Transformation

The next step is the transformation to DDL commands. We started by making “CREATE TABLE” commands for each entity in the ER diagram. We gave every attribute the right type and made sure to add NOT NULL if the attribute is not allowed to be empty. Furthermore, we made the right attributes primary key and we connected the attributes that are foreign keys. The next step was to make sure that the database meets the requirements, we added different types of codes. The first requirements we added were:

- Each movie edition is connected to exactly one movie.
- Each hall is connected to exactly one branch.
- Each seat is connected to exactly one hall.
- Each customer is connected to exactly one person.
- Each employee is connected to exactly one person.
- Each contract is connected to exactly one employee.
- Each reservation has exactly one existing seat, customer and showing.
- The product stock is connected to an existing branch.
- The product stock only has products that exist in the product table.

We did this already partly by adding constraints that connect the foreign keys. This made sure that all these tuples are connected to one tuple of another table. Because the attributes can’t be multivalued, the entities are connected to exactly one attribute. By adding “ON DELETE CASCADE” at the end of these constraints, we made sure that when the connected tuple is deleted, that tuple is also deleted so that it can never be connected to a non-existing tuple.

We then started to focus on the next requirements of the database:

- There are not multiple people with the same email address.
- There are no multiple instances of the same person in the customer table.
- There are no multiple instances of the same person in the employee table.
- There are not two reservations on the same seat for the same showing.
- There are not multiple tuples with the same combination of product and branch in the product stock.

We made sure that the database meets these requirements to add “UNIQUE” to the attributes that may not have multiple values in the table.

After this we still had two requirements we had to add. For these requirements we needed triggers. The first requirement we made triggers for is to make sure that showings are not at the same time in the same hall. For this we added two triggers, one before updating the showing table, and one before inserting into the showing table. Each trigger checks if the new start time or end time is between the start and end time of another showing. Furthermore, it checks if the start time of the other showings are between the start and end time of the new showing. If one of these checks are true, the trigger returns an error. Another requirement we still have to meet is that the showing types have to correspond with the hall type the showing is in. For this requirement, we also made two triggers that will be activated before the showing table will be updated and before something is inserted to the

showing table. The triggers match the type of the movie edition that will be shown, and the type of the hall that the showing will be in and return an error when this is not the case.

The database now meets all requirements, but we still noticed a problem. By some attributes, it was possible to give values that do not make sense. This was the case by the email address attributes, the type attributes and the attributes where a country or language needs to be entered. To make sure that only right values can be entered we added “CHECK” constraints to these attributes. For the email address attributes we checked the format of the value and for the other attributes we checked if the value is in a list with options.

Finally we took a look at normalization, we concluded that most of our schemas were in 4NF. The only table that was not in 4NF is the movie table because each movie can have multiple genres and directors. To fix this, we splitted this table into three tables: Movie, Genre and Director. In the table Genre, we made the combination of Movie_id and Genre the primary key and we made sure that each tuple has a unique combination of these by adding a “UNIQUE” statement. By Director, we made the primary key the combination of Movie_id and Director and we also made sure that this combination is unique for each tuple by adding a “UNIQUE” statement. To make sure that both tables are connected to an existing movie, we made the Movie_id in both tables a foreign key that references the Movie_id in the Movie table and we added “ON DELETE CASCADE” to the command to make sure that the information is deleted when the movie is deleted.