Conceptual Model

Data Storage Paradigms, IV1351

Group 50

2024-11-07

Project members:
[Allan Al Saleh, Allan2@kth.se]
[Mostafa Faik, mfaik@kth.se]
[Derfesh Mariush, derfesh@kth.se]

Declaration:

By submitting this assignment, it is hereby declared that all group members listed above have contributed to the solution. It is also declared that all project members fully understand all parts of the final solution and can explain it upon request. It is furthermore declared that the solution below is a contribution by the project members only, and specifically that no part of the solution has been copied from any other source (except for lecture slides at the course IV1351), no part of the solution has been provided by someone not listed as a project member above, and no part of the solution has been generated by a system.

Introduction

The musical school Soundgood holds ensembles, individual- and group lessons, which the students can join. The student pays for each lesson they join and the teachers get payed depending on the amount of students and lessons (and other criteria).

Our aim for this project was to create a conceptual model of a database for the fictional school Soundgood. The data base contains the personal information of the students and instructors. Furthermore, it holds the data for school services such as: lessons, instrument rental and payment handling.

Literature Study

To gain the foundational knowledge for this task, we relied on these resources:

- Course videos(Conceptual model): These provided an overview of UML and ER modeling, along with guidance and a brief introduction on using the Astah tool. They explained the basics of entities, attributes, relationships and cardinalities/ multiplicities, which we used to structure our model.
- Textbooks: We referred to the book *Fundamentals of Database Systems* (7th edition by Elmasri and Navathe), particularly chapters on conceptual modeling, ER diagrams and UML. This helped clarify the roles of classes, relationships and the application of cardinality to limit the number of entities in relationships. Furthermore, we used the category list found in *A First Course in Object-Oriented Development* (March 1, 2024 by Leif Lindbäck).

From these resources, we learned how to identify entities based on requirements, establish relationships with cardinalities and use UML to structure our model.

Method

We started creating the conceptual model by finding the nouns in Soundgoods demands. We did this together as a whole group, to decrease the risk of missing any piece of information. When we were done with this step, we proceeded to look through the category list (which we found in Leifs book) and go through the demands a second time, enabling us to find nouns/entities between the lines.

With all information we need, we proceeded to divide the nouns to classes and attributes. The procedure behind sorting them is that we tried to decide if the noun can have any of the other nouns as an attribute. If it does, then it is a class, otherwise it is an attribute. After that we combined them together, with the goal of creating the fewest amount of classes and attributes, only leaving the most necessary. A suggestion is to always look for ways to combine the classes together.

Lastly, we created the relations between the classes and assigned the cardinality of the relations and the attributes. At times, our knowledge of UML and conceptual models weren't enough to fully implement a certain cardinality, which leads us to adding notes for clarity and efficiency.

Result

Our design of the conceptual model for the *Soundgood Music School* can be seen in the figures below. The model is mainly structured and built around the *Lesson* class, where other classes are connected together, and with the *Lesson* class using relationships.

Figure 1:

In this concept model we used inheritance, which increased the amount of classes to 10. The benefit is that the attributes of some classes are shared, which results in classes that appear to be smaller and easier to read. Instead of using one lesson class, we have four, the main lesson class and the subclasses, which are Ensembles, Group lessons and Individual lessons.

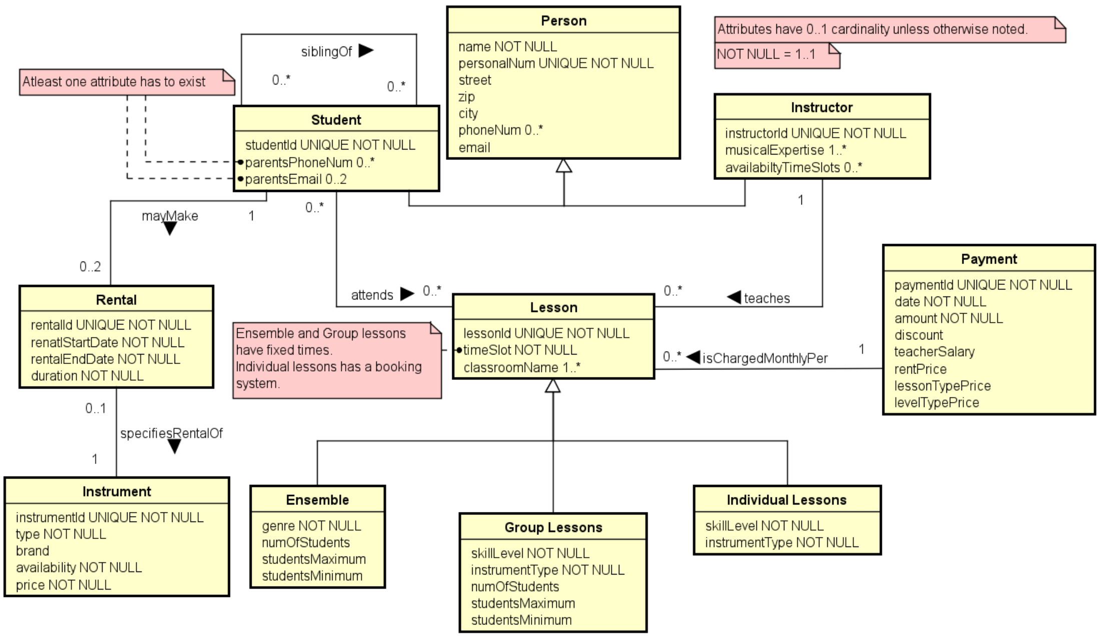

Figure 1: Project designed with inheritance.

Figure 2:

After finishing the model we ended up with 6 classes: Student, Instructor, Lesson, Rental, Instrument and Payment. The student and instructor classes holds the private information for these people. Which includes data such as personal numbers, name, contact information, address and IDs.

The Rental class stores the rental ID, the start and end date of the rented instruments and the duration. While the Instrument class tells us the instrument type, ID, brand and price. The lesson class stores data on lessons, for example the students currently in class, difficulty, which type it is and etc. The payment class holds the costs/payment for everything from instrument rent, lesson costs and teacher's salary.

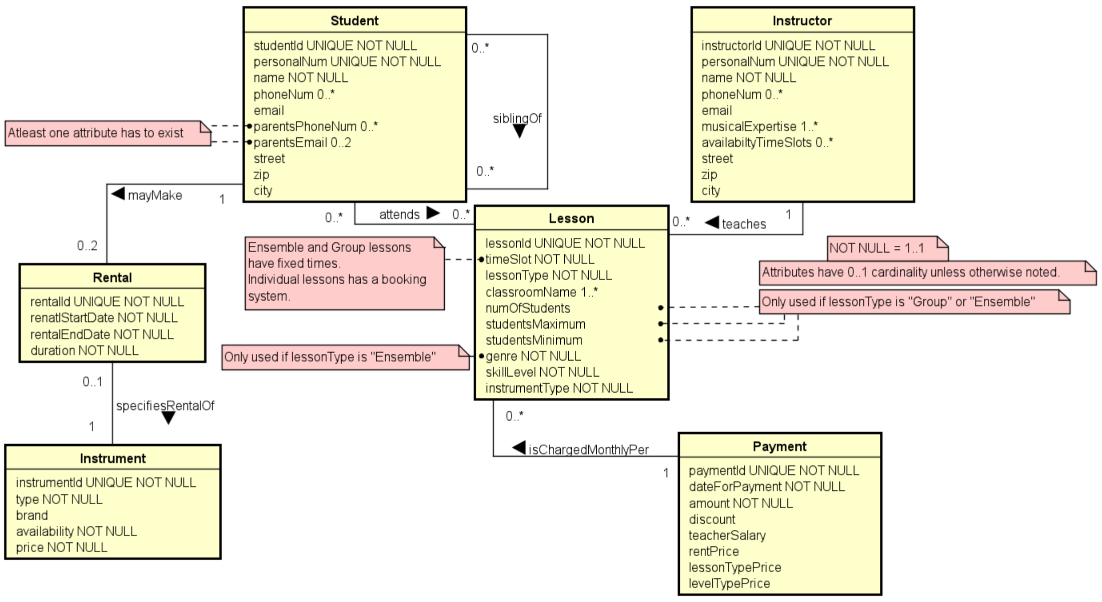

Figure 2: Project designed without inheritance.

Discussion

Difficulties

Regarding attributes and their cardinality, we had a lot of discussions around them, specifically we were unsure at first whether an attribute should be NULL or not. For some attributes, it was easy to implement, such as the amount of instruments we could rent or when an attribute has to be unique. The complexion lies in the attributes which can be deemed as NOT NULL (cardinality at least 1), because we must think which attributes are the bare minimum for the class. In the end after asking, discussing and doing some research, we concluded and decided the cardinality, that exists in the figures.

The pros of having an attribute that can be null, is that we are not forcing the system admin to always include a value for the attribute. However, even if we were to make an attribute NOT NULL, the admin could for example just have an empty string, or the integer 0 or circumvent the data type in any different way, which makes or choice meaningless.

Criteria Analysis

The CM fulfills *Soundgood Music School's* requirements by making the essential business operations like student enrollment, lesson scheduling, instrument rentals and flexible payments. The multiplicities and attributes ensure accurate tracking of lessons, payment and rental uses.

Inheritance vs. Non-Inheritance

Using inheritance offers several advantages:

- Reduced Redundancy: shared attributes like name and contact details are stored once in person.
- Clear Hierarchy: relationship between Person, Student and Instructor provide a logical structure.

In contrast, the non-inheritance model has benefits:

- Simplicity: the structure is straightforward, as each class is independent.
- Less Dependencies on Hierarchies: attributes are accessible directly in Student and Instructor.

The inheritance model was chosen for its organized structure and reduced redundancy, which enhances scalability and maintainability.

Completeness of Information

The model includes all entities and relationships needed by the *Soundgood Music School*, addressing each requirement. Each function (student enrollment, lesson scheduling, instructor management and instrument rentals) is represented, ensuring the database will support the school's operational needs. Example:

- The Lesson entity allows for different types of lessons (individual, group and ensemble) with specific attributes, like skill level and time slot.

Easy Access to Related Information

The model is structured so that information can be retrieved with minimal navigation across entities:

- Most relevant data is directly connected (e.g., Student and Lesson), keeping paths short.
- The structure simplifies common queries, such as finding lessons associated with an instructor or a student's enrolled lessons, due to direct relationships.

Number of Entities and Relevance

The model uses a reasonable number of entities:

- Each entity serves a specific purpose, and every entity has attributes.
- Person as a parent class combines shared attributes, avoiding redundancy while still capturing Student and Instructor differences.
- All important entities are included; for example, Rental tracks instrument rented by students without complicating the Instrument or Student entities.

Attributes and Cardinality

Attributes are defined for all essential data and the cardinality is specified accurately:

- Attributes are marked with appropriate cardinality, like NOT NULL for identifiers and primary fields.
- Unique identifiers, such as studentId and InstrumentId are marked as UNIQUE to ensure each entry is distinct.

Relationship and Multiplicities

The model incorporates necessary relationships with clear multiplicities and names:

- Relationships connect all relevant entities, such as Student to Lesson and Instrument to Rental.
- Multiplicities are well-defined. For instance, Instructor-Lesson relationship have multiplicities 1 and 0..*, ensuring each lesson has one instructor, while an instructor can teach none (if there is no lessons) and multiple lessons.
- Each relationship is named descriptively, like "attends" for Student to Lesson, making the model readable and intuitive.

Naming Conventions and Notation

All entity and attribute names follow naming conventions:

- Names use standard, descriptive terms(e.g., Person, Instructor and Lesson) to maintain clarity.
- Notation in UML is consistently applied, using inheritance where appropriate and clear multiplicities across relationships.