Project Report

Data Storage Paradigms, IV1351

Noel Tesfalidet Noelt@kth.se

2022-05-05

1 Introduction

The purpose of this Project was to design the database and an application for Soundgood Music school. The database handles all information that the school use and the application handle the instrument renting transactions. I developed the database and application with Sam Khosravi. The project was split into four tasks (Conceptual Model, Logical and Physical Model, SQL, Programmatic Access). The Conceptual model was made with a program called Astah and displays relationships, entities, and the attributes. The conceptual model shows all the entities that are important to the business and the relationships between them. The main goal for the model is to establish an overall view of the school. The Logical model is a model that builds on the conceptual model but focus more on the actual database and the specific implementations of the relationships. We also refined our logical model so that it has some properties of a physical model so that the model is as close to the actual database. In conclusion a single model was made that is a mixture between logical and physical model. In the SQL task we made our model to a database and then wrote some queries on the database that both the school will run manually to generate rapports and some quires that will be run programmatically through the website. To be able to get some information and test the queries there had to be some dummy values and that was also created in this task. The last task is about programmatic access and a program that can access the database was implemented. The program handles the instrument rentals system. The program can add and terminate rentals and get show which instruments is available to rent.

2 Literature Study

2.1 Task 1 Conceptual model

The first thing I did before starting this task was to watch the lecture on introduction to databases and the lecture on concept models that was posted on the Canvas course page. As mentioned in the introduction the model was made in Astah and I was not familiar with the program, so I had to research the program before starting with the task. Most of the information I gathered was from various websites and videos posted on YouTube. I also got some information from the Course literature about inheritance.

2.2 Task 2, Logical and Physical Model

As in task one I started with the lectures about the topic and then filled in the information gasps with the course literature and the internet. I watched some additional videos on YouTube about the Normalization forms and a website called geeksforgeeks helped a lot with the understating what keys was and the differences between them.

2.3 Task 3, SQL

I started with the lecture about SQL and then I did the non-graded quiz. The lecture was great for understanding OLAP and other overall basic information but I did not fell that it was enough to write harder queries. I got some questions wrong on the quiz so I did the website

W3school questions and I looked up the syntax. After that I did the lab questions and then I did the quiz on Canvas again and I went a lot better.

2.4 Task 4, Programmatic Access

For this task I relied mostly on the lectures about database application. I felt that the lecture was enough to at least start with the task and there was a lot of trails and end errors. The first problem was to create a connection to the database and after a couple of hours I found a page on Stackoverflow that helped me. I used the IDE IntelliJ and I had to create a path to the driver jar file. I also used the JDBC-Bank GitHub as a cheat sheet whenever I got stuck and I could not find anything on the internet.

3 Method

3.1 Task 1 Conceptual model

The purpose of the conceptual model is to model the reality and get a picture of how the business is built. The program that was used is called Astah. The first step for creating the conceptual model is to identify nouns all nouns from the project instruction text. These nouns will potentially be entities in our final model. The next step is to go through a list that will help us check that all entities have been found and even get more potential entities. After that a cleanup process begins and the unnecessary entities gets removed and some of the entities that are left gets more appropriate names. The Next step is to add attributes to the entities. Attributes can only be represented as String, Boolean, Number and Time so I went through all the entities and found attributes that would fit in.

After I got all entities and attributes done I started to add all relationships between the entities. The relationships also got a small text that tells us about the reason for the relationship. Crow foot notation was used to be able to know the cardinality of the relations. The last step was to add the cardinality of every attribute and write a small note if the attribute cannot be null.

- **Transactions**, selling or buying a product or service
- **Products or services**, what is sold or bought in the transaction
- **Roles** of peoples and organizations involved in the transaction
- **Places**, maybe where a transaction is performed
- **Records of a transaction**, for example contract, receipt
- **Events**, often with a time and place
- **Physical objects**
- **Devices**, are probably physical objects
- **Descriptions** of things
- **Catalogs**, where the descriptions are stored
- **Systems**, software or hardware that is collaborating with the system for which we are creating the DM
- **Quantities and units**, for example length, meter, currency, fee
- **Resources**, for example time, information, work force

Figure 1. Picture of the category list

3.2 Task 2, Logical and Physical Model

The model that was created in this task was made from the conceptual model that was made in the previous task. The differences between the conceptual model and the model created in this task is that it provides more details about the database for example Attribute datatypes and primary keys. As mentioned in the introduction, only one model was created in this task and it's a combination between a logical and physical model but it's mainly a logical with an enough characteristic of a physical model to create a database from the model. The logical model defines what to store and is not adapted to any particular DBMS (we used PostgreSQL). The logical model also does not handle physical storage like views for example. The Physical model is adapted to the specific DBMS and that's why our created model is a mixture of both models.

The first step was to create a table for each entity that was in the concept model and then insert a column for each attribute with cardinality 0..1 and 1..1. The attributes with higher cardinality needed their own table so we created a table to all those attributes. The next step was to declare datatype and since we use PostgreSQL there are no performance impact if we used Char or Varchar so the only datatypes we used is Varchar and timestamp (no need for text in our database). After that, small notes were added to tell if a certain attribute was unique or not allowed to be null.

The model needs primary and foreign keys so that was added. First the primary keys were added to all strong entity and a strong entity is an entity that is not dependent on any other entity in the model. Since we use Postgres we left notes that's said with id should be serial meaning that the id will be increment automatically when adding another row in the table. Foreign keys were added to the entities with one-to-one or one-to-many relationships to an entity that had a primary key. After that we consider the foreign key constraints to the weaker entities. There are three constraints Default is not action at all, allow which is to Set to null and lastly Delete which means that the row automatically deletes when it's removed from the strong entity.

The last step was to handle the many to many relationships. This problem was solved by adding a cross reference table for each many to many relations.

After all these steps the model should be normalized (3NF). 1NF is that each table contains max one value and no duplicate rows. 2NF is that the table is normalized to 1NF and that all primary keys is a single column that is independent. And finally, 3NF is that the table is normalized to 2NF and has not transitive functional dependencies.

We also created in a script that create the entire database in this task. The Astah program had had a function that directly turned our model into SQL code and from there we added small changes manually. Another script hade to be made to insert data into our database. The script contains some insert into queries and the dummy data was received from a website called moockaro (random data generator) and some data we entered manually.

3.3 Task 3, SQL

The goal of this task is to create OLAP queries and we mainly wrote some queries that executed certain functions that were given to use in the project instruction page in canvas. The code was written in Visual Studio Code and I used PostgreSQL (PgAdmin/Psql (shell)) as the database management software. We had to make more dummy data to test the quires and for that we used moockaro again. Our testing method was fairly simple, we just calculated the results manually and then we confirmed that the queries output gave us the same result. There was no step-by-step method in this task, we just did the queries one after another and made changes to the database created in task two when it was necessary.

3.4 Task 4, Programmatic Access

In this task we handle some OLTP queries that was given to us from the project instructions page on canvas. We also had to create a program that can execute SQL statements from the program on our database. We wrote that program in Java and I used an IDE called IntiliJ and we chose not to use any code from the GitHub Repository. The first thing we did was to connect the driver manager to the database. After the database was connected we made a switch case so that enter in which queries that should be executed. That connections were used to create all the statements in all the methods. We made prepared statements () for all our queries to improve performance and to protect against SQL injection attacks. Lastly we made the logic that would run the queries, print out the results that we wanted and made sure that restrictions were in place for example restrict the number of rentals a student can have.
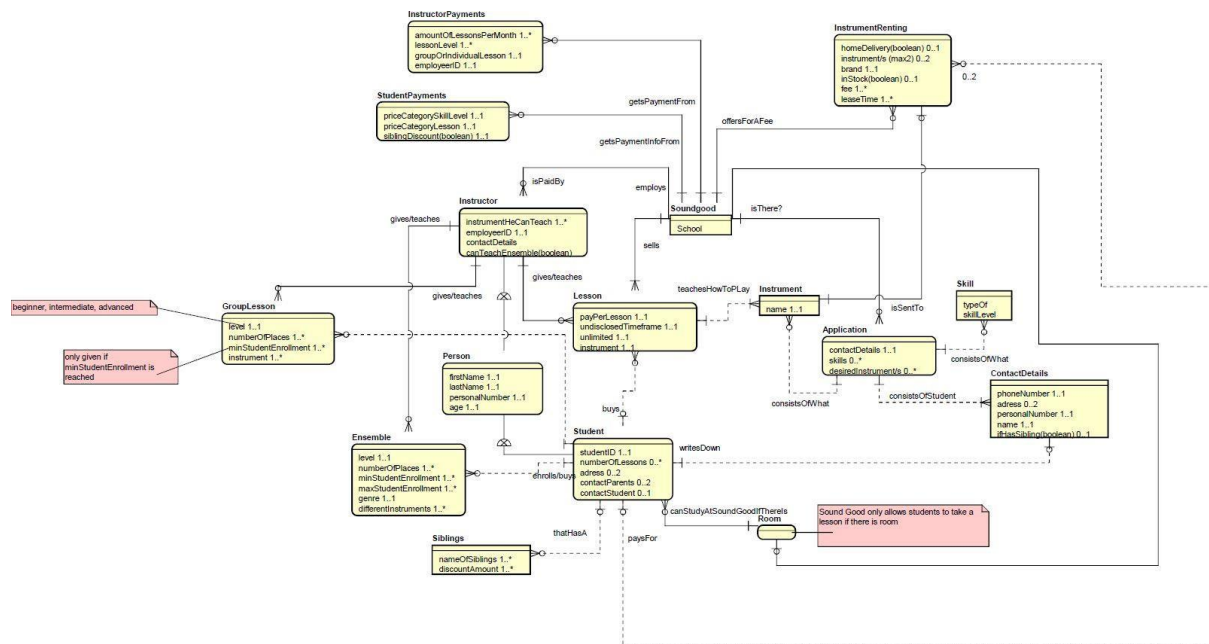
4 Result

4.1 Task 1 Conceptual model

Figure 2. Picture of the conceptual model

Link to the GitHub: https://github.com/NoelMT/Soundgood-Music-School-Project/tree/main/Task1

In this task a conceptual model for the Soundgood music school database and the model that was created can be seen in figure 2. The model contains in total sixteen entities.

Soundgood:

This entity connects most of the other entities to the school. Its only attribute is school which contains the school's name. It has not really any other purpose then to connect everything to the school.

Intrument_renting:

This entity handles the instrument renting side of the database. It contains the attributes homedelivery, Instruments, brand, inStock, fee, leaseTime. This entity holds information about instruments that are available to rent and some information about price. It also contains information about when the instrument was returned by a student.

InstructorPayments:

InstructorPayments contains four attributes, amountOfLessonsPerMonth, lessonLevel, groupOrInduvidualLesson and employerID. The plan for this entity was to hold data of payments that have been made from the school to an instructor and payments that is going to be made from the school to the instructor. The table also contains how many lessons an instructor has taken per month.

StudentPayments:

Student Payments only holds three attributes, priceCategorySkillLevel, priceCategoryLesson and siblingDiscount. These three attributes can be used to calculate how much a student is supposed to pay the school. It contains if a student has a sibling discount and at which skill level the student is taking lessons.

Instructor:

This table contains all information about the instructors at the music school. The attributes are, instrumentHeCanTeach, employeerID, contactDetails, canTeachEnsemble. Since not all instructors can teach ensemble lessons we have a Boolean attribute that shows which instructors can teach ensemble lessons. We also got some contact details to the instructor and employeerID to identify each instructor. Lastly we got an attribute to tell what instrument each instructor can teach. This entity is inherited by the person class and person contains information like first and last name, age, and personal number.

Student:

Student is similar to Instructor, it contains information about the students in the school. The entity contains the following attributes, StudentID, NumberOfLessons, address, contactParents, contactStudent. StudentID is used to identify each student and we got some contact information about the student and the students' parents. The entity also contains the number of lessons a student has taken in total. This entity is inherited by the person class and person contains information like first and last name, age, and personal number. Student also have a relationship with an entity called Siblings. That entity holds information about that students' siblings if the students have any siblings enrolled in the school. The sibling entity purpose is to get the discount amount for the student.

Lesson:

The SoundGood musical school offers three types of lessons (lessons, group lessons and ensemble). Each of these lesson type got their own entity in our conceptual model. Normal lessons have following attributes, Payperlesson, undisclosed_timeframe, unlimited,instrument. This class can hold unlimited number of lessons and have information about the price of a lesson, at which time the lesson was held and which instrument was used at the lesson. The ensemble entity contains information about at which level the ensemble lesson was held at, the number of seats that the ensemble lessons can take, the minimum students that are needed to have an ensemble and the maximum number of students that is allowed, the genre of the ensemble lesson and lastly all instruments that was used in ensemble lesson. Group lesson is similar to ensemble. The information that is available is the following, which level the lesson was held at, number of seats in the group lesson, the minimum number of students that needs to be enrolled in the lesson and which instrument that was used.

Application:

Application contains three attributes, contactDetails, skills and desired instrument. This class handles the applications of new potential member. Before a new student is accepted the entity checks that there is room for new student. Contact details about the potential student is gathered in a connected relationship called ContactDetails. In that entity information like phone number, address, personal number, name and even if the student already has a sibling in the school. It also collects information the current skill level of the potential new student.
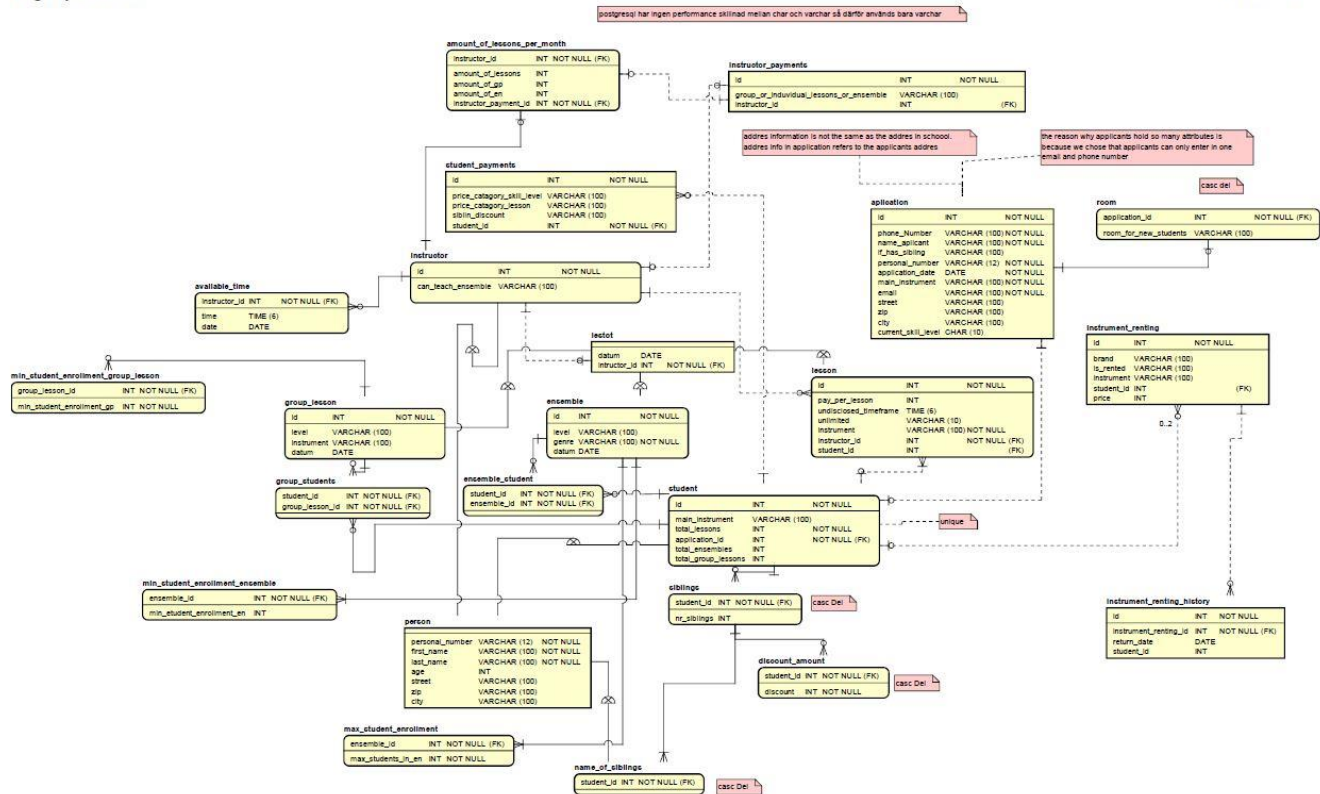
Figure 3. Picture of the Logical/physical model

4.2 Task 2, Logical and Physical Model

Link to the GitHub: https://github.com/NoelMT/Soundgood-Music-School-Project/tree/main/Task2

In this task the logical/physical model was created from the conceptual model that was made in the previous task. The finished model can be seen in figure 3 and it contains in total twenty-three entities in total.

Application:

Application still handles the applications from the new students. The following information is available from this entity, phone number, name, if the applicant has siblings in the school, personal number, application date, main instrument, email, address information and current skill level. Most of these attributes are not allowed as null values, address and if the student have a sibling is the only attributes that are allowed to be null. The table primary id is application id which allow us to identify

each application. It is connected to another table that use application id to see if there is enough room for a new student.

Instructor:

This table contains all instructor in the school. The primary key is instructor_id and there is also an attribute to tell which instructors can hold ensemble lessons. Person is inherited so that we can get some contact information of each instructor. There is a table called available_time connected and with instructor id as foreign key. The purpose for this table is that instructors can enter in at which time and date that they are available. Instructor_payments also use instructor id as a foreign key while having its own primary key so that the school always can identify payment records. We also have a table called lestot and it contains every single lesson that have been held and is going to be held in the school.

Lessons:

All three lesson types have their own entity and all of them is inherited by lestot. Lesson has information about the lessons, price per session, the date and time of the lesson and also which student attended the lesson. Student id is an FK in lesson. Ensembles have information about the level of the ensemble, genre, and the date that the lesson was held. From there we have a table called ensamble_student who holds information about which ensemble each student has attended. The table use student id and ensemble id as primary keys, this table solves the many-to-many relationship between ensemble and student. A similar solution was made to group lesson (group_students). Group lesson is similar to ensemble id but have a column that shows which instrument that was used for each lesson.

Student:

Student is also inherited by person and the table handles all information about the students at the school. The primary key is student id and the information available in the table is the students' main instruments, the total number of lessons each student has taken, application id as a foreign key so that the application is connected to each student. Students have a relationship with siblings as well, siblings only contain the nr of sibling a student has and use student id as the primary key. From there we have a table that calculate the discount amount and a table containing information about the sibling.

Instrument_renting:

Instrument renting handles the data about the instrument rentals in the school. Each student is allowed to rent max two instrument. The table have instrument id as the primary key to be able to identify all instrument that the school have. Information like brand, instrument, price and if the instrument is available to rent is found in this table. Student id is an FK because the school needs to know which student has rented which instrument. All rented and returned instruments have a table called instrument_renting_history. The table have instrument_renting_history_id as primary key and the table contains information about which student have rented an instrument and at the date that the instrument was returned.

Script that creates the database

This script was created with the Astah program. Astah have a function that automatically exports the model to SQL code. Note that not all tables are being used and some of them do not even exist in the model anymore since some changes were made after the export was done. The script can be found here: https://github.com/NoelMT/Soundgood-Music-School-Project/blob/main/Task2/finaldb.sql

Script that inserts data

This script was created manually but some data was gotten from a random data generator called Moockaro. Note that we have kept the number of inserts has been kept minimal but all queries have been tested and in a couple of cases we have updated some rows manually to be able to get the outcome that we were testing.

```
insert into student (personal_number, first_name, last_name, age, street, zip,
city, id, main_instrument, total_lessons, aplication_id, total_ensembles,
total_group_lessons) values ('608336704864', 'Sher', 'Rawsen', 9, '19 Scoville
```

```sql
Hill', '3', 'Longgang', '3372', null, 1, '2213', 0, 0); insert into student
(personal_number, first_name, last_name, age, street, zip, city, id,
main_instrument, total_lessons, aplication_id, total_ensembles,
total_group_lessons) values ('165487277245', 'Elihu', 'Milburn', 29, '72462
Anthes Avenue', '6', 'Sukkozero', '3902', null, 2, '1260', 0, 0); insert into
student (personal_number, first_name, last_name, age, street, zip, city, id,
main_instrument, total_lessons, aplication_id, total_ensembles,
total_group_lessons) values ('774429691466', 'Allie', 'Nancekivell', 14,
'82910 David Court', '193', 'Petaling Jaya', '1082', null, 1, '6065', 0, 0);
insert into student (personal_number, first_name, last_name, age, street,
zip, city, id, main_instrument, total_lessons, aplication_id,
total_ensembles, total_group_lessons) values ('828944401352', 'Vin',
'Muneely', 19, '38 Porter Center', '33837', 'Lukou', '8631', null, 1, '0948',
2, 1); insert into student (personal_number, first_name, last_name, age,
street, zip, city, id, main_instrument, total_lessons, aplication_id,
total_ensembles, total_group_lessons) values ('859800086312', 'Elsinore',
'Van Hesteren', 28,
'7364 Eliot Street', '8502', 'Gaocun', '7491', null, 2, '3854', 1, 1);
 insert into
instrument_renting(id,brand,is_rented,instrument,price,student_id) values
('1','yamaha',TRUE, 'piano',1200,'3372'); insert into
instrument_renting(id,brand,is_rented,instrument,price,student_id) values
('2','pioner',TRUE, 'gitar',400,'3902');   --3902 insert into
instrument_renting(id,brand,is_rented,instrument,price,student_id) values
('3','Arvada',FALSE, 'fiol',600,null);




insert into instructor (personal_number, first_name, last_name, age, id,
street, zip, city, can_teach_ensemble) values ('997137283954', 'Beverlee',
'Gritsaev', 51, '2482', '20364 Spaight Place', '8164', 'Sanankerto', false);
insert into instructor (personal_number, first_name, last_name, age, id,
street, zip, city, can_teach_ensemble) values ('405608868629', 'Ivory',
'Habbema', 26, '9367', '611 Hooker Road', '89', 'Huanshan', true);

insert into lesson (id, instructor_id, student_id, instrument, datum) values
('1',  '2482', '3372', 'piano', '2022-01-07');
insert into lesson (id, instructor_id, student_id, instrument, datum) values
('2',  '2482', '3902', 'piano', '2021-02-02'); insert into lesson (id,
instructor_id, student_id, instrument, datum) values
('3',  '2482', '3902', 'piano', '2021-03-12');
```

```
insert into lesson (id, instructor_id, student_id, instrument, datum) values
('4', '2482', '1082', 'piano', '2021-04-18'); insert into lesson (id,
instructor_id, student_id, instrument, datum) values
('5',  '9367', '8631', 'piano', '2021-06-09'); insert into lesson (id,
instructor_id, student_id, instrument, datum) values
('6',  '9367', '7491', 'piano', '2021-10-10'); insert into lesson (id,
instructor_id, student_id, instrument, datum) values ('7', '9367', '7491',
'piano', '2021-12-05');
 insert into ensemble (id, level, genre, instructor_id, datum, max_seats)
values ('3363', 'Advanced', 'gospel band', '9367', '2021-03-07', '1');
insert into ensemble (id, level, genre, instructor_id, datum, max_seats)
values ('0126', 'Beginner', 'punk rock', '9367', '2022-01-17', '2'); --
change date to any date in the next week to be able to test case insert into
ensemble (id, level, genre, instructor_id, datum, max_seats) values ('6590',
'Intermediate', 'punk rock', '9367', '2021-05-07', '5');
 insert into ensemble_students(student_id, ensemble_id) values
('7491','0126'); insert into ensemble_students(student_id, ensemble_id)
values ('8631','3363'); insert into ensemble_students(student_id,
ensemble_id) values ('8631','6590');


insert into group_lesson (id, level, instructor_id, datum) values ('1',
'Beginner', '2482', '2021-02-07'); insert into group_lesson (id,
level, instructor_id, datum) values ('2', 'Advanced', '9367', '2021-
04-07');
 insert into group_students(student_id, group_lesson_id) values
('8631','1'); insert into group_students(student_id, group_lesson_id)
values ('7491','2');
UPDATE lesson SET instrument = 'piano';
```

4.3 Task 3, SQL

Link to the GitHub: https://github.com/NoelMT/Soundgood-Music-School-Project/tree/main/Task3

The first assignment of this task was *"Show the number of lessons given per month during a specified year. It shall be possible to retrieve the total number of lessons per month (just one number per month) and the specific number of individual lessons, group lessons and ensembles (three numbers per month). This query is expected to be performed a few times per week."*

The assignment was solved with these three queries:

```sql
        SELECT 'This is single lessons per month in year
2021'; select extract(MONTH from lesson.datum) AS MONTH,
extract(year from datum) AS year,
```

```
  count(*) from lesson where extract(year from datum) = 2021 group by
extract(MONTH from datum), extract(year from datum);


        SELECT 'This is group lessons per month in year 2021';
select extract(MONTH from datum) AS MONTH,  extract(year from
datum) AS year,    count(*) from group_lesson where extract(year
from datum) = 2021   group by extract(MONTH from datum),
extract(year from datum);
        select 'This is ensamble lessons per month in year 2021'; select
extract(MONTH from datum) AS MONTH, extract(year from datum) AS year,
count(*) from ensemble  where extract(year from datum) = 2021   group by
extract(MONTH from datum), extract(year from datum);
    SELECT 'This is table with all lessons per month in year 2021'; select
extract(MONTH from datum) AS MONTH, extract(year from datum) AS year,
count(*) from lestot   where extract(year from datum) = 2021
group by extract(MONTH from datum), extract(year from datum);


The first select statement selects three columns, the month and year from column datum (lesson) and
the number of elements selected using count(*). The data is selected from the table lessons that
contains all single lessons. And where clause makes sure that only the lessons that was had in 2021 is
selected therefore meeting the requirement that only lessons counted is the was taken during a
specified year (2021). We also group by month and year so that we count all the lessons during each
month of the year 2021.
```

The second and third select statements work exactly the same as the first one but instead of getting the
data from the single lessons they get the data from group_lesson and ensemble. The last statement is
also similar to the previous selects but this statement gets the data from lestot which is inherited by all
three of the lesson types. That means that all of the lessons and their respective date will be in in lestot
and from there we have access to all lessons that have been held in 2021.

The second assignment of this task was *"The same as above but retrieve the average number of
lessons per month during the entire year, instead of the total for each month."*

```
select extract(MONTH from datum) AS MONTH, extract(year from datum) AS year,
(CAST(count(*) AS DECIMAL(4,2) )  / 12) as  Average
from lestot where extract(year from datum) = 2021 group by extract(MONTH from
datum), extract(year from datum);

--Avarage of single lessons
select extract(MONTH from lesson.datum) AS MONTH, extract(year from datum) AS
year, (CAST(count(*) AS DECIMAL(4,2) )  / 12) as  Average from lesson where
extract(year from datum) = 2021 group by extract(MONTH from datum),
extract(year from datum);

--Avarage of group lessons
```

```sql
 select extract(MONTH from datum) AS MONTH, extract(year from datum) AS year,
(CAST(count(*) AS DECIMAL(4,2) )  / 12) as  Average  from
group_lesson where extract(year from datum) = 2021 group by
extract(MONTH from datum), extract(year from datum);

--Avarage of ensamble select extract(MONTH from datum) AS MONTH,
extract(year from datum) AS year,
(CAST(count(*) AS DECIMAL(4,2) )  / 12) as  Average from ensemble where
extract(year from datum) = 2021 group by extract(MONTH from datum),
extract(year from datum);
```

All of the select statements work exactly the same as the previous assignment. The only difference is that the average lessons per month in a year is being counted instead of just counting the total number of lessons. We have a small number of lessons so we had to cast to a decimal to be able to see the average and then we divide by 12 (months per year).

The third assignment of this task was *"List all instructors who has given more than a specific number of lessons during the current month. Sum all lessons, independent of type, and sort the result by the number of given lessons. This query will be used to find instructors risking to work too much and will be executed daily."*

The assignment was solved with this query:

```sql
select instructor_id ,extract(month from datum) as month, extract(year from
datum) as year,count(instructor_id) as antal from lestot where
extract(month from datum) = extract(month from current_date)  and
extract(year from datum) = extract(year from current_date)  group by
instructor_id,extract(month from datum),extract(year from datum)  having
count(*) > 0 order by count(*);
```

The select statements columns will be instructor_id, the month and year extracted from lesson.datum and a count of the number of times an instructor have had a lesson. Since the assignment ask for all lesson types we gather the data from lestot and the statemen only picks lessons that occurred in the current month. We get the current month and year by extracting month and year from current_date. Then we group by instructor id so that instructors have a single row for them self and a having clause that only picks the instructors that have over zero lessons this month. Larger than zero will get all teacher that have taught a lesson this month but that number could be changed to anything. Lastly we order by the number of lessons that each instructor have had this month meaning the instructor that have had the most taught lessons will show up at the top.

The fourth and last assignment of this task was *"List all ensembles held during the next week, sorted by music genre and weekday. For each ensemble tell whether it's full booked, has 1-2 seats left or has more seats left"*

The assignment was solved with this query:

```
select id as ensemble_id , datum ,genre,instructor_id,id,max_seats,
CASE
WHEN  (max seats - boked) = 0
THEN 'full boked'

WHEN  (max seats - boked) = 1 or (max seats - boked) = 2
THEN '1-2 seats left!'

WHEN  (max seats - boked) > 2
THEN 'Seats left'

END
from ensemble
inner join booked students on ensemble.id = booked students.ensemble id
where DATE PART('week', datum) = (CAST(to char(current date, 'WW') as
INTEGER)+1)
order by genre, datum;
```

The assignment was solved using a case statement for each case that can occur. The select statement will have 5 columns, ensemble id, date, genre, instructor id and the max seats. Then we use the case statement to decide which output is going to be in the last column that says the status of number of seats left. To check how many seats is left we use boked, boked is created in a view created in the finaldb.sql (line 294 in GitHub) file and stands for the number of booked students. The number is retrieved by doing a select count from ensemble_student who is grouped by ensemble id. Then an inner join is made on the ensemble id. Then we can easily se the number of seats left by subtracting max number of seats with the booked seats. We only pick ensemble lessons that occurs next week and we do that in the where clause. We check that the date of the ensemble is in the next week by taking out the current week number (1-53) and then adding one to it. Lastly we order by genre and then the date.

```
SQL Shell (psql)

final=# \i q1.sql
                        ?column?
---------------------------------------------------------
 This is single lessons per month in year 2021
(1 rad)


 month | year | count
-------+------+-------
     2 | 2021 |      1
     3 | 2021 |      1
     4 | 2021 |      1
     6 | 2021 |      1
    10 | 2021 |      1
    12 | 2021 |      1
(6 rader)


                        ?column?
---------------------------------------------------------
 This is group lessons per month in year 2021
(1 rad)


 month | year | count
-------+------+-------
     2 | 2021 |      1
     4 | 2021 |      1
(2 rader)
```

```
                         ?column?
---------------------------------------------------------------
 This is ensamble lessons per month in year 2021
(1 rad)


 month | year | count
-------+------+-------
     3 | 2021 |     1
     5 | 2021 |     1
(2 rader)


                         ?column?
----------------------------------------------------------------
 This is table with all lessons per month in year 2021
(1 rad)


 month | year | count
-------+------+-------
     2 | 2021 |     2
     3 | 2021 |     2
     4 | 2021 |     2
     5 | 2021 |     1
     6 | 2021 |     1
    10 | 2021 |     1
    12 | 2021 |     1
(7 rader)
```

4.4 Task 4, Programmatic Access

Link to the GitHub: https://github.com/NoelMT/Soundgood-Music-School-Project/tree/main/Task2
The files to create the database and insert data can be found in task 2 in the same repository.

The class is named Main and the first thing that happends when the main function is run is a greating and instructions to see all available commands. Then the person running the application is being asked to enter in DBMS password and the name of the database.

```java
public Connection accesDB(String s, String s1) throws SQLException,
ClassNotFoundException {
        Class.forName("org.postgresql.Driver");
return
DriverManager.getConnection("jdbc:postgresql://localhost:5432/"+s1,
                "postgres", s);
}
```

After entering in that information, a function called accesDB gets called and there the driver gets retrieved, then that driver gets connection to the database that was entered in before and returns that connection object.

```
private static void prepareStatements(Connection c) throws SQLException {

    getAvailInsSTM = c.prepareStatement("select * from instrument_renting
where is_rented = FALSE and instrument = ? ");

    getallSTM = c.prepareStatement("select  *  from  instrument_renting  where
is_rented = TRUE");

    nrRentalsSTM = c.prepareStatement("select count(*) from
instrument_renting where student_id = ?");

    addRentalSTM = c.prepareStatement("update instrument_renting set
is_rented = TRUE, student_id = ? where id = ?");

    revertSTM = c.prepareStatement("update instrument_renting set is_rented
= FALSE, student_id = null where id = ?");

    getStudIdSTM  =  c.prepareStatement("select  *   from  instrument_renting
where id = ? ");

    addHistorySTM = c.prepareStatement("insert into
instrument_renting_history(instrument_renting_id,return_date,student_id)
values(?,CURRENT_DATE,?)"); }
```

When the driver has connected successfully to the database the program sets auto commit to false so that we could choose when to commit. As mentioned previously in the rapport we choose to use prepared statements, above is all of those statements. After that a while loop and a switch case starts to be able to enter in which command is going to be ran.

The first assignment of this task was *"It shall be possible to list all instruments of a certain kind (guitar, saxophone, etc) that are available to rent. Instruments which are already rented shall not be included in the listing. The listing shall show brand and price for each listed instrument."*

The command that handles this first assignment is "list". When the command is entered in it will as the user for a certain kind of instrument. After that a method called rentalPrint() gets ran. That function handles the printing of the available instruments. To print the result, it calls another function called getAvailIns to get the ResultSet. getAvailIns will try to firstly set the instrument that was asked for in the prepared statemen getAvailInsSTM. After that the result set of the query is returned back all the way to rentalPrint and the price and brand is of the available instruments gets printed. In the getAvailIns function there is a try catch block so that if the query fails the catch block will run printStackTrace() and from there we can see where the error occurred and it gets easier to rollback if necessary. This is an example of consistency.

The second assignment of this task was *"It shall be possible to specify which student is renting the instrument, and which instrument is being rented. Since different instruments of the same kind might have different prices, it must be possible to specify exactly which particular instrument to rent, not just any instrument of the desired kind. Remember that a student is not allowed to rent more than two instruments at the same time, your program must check that this limit is not exceeded."*

The corresponding commands is show and add. Show will print out all the instruments that are currently rented and by who the instrument is being rented by. The function showRent gets the resultSet in the same way as the previous assignment but it calls the getallSTM instead .The Add command is used to enlist new rentals. After entering the command, a student id and instrument id will have to be entered in. Then a function called addRent is called and there an if statement checks that the student doesn't already have to ongoing rentals. A function called nrRentals will return the number of rentals of the student id that was entered in earlier. If the student has less than 2 rentals a function called addrentals is called and sends student id, instrument id and the connection to the database. In addrentals a try catch block will try to firstly set the student id and instrument id into the query. Then the update query execute is and lastly after all reads and writes we commit. This is a property of ACID since we only commit after everything have run successfully and if something fails nothing will happen to the table.

The Third and last assignment of this task was *"It shall be possible to terminate an ongoing rental. You are free to decide how the user specifies which rental to terminate. You are not allowed to delete all information about a terminated rental from the database. Instead, the database must still contain all information about the rental, but also show that the rental has been terminated."*

The corresponding command for this task is term. The table instrument_rental contains all instruments that are rented and are not rented so what happends is that if a rented instrument is getting terminated the code makes is_rented false and the student id is removed from the instrument. We add a row in instrument_renting_history which contains the date that the rental was terminated, student id and instrument id. We specify which instrument rental is getting to terminate by selecting the instrument id that is going to be terminated.

All Commands

```
help

-> quit
-> list |enter in a instrument and get brand and price of available instrumnets|
-> show |shows all intruments that are rented and by who|
-> add |Lets a student rent an instrument by entering student id and then instrument id|
-> term |Terminate a rental by entering instrument id and adds it to the rental history|


ENTER NEW COMMAND
|
```

```
ENTER NEW COMMAND
list
choose instrument:
fiol
--------------------------------------------------
Brand: Arvada price: 600
--------------------------------------------------
ENTER NEW COMMAND
```



```
ENTER NEW COMMAND
show
StudentID: 3372 IntrumentID: 1
StudentID: 3902 IntrumentID: 2
ENTER NEW COMMAND
```

5 Discussion
5.1 Task 1 Conceptual model

Are naming conventions followed? Are all names sufficiently explaining?

Is the notation (UML or crow foot) correctly followed?

Is there a reasonable number of entities? Is some important entity missing?

Are there attributes for all data that shall be stored? Is cardinality specified for all attributes?

Are all relevant relations specified? Do all relations have cardinality at both ends and name at least at one end?

Are all business rules and constraints that are not visible in the diagram explained in plain text?

The conceptual model uses two naming conventions. Entity titles use the naming convention upper camel case meaning that the multiple words are joined to one word and the first letter of every word is capitalized for example UpperCammelCase. The attributes follow normal camel-case syntax, its similar to the title naming convention but the first letter of the first word is lower case. One of the most challenging parts of this model was to find good names so that anyone could understand the model. It's hard to know if the names used in the model is easy to understand for everyone but I used a friend that don't have this course to test if the model is understandable to a person who don't know the business and most of it was easy to understand and he got a good first picture of the business from just looking at the model. Therefore, I'm confident to say that the names and the model is sufficiently explaining the reality of the business.

The crow-foot notation was taken from the lecture published on Canvas and it was followed to the best of our ability. It's hard to know what a reasonable number of entity since we followed the steps that were mentioned in the method chapter but we felt that everything important is in the model. After the whole project is finished, looking back at the conceptual model that was made in the beginning of the course I mainly notice that some other relationship should exist and some entities could be removed. The Soundgood entity should be removed since it does not do a lot and only connects entities. There should also be a relationship between student and student payment. The model does not cover the instrument renting history and an entity to keep those records should be added. Another thing that I'm not happy with is the person and contact details entities. Student and instructor inherits person but I also think that application should inherit person since it's a person that fills in the application. The advantage of using inheritance is that there will be less repeating redundant attributes so instead of adding the same attributes to multiple tables we could reuse the same attributes with inheritance. Another thing that we found useful is that we could get list of all rows independent of type for example in our conceptual model all persons are enlisted in person regardless of if the person is a student or instructor. This property was very useful for many queries. The disadvantage of inheritance is that its slower than normal tables and sometime inherited attributes will be left empty. Another thing that might be a disadvantage is that the table can harder to understand especially if the viewer doesn't understand inheritance. A version of how the model would look without inheritance can be found just below *(Figure 4)* . If we chose to inherit from person to contact details we could make some small changes and remove contact details completely which makes the model simpler and easier to understand. The room entity is also not a good solution to see if there is space in the school for new students. Room should be removed and an attribute to Soundgood which keeps track of the total number of students in the school could be added, that way we remove an entity and we make the Soundgood entity more useful. If I had the chance to do the whole model again now after the whole project I would add more notes and explain more of the business rules and constraint. There are some notes but I feel like it would be easier to understand if it was explained in plain text, for example that each student is allowed to rent max two instruments and it is explained if you look at the relationship but it would be easier if a note were added.
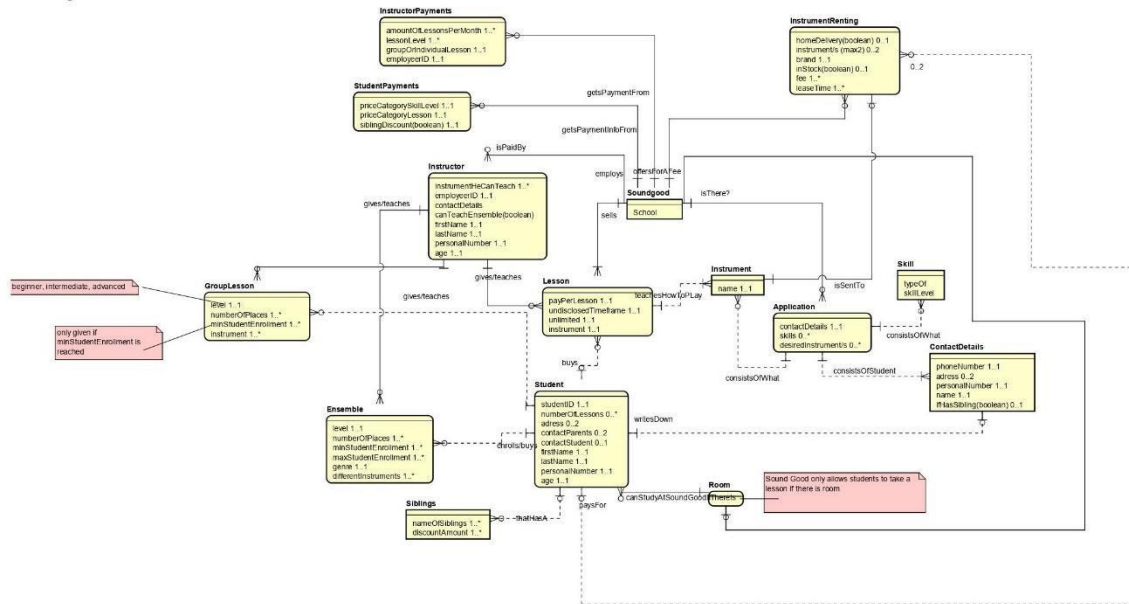
Figure 4. Picture of the Conceptual model if inheritance was not used

I thought that this task was very hard since I the course had just begun and I did not understand how anything of this was relevant to the database. I was also not comfortable with an assignment that had no single correct answer. The model could be made with many different solutions and that was hard for me in the beginning but after a while I started to get an intuition for what solutions was better and how to find those solutions.

5.2 Task 2, Logical and Physical Model

Are naming conventions followed? Are all names sufficiently explaining?

Is the crow foot notation correctly followed?

Is the model in 3NF? If not, is there a good reason why not?

Are all tables relevant? Is some table missing?

Are there columns for all data that shall be stored? Are all relevant column constraints and foreign key constraints specified? Can all column types be motivated?

Can the choice of primary keys be motivated?

Are all relevant relations correctly specified?

Is it possible to perform all tasks listed in section one, Project Description, above?

Are all business rules and constraints that are not visible in the diagram explained in plain text?

The naming convention is different from the naming convention used in the conceptual model. This naming convention is called snake case and is normally used for variables, subroutine names and file names. Snake case means that all letters are lower case and if the name use multiple words there need be an underscore between the words. The naming convention was not used one time in the model. I and that is the table lestot. Since we followed the steps that were mentioned in the method chapter the model should be normalized to 3NF. I know that each table only contains max one value in each model, there are no duplicates and all primary keys is a single independent column (except group_students and ensemble student). I cannot see any transitive functional dependencies so the model should be in 3NF. All necessary tables exist in the model but something that might be good feature is the possibility to add multiple e-mails or phone number. The reason we chose that our database only could store a single e-mail or number is because we thought the school only needed one number and email. Allowing multiple emails would cause a many to many relationships and we wanted to keep those to a minimum. Room can also be changed so that instead of having its on table the number of students in the school and the maximum number of students allowed in the school would be saved as column in the school entity. I also was not that happy with the sibling implementation. Instead of having the siblings and name of sibling's tables we could add personal number or the student id of the sibling in siblings and remove one attribute and name of sibling since we could identify the sibling with the id. The solution used now will cause the database to save the personal information about the same student twice in two different tables (student and name of sibling).

Since PostgreSQL don't suffer any performance differences between varchar and char. That is why varchar is used the most in the model. The only columns that are not varchar are all id's (Keys) and they are set to int. We also use data and timestamp, date is used on the columns that we know that we don't need the exact time and timestamp when we need the time or not sure if it would be needed. Personal number was set to only varchar(12) since we already know the length of a personal number is 12 characters, there is no need for varchar(100). All primary keys can be motivated but there might be some redundancy since personal number also can be used as a primary key but the solution of using instructor id and student id helped with the queries that was made in the later tasks.

Most of the constraints have been left of default since we don't know exactly what the school want in the records. A couple of tables have been set to delete because the seemed completely unnecessary records to keep. The model should be able to perform all tasks that are listed in the project description. More notes should be added so that it's easier to see the rules and constraints but I believe that all necessary constraints are visible in the model.

5.3 Task 3, SQL

Four queries were made in this task but only one of them use view. The three first queries were made before reading what was going to be in the discussion and the after reading the entire task and realizing that view and materialized view is useful depending on how often the query is going to be ran we used started to think about using view in the last query. The first query is only runned a few times a week and it will be used to generate analysis reports so we should have used a view on that query. Views update on a frequent basis which makes sure that we get the newest data to the report . Since the query will be ran infrequently and needs the most updated data it is a better choice than materialized view. The second query should also use views for the same reasons. The third query is a bit more complicated, it will be running daily and data is going to be accessed frequently so it might be better to use a materialized view. But materialized views don't get updated so the view have to be refreshed all

every time the query is ran. So materialized view might be better since the query will be ran often but it has to pay an update cost every time. It's uncertain which is better depending on the update cost. The last query will be going to be ran daily and also needs to be accurate which the newest data have to be used. Student can enlist ensemble lessons all the time meaning that the materialized view will have to be refreshed everytime because we don't want student to enlist in ensemble lessons that are full. Even if it has to be refreshed a lot I still think that materialized view is the better choice since it will be running so frequently. No views were used in the first three queries but the last one use materialized view because of how often the query would be ran and we needed a table with the number of students enlisted in every ensemble lessons.

Disregarding how often the queries will be run not using any view on the first three queries have some advantages. Since the queries are fairly simple we don't have to create a view insert the data into the view

5.4 Task 4, Programmatic Access

Are naming conventions followed? Are all names sufficiently explaining?

Is transaction management correct? Are there ACID transactions, which are committed or rolled back correctly?

Is the program working as expected and does it meet all functional requirements listed in this task?

The program that was made in this task was written in mainly java therefore the naming convention used is camel case. Most functions sufficiently explain the what the function does but the commands are not always self-explanatory. That's why the help command was added. When entering "help" the user will get a text list explaining what each command does and what the user is expected to enter in. The program only commits after all necessary queries have been prepared meaning that the program only commits after every function have run successfully, this is a property of atomicity since ether the whole transaction works or nothing happends. This property was implemented by using try catch blocks and dividing up each step of the transaction in different functions. In the catch statements we do a stack trace and we rollback to where the error occurred. The program meets all the requirements and work but it's still unsure what happends when the user enters in student or instrument id that do not exist. As long as the user enters in data that exist in the DB the program will work as expected. I spent a lot of time trying to get the driver to work. The IDE could not find the driver JAR file automatically even if it was placed in the same directory. After many ours I realized that the IDE needed the path inserted manually. Another problem was that the program was committing automatically and it took some time to find out that the program needs to be set so that it does not auto commit.

GitHub: https://github.com/NoelMT/Soundgood-Music-School-Project