Project Report - Seminar 1

Data Storage Paradigms, IV1351

2024-11-03

Project members:
Simon Lieb Fredriksson, simonlf@ug.kth.se
Sebastian Taavo Ek, sebte@ug.kth.se
Adalet Adiljan, adalat@ug.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.

1 Introduction

The task for this seminar was to design the conceptual model, in the shape of an entity-relation diagram, for the database project in the course IV1351. The requirements for the model are that it accurately depicts the entities and relationships of the "Soundgood music school", a hypothetical client requesting a database from us, found on the project description section of the course's canvas page. Once the conceptual model has been completed to an acceptable degree of accuracy and usability, it will later be used to inform the future design of the database in subsequent seminars.

The model needs to contain all the information needed by Soundgood, be easy to collect information from related to all the major entities, have a reasonable number of entities to begin with, and have attributes for all the data that needs to be stored.

Importantly, the ER-model has to depict as many of the entities and relationships in the UoD (Universe of Discourse) as possible. Failing to capture the entire context

could be detrimental to our end product. We therefore set out to methodically perform noun-identification and using a category list to identify our entities, and then reason our way iteratively through what needs to be an entity, attribute or relationship.

With the complete model in hand, we then analyze and evaluate our findings further in the discussion section of this report.

2 Literature Study

Before beginning the initial designs of our ER-model, we first set out to gather some foundational material in the course literature. Since a database can be summarily described as a collection of related data [1, page 4], we knew from the start that the key to a well-designed database would be a detailed and accurate model for the relationships that exist between the data in our UoD [1, page 5].

Importantly, the course book (Fundamentals of Database Systems by Ramez Elmasri and Shamkant B. Navathe) very clearly states that the conceptual model, also called schema, is a concise description of the data requirements of our end-users. [1, page 61] Leif Lindbäck, administrator for this course, says something very similar by explaining that the ER-model is not meant to model a database, but to model the data present in the real context that a future database is meant to serve. [2]

2.1 Entities

In order to design the ER-diagram, we first had to understand the definition of its parts. Entities, as we understand it from the literature, are things with an independent existence in the world - be it physical or immaterial and more abstract. They could be objects like a building or a person, but also phenomenon like a course or a degree. [1, page 63] Entities are largely defined by their attributes, and if we can't find any attributes for an entity candidate in our conceptual model then it is not interesting to us.

2.2 Attributes

Attributes, differently to entities, are descriptors or properties of other larger things. Interestingly, and new to us, attributes can be either simple (also called atomic) in cases where they represents simple data like strings, numbers and booleans, but they can also be composite. The book gives the example of an attributes called "Address", which might in turn be composed of attributes such as a street name, zip code and city. [1, page 65] This concept was initially quite challenging to accept - since "Address" seems to be a prime candidate for an entity rather than an attribute if it has attributes of its own, but we gather that depending on how you decide to build your database in the end, you might prefer to represent the data as an entity or an attribute in the conceptual model given your product's needs.

Attributes can also be presented as single-valued or multi-valued. The book gives the example of a car having an attribute "Colors", which could be populated by multiples.

If a multi-valued attribute has any constraints on the amount of values it can hold, we should declare that constraint properly in the model. [1, page 66]

We also came across the rather important term "Key attribute", which seems to signify the defining attribute of an entity such that it is unique among all other entity instances in its entity set. The book gives the example of a "Person" entity who might have the key attribute "Social security number", since it is unique and could never be common between two entities of the type. [1, page 68] This sounds relevant to our own ER-model, where a "Person" entity will probably have the same key attribute.

2.3 Relationships

Given that we are working with the idea of a relational database, this section might be the most important of all. Relationships indicate how the entities in our UoD relate and or interact with one another. Interestingly to us, the book also reveals that many bits of data that might have initially been registered as attributes in the early stages of the ER-model, might later be transformed into relationships as the model matures. [1, page 72] It is not entirely clear to us what the perk is of over-indexing on attributes versus relationships (since they look to be closely enough related), but it seems in any case preferable to have clearly defined relationships between entities if our end goal is to create a specifically relational database. The book also explains that relationships, just like entities, can have attributes of their own - although we were not sure enough on this concept to implement it in our own design which can be found in later sections of this report. Relationships also declare their cardinality - indicating how many relationships an entity is allowed to participate in.

Finally, a relationship can be either identifying or non-identifying - where an identifying relationship indicates that the target entity could not stand to exist as an entity on its own, were it not for the parent. Another way of saying this, as the book does, is that the weak entity (the entity being defined by the relationship) has no key attributes of its own, while the stronger entity does. Also, similarly to how some attributes can sometimes be re-imagined as relationships as the model matures - some weaker entities in a identifying relationship can also be re-imagined as complex attributes in the defining entity. The book, again, mentions the example of the "Address" entity, which might have an identifying relationship with a "Person". We could instead re-imagine the "Address" as a complex attribute (both multi-valued and composite) in the "Person" entity, and remove the relationship and "Address" entity altogether.[1, page 79] It is not entirely clear to us when this is a better or worse choice for the model as a whole.

3 Method

We followed the recommended methodology given by Leif Lindbäck in his pre-recorded lectures, and replicated in many ways the same process we had used in an earlier course for the production of a domain model. The first step was to perform a noun-identification process on the project description, in which every key noun is extracted and taken as a

future entity candidate. We would only cull these candidates at later stage, and for the moment we considered even the most ludicrous of options.

After performing the noun identification process, as well as considering any other candidates that a category list might produce, the next step was to start considering which of the generated candidates were more appropriate than others - and to consider which nouns were more useful as relationships or attributes than entities. For example, a candidate such as "Instrument rental" is clearly better represented as a relationship between a student and an instrument, where the relationship is defined with a cardinality of "one student rents 0 to many instruments". It was also advised by the tips-and-tricks document that went with the task for seminar 1 to avoid entities that are completely without attributes. The data in our model is present in the form of attributes - so if an entity is completely empty it has little use to us as database programmers. When selecting our candidates we were also careful to avoid any derived data which could be discerned from other data in the model. A lesson might have a "startTime" and "endTime" for example, but then it does not need an attribute "duration".

The last and final step was to review our findings and revisit the project description to make sure that we hadn't forgotten anything or overlooked something obvious. We went through each section of the project and double checked that the data necessary to perform the tasks of the future software was all present in our ER model - keeping in mind that we are still not modeling a database, but rather the relationships and entities at play in the real situation being considered.

The review sent us down a path of making sure that all the cardinalities were correct, and that all attributes were marked as either unique or "not null" in the necessary places. We also reconsidered many entities and their relations and ended up re-constructing large parts of the ER-diagram involving finances. This was an incredibly challenging exercise and we're still not entirely sure if the end result is satisfactory, but its the best we could do.

The final result was modeled in Astah as a UML entity-relation diagram. Find the diagram in the next section of this report.

4 Result

Following is the end-product of our diagram as of the time before seminar 1, after which we will almost certainly make changes to it according to the feedback we receive from our teachers and peers.

As you can see in the image, there are a couple of strong entities around which the whole diagram revolves. These are "Person", "Lesson", "Instrument", and the different types of "Payment". We first experimented with having a super-entity called "Payment" from which the three different sub-types (rental, lesson and instructor) could inherit a lot of their common attributes, but it ended up making the model a lot harder to read at a glance, as well as complicating the relationships around them. What relationships go to the super-entity rather than the child entity? Is it preferable to generalize or to be more specific? As we weren't entirely sure, we made them separate entities in their

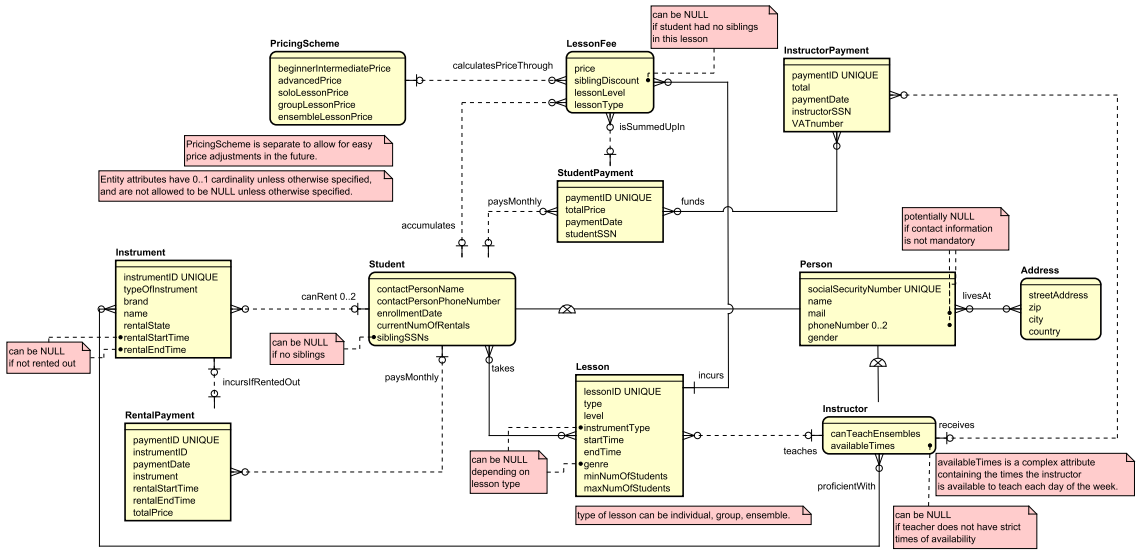

Figure 1: Our conceptual model in the shape of an ER-diagram.

own rights and tried to spread them out in the diagram to the corners where they had the most relations for the sake of readability.

Furthermore, we tried our best to identify which relationships were identifying versus non-identifying, and reasoned that any entity that could not possibly exist on its own merits must have an identifying relationship with a stronger one. Some examples of this included the entity "Address", which could not exist on its own but rather exists as a complex attribute to every "Person" in the database. Earlier on we mentioned in the course literature how weaker entities have a tendency to qualify as complex attributes to the stronger entity in its relationship, and that seemed to be true in our case here. [1, page 79] Another example is that of "LessonFee" which is generated for each student by a Lesson - since every student attending that lesson needs to pay for attending it. In this way, we reasoned that "LessonFee" could not exist without the "Lesson" entity, and because of this we can say that it has a defining relationship to it as the weaker part.

Other relationships are marked with dotted lines to indicate non-identifying relationships, in which entities with unique identifiers relate to one another. For entities such as "Person", the unique identifier is clearly something like a social security number - but for other entities such as "InstrumentRental" it was significantly harder to think of an attribute that could be used as a unique key. In these cases, we argue that each instrument rental must in some way be different from others in the context (and later also in the software) through some arbitrary identifier. For entities such as these, we therefore added an "....ID" attribute, which has to be different for each instance of that entity type. We're not sure if this is a heavy-handed approach, or if this somehow breaks the rule that says that we should not yet be modeling an actual database but rather the

UoD in question - but for now it serves as a solution to a somewhat difficult conundrum.

All in all, we believe that we meet the requirement of representing all the information contained in the project description through our entities, relationships and attributes. With some small caveats that will be discussed in the final discussion section.

Since we used inheritance in our model, it was also mandatory to include a diagram of our conceptual model which does not employ the use of inheritance, in order to illustrate the differences. To that end, here is our same model without using inheritance:

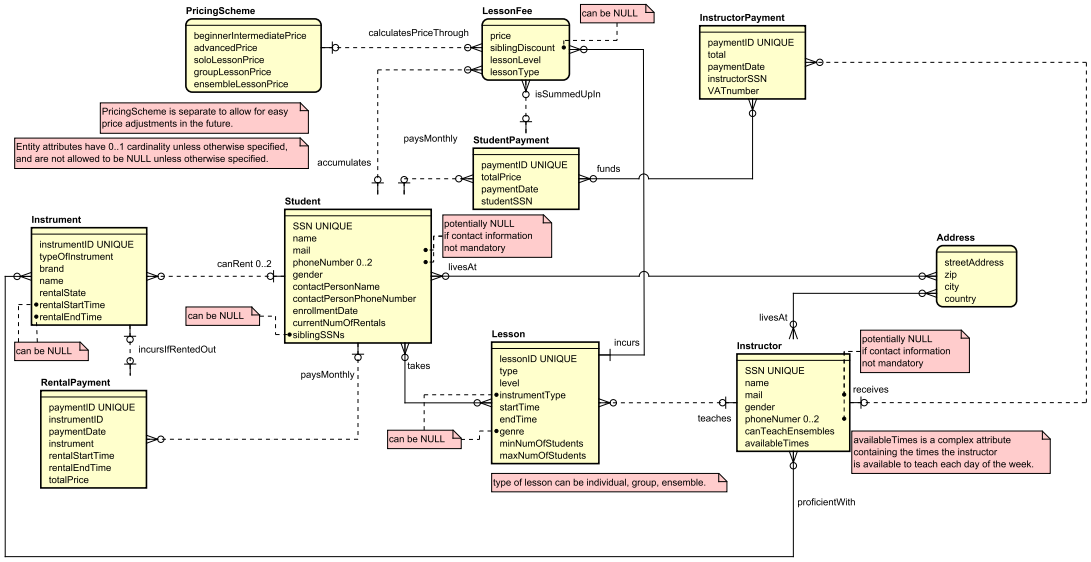

Figure 2: Our conceptual model in the shape of an ER-diagram. No inheritance.

5 Discussion

To be frank - this was quite a challenging seminar task for the whole group. The rules given for how to construct a conceptual model are pretty strict, and the definitions for each of the components (entities, attributes, relationships) are relatively clear as well - but to put it all together can feel like a subjective endeavor to say the least of it. There are no doubt as many ways to create a conceptual model of a UoD for a database as there are database designers, and that subjectivity and variance can make it challenging for a learning student to know exactly what to do. As we discovered in the course literature and preparation lecture videos; some data could be represented as attributes or relationships depending on what we need them to be. Other bits of data could be either an entity or a relationship (with attributes of its own). As a project group, we are still not entirely sure how to tell right from wrong in this situation, and know what best suits the design and the future of the project. Hopefully this is something that becomes clearer as future seminar tasks make use of the conceptual model to different ends.

As for the topic of inheritance - it was requested to briefly talk about what the diagram looks like with or without inheritance involved. Some obvious perks of the inheritance strategy seem to be that we don't have to re-write attributes that are common between sub-entities, such as a name, address, phone number and social security number, which are all common to the "Person" type of entity - but have to be written out explicitly in each sub-entity ("Instructor" and "Student") if the super-entity is removed. The same goes for relationships that sub-entities might have in common. This makes the model easier to read at a glance, and reduce the amount of relationship clutter.

Another perk could be, and here we are somewhat speculating, that a proper use of inheritance might facilitate queries in our future database that operate on a general class of data (the super entity) but still accurately reflects the contents of its sub-entities. That way you would not have to write a query method for "Student" and "Instructor" separately if all you were looking for was a name and social security number, since they have them in common. You could instead just query the "Person" type.

5.1 Self evaluation

1. **Does the CM contain all information needed by Soundgood?**
   Answer: *To our knowledge, yes.*

2. **Is it easy, that is a reasonable number of hops, to collect information related to all of the major entities (student, lesson, instructor, etc)?**
   Answer: *Yes. We've made efforts to keep the ER-diagram readable but still sufficiently interconnected between the entities.*

3. **Does the CM have a reasonable number of entities? Are important entities missing? Are there irrelevant entities, for example, entities without attributes?**
   Answer: *We are not sure what constitutes a reasonable number of entities, but we believe that ours is within the scope for what is acceptable. There are no entities without attributes.*

4. **Are there attributes for all data that shall be stored? Do all attributes have cardinality? Is the cardinality correct? Are the correct attributes marked as NOT NULL and/or UNIQUE?**
   Answer: *We believe that all data that needs to be stored by the future database is represented in the diagram. The one exception might be some sort of "stock" for the instruments. We currently have no way to keep track of how many instruments are in stock as opposed to in circulation as rentals. It was not clear in the project description if this was a must. All attributes do have cardinality, and we have employed the use of "UNIQUE" but not "NOT NULL".*

5. **Does the CM have a reasonable number of relations? Are important relations missing? Are there irrelevant relations? Do all relations have cardinality at both ends and name at least at one end?**

Answer: *It's very hard to tell when you're new to the topic whether or not there is a reasonable number of relations - but to our eyes we have enough relations to accurately depict the project description from the project page on canvas. There is cardinality at both ends, and all relations have at least one name.*

6. **Are naming conventions followed? Are all names sufficiently explaining?**
   Answer: *Java conventions have been followed and the names are descriptive.*

7. **Is the notation (UML or crow foot) correctly followed?**
   Answer: *We believe that we have followed the notation correctly. We tried our best to follow Leif Lindbäck's example in the pre-recorded lecture videos.*

8. **Are all business rules and constraints that are not visible in the diagram explained in plain text?**
   Answer: *We've tried to make use of notes when possible to cover constraints that the diagram does not show.*

9. **Is the method and result explained in the report? Is there a discussion? Is the discussion relevant?**
   Answer: *The mentioned sections are included. We hope that they are relevant.*

6 Comments About the Course

It was recommended to spend about 20 hours (at least) on the project task in order to do it properly, and we can pretty confidently say that we've spent at least that time to produce our model and this report.

Other than that, we can only say that the start of the course feels only somewhat overwhelming with the amount of new terminology and concepts that we need to grasp - and this may have effected the quality of our conceptual model. There are no doubt entire concepts that we forgot to include or didn't illustrate properly with UML, and we'll just have to accept that this is something to fix after the fact, since it's not possible to know what you do not know.

References

[1] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems*, 7th ed., Pearson, 2016.

[2] Leif Lindbäck, *Conceptual Model, Part 1*, YouTube. Available 2024-11-05. URL: https://www.youtube.com/watch?v=LhTAAlg8ABc