text{“room\_id} \longrightarrow\!\!\!\!\!\rightarrow \text{room\_id, room\_size”},$$

which we obtain by applying Replication on $\text{room\_id} \longrightarrow \text{room\_id}^+$, is a 4NF violation. We split using this violation and obtain $R_{1,2,1,2,1}$ and $R_{1,2,1,2,2}$:

$$R_{1,2,1,2,1}(\text{Ri, Rs}); \qquad R_{1,2,1,2,2}(\text{St, Fi, Ri}).$$

The relational schema $R_{1,2,1,2,1}$ is binary and, hence, is in 4NF.

(g) The multivalued dependency

$$\text{“screening\_time, room\_id, subscriber\_id} \longrightarrow\!\!\!\!\!\rightarrow \text{screening\_time, subscriber\_id, film\_score, room\_id”},$$

which we obtain by applying Replication on screening_time, room_id, subscriber_id $\longrightarrow$ (screening_time, room_id, subscriber_id)$^+$, is a 4NF violation. We split using this violation and obtain $R_{1,2,2,1}$ and $R_{1,2,2,2}$:

$$R_{1,2,2,1}(\text{St, Si, Fs, Ri}); \qquad R_{1,2,2,2}(\text{I, St, Si, Ri}).$$

Next, we evaluate whether each resulting non-binary relation scheme is in 4NF. Hence, we have the following additional steps:

(h) $R_2(\text{I, P, Rp})$. We have “id $\longrightarrow$ ID”. We apply Replication on “id $\longrightarrow$ ID” to obtain “$id \longrightarrow\!\!\!\!\!\rightarrow ID$”. Next, we apply Transitivity on “$id \longrightarrow\!\!\!\!\!\rightarrow ID$” and “ID $\longrightarrow\!\!\!\!\!\rightarrow$ product” to obtain “id $\longrightarrow\!\!\!\!\!\rightarrow$ product”. As id is not a key of R_2, the multivalued dependency “id $\longrightarrow\!\!\!\!\!\rightarrow$ product” is a 4NF violation. We split using this violation and obtain $R_{2,1}$ and $R_{2,2}$:

$$R_{2,1}(\text{I, P}); \qquad R_{2,2}(\text{I, Rp}).$$

Both $R_{2,1}$ and $R_{2,2}$ are binary and, hence, are in 4NF.

(i) The introduced multivalued dependencies do not introduce new dependencies for the relational schemas $R_{1,2,1,2,2}(\text{St, Fi, Ri})$, $R_{1,2,2,1}(\text{St, Si, Fs, Ri})$, $R_{1,2,2,2}(\text{I, St, Si, Ri})$. Hence, these schemas are in 4NF.

The following functional dependencies hold in each relational schema of the resulting decomposition:

Relation Scheme	Functional Dependencies
$R_{2,1}$(id, product)	
$R_{2,2}$(id, room_prop)	
$R_{1,1,1}$(subscriber_id, sub_start)	"subscriber_id ⟶ sub_start".
$R_{1,1,2}$(sub_start, sub_duration)	"sub_duration ⟶ sub_start",
	"sub_start ⟶ sub_duration".
$R_{1,2,1,1}$(film_id, film_length)	"film_id ⟶ film_length".
$R_{1,2,1,2,1}$(room_id, room_size)	"room_id ⟶ room_size".
$R_{1,2,1,2,2}$(screening_time, film_id, room_id)	
	"screening_time, room_id ⟶ film_id".
$R_{1,2,2,1}$(screening_time, subscriber_id, film_score,room_id)	
	"screening_time, room_id, subscriber_id ⟶ film_score".
$R_{1,2,2,2}$(id, screening_time, subscriber_id, room_id)	
	"id ⟶ screening_time, subscriber_id, room_id".

We note that non-trivial multivalued dependencies hold in this decomposition (that cannot be derived from the above functional dependencies).

This decomposition is lossless-join, as we have used the DECOMPOSE-4NF decomposition algorithm (which guarantees a lossless-join decomposition). This decomposition *is not* dependency preserving: the two multivalued dependencies "id ⟶» product" and "id ⟶» room_prop" and the functional dependency "film_id, subscriber_id ⟶ film_score" are not preserved as these dependencies cannot be derived from the functional dependencies in the above table.

> **Remark.** The DECOMPOSE-4NF decomposition algorithm *cannot* guarantee that the decomposition is dependency preserving, as there are cases in which the only lossless-join 4NF decompositions are not dependency preserving.

Finally, we decompose the example dataset according to the relational schema obtained from the decomposition algorithm:

$R_{2,1}$	I	P
	1	ticket
	1	3D
	2	ticket
	2	3D
	3	ticket
	3	IMAX

$R_{2,2}$	I	Rp
	1	3D
	1	Dolby
	2	3D
	2	Dolby
	3	IMAX
	3	4D

$R_{1,1,1}$	Si	Ss
	1	Oct. 1
	2	Oct. 3

$R_{1,1,2}$	Ss	Sd
	Oct. 1	31
	Oct. 3	29

$R_{1,2,1,1}$	Fi	Fl
	5	120
	9	99

$R_{1,2,1,2,1}$	Ri	Rs
	7	medium
	3	large

$R_{1,2,1,2,2}$	St	Fi	Ri
	Nov. 1, 1pm	5	7
	Nov. 7, 2pm	9	3

$R_{1,2,2,1}$	St	Si	Fs	Ri
	Nov. 1, 1pm	1	great	7
	Nov. 1, 1pm	2	awful	7
	Nov. 7, 2pm	2	not-scored	3

$R_{1,2,2,2}$	I	St	Si	Ri
	1	Nov. 1, 1pm	1	7
	2	Nov. 1, 1pm	2	7
	3	Nov. 7, 2pm	2	3

6. Does any of the above three decompositions of **Order** resolve all design issues of **Order**? If so, explain which decomposition(s) resolve all design issues. Else, provide an example of a design issue that was not resolved by decomposition.

Solution:

Not all issues are resolved: the *sub_duration* (subscription duration) attribute can be derived from the *sub_start* (subscription start) attribute. Hence, the attribute *sub_duration* can be eliminated entirely.

Furthermore, the above decompositions store the room properties with the order identifier. It is likely that this information is unrelated to each other, and, hence, this information is better modeled via a separate relationship between “room_id” and “room_prop”.