Task 2, version 2

Datalagring, IV1351

Adrian Jonsson Sjödin
adriansj@kth.se

April 8, 2023

Contents

1 Introduction

The purpose of task 2 was to turn the conceptual model of the Sound Good Music School database developed in task 1, into a logical model. That is a model that we can use when creating our database in MySQL. The creation and reasoning behind this logical model is what will be covered in this report.

2 Method

To turn our conceptual model (see fig. 2.1) into a logical model we mostly followed the steps laid out in the lecture about logical and physical models.

First we copied our conceptual model into the logical model and then proceeded to delete all relationships so that we where only left with the tables, and after this we went through each table and made sure they where 1NF and no row contained duplicated data. This resulted in some new tables. After this we specified the type for each column and whether or not it should allow NULL.

Then the next step was to achieve 2NF and start creating the Primary Keys and draw the relationships between the tables, and consider which Foreign Keys can be allowed to be NULL. Drawing the relationships resulted in some tables being deleted.

The last step was to split up some tables a bit more to try and achieve 3NF.

Having the logical model done we created the database using Astah's export to SQL function. Unfortunately you can not directly use it since it gave quite a lot of errors, but it gave a good foundation to build upon.

Having this foundation to build upon we went through it from top to bottom while looking at the logical model to make sure everything is implemented correctly. During this process we did some changes to the logical model when discovering mistakes we had overlooked.

For writing the SQl scripts that created the databases and populated it with data MySQL Workbench and DbGate was used. The logical model was created in Astah Pro.

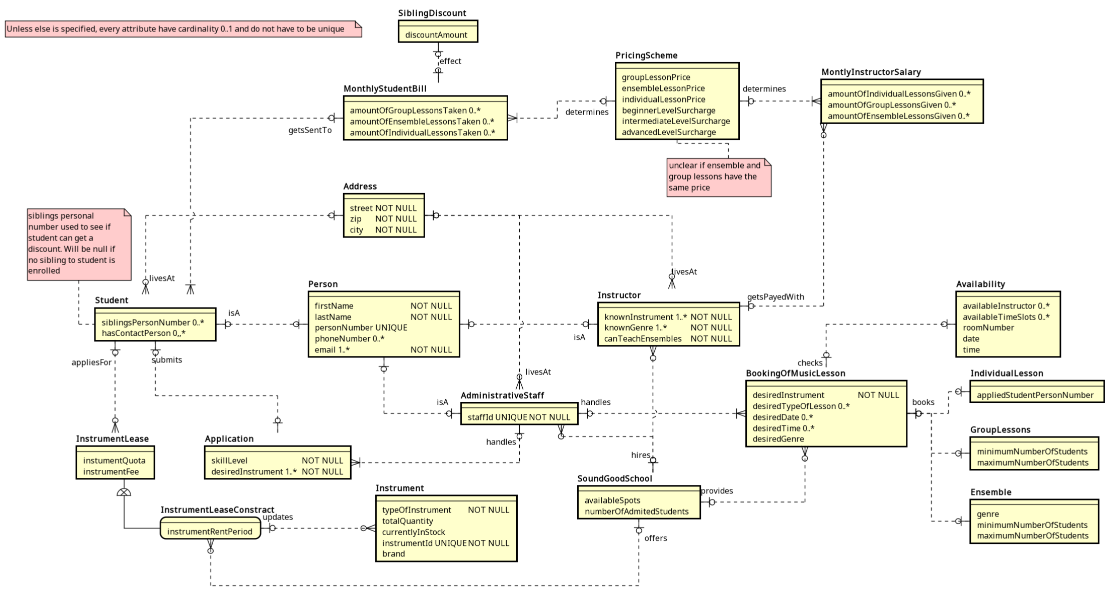
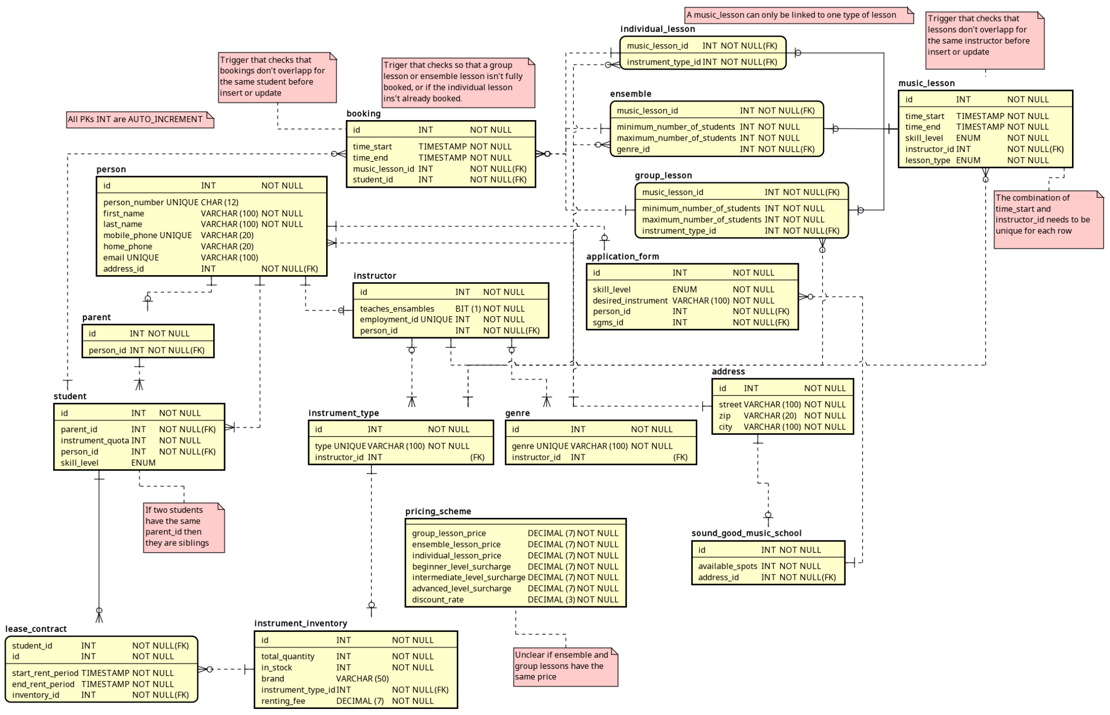


Figure 2.1: Conceptual Model for the SoundGood music school database

3 Result

In figure 3.1 we can see the finished logical model. I tried to have as little duplicated data as possible which is why I decided to have a table ”addresses” instead of having an address column in ”person” and ”sound_good_music_school”. They will instead have a Foreign Key to the address, which will keep the table smaller and save memory since we will not have multiple rows with the same values for people living at the same address. The same reasoning is behind the ”parent” and ”student” table with them having a Foreign Key to ”person”, which we can use to retrieve their data. The most important part of the model is arguably how lessons are booked. As can be seen in figure 3.1 we have a table ”music_lesson” which contains all the available lessons. As can be seen in the ”booking” table, a booking will have a foreign key to the corresponding music lesson and this key will also be the primary key for the corresponding type of music lesson. Furthermore we added some triggers to the database that makes sure a student can not have two bookings on the same time and day, and to make sure that lessons that are full can not be booked.

The SQL scripts can be found here: GitHub.



A music_lesson can only be linked to one type of lesson
Trigger that checks that lessons don't overlapp for the same instructor before insert or update
Trigger that checks that bookings don't overlapp for the same student before insert or update
Triger that checks so that a group lesson or ensemble lesson isn't fully booked, or if the individual lesson ins't already booked.
All PKs INT are AUTO_INCREMENT
The combination of time_start and instructor_id needs to be unique for each row
If two students have the same parent_id then they are siblings
Unclear if ensemble and group lessons have the same price
individual_lesson
music_lesson_id INT NOT NULL(FK)
instrument_type_id INT NOT NULL(FK)
ensemble
music_lesson_id INT NOT NULL(FK)
minimum_number_of_students INT NOT NULL
maximum_number_of_students INT NOT NULL
genre_id INT NOT NULL(FK)
group_lesson
music_lesson_id INT NOT NULL(FK)
minimum_number_of_students INT NOT NULL
maximum_number_of_students INT NOT NULL
instrument_type_id INT NOT NULL(FK)
music_lesson
id INT NOT NULL
time_start TIMESTAMP NOT NULL
time_end TIMESTAMP NOT NULL
skill_level ENUM NOT NULL
instructor_id INT NOT NULL(FK)
lesson_type ENUM NOT NULL
booking
id INT NOT NULL
time_start TIMESTAMP NOT NULL
time_end TIMESTAMP NOT NULL
music_lesson_id INT NOT NULL(FK)
student_id INT NOT NULL(FK)
person
id INT NOT NULL
person_number UNIQUE CHAR (12)
first_name VARCHAR (100) NOT NULL
last_name VARCHAR (100) NOT NULL
mobile_phone UNIQUE VARCHAR (20)
home_phone VARCHAR (20)
email UNIQUE VARCHAR (100)
address_id INT NOT NULL(FK)
application_form
id INT NOT NULL
skill_level ENUM NOT NULL
desired_instrument VARCHAR (100) NOT NULL
person_id INT NOT NULL(FK)
sgms_id INT NOT NULL(FK)
instructor
id INT NOT NULL
teaches_ensambles BIT (1) NOT NULL
employment_id UNIQUE INT NOT NULL
person_id INT NOT NULL(FK)
parent
id INT NOT NULL
person_id INT NOT NULL(FK)
address
id INT NOT NULL
street VARCHAR (100) NOT NULL
zip VARCHAR (20) NOT NULL
city VARCHAR (100) NOT NULL
student
id INT NOT NULL
parent_id INT NOT NULL(FK)
instrument_quota INT NOT NULL
person_id INT NOT NULL(FK)
skill_level ENUM
instrument_type
id INT NOT NULL
type UNIQUE VARCHAR (100) NOT NULL
instructor_id INT (FK)
genre
id INT NOT NULL
genre UNIQUE VARCHAR (100) NOT NULL
instructor_id INT (FK)
pricing_scheme
group_lesson_price DECIMAL (7) NOT NULL
ensemble_lesson_price DECIMAL (7) NOT NULL
individual_lesson_price DECIMAL (7) NOT NULL
beginner_level_surcharge DECIMAL (7) NOT NULL
intermediate_level_surcharge DECIMAL (7) NOT NULL
advanced_level_surcharge DECIMAL (7) NOT NULL
discount_rate DECIMAL (3) NOT NULL
sound_good_music_school
id INT NOT NULL
available_spots INT NOT NULL
address_id INT NOT NULL(FK)
lease_contract
student_id INT NOT NULL(FK)
id INT NOT NULL
start_rent_period TIMESTAMP NOT NULL
end_rent_period TIMESTAMP NOT NULL
inventory_id INT NOT NULL(FK)
instrument_inventory
id INT NOT NULL
total_quantity INT NOT NULL
in_stock INT NOT NULL
brand VARCHAR (50)
instrument_type_id INT NOT NULL(FK)
renting_fee DECIMAL (7) NOT NULL

Figure 3.1: The logical model for the SoundGood music school database

4 Discussion

While converting the Conceptual Model into a Logical Model I noticed quite a few flaws that needed correcting. For starters I had the administrative staff in the database, which is not a requirement since their roll was to use an application that interacts with the database. They handle the bookings and such but the task here was not to model a booking system but rather a database system.

I also added a table ”parent” since I realized having an attribute ”hasContactPerson” is not the best way to store data about a students parents. Instead ”student” now have a Foreign Key referencing ”parent” and ”parent” have a Foreign Key referencing ”person”. This will also help us calculate which students are siblings since they will have the same Foreign Key to parents.

Another big change is how instruments and the leasing of them is handled. I moved the ”instrumentQuota” into ”students” and ”instrumentFee” into ”instrument_inventory” thus removing the ”InstrumentLease” table seen in fig. 2.1. The ”instrumentLeaseContract” then became the ”lease_contract” and since the same student can rent multiple instrument we hade the make the Primary Key a Composite Key using the Foreign Key from ”students” and a Surrogate Key. So each row in this table would represent a lease contract. For the instruments I decided against assigning each single instrument its own serial number and instead went with the approach of just storing how many instrument of a certain type we have. This is of course something that would have to be discussed with the client, but doing it this way simplified the design.

Speaking of the instrument type, we also created a new table just for that since it helped reducing the duplication of data in the ”instructors” table, and we can use it in our ”instrument_inventory” table and the lesson tables. The reduction of duplicate data is also the reasoning behind the ”genres” table.

In fig. 2.1 we can see that we had two tables named ”BookingOfMusicLesson” and ”Availability”, these have been removed and the tables ”booking” and ”music_lesson” have been created instead. The ”booking” table is quite straightforward, it contains a foreign key to the music lesson, the time slot that they've booked and the id for the student taking it. The foreign key ”music_lesson_id” tells us which lesson in the ”music_lesson” table the student have booked and it is also the primary key in the three different lesson types tables. This tells us which lesson a student have booked and we can also see how many students have booked a group lesson or ensemble lesson by simply looking at the primary keys in for example the ensemble table and count gow many rows there are in bookings containing that key.

One final thing about the choice of data types and constraints. I used MySQL which is why I went for *CHAR* in the once case where I knew exactly how long a string should be. I also used *DECIMAL* since it makes it possible to define exactly how many decimals are allowed. For the constraints only NOT NULL and UNIQUE are indicated in fig. 3.1, but all *INT*s and *DECIMAL*s have the constraint that they have to be larger or equal to zero.