Logical and Physical Model

Data Storage Paradigms, IV1351

2024-11-21

Project members:
[Anton Persson, antope@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

This report covers the creation of a logical model based on the conceptual model from Task 1 of the project. The purpose of this model is to increase the detail of the conceptual model so that it can be translated into an actual database using SQL. In this course only a logical model is created, however this logical model has features from the physical model which allows us to create a real database from it.

2 Literature Study

The Author started by going going through the course lectures on relational algebra, SQL, normalization, logical and physical models. The author gained understanding about the concepts and making of making a logical model and the relational algebra/normalization which together build the physical database. This understanding allowed the author to complete task 2.

3 Method

To begin creating the logical model for the database a modeling software is needed. Because Astah was choose for the first task of the project it is reasonable to use here as well. The software used to create the functional database itself was PostgreSQL.

The process of creating the logical model contains a number of steps that are used to "translate" the conceptual model into this new model which more accurately represents the physical database that will be created.

The first step that is taken is to create a "relation" or a table for every entity in the conceptual model.

The second step is to look at every attribute in each entity and translating them into columns in the table. This however can only be done for "atomic" values, the attributes that can only hold one value.

The next steps is to create new tables for all "multi-variable attributes", specify the type or "domain" of all columns and consider all column constraints. Following this all "strong entities" in the model shall be given primary keys in the form of surrogate keys.

The final set of steps to create the logical model handle connecting the tables to each other following the relationships in the conceptual model. Here "many to many" relationships need to handled differently, a new table has to be created between the two strong entities to act as a middle man. this table uses both foreign keys as its primary key in order to create relations between tables. The "new" relationships between the tables created from multi variable attributes is resolved by using the foreign key of the strong entity they were part of and the attribute combined.

To finish the model we verify that the model is normalized by going through the conditions that the levels of normalization require, and we verify that all of the planned operations are able to be preformed on the data.

After completing the model we only need to implement the model into an actual database using SQL and PostgreSQL, then filling it with some testing data.

4 Result

`https://github.com/Smullbabo/IV1351`

The authors solution for a logical model based on the conceptual model from task 1 can be seen in figure 1.

This model is very similar to the conceptual model because they share the vast majority of tables and relationships. There are however some key distinctions.

In the logical model the "Student" and "Lesson" table have a new helper table between them because a student can attend multiple lessons and a lesson can have multiple students, this creates a problem because we can not directly model a "many to many" relationship. This new helper table lets us cross reference their ids in order to find all of the lessons a student has attended and also every student that attended a lesson. This table works by having a primary key which is the combination of the foreign keys from student and lesson.

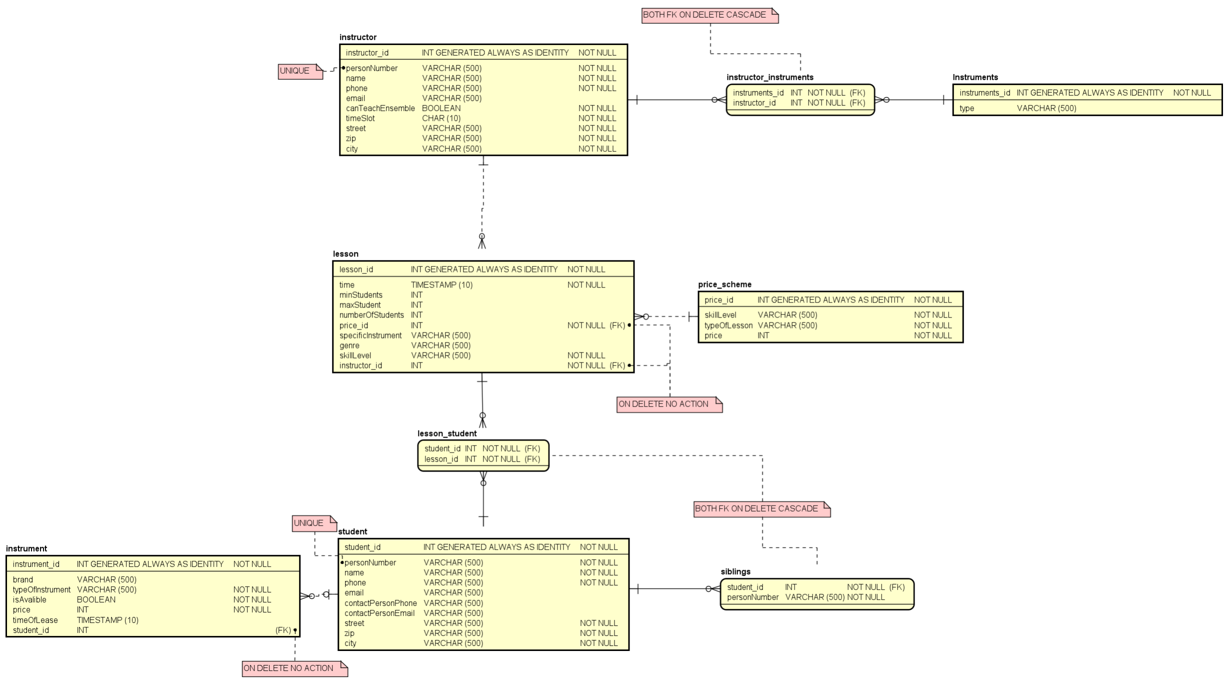
BOTH FK ON DELETE CASCADE
instructor
instructor_id INT GENERATED ALWAYS AS IDENTITY NOT NULL
UNIQUE
personNumber VARCHAR (500) NOT NULL
name VARCHAR (500) NOT NULL
phone VARCHAR (500) NOT NULL
email VARCHAR (500)
canTeachEnsemble BOOLEAN NOT NULL
timeSlot CHAR (10) NOT NULL
street VARCHAR (500) NOT NULL
zip VARCHAR (500) NOT NULL
city VARCHAR (500) NOT NULL
instructor_instruments
instruments_id INT NOT NULL (FK)
instructor_id INT NOT NULL (FK)
Instruments
instruments_id INT GENERATED ALWAYS AS IDENTITY NOT NULL
type VARCHAR (500)
lesson
lesson_id INT GENERATED ALWAYS AS IDENTITY NOT NULL
time TIMESTAMP (10) NOT NULL
minStudents INT
maxStudent INT
numberOfStudents INT
price_id INT NOT NULL (FK)
specificInstrument VARCHAR (500)
genre VARCHAR (500)
skillLevel VARCHAR (500) NOT NULL
instructor_id INT NOT NULL (FK)
price_scheme
price_id INT GENERATED ALWAYS AS IDENTITY NOT NULL
skillLevel VARCHAR (500) NOT NULL
typeOfLesson VARCHAR (500) NOT NULL
price INT NOT NULL
ON DELETE NO ACTION
lesson_student
student_id INT NOT NULL (FK)
lesson_id INT NOT NULL (FK)
BOTH FK ON DELETE CASCADE
UNIQUE
student
student_id INT GENERATED ALWAYS AS IDENTITY NOT NULL
personNumber VARCHAR (500) NOT NULL
name VARCHAR (500) NOT NULL
phone VARCHAR (500) NOT NULL
email VARCHAR (500)
contactPersonPhone VARCHAR (500)
contactPersonEmail VARCHAR (500)
street VARCHAR (500) NOT NULL
zip VARCHAR (500) NOT NULL
city VARCHAR (500) NOT NULL
instrument
instrument_id INT GENERATED ALWAYS AS IDENTITY NOT NULL
brand VARCHAR (500)
typeOfInstrument VARCHAR (500) NOT NULL
isAvalible BOOLEAN NOT NULL
price INT NOT NULL
timeOfLease TIMESTAMP (10)
student_id INT (FK)
ON DELETE NO ACTION
siblings
student_id INT NOT NULL (FK)
personNumber VARCHAR (500) NOT NULL

Figure 1: The authors solution to the logical model

This type of ”helper table” is also used when dealing with the types of instruments a instructor can play. This is not strictly necessary as this is a one to many relationship but in order to prevent their being multiple of every instrument for each instructor. For example ”Erik” could play guitar and piano but so does ”Magnus” then there would be 4 rows in that table. We could also have different spellings so that only ”Erik” is connected to 4 rows like this ”Guitar”, ”guitar”, ”Piano” and ”piano”. This makes sure that we only store types once and the primary key of the helper table allows us to find which instruments each instructor can play.

The last new table that has been added is the ”sibling” table and is used because we cannot have multi variable attributes unlike the conceptual model. This new table simple has a primary key made up of the foreign key from student and a person number that connects back to another student.

Next the constraints for keys needed to be implemented. For both the ”helper tables” it was decided that the should be ON DELETE CASCADE because if one of the ”parents” are deleted that row has no use and should be deleted. For example if a lesson is deleted there is no point in having a student connected to nothing. This was also decided for the ”sibling” table as if one of the siblings is no longer a student that entry is no longer useful to us. As for the remaining foreign keys it was decided that they should be ”ON DELETE NO ACTION” because for example if an instructor is removed we might still want a record of the lessons for student payments. we also have this for the instruments students can rent because if a student is removed the real life instrument is not removed and is simply returned and can be sued again.

The final step of the task was to implement the logical model in PostgreSQL as an actual database, this is quite simple however as the majority of the SQL code needed to create the database can be exported directly from astah as a SQL-file. From this file we can add the ”INT GENERATED ALWAYS AS IDENTITY ” as the primary keys of the tables who do not have a foreign key as a part of their primary key. This makes sure that all the primary keys are unique and we don’t have to manually add any ”ids”. All that is left is to add the constrains we set as notes in the model to the SQL file.

Finally the database was filled with example data generated by chatgpt.

5 Discussion

The model created in this rapport and the database that it describes follows a standard notation and all the tables has sufficient explanations withing they just as they have in the conceptual model. As it is based on the conceptual model this one also uses correct crowfoot notation.

The model might not be modeled in 3NF because both ”instructor” and ”student” have street, city and zip in them which could be argued to be derived from each other in ways. Here however it was deemed that creating an ”address” table would introduce more potential problems and conflicts than it would solve.

The columns are all relevant and needed as they correspond to a attribute in the conceptual model, the constraints and types are all relevant and can be motivated based on the specific attributes relationships with other attributes, the real world and appropriate data types.

All the operations listed in the project description are able to be preformed by using various SQL queries in order to get a desired outcome, for example it is possible to SUM all of the prices of the lessons for each student in order to facilitate payments. Here you can also make sure to add if they have a sibling to qualify for a discount. The same is true for the instructor for payments of course.

All primary keys in the model are unique as most are surrogate keys generated buy the database itself and the few tables who do not have such a primary key have a primary key made up of 1 or more foreign keys as explained in the method part of the report.

The businesses rules that are not apparent are as discussed all related to payments, these functions can however be derived from the data present in the tables. The amount to be payed by a student can be calculated from the tables ”student”, ”lesson” and ”price-scheme” as can the cost of rented instruments for a given student. This all depends on what kind of system one wants to implement on top of the database but all transactions of the business are doable with the given database. It might have been easier to have a table or a value for each students payment as it would be easier to understand as a non technical user but this data would be derived and therefore a duplicate of the data already in other tables.