Seminar 1 rehandin

Edvin Frosterud

November 2022

New fixes and or additions will be in italics. Images have been replaced with new iterations, if such exists/was necessary. Lesson scheme and cost per lesson are not handled properly. Niharika Gauraha , 15 Nov 2022 at 9:38 It is difficult to find out who is sibling of who. Niharika Gauraha , 15 Nov 2022 at 9:44 Actual lessons (booking) and lesson plan (scheme) should be separate entities. In this CM a lesson can not exist if a student does not book it. How would a student decide to book a lesson? Niharika Gauraha , 15 Nov 2022 at 9:47 It is difficult to find out who is sibling of who. Niharika Gauraha , 15 Nov 2022 at 9:47 In CM with inheritance, lessons are not associated with students. And pros and cons of with/without inheritance are not discussed.

1 Introduction

Seminar 1 is in practice about how to plan a workspace for a database. Using similar techniques to the previous OOD course. I shared my solution and worked intensily with Yas Asghari but also discussed with a lot of different people and gave each other criticism and ideas. The premise is to make an ER-diagram using a program of our choice that represents the Soundgood music school. A bunch of mandatory criteria were laid out for us, but we also slightly took some liberty in the implementation.

This phase of the project was just planning, and in theory would give us a great foundation to actually use for a database for the full project. The modeling phase, in my opinion was very boring and stressful, as the other course we have also had insane first week assignments. Our ER-diagram was made pretty quickly, but we did make multiple iterations of it after feedback from our peers.

2 Method

The editor used in the project planning phase was Astah. It's an advanced chart- diagram modeler where we could make relatively easy ER-diagrams. We first watched the online video lectures to be able to learn about and how to implement a basic ER diagram, which did follow similar steps to the old UML diagrams we've done in previous courses. These steps are noun identification, category list, cleaning up and then adding attributes and associations.

2.1 Noun identification

The idea of noun identification is to take the text given to the supplier by the customer request, in this case the Project Seminar 1 instructions, giving us about a 100 hits, but a lot of those were duplicates. Manual labour was used to add every single one as a entity in the ER-diagram. At this current stage, none of them had attributes, which would be later added from other nouns and the concepts and ideas the team could come up with that needed to fill in the gaps. The duo ended up with about 30 from this step, but that would be heavily reduced due to the other steps later down the line.

2.2 Category list

The idea of the category list is to add unnamed, but obvious entities to the ER-diagram. These categories are meant to give you some idea of entities that are often missed in the description, therefore giving you a hint what could be moved. Some of the entities here where also later implemented as attributes or removed completely. Some of the categories that weren't explored a lot was transactions that isn't renting an instrument. As the customers instructions didn't include receipts and invoices, we didn't do that, but in case of a real project I think that would be external in their banking system either way.

Around here we had a bit of a snag. Edvin's laptop broke, don't have to go into details but basically lost all the entities already picked out, hence why he felt like we missed some stuff. If it could be redone within the time frame he'd love to add more category list based entities.

2.3 Removing unnecessary and duplicated

A lot of the Entities at this stage were plural versions of singular terms, therefore it was decided to change all Entities to singular versions, getting rid of a lot of duplicated. Some entities also made more sense to be an attribute as they as an entity wouldn't have attributes of their own, like Address which in this case can be stored a basic datatype, String.

2.4 Relations and cardinality

A relation, is the concept of a child and a parent Node. The idea is that a "parent" node can see the scope of all it's children, therefore having access to them. There's a couple if types of relations. Identifying, and non-identifying, and they also contain a cardinality. Cardinality is the concept of relations with a limited scope- usually from 0 to x where x is a limited amount of children of that type of child each parent can have. We have lots of different cardinalities, but as far as this assignment the way to add cardinality 2 wasn't defined in Astah, so 0-many had to be used for the instrument leasing where according to instructions the maximum were actually two.

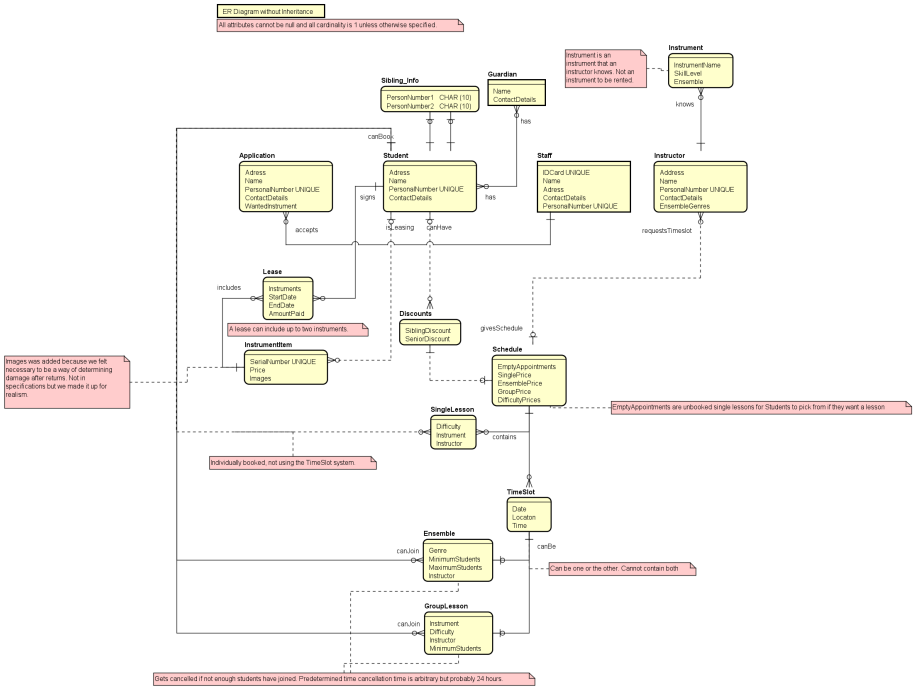
Relations and cardinalities were added and refined over a couple of iterations. Some attributes had to be labeled with a UNIQUE tag as to not merge data.

We decided that all data is not nullable even though technically I think some of it is possible to change in case a parent perhaps wants to not give out their telephone number or similar.

2.5 Method changes

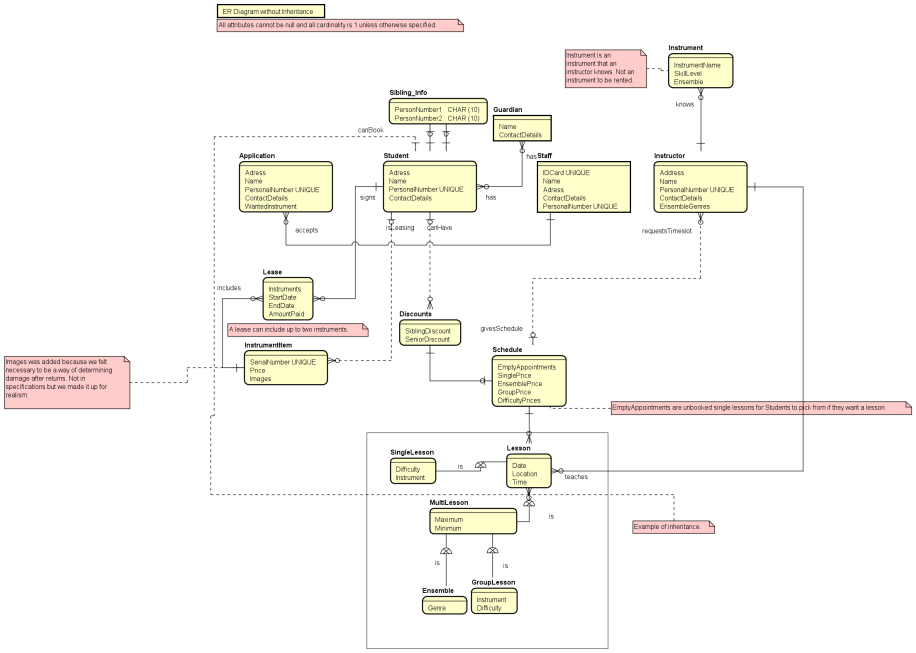
I've previously described the how the newer ER-diagram was made in Seminar 2. In practice, we redid the whole thing, which was frustrating and annoying. Since seminar 2 had slightly different specifications, requiring a logical model instead of a conceptual, some changes. We moved lesson pricing to schedule and cleared up the system completely by removing BookingManager. Sibling info was clarified by having a seperate entity that stores two siblings. This would only confirm if you had a sibling or not, but it also is possible to follow a chain of siblings incase you want to know if two people are siblings. An alternative is storing every sibling for every sibling, but that seems like a lot of data. This approach does however require some more logic for making a new sibling-student, as you have to find the end of the chain, sort of like a linked list. I also seperated the schedule and booking system from student, so that a lesson can exist without a student booking it.

3 Result



Here we can see the no Inheritance version. We've reduced our entities to a very few amount. Some notes help you understand the reasoning and logic behind each relation. Note that also in the Notes there is a note saying all are non-nullable if not otherwise noted. This is the inheriting version. Only implementation of inheritance is with the lessons type at the bottom. Same thing with nullable objects.

Changes are discussed above in method.


ER Diagram without Inheritance
All attributes cannot be null and all cardinality is 1 unless otherwise specified.
Instrument is an instrument that an instructor knows. Not an instrument to be rented.
Instrument
InstrumentName
SkillLevel
Ensemble
knows
Sibling_info
PersonNumber1 CHAR (10)
PersonNumber2 CHAR (10)
Guardian
Name
ContactDetails
canBook
has
Application
Adress
Name
PersonalNumber UNIQUE
ContactDetails
WantedInstrument
Student
Adress
Name
PersonalNumber UNIQUE
ContactDetails
Staff
IDCard UNIQUE
Name
Adress
ContactDetails
PersonalNumber UNIQUE
Instructor
Address
Name
PersonalNumber UNIQUE
ContactDetails
EnsembleGenres
signs
has
accepts
hLeasing
canHave
requestsTimeslot
Lease
Instruments
StartDate
EndDate
AmountPaid
includes
A lease can include up to two instruments.
Discounts
SiblingDiscount
SeniorDiscount
givesSchedule
InstrumentItem
SerialNumber UNIQUE
Price
Images
Images was added because we felt necessary to be a way of determining damage after returns. Not in specifications but we made it up for realism.
Schedule
EmptyAppointments
SinglePrice
EnsemblePrice
GroupPrice
DifficultyPrices
EmptyAppointments are unbooked single lessons for Students to pick from if they want a lesson
SingleLesson
Difficulty
Instrument
is
Lesson
Date
Location
Time
teaches
MultiLesson
Maximum
Minimum
is
is
is
Ensemble
Genre
GroupLesson
Instrument
Difficulty
Example of inheritance.

4 Discussion

Oh, how this assignment was a mess for me and my partner. We do have a laptop each, but mine breaking down after the first day without backup certainly learned us a lesson. Astah is also very bad on M1 Macbooks, so I believe that program to stop being recommended. We did this with little time and a lot of stress due to our other course completely breaking the mold and having an assignment without lectures or tutorials in 3 new programming languages in one week. Either way, I think our results speak for themselves. There's probably some things to improve, but in my opinion we did pretty well considering the bumps we encountered. Both our Conceptual Models include the necessary noted data that the instructions required.

- The only slightly discuss able traits I can see is our way of detecting Siblings. Siblings are currently found by checking the Guardian/Contact Person and seeing what Child nodes they have, but this approach is very rudimentary as it doesn't take into account people without a legal guardian.
- I believe most important information is very central, and this is without connecting a circular loop due to how ER Diagrams in Astah blocks us from doing that.
- All our entities are relevant, but I can see criticizing the use of the BookingManager, but in our theoretical situation this is where global data for multiple Nodes exists, like the difficulty prices, the prices for different types of lessons and access to the child, Schedule. It could be argued that it can be encorperated into the Schedule, but we didn't think a bunch of money stuff fits. This would also handle data for outside logic calculating the amount of money each person has ordered at the end of the month.
- All our attributes are made sure to not be duplicated as much as possible. Unless I missed something there is attributes for everything at this point in time.
- Everything is by default not null- so we didn't mark it. All our uniques have the tag UNIQUE.
- I believe our relations might be a bit cluttered, but they're not too many. If I had the chance I'd redo it again just to clean the look up, but time is a factor.
- In my opinion no relations are missing. Yes, they have cardinality but also names of the action they perform.
-
- Naming conventions are followed, even though I never heard them specified anywhere we did saw how they were handled by peers and video so we altered our attributes.

- We tried adding lots of notes but it looks cluttered, but the most important logic and business rules are there, and comments where an action was hard to describe in few words.

- Yes. It's above.

Inheritance and non-inheriting examples are included in the results. Subtypes, the line for inheritance does however look different, so I'm wondering if that's an option or something in Astah?

In my opinion, as a person whos mostly been working in OOD I think subtypes and inheritance is a lot cleaner. Giving the option to store data of a similar type with their respective functions working on all of them. I'm not sure how the rest of the project looks, but I'd like to redo it with all inheritance, a possibility def. exists for People to be one major class as they share so many attributes. It is generally harder to implement as we'd have to be careful with making changes to the superclass but I think the upside of less clutter and duplicated data makes up for it. All the lesson types inherit from the larger supertype Lesson giving us more diversity, and in the case of this we could have multiple lessons already setup in classrooms and locations, including a tutor in there and allowing for the students to use it as either an Ensemble or a GroupLesson depending on what has more signups.

As it was required for 10p, heres a little bit more discussion about the pros/cons of inheritance. Inheritance, as a concept comes with a bunch of upsides. It allows our code to be less repetitive, as the child entity will inherit attributes from the parent. Inheritance can increase coupling, which generally is something you want to avoid, but it does basically always increase cohesion which is demanded. By using subtypes we minimize the duplicate information as much as possible and allows for faster implementation of new lesson types and or global changes to all lessons. Subtypes, at least generally in programming can also be overwritten, which does make any downsides that subtypes have very debatable. One of the things that cannot be overwritten is overflowing attributes, which the only fix I'm aware of is more subtypes. Great example of this is the Lesson-¿TimeSlot-¿Ensemble, GroupLesson since we made an entire timeslot as to not have extra attributes on the regular single lesson, which slows down query time in SQL. I do think our inheriting diagram is a lot clearer, but this is debatable if its from the inheriting aspect of it, or just working more on the readability of it. Encapsulation isn't affected.

5 Sources

Youtube channel Leif Lindbäck(Link: https://www.youtube.com/channel/UCs3aBf$_5KdxjKnNVFk_HrSQ$)