Project Report Task 2
Data Storage Paradigms, IV1351

2024-11-21

Project members:
Jonathan Värild, varild@kth.se
Oscar Caddeo, ocaddeo@kth.se
Elias Holm, eliholm@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

2 Literature Study

3 Method

4 Result

The result of our combined physical and logical diagram of our database can be seen in Figure 1 below. Its structure is very alike to our Conceptual Model Diagram since it already was fairly normalized and compatible with a database structure. The first table that we implemented was the **person** which uses an automatically incremented

integer ID **school_id** as primary key. It is automatically incremented using the built-in SQL feature **GENERATED ALWAYS AS IDENTITY** which ensures that we can't insert values to it. The **school_id** is widely used in the database since it is used to identify different persons. Their personal ID number is only used to identify their **Person** row with their real life identity. This ensures that we in special cases can update the personal ID number of a person if it were to change for some reason. The table have two checks using regex to ensure that the **personal_id_number** and **zip_code** are correctly formatted.

The **student** table has a relationship with **contact_details** which store personal or relative contact details for each person. Each contact details have a unique ID that is also **GENERATED ALWAYS AS IDENTITY**. This is also the primary key of each contact details. Since we don't require both a phone number and an email address we have added a check that ensures they are not null at the same time.

Next, we have the tables **instructor** and **student** which uses a foreign key to person which connects them. This allows us to divide each person into roles and also apply additional columns or relations to each role such as **can_teach_ensamble** or the table **sibling**. The **instructor** and **student** row will be automatically deleted if the **person** reference is deleted since it is defined as **ON DELETE CASCADE**.

One of our main issues with the Conceptual Diagram Model was that we didn't think about that 3 students with 3 different parents may not be siblings with everyone. For example, student 1 may be sibling with student 2 which share the same mother and the same father. At the same time, student 3 may also be sibling with student 2 by sharing father. Sibling 1 and 2 may however have different fathers and mothers which don't make them siblings. This made it impossible for them all to share a single sibling ID that groups them together. To fix this, we implemented a new table called **sibling** that takes two foreign keys which both are **student_school_id**s which can link two students together as siblings. To ensure that you cannot insert bidirectional data we created a trigger which automatically ensures the first school ID is the smallest one and the second school ID is the largest one. These two IDs build the primary key which ensures only one sibling relation can be saved for two students. We also added a check to ensure that a student cannot be a sibling with themselves. If any of the students are removed from the database, the sibling relationship is automatically removed since both foreign keys are defined as **ON DELETE CASCADE**.

The **instructor** table has a relationship with **instrument_skill** which defines what instrument skills each instructor has. The **instrument_skill** table in turn has a foreign key to **instructor** which links them together. If the **instructor** row is deleted, so is the **instrument_skill** row since the foreign key is defined as **ON DELETE CASCADE**. The **skill_level** column also has a check which enforces that the value is between 1 and 3 indicating the different skill levels beginner, intermediate, and advanced.

Next, we have the **activity** table which store general information about all types of activities like individual lessons, group lessons and ensembles. This table also has a generated integer ID. It has a check to ensure that the start and end time of each activity is in the correct order. The **activity** table also has relationships with the **lesson_individual**, **lesson_group**, and **ensamble** tables. These tables help to group

each activity into the different types. This is also a small change from our conceptual model diagram which used Lesson to define both an individual and group lesson but with different values to their min and max student attributes. To ensure that we don't have a table with a lot of NULL values for min and max students, we decided to split the tables up instead which removes this need. Both **lesson_group** and **ensamble** share checks to ensure the min students are above 1 and that max students is larger that the min students. All 3 tables share **ON DELETE CASCADE** on **activity_id** which ensures that their row is deleted if the activity row is deleted. The tables **lesson_individual** and **lesson_group** share the check to ensure that the skill level is between 1 and 3 in the same way as it's defined for the **instrument_skill** table. The **activity** table has **ON DELETE RESTRICT** on all foreign keys to ensure that these are not removed before all their references are replaced or removed. The **activity** table also have a trigger that ensures a instructor isn't booked on overlapping times since the instructor can't be at two places at the same time.

The **activity** table also has a relationship to the **location** table which is a table of all possible locations where you may host an activity. The **location** table has a check to ensure that not both **room_name** or **video_link** are NULL since a location may be either physical, online, or both.

Next we have the **pricing** table which hold pricing information for each activity type based on its type and skill level. It also holds extra information such as sibling discount percentage, if its active and a unique ID. This fulfills the requirement to be able and update the price of a certain activity but still be able to see the old price after its updated. Each activity store the id to the pricing row that was active at that time. If you wish to update the pricing you simply set the current row's to active column to false and insert a new row matching the same activity and skill level. The **pricing** table also has a number of checks such as that **activity_type** is between 1 and 3 which indicates individual lesson, group lesson, and ensemble. It has a check to ensure skill level is between 1 and 3, that the price isn't negative, and that the sibling discount is a number between 1-100 indicated by a discount percentage.

The **booking** table has a relationship with both the **activity** and the **student** table and is used to store what activities each student is booked for. Both foreign keys are set to **ON DELETE CASCADE** which ensures that a booking is removed if its student or activity is removed.

Looking in the other direction of **person** we have the **rental** table which keeps track of all rentals made by each student. It has foreign keys to the student **school_id** and instrument **inventory_id** which are defined as **ON DELETE CASCADE** since there is no meaning to keep track of a rental that cannot be linked to a person or instrument. The table is linked to person to ensure that both students and instructors should be able to make rentals. To fulfill the requirement on that each student only should be able to have two active rentals we have added a trigger that checks this before a row is inserted or updated. Another check ensures that the rental time does not exceed 12 months.

Finally, we have the **instrument** table which store information about all instruments that the school has in inventory available for rental. All instruments of the same brand and model are stored in the same row and are instead kept track of using a **quantity**

column that defines how many instruments there are. The table has two checks to ensure that the quantity and price is over 0.
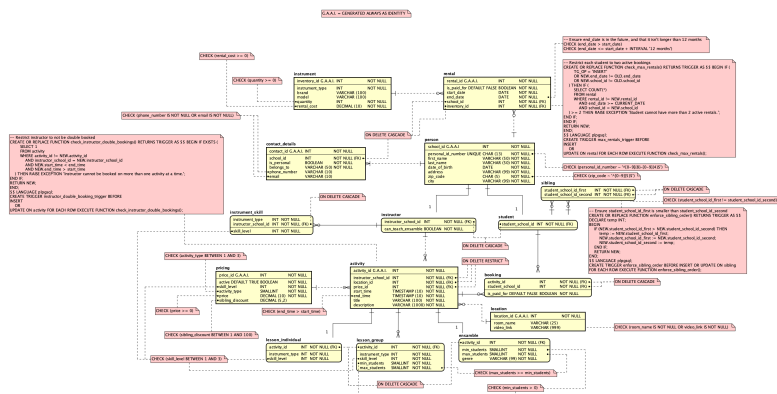

Figure 1: The physical/logical diagram that we came up with.

5 Discussion

5.1 Question 1

5.2 Question 2

5.3 Question 3 ...

6 Comments About the Course