Project Report - Task 2

Data Storage Paradigms, IV1351

Rozhan Asadi - rozhana@kth.se

Fall term 2022

Introduction

Previously in the last task, a conceptual model was made for the sound good music school, which offers different types of music lessons to its students. The point of making the conceptual model was to be able to start our journey to make a database for the sound good music school in a way which is useful and efficient. In this task, we will be explaining how the second step of the journey goes. This step consists of making a physical model and then making the database based on that model. A physical model shows how the data will be implemented in detail which makes it easy to make the database in the real world. In a physical model, we will have entities, relations and attributes similar to a conceptual model. After having a physical model, we can clearly see the technical and the performance requirements of the database. This project and task were done by the writer of this report alone, Rozhan Asadi.

The changes that were needed

In the last SQL code that was submitted for seminar 2, the primary keys for the database were not automatically generated, and they were generated manually. The code was changed so that the primary keys are serials automatically generated by the database. The only stored about the payments is the last date of the payment so the next time that the person needs to pay (the next pay date) can be derived from the database. The script for inserting data was added for the seminar 5 submission.

Method

This chapter explains how the physical model and the database was made in this task step by step. The physical model was made with the software "Astah" just like the conceptual model which was made in the last task. To make the physical model, we need the conceptual model to start with.

The implementation of the physical model is divided into different main steps. In the first main step, we start by making a table or an entity for each

entity that we have in the conceptual model. Then we create a column or, better to say, attributes for each attribute in the conceptual model that has the cardinality of 0..1 or 1..1 (having at most one value). There are also attributes that can have more than one value, for example, the phone number for each person. A person can have several phone numbers that we can contact them with. "What should we do with those attributes?" you may ask. The answer is that we make a new entity for each of them in the physical model. As it is known, all the attributes have a domain that defines them. We can take the attribute "firstName" as an example which has CHAR as its domain in the physical model made in this task.

Now that we know what columns the tables will have, there are two more things to note about the attributes. We need to define which attributes are UNIQUE and which attributes should be NOT NULL in the database. All IDs or attributes that make big problems if they are repeated in different rows (tuples) in the table, should be declared as UNIQUE. On the other hand, there are attributes that should never be null, or else the database will have a problem keeping itself together. Those attributes in our current database are the IDs since the IDs are implemented to connect the database's tables.

Then we move to the second main step. This step starts with making primary keys for all the strong entities. Strong entities are the entities that are important and other entities depend on them. These entities need the primary key so that the key can connect them to the relative entities. Primary keys can be either a real-world id, a business id, or a surrogate key.

Then we move on to creating the relations and connecting the entities. Dealing with one-to-one and one-to-many relations is different from dealing with many-to-many relations. For one-to-one and one-to-many relations, we first create the relation, then we make the primary key of the stronger entity, the foreign key for the weak entity. Then we consider the foreign key's constraints. For the many-to-many relations, we start by making a cross-reference table for the relation, using the primary key as the foreign key. After considering the foreign key's constraints, we move to add more columns to the cross-reference table if possible.

Result

In this section, we will discuss the results of the task. The images of the physical model can be seen in figure 1 and figure 2. To view the physical model completely, visit my Git repository: Click Link.

While comparing the physical model with the conceptual model, we can easily see that some new entities were added, such as; phone and email. All the entities include either the ID of the person or the ID of the lesson to connect them and make them reachable. All the IDs should be UNIQUE so that the information about the lessons and people do not get mixed up together. The IDs should also be NOT NULL so the database does not get discrete. The domain of every attribute is declared. In the logical model, the domain for the IDs was

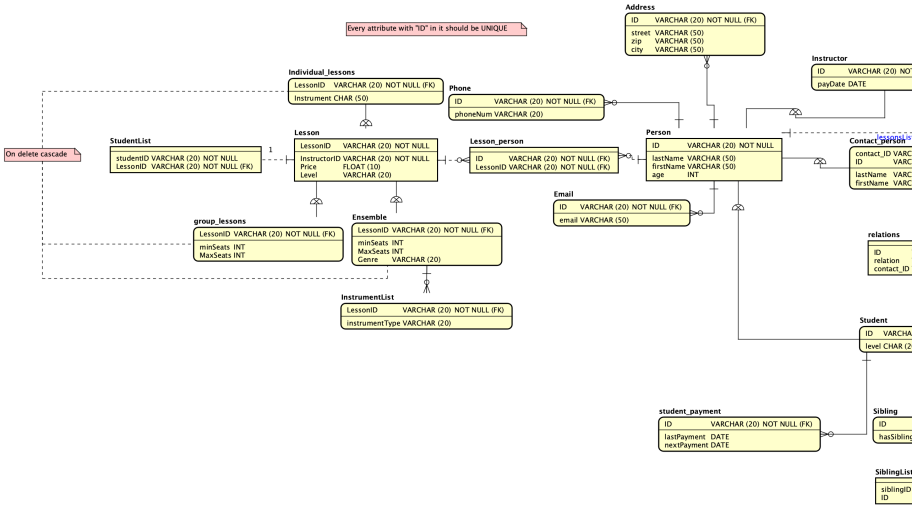

Figure 1: Logical and physical model for the sound good music school database

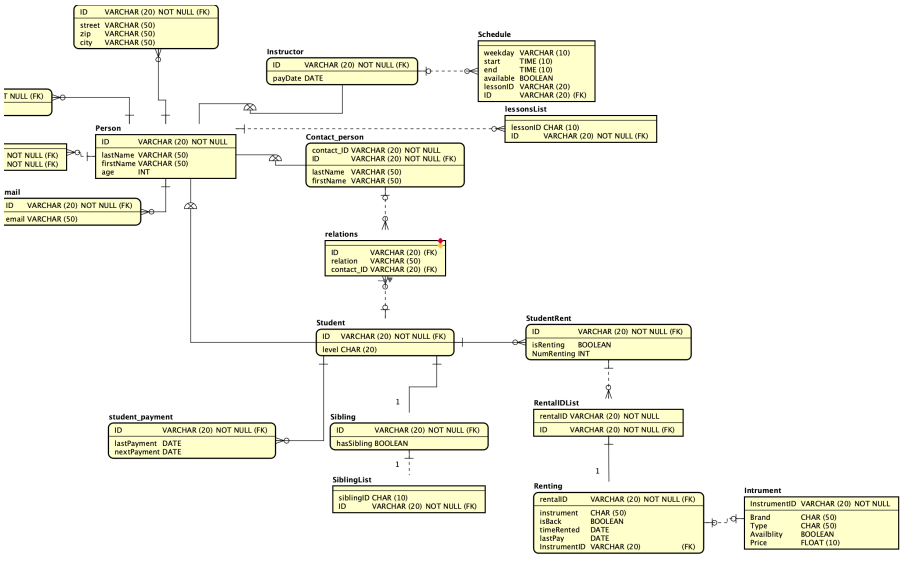

Figure 2: Logical and physical model for the sound good music school database

declared as VARCHAR but then in the SQL code, to create the tables, the IDs were declared as serial so that they would be auto generated by the system.

The persons ID, the lesson ID, rental ID and the application ID were used as the primary key for the database. The entities are connected with relations. The relation between person and lesson was many-to-many since one person can have several lessons and one lesson has several students. For the relation

between person and lesson a cross reference table was made which connects the people to their lessons.

One other thing which was needed for this task was making the database. That was done using MYSQLWorkbench, a GUI tool. The database code can be seen in the Git repository.

Discussion

There are many reasons that I think this physical model is a good model. I tried to name the entities in a way which is easy to understand and consistent. As for the notation of the model I chose the IE notation to have the crow feet notation. The model is in 3NF since it was made with the mentioned steps which was learned by the tutorial available on Canvas page. All needed entities and attributes were added. There are many entities in the model but the reason is that they were needed. The business rules can be seen as note in the model. Only the last payment and the next payment of the students are saved in the database which are the only payment data needed to be stored without wasting space in the database. It was tried to only include necessary relations in the model. All the tasks mentioned in the project can be performed. Unnecessary entities, attributes and relations do not exist and all the data in the database are needed and will be used to derive the other needed data by the system.

Inheritance

In the logical model it is harder to implement inheritance so we try to make the database in a way to include the inheritance in the model. Since inheritance was not covered in the class I had to study the book and search in the internet to try to understand. Honestly, not having information about inheritance in the lectures makes it hard to work on the higher grade part of the assignment. However, I tried my best to include inheritance in the model in a way that it is correct and possible. One way of showing inheritance in the model is to use the ID of the parent entity as the key to connect it to the child entities. If inheritance was not used then all the attributes that are currently used in the child entities would have been in the parent entity and the parent entity would have a big table were not all columns/attribute were needed for each row. The advantage of having inheritance in the model is that we prevent storing useless data and wasting space in the memory while making and maintaining the database.

Having inheritance in the logical model means that the child entity inherits the attributes, unique keys and references from the parent entity. Which means deleting an inherited attribute on a child entity removes it from the parent entity. Inheritance is a good choice to avoid repetition while designing the entities but it can also cause some limitations. Using too much inheritance can make the logical model complex, hard to understand and hard to handle. Inheritance can only represent and show certain types of relationships between entities, such as hierarchical relationships, therefor it will not be suitable for representing

some other types of relationships. Knowing this fact, we need to pay attention to where we use inheritance. Similar to what was discussed about inheritance in conceptual models (task 1), we can conclude that inheritance is a powerful tool that can simplify and improve the representation of complex systems in logical models, but it should be used with caution to avoid introducing extra complexity or oversimplifying the model.