Seminar 1

Data Storage Paradigms, IV1351

Ayda Ghalkhanbaz
aydag@kth.se

November 9, 2023

Contents

1 Introduction

The first task of the project is to design a conceptual model for the SoundGood music school. This school sells music lessons to those who want to learn to play an instrument.

All the information about students will be stored once their application is submitted and they are offered a place. Students are offered different types of lessons such as individual and group lessons, as well as ensembles. They are allowed to take as many lessons as they desire, and the bill for the lessons taken during a month will be paid within the upcoming month. Furthermore, the students have the opportunity to rent up to 2 instruments for a maximum period of 12 months.

The instructors get their salary in a similar way to the student's payment; that is, their salary is paid for the lessons given during the previous month. A more detailed description of the requirements is provided on canvas.

This report covers mostly how the conceptual model was designed, and the results will be discussed in a later section. This task was done individually.

2 Literature Study

Sufficient knowledge in order to complete this task was acquired by watching the videos about Conceptual Model on canvas and reading chapter 3 of the book *'Fundamentals of Database Systems'*. This provided the basics to understand the concept and start the design of a conceptual model.

3 Method

The used method for completing this task involved the following steps:

1. Noun identification
2. Category list
3. Remove unnecessary entities
4. Find attributes
5. Find associations

While this method is generally used to design a domain model, it can also be adapted to design a conceptual model. However, since the conceptual model illustrates the data that exists in reality, some other points should be considered in addition to the list above. These include ensuring that all entities (data) have defined attributes where each attribute has cardinality, and identifying all relations and their cardinality.

The first step was to identify each noun in the project description and categorise them as entities. In order to find more entities and check if any entity candidate is missed or not, the category list provided in chapter 4 of the book *'Object Oriented Development'* was used. The third step was to remove unnecessary entities and check if they can be attribute candidates instead. Step 4 was finding all attributes to the remaining entities. Finally, every relation to the attributes was found and their cardinality was determined.

4 Result

The conceptual model designed for the mandatory part of the task is demonstrated in figure 7.1 while the model for the higher-grade part is illustrated in figure 7.2. The model is built around the persons such as students and instructors.

In the higher-grade model, the inheritance is applied in two distinct instances which is explained in the description of the corresponding classes. The inheritance can be applied only if the super class is a generalisation of the subtype, the subtype is a special case of the super class with its own attributes and all the super class's attributes are relevant to the subtype.

4.1 Person

The person class (entity) consists of general attributes that are common for every person; such as name, person number, phone number etc.

Since Instructor, Student and ContactPerson are persons, meaning that they all have name, person number, phone number and email then they can inherit the attributes from the Person class. The relations are demonstrated in figure 7.2.

4.1.1 Student

In addition to the general attributes of a person, a student has two further attributes; a student ID and person number of the student's siblings, where both of these attributes are unique.

The sibling's person number is stored since the student gets a discount if his/hers sibling(s) are taking lessons as well. This makes it easier to indicate which students are siblings.

4.1.2 ContactPerson

ContactPerson class stores the contact details of relative(s) to a student. A student can have at least one contact person stored, therefore the relation's cardinality is 1 to more.

4.1.3 Instructor

An instructor is a person who additionally has the attributes employment ID (which is unique), the instruments he/she can play and a boolean indicating if the instructor can hold ensembles.

4.2 Address

This class includes attributes indicating an address which a person can live at.

4.3 Application

The Application class has two attributes; *skillLevel* and *instrumenToLearn*. These are the information that a student will provide during the registration process at the school. *SkillLevel* is the student's present skill and can be beginner, intermediate or advanced and *instrumentToLearn* is the instrument that student desires to learn.

4.4 InstructorAvailability

This class stores the attributes indicating the time and date that an instructor is available and can be booked for a lesson. Its relation to the Instructor class is a 0 to more, since an instructor can have 0 or more available time slots.

4.5 Lesson

Just like the Person class, the Lesson class stores general attributes that are common for different types of lessons. Each lessons has an unique *ID*, *nameOfInstrument* and the *level* in it's given. This class has a relation to the InstructorAvailability that checks if any instructor with specific attributes is available or not.

Lessons might be given individually, in group or as ensembles. This means that Lesson can be a super class to the IndividualLesson, GroupLesson and Ensembles. These types of lesson inherit from their super class and fulfill all the requirements of being subtypes. The inheritance relations are shown in figure 7.2.

4.5.1 IndividualLesson

An individual lesson will be booked for a student with a unique ID at the desired time and date. All other useful information such as level and the instrument type is determined already in Lesson class.

4.5.2 GroupLesson

A group lesson might differ slightly from an individual lesson. Group lessons have limited spots and they are not given if the required minimum number of students don't participate. These lessons are given only at scheduled time.

4.5.3 Ensembles

Ensembles, much like group lessons, have the requirements for both minimum and maximum number of students and are given only in scheduled time and dates. Furthermore, they include an attribute named genre.

4.5.4 LessonBooking

This class acts as a bridge between a student and a lesson. A student can book several lessons and each lesson can be booked by several students (excluding the individual lessons), which makes this relation a many-to-many relationship. However, without LessonBooking class, it will be hard to distinguish between students and their booked lessons. The LessonBooking class stores both the student's and the lesson's ID in order to keep the bookings organised.

4.6 LeaseContract

Since students can rent up to 2 instruments during a 12 moths period, a class LeaseContract is useful to manage the contracts. This class has a contractNumber (which is unique), a rental period indicating how long the instrument is leased, a boolean exceedTheLeaseLimit in order to keep track of numbers of leased instruments, and a start date. This class has a relation to the Student class that has the cardinality of 0 to more, meaning that a student may not or may rent up to 2 instruments, while every contract should be related to a student.

4.6.1 Instrument

The Instrument class represents an instrument by storing instrumentID (which is unique), the name of the instrument, the instrument's brand and the quantity of the instrument in the stock. This class has a relation to the LeaseContract, since by renting an instrument, the quantity of the item should be updated.

4.7 PriceScheme

The PriceScheme class consists of prices for each type of lesson in every skill levels. This class determines the instructor's salary as well as student's bill.

4.8 StudentBill

This class includes the number of taken lessons in any category by student during a month. It also has an attribute indicating the fee that student should pay for the leased instruments (it is 0, if the student hasn't rent any item). The relation between StudentBill and PriceScheme has the cardinality of 0 to more since the priceScheme can determine multiple bills.

4.8.1 Discount

Discount is a class with one attribute indicating the percentage of discount and it can affect on the StudentBill if the student has sibling(s).

4.9 InstructorSalary

The instructor's salary is calculated based on the number of given lessons in any category and it is not affected by the sibling discount. The cardinality of its relation to the PriceScheme is 0 to more for the same reason as explained in the StudentBill Class.

5 Discussion

According to the guidelines introduced in the videos about Conceptual Model on canvas, the general naming convention for entities/classes is upper camel case (e.g. ContactPerson) and for the attributes is lower camel case (e.g. contractNumber), which is considered when naming the attributes and entities.

To ensure that all necessary entities are included, the entire project description was summarised, the details were collected and corresponding entities were created. Following this, the unnecessary entities were either removed or categorised as potential attribute candidates to existing entities. Therefore, it is believed that the number of entities and attributes are sufficient and they cover all details of the project. However some slight entities such as Address or ContactPerson were added to the model during final steps.

For some unknown reasons, NOT NULL didn't appear in the entities in front of attributes but this was noted in a comment box in the model. All attributes have the cardinality of 1, excluding the ones with a specified cardinality. All relations have a name at one end.

As mentioned earlier, the inheritance was applied on two instances in the model. It is noteworthy that inheritance can be used only if the relation between two entities can be defined as a "isA" relationship. In this model, there were two entities that fulfilled this requirement, that is, the Person class and the Lesson class. For instance, the Lesson class has relations to the IndividualLesson, the GroupLesson and Ensembles. As we can see in figure 7.1, the relation's name is "isA", which means that those classes with a relation to the Lesson class can inherit all its attributes (see figure 7.2).

There are several advantages of using inheritance. For instance, inheriting the attributes from a super class reduces the redundancy of the common attributes between entities. In this way the subtypes will include only their own specific attributes which improves the readability for the user. Furthermore, editing the common attributes between several entities will be simpler, since we only need to change the super class's attributes.

Despite its advantages, inheritance has also disadvantages. Using inheritance doesn't affect the database and in fact it can make the solution more complicated. Thus, using inheritance is a trade-off and it is preferred when there is a purpose behind using it.

In general, designing a conceptual model is an important step of creating a database, although it is a time consuming part. Missing entities/attributes or finding wrong relations become more common as the project details grow. However, following a step-by-step guideline will minimise the mistakes.

6 Comments About the Course

Completing this task (including lectures, tutorials etc.) took approximately 20-30 hours and I think this is normal time spent on a completely new concept. Learning the basics needs time to sit in, especially when there are multiple choices of materials to read/watch.

7 Attachments

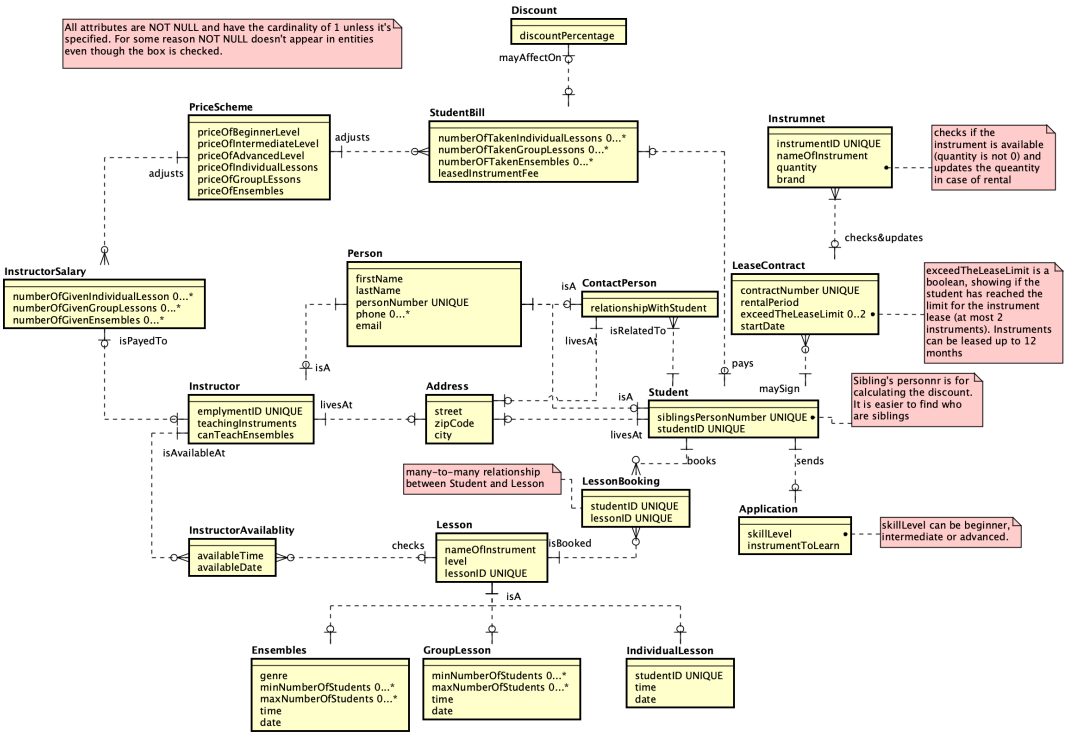

Figure 7.1: Conceptual Model designed for the SoundGood music school project (*Without* inheritance)

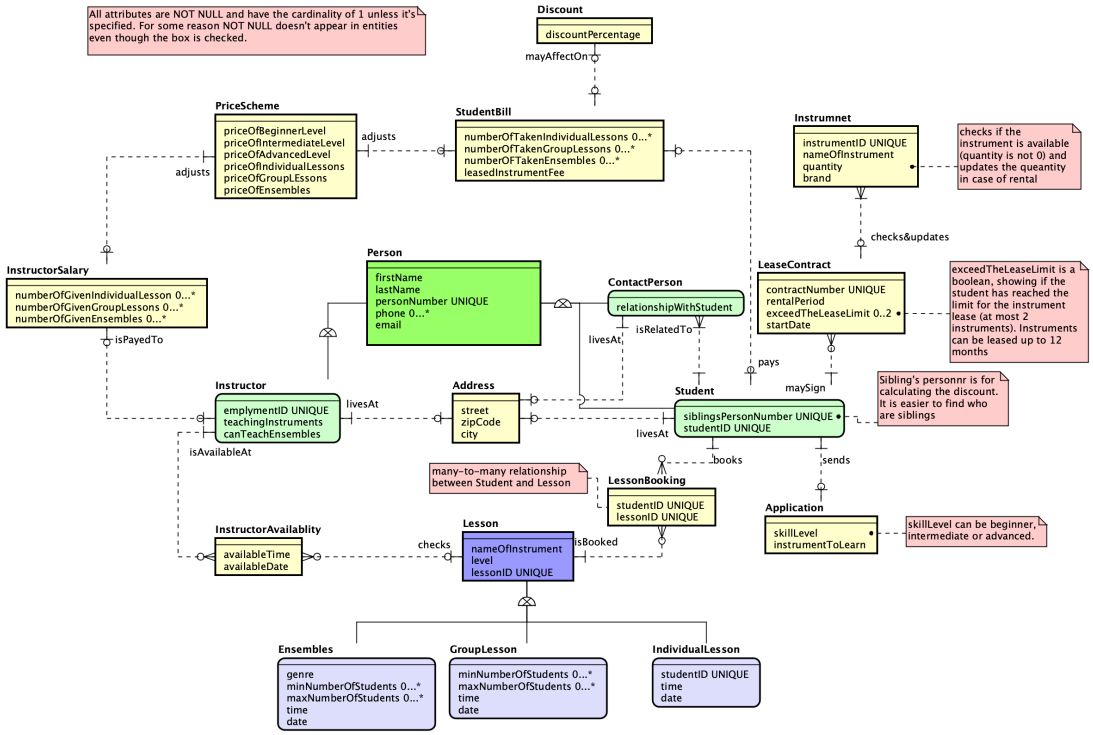

Figure 7.2: Conceptual Model designed for the SoundGood music school project (*With* inheritance). The super class (Person and Lesson)is highlighted with darker colour while the subtypes are highlighted with a lighter colour