Project Report Seminar 2

Data Storage Paradigms, IV1351

Teoman Köylüoglu and Teoman@kth.se

Date: 2023-11-23

1 Introduction

This section of the project will, using the conceptual model as foundation, henceforth be used to create the Logical Model. Whereas the conceptual model serves as a high level abstract model of the database, it does not dwelve into the details and the types of data the database involves. The Logical Model is the final step between a model and a database. Primary keys, foreign keys and their constraints are considered as well different kinds of datatypes to be stored in the database.
Seeing as all the data mentioned in the task, building upon the previous model, some changes had to be made to fit all of the criteria put forth by the music school. Therefore some things were revised such as added entities and many-to-many relationships were hereon considered of importance wherein the CM they were not.

Lastly, when the Logical Model is completed, and the database is to be filled with data. The ER diagram is imported into SQL, a database is then created in the Shell and the empty tables are thereafter filled with generated data for each table and its attributes. All while leading into the forthcoming task wherein querying the database will be the main objective.

This project is a joint collaboration with students Julius Larsson and Victor Naumburg, and a higher grade is sought on this task.

2 Literature Study

To gather sufficient information in order to complete this task, and with the intent of fulfilling the learning objectives of the course, the textbook as well as Leif's videos showed to be of great benefit in traversing the task.

The textbook, covering the topic of normalisation throughout chapters 14 and 15, gave giving a clear guideline on what is considered to be a good and bad relational model. Some examples of this is the decomposing of multi valued attributes, into its separate

entity. And in accordance with the first guideline; Keeping the meaning of each entity clear via not including different entity types' attributes into one relation.
Chapter 14 also makes a clear point in avoiding having entity types which would lead to exaggerated NULL values in its tuples. This was of great benefit, and also seemed rather natural, as it would not make sense to have tables with an exaggerated number of attributes anyhow in a relational model.
One of the biggest takeaways, and something which was noted through the reading of the textbook and subsequent querying in SQL, was that having the same attribute names for the primary key and its respective foreign key proved to be of great value when doing joins, more specifically equi-joins. This was a convention employed through the designing of the Logical Model.

The main source of information, for this task, however was Leif's videos on the implementation of the Logical Model. Leif provides a clear methodology consisting of eleven steps which as a result made the implementation of the Logical Model a much more feasible task. Details on what the eleven steps are, is covered in the Method section.

3 Method

To make the implementation of the Logical Model, and the subsequent transition from the Conceptual Model to the Logical Model easier, the diagram editor Astah was continued in use. The use of IE notation (crow foot) was similarly not changed. Therefore there is no change in methodology on specifics on the software side, excluding the addition of a Command Line Interface to reach Postgres and create the database towards the end of the task, and naturally the use of the SQL query language.

To meet the requirements in the description and task, Leif's eleven steps method was employed. The first step consisted of transforming the entities into tables, seeing as they have different naming conventions. Since multivalued attributes are not permitted, every such attribute was made as its separate entity with a relation to its original entity. The fourth step then consisted of defining a datatype for each of the attributes, such as CHAR, VARCHAR, INT and TIMESTAMP etc.

Furthermore, after having defined the datatypes for each attribute, the next step was to consider their respective column constraints. Leif provides a meaningful discussion on what is reasonable when defining them, considering their real-world semantics and using common sense. For example, it wouldn't make sense for first and last name to be unique when a social security number already fills this purpose. However considerations with regards to them not being NULL or not is meaningful however.

The next step, as provided by Leif, is to create the different relations and assign primary keys to strong entities while preferring surrogate keys for those entities. The general

convention is that weaker entities do not have surrogate keys unless they are stronger to another entity, and as such having a meaning on their own. The primary key is always inherited as a foreign key in the weaker entity, and may be used as a primary key for the weaker end if there is a one-to-one relationship, or in combination with another foreign key or attribute to create a primary key. Foreign key constraints are then needed to ensure proper deletion and update rules for foreign keys which are highlighted in the diagram via comments and then programmed into the database.
As for many-to-many relationships between two entities, they are solved via creating a new entity that resembles a bridge connecting the two. The primary key in this new entity will be a set of the two primary keys from the respective entities.

Lastly, the method of creating the database is done via connecting a command line interface to the Postgres application, writing code that creates the database and the exporting the ER diagram from Astah onto the database which is then filled with generated data.

4 Result

To meet the requirements of the task, firstly the Conceptual Model had to be converted into a Logical Model. To realize this, a new Astah file was created which essentially re-constructed the Conceptual Model in the form of the Logical Model. Several changes had to be made; Initially, each entity name had to conform to the Logical Model's conventional standards, that is names were uniformally in lowercase with an underscore between spaces of words.
In the previous task the CM was mainly modeled using inheritance with the non-inheritance model showcased as an alternative. It was decided for this task that the non-inheritance model would be used instead, seeing handling cardinalities for non-inheritance relationships would be more reasonable.

A general convention which will be used throughout the diagram is for the use of primary keys for strong entities. These primary keys will be assigned as surrogate keys that have a naming convention that of the entity name. In the case of weak entities, when they are in a one-to-one relationship, the foreign keys will be unique and act as a primary key. In any other case, such as for the "time_available" entity (weak entity in a one-to-many relationship) the primary key will consist of a combination of attributes.
If a weaker entity is stronger to another entity, then it will be assigned a surrogate key as well, seeing as it's meaningful irrespective of its stronger entity.

Referential integrity constraints for selected foreign keys are set to "ON DELETE CASCADE" and is displayed in figure 8. In the event a stronger entity has one of its tuples or records erased, this change ought to be reflected in the tuples referencing the tuple being deleted. ON DELETE CASCADE means that the corresponding records referencing the deleted record is deleted alongside it. This constraint upholds the IC-rules maintained

in a relational database.

After the tables were created, as mentioned in the method section of the report, each attribute of zero-to-one cardinality were included as they would. Multivalued attributes were created as their separate entity with a relation to its predecessor. In the Conceptual Model, phone number remained an attribute of the "Person" entity. An instance of a person may however have several phone numbers, and thus as a result, phones as well. This is a multivalued attribute and has to be created as its separate entity. After having created the "phone" entity with a phone number as its attribute, it is also noted that one single instance of a phone number may be attributed to several persons (landline). Since this then becomes a many-to-many relationship it is resolved via connecting the two entities with a "bridge" entity "person_phone" containing foreign keys referencing each surrogate primary key in the respective entities.

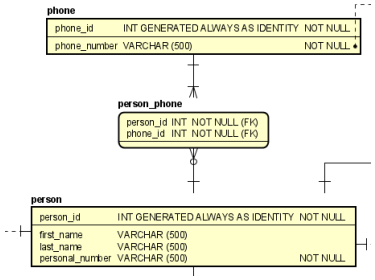

Figure 1: Subdiagram of figure 8, Included to illustrate person_phone relationship.

A similar reasoning is made for the "instrumentCompetence" entity of the CM. One person may have several instances of instrument competence, and an instrument competence can be attributed to many students. Another bridge of this sort connecting "person" and "instrument_competence" and its many-to-many relationship holding the primary keys of each as foreign keys is made.

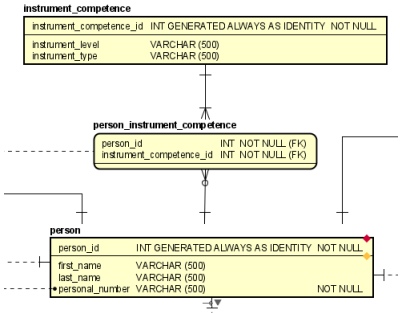

Figure 2: Subdiagram of figure 8, Included to illustrate instrument_competence relationship.

Proceedingly, a student may take several lessons, and a lesson may hold an arbitrary number of students. Using the same reasoning as above, the following result is obtained.

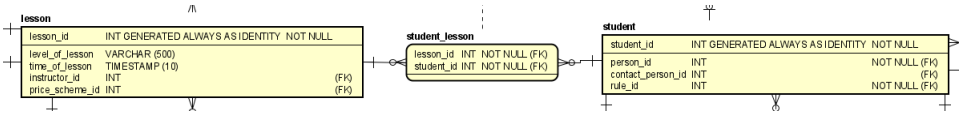

Figure 3: Subdiagram of figure 8, Included to illustrate student_lesson relationship.

In a lecture it had been suggested that the address entity may be merged into its connecting relations, however it was deemed appropriate to keep it as such and let it remain a many-to-many relationship. A person encapsulates students, instructors as well as contact persons.

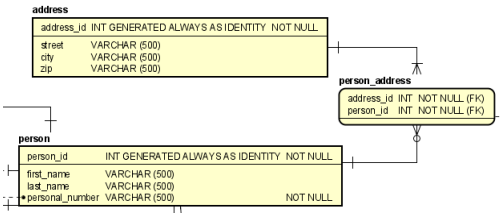

Figure 4: Subdiagram of figure 8, Included to illustrate person_address relationship.

The sibling relationship is modeled as a relation to self, as the task requires knowledge of each student's sibling in order to issue the discounts, a relation to self provides a table in the database wherein such information is readily displayed via a simple select-from query.

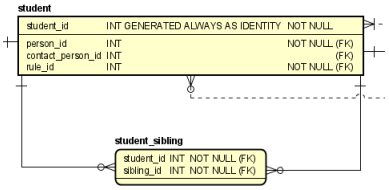

Figure 5: Subdiagram of figure 8, Included to illustrate student_sibling relationship.

Changes to the "priceScheme" entity of the Conceptual were substantial. As the task requires the fact that prices may change and that students still have to be billed the original amount they paid at the time. Records showing the time of a specific price scheme is needed. A "price_time" attribute was added with the datatype of "TIMESTAMP", this is because not only does it hold the time record but also its specific date. As a result, older lessons may still be associated to an older instance of the price via this

solution as both lessons and the price scheme have a "TIMESTAMP" and an associated price. Additionally, the "Price" entity from the Conceptual Model was no longer needed as it is data which may be derived from the price scheme.

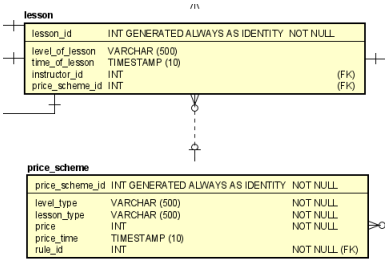

Figure 6: Subdiagram of figure 8, Included to illustrate lesson and price scheme relationship.

In the conceptual model, constraints on the cardinality or rules regarding some relationships were highlighted with a comment. In a logical model, which will be later exported to a database, data storing the actual constraints have to be enforced and stored as actual data. Three kinds of tangible rules had been identified in the description of the music school. Students may only rent up to two instruments at a time, students under the age of 18 should have a contact person and data storing the discounts. To solve this, a generic entity "rule" with generic attributes encompassing the three different constraints was created. To separate the different rules, and the fact that the entities e.g "price_scheme" and "rental" have total participation in the rule entity, a surrogate primary key "rule_id" is befitting.

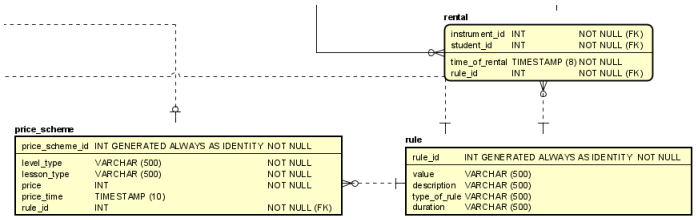

Figure 7: Subdiagram of figure 8, Included to illustrate rule_id relationship.

The "type_of_rule" attribute specifies the type of rule being enforced, e.g "Rental, Discount or Contact Person". The description describes the nature of the rule, "Amount of Rental", "Age Limit, and "Student Sibling Discount". The value is then specific for each of the rules, "2, 18, and %10". The duration attribute specifies the duration of the rule, or contract in the case of rental. In the case of rental the duration would be 12 months, with allowance to be changed if need be. Note that the contact_person table does not store data regarding the rule specification, instead this data is available in the

rule_id table. To see if a student is applicable for a discount, a query can be made for the contact_person table to see who's a sibling of whom, their amount of lessons during a month can be seen from the cardinality of relational instances and since the "rule_id" table is related to the "price_scheme" table the discount value may be aggregated to the price.

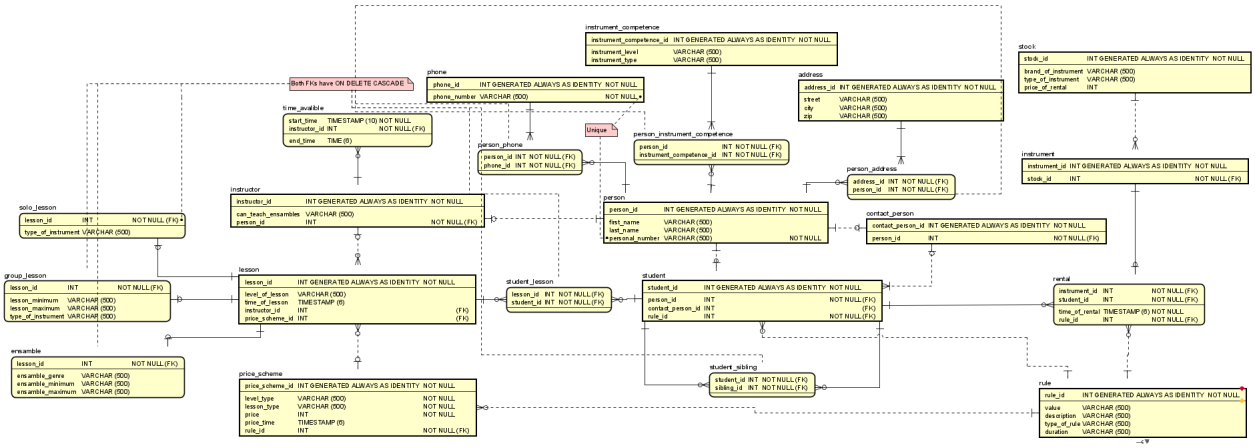

Figure 8: Complete Logical Model.

A link to the Github repository

5 Discussion

All while designing the Logical Model, especially in the beginning, it was decided to adhere to some core principles by which the Model shall be based on. One of these was the introduction of a primary key convention. The primary key convention consisted of the idea that surrogate keys are only assigned to strong entities. A weaker entity would also be assigned a surrogate (primary) key if they were stronger to a different entity. For example the instructor entity is a weak entity to "person", but also a strong entity to "time_available" and "lesson", resulting in a surrogate primary key for the instructor id. Every surrogate key was named after their entity, e.g "instructor_id". This convention was especially important seeing as when the relation between lesson and instructor is drawn, and Astah automatically assigns a foreign key reference to the primary key in instructor, it may be tempting then to change the name of the foreign key, however keeping the referencing name identical to the PK was a conscious decision. Via doing this, natural joins in SQL are optimized as the join condition may be omitted in the query when they are of the same name. The model and the tables are in addition optimized for comprehension for a third party. Keeping small yet important conventions such as this proved to be a great factor in the designing, as keeping a clear structure lead to less debating amongst the collaborators with regards newly created entities and their respective keys.

5.1 Normalisation

Moving forward, normalisation playing a central role in the designing of the Logical Model, several steps were taken to ensure that the final model is normalised. As mentioned in the method section and in the Literature study, Leif provided eleven steps for designing a Logical Model. What was not mentioned is that, doing those steps will, indirectly, lead to a normalised model if done properly. One of the initial steps taken was to create new tables for attributes with a cardinality of greater than one. This had to be done for the phone number attribute of the "person" entity. This is the essence of 1NF; which states that only single atomic attributes are allowed in a relation. There are several ways of going about solving the issue of multivalued attributes. The one opted for in the model made use of decomposition into a separate entity with a relation drawn to "person". An alternative option could have been to, instead of decompositing into a separate entity, had it been known a maximum value of the cardinality to replace the "phone_number" attribute with atomic attributes "phone_number 1, phone_number 2, etc..". However seeing as a person may have an unrestricted amount of phones, it was better suited to consider the first alternative opting for a many-to-many relationship, as shown in the result section. Aside from aggravating queries on these attributes, as well as reducing redundancy, it would not make sense to restrict the number of phones a person may have, without a specific request from the client in the case the database only considers primary phone numbers, needed only for reaching the individual.
While it is true that a Logical Model opts to model a mini-world, it could be argued that

storing multiple phone numbers for a person, student or instructor (besides personal and work phones) could be considered a waste of storage and meaningless in a music school not dealing in indexing of said numbers.

Furthermore, the Logical Model manages to effectively uphold functional dependency and thus 2NF as well. Almost all, besides from the many-to-many bridge entities, have a singular attribute as primary key, meaning the second normal form test need not even be applied to most of the model! As the model at this point already satisfies 1NF, and most entities constitute a singular key, which in turn eliminates the possibility of partial dependency, then only some of the entities need be checked for functional dependency. It may be mistakenly assumed that having the "personal_number" remain as unique may uniquely identify the remaining attributes and therefore violate the second normal form. Note however that partial dependency is only related to wherein a non-prime attribute depends only on a subset of the primary key set. It is thus concluded that functional dependency is still upheld for this relation.

A similar reasoning can be made for the remaining entities; The address_id in the "address" entity cannot be identified via way of one of the singular attributes "street, city or zip". And as such, functional dependency and 2NF is upheld. Note that address has one singular attribute as primary key, which means this test was unnecessary to begin with.

As for rental, which has a combination of two foreign keys as its primary keys, further consideration has to be made.

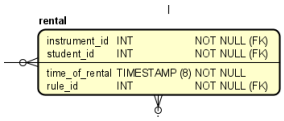

Figure 9: Subdiagram of figure 8, Included to illustrate attributes of rental.

Is it possible for any of the non-prime attributes of rental to be identified by a single key-attribute?
A "time_of_rental" cannot be inferred from either of the attributes. However, as for rule_id, since the rule_id is the same for all instruments (in this database, seeing as there is one rule of rental pertaining to rental period etc.), rule_id can then effectively be inferred from "instrument_id" alone which means there is a partial dependency in this relation. This violates the second normal form, and would normally have been dealt with accordingly (in the form of a surrogate key). Doing so however, would violate the stipulated convention of only having surrogate keys for stronger entities. Rental is not a strong entity, neither is it causing any obstruction to the database leaving it as it is.

It is also of wisdom; The music school may decide to at some point have different

rules for rental, say, students taking several types of lessons spanning several types of instruments may need to rent more than two instruments at a time. In that case, there would no longer be a partial dependency as not every student will have the same rule_id applied onto them.

Most of the model can be considered to be in 3NF. The entities pertaining to the lessons strictly uphold the third normal form. For example, it may be mistakenly thought to be the case that the attribute "price_scheme_id" is dependent on the subset of non-prime attributes "level_of_lesson" and "time_of_lesson". Two lessons of the same level may take place on the same time (as they may be of different type). Likewise, none of the attributes in either type-of-lesson entity has transitive dependency. The same is said for the instructor identity; An instructor being able to teach ensembles says nothing about who they are in the database.

Some sacrifices with regards to 3NF have to be made in order to preserve the structure of the model, and to store the data necessitated by the music school. For example, the "price_of_rental" attribute's value is dependent of the brand and the type of instrument in the table. And since their being tied to the same stock_id, the price of every identical stock_id will be the same. Price, still has to be included as an attribute, and had it been inserted into the "instrument" entity instead, then the price would still be dependent on the stock_id as well as the primary key "instrument_id".
To include it in the "price_scheme" would lead to exaggerated null values and the intended meaning of its entity. The course book further mentions (p. 476) that *"database designers need not normalize to the highest possible form. relations may be left in a lower normalization status, such as 2NF, for performance reasons"*.
Leif similarly stresses this point in his lectures; And as for the Logical Model, normalisation was sought whenever possible, but extravagant measures to adhere to higher order principles was taken on its merit in relation to circumstance.

Another consideration that was taken into account when designing the database, was the use of ENUMs with regards to the difficulties of lessons. As the music school only specifies the use of three levels of difficulty, it would naturally make sense to limit the datatype of the those three columns to only include one of the three values specified. Via doing this, constraints are put on the user when inserting these values by way of enforcing and rejecting typos, and thus maintaing the integrity of the database.

Benefits of storing all data in the database are tremendous. It makes the process of joining tables simple, as storing all data necessarily (for a sound relational model) entails the data is organised thereby ensuring quick retrieval of information. Updating and changing information and rules pertaining to the music school becomes easier as the application or any code surrounding it may be bypassed to the database itself. Since all data is stored in the database, changes will automatically be reflected on the software level.

Some obvious disadvantages is the increase of complexity, which has already been noted by the designer during this project. Maintaining data integrity constraints can quickly become a tedious endeavour when considering multiple tables, their referential integrities and maintaining a high database standard.
Databases that centralize all their data can also become more vulnerable to security threats, which would put further constraints on the operation and underlying security to protect such an endeavour.
Sounds like a job for an engineer.