Exercise 1

Consider the entity relationship diagram from exercise sheet 3:

Refine the relational schema that you created in sheet 3 from the ER-Diagram. Underline keys and find appropriate data types. As a reminder, here is the un-refined schema:

City : {[name : string, state : string]}  
Station : {[name : string, #platforms : integer]}  
Train : {[trainNo : integer, #wagons : integer]}

For the relationships in the model, we create the following relations:

located_in : {...}  
start : {...}  
end : {...}  
connects : {...}
Solution:

During refinement, we merge relations for binary relationships into relations for entities, if the relations have the same key and it was a 1:N, N:1 or 1:1 relationship in the ER-model. Note: A binary 1:N relationship can be merged into the entity with the N next to it.

Doing so we can merge the (4) relation into (2). (5) gets merged into (3). And same for the end relation, which also gets merged into train.

\[
\begin{align*}
(4) & \rightarrow (2), \\
(5) & \rightarrow (3), \\
(6) & \rightarrow (3)
\end{align*}
\]

Thus, we end up with the following schema:

- **City**: \{\[name : string\], \[state : string\]\}
- **Station**: \{\[name : string\], \[#platforms : integer\], \[cityName : string\], \[state : string\]\}
- **Train**: \{\[trainNo : integer\], \[#wagons : integer\], \[startStationName : string\], \[endStationName : string\]\}
- **connects**: \{\[trainNo : integer\], \[fromStationName : string\], \[toStationName : string\], \[departure : date\], \[arrival : date\]\}

In our model the train number is uniquely identifying a connection between two cities (possibly involving several stations). An ICE starting in Munich (startStationName) and going to Berlin (endStationName) has a unique train number. When the train returns it has a different train number. Therefore, in the connects relation, the (trainNo, fromStationName)-pair and the (trainNo, toStationName)-pair are both valid keys (as they are both uniquely identifying a tuple in the relation).

Exercise 2

For additional practice, consider the hospital example, again. This time take the entity relationship diagram and transform it into a relational schema. Then, optimize it by eliminating relations.

This is obviously a large example but practice is very helpful. However, if you want to save time, you could focus on the difficult parts: employs, works, consists_of, Doctors + has
Solution:

a) Create a relational schema

The un-refined translation yields the following relations for the entities in the model:

- Hospital : { [address : string, #beds : int] } (8)
- Department : { [address : string, name : string] } (9)
- Room : { [address : string, name : string, roomNo : int] } (10)
- Employee : { [id : int, salary : int] } (11)
- Nurse : { [id : int] } (12)
- Doctor : { [id : int, area : string] } (13)
- Shift : { [date : date, from : time, to : time] } (14)
For the relationships in the model, we create the following relations:

- **consists_of**: `{[address : string, departmentName : string]}` (15)
- **contains**: `{[address : string, departmentName : string, roomNo : int]}` (16)
- **employs**: `{[address : string, id : int]}` (17)
- **supervises**: `{[nurseId : int, doctorId : int]}` (18)
- **doctor_has**: `{[doctorId : int, address : string, departmentName : string, roomNo : int]}` (19)
- **runs**: `{[doctorId : int, address : string, name : string]}` (20)
- **works**: `{[employeeId : int, date : date, from : time, to : time, address : string, name : string]}` (21)

There are several alternative translation options:

1. The is_a relationship could have also been translated by merging the attributes of the Employee into the Nurse and Doctor relation:
   - Nurse : `{[id : int, salary : int]}
   - Doctor : `{[id : int, area : string, salary : int]}

2. In the 1:1 relation has between Doctor and Room we could have also chosen the key of the Room as a key.

**b) Refine the relational schema**

Next, we refine the relational schema by combining relations.

All binary relations with 1:1, 1:N, N:1 can be refined in the following way:

First, we can eliminate all relations that originate from weak relationships in the ER-model. In this case we do not have to add additional keys to the entity we merge them into because they already have this key because they are weak entities:

\[ \text{consists_of} \rightarrow \{ \text{contains} \}, \text{employs} \rightarrow \{ \text{supervises} \} \]

Next, we take care of the has relation between Doctor and Room. This is a 1:1 relation and can therefore be merged into Doctor or Room. We choose to merge it into room, as this requires us to only add one attribute to Room instead of four to Doctor:

\[ \text{doctor_has} \rightarrow \{ \text{runs} \} \]

Now, there is no binary relation left with a 1:1, 1:N or N:1 functionality. Therefore, we are done and end up with the following relational schema:

- **Hospital**: `{[address : string, #beds : int]}
- **Department**: `{[address : string, name : string]}
- **Room**: `{[address : string, name : string, roomNo : int, doctorId : int]}
- **Employee**: `{[id : int, name : string, roomNo : int]}
- **Nurse**: `{[id : int]}
- **Doctor**: `{[id : int, area : string]}
- **Shift**: `{[date : date, from : time, to : time]}`
For the relationships in the model, we create the following relations:

employs : { [address : string, id : int] }

supervises : { [nurseId : int, doctorId : int] }

runs : { [doctorId : int, address : string, name : string] }

works : { [employeeId : int, date : date, from : time, to : time, address : string, name : string] }