Project Report

Data Storage Paradigms, IV1351

Noel Tesfalidet [Noelt@kth.se](mailto:Noelt@kth.se)

2022-01-06

![A picture containing text

Description automatically generated]()

# 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.

//Kanske flyttas till method

Then after I finished the model I compared it to my project partners model and we discussed about our solutions and merged the models and chose the best of parts of our individual model.

## 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.

Text

Description automatically generated

*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 its mainly a logical with a enough characteristics 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 (CREATE GIT REPOSITORY)

## 4.1 Task 1 Conceptual model

![Diagram

Description automatically generated]()

*Figure 2. Picture of the conceptual model*

REQIREMENTS

Create a conceptual model for the Soundgood music school database

The diagram must be made either in UML or in one of all possible crow foot notations (for example IE notation)

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.

IntrumentRenting:

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.

## 4.2 Task 2, Logical and Physical Model

Diagram, schematic

Description automatically generated

*Figure 3. Picture of the Logical/physical model*

Link to the GitHub[: https://github.com/NoelMT/Soundgood-Music-School-Project/tree/main/Task2](:%20https:/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.

School:

This entity has been expanded from the last model. It now contains more information about the school. The four attributes are school\_name, street, zip, city. School has no keys but are connected to a lot of core entities.

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. Student have a relashionship with siblings as well, siblings only contains 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 a 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. That 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 is being used and some of them do not even exist in the model anymore since some changes were made after the export was done.

CREATE TABLE Instrument (

 name VARCHAR(100) NOT NULL

);

CREATE TABLE instrumentGP (

 intrument\_in\_group\_lesson VARCHAR(100)

);

CREATE TABLE instrumentHeCanTeach (

 Attribute9 CHAR(10)

);

CREATE TABLE number\_of\_lessons (

);

CREATE TABLE number\_of\_places (

 number\_of\_Students\_In\_GP INT

);

CREATE TABLE person (

 personal\_number VARCHAR(12) NOT NULL,

 first\_name VARCHAR(100) NOT NULL,

 last\_name VARCHAR(100) NOT NULL,

 age INT,

 street VARCHAR(100),

 zip VARCHAR(100),

 city VARCHAR(100)

);

CREATE TABLE school(

 school\_name VARCHAR(100) NOT NULL,

 street VARCHAR(100),

 zip VARCHAR(100),

 city VARCHAR(100)

);

CREATE TABLE Skill (

 type\_of VARCHAR(300) NOT NULL

);

CREATE TABLE skills (

 level\_of\_skill VARCHAR(100)

);

CREATE TABLE aplication (

 id INT NOT NULL UNIQUE,

 phone\_Number VARCHAR(100) NOT NULL,

 name VARCHAR(100) NOT NULL,

 if\_has\_sibling VARCHAR(100),

 personal\_number VARCHAR(12) NOT NULL,

 aplication\_date DATE NOT NULL,

 main\_instrument VARCHAR(100) NOT NULL,

 email VARCHAR(100) NOT NULL,

 street VARCHAR(100),

 zip VARCHAR(100),

 city VARCHAR(100),

 current\_skill\_level CHAR(10)

);

CREATE TABLE ContactDetails (

 phoneNumber VARCHAR(10),

 personalNumber VARCHAR(10),

 name VARCHAR(10),

 ifHasSibling CHAR(10),

 id INT UNIQUE

);

CREATE TABLE instructor (

 id INT NOT NULL UNIQUE,

 can\_teach\_ensemble VARCHAR(100)

)INHERITS (person);

ALTER TABLE Instructor ADD CONSTRAINT PK\_Instructor PRIMARY KEY (id);

CREATE TABLE instructor\_payments (

 id INT NOT NULL UNIQUE,

 group\_or\_induvidual\_lessons\_or\_ensemble VARCHAR(100),

 instructor\_id INT

);

ALTER TABLE instructor\_payments ADD CONSTRAINT PK\_instructor\_payments PRIMARY KEY (id);

CREATE TABLE room (

 aplication\_id INT NOT NULL,

 room\_for\_new\_students VARCHAR(100)

);

ALTER TABLE room ADD CONSTRAINT PK\_room PRIMARY KEY (aplication\_id);

CREATE TABLE student (

 id INT NOT NULL UNIQUE,

 main\_instrument VARCHAR(100),

 total\_lessons INT NOT NULL,

 aplication\_id INT NOT NULL,

 total\_ensembles INT,

 total\_group\_lessons INT

)INHERITS (person);

ALTER TABLE student ADD CONSTRAINT PK\_student PRIMARY KEY (id);

CREATE TABLE student\_payments (

 id INT NOT NULL UNIQUE,

 price\_catagory\_skill\_level VARCHAR(100),

 price\_catagory\_lesson VARCHAR(100),

 siblin\_discount VARCHAR(100),

 student\_id INT NOT NULL

);

ALTER TABLE student\_payments ADD CONSTRAINT PK\_student\_payments PRIMARY KEY (id);

CREATE TABLE amount\_of\_lessons\_per\_month (

 instructor\_id INT NOT NULL,

 amount\_of\_lessons INT,

 amount\_of\_gp INT,

 amount\_of\_en INT,

 instructor\_payment\_id INT NOT NULL

);

ALTER TABLE amount\_of\_lessons\_per\_month ADD CONSTRAINT PK\_amount\_of\_lessons\_per\_month PRIMARY KEY (instructor\_id);

CREATE TABLE available\_time (

 instructor\_id INT NOT NULL,

 time TIMESTAMP(6),

 date DATE

);

ALTER TABLE available\_time ADD CONSTRAINT PK\_available\_time PRIMARY KEY (instructor\_id);

CREATE TABLE lestot(

    instructor\_id INT NOT NULL,

    datum DATE

    );

CREATE TABLE ensemble (

 id INT NOT NULL UNIQUE,

 level VARCHAR(100),

 genre VARCHAR(100) NOT NULL,

 max\_seats INT

)INHERITS(lestot);

ALTER TABLE ensemble ADD CONSTRAINT PK\_ensemble PRIMARY KEY (id);

CREATE TABLE group\_lesson (

 id INT NOT NULL UNIQUE,

 level VARCHAR(100),

 instrument VARCHAR(100)

)INHERITS(lestot);

ALTER TABLE group\_lesson ADD CONSTRAINT PK\_group\_lesson PRIMARY KEY (id);

CREATE TABLE instrument\_renting (

 id INT NOT NULL UNIQUE,

 brand VARCHAR(100),

 is\_rented BOOLEAN NOT null,

 instrument VARCHAR(100),

 price int,

 student\_id INT

);

ALTER TABLE instrument\_renting ADD CONSTRAINT PK\_instrument\_renting PRIMARY KEY (id);

CREATE TABLE instrument\_renting\_history (

 instrument\_renting\_id INT NOT NULL,

 return\_date DATE,

 student\_id INT

);

ALTER TABLE instrument\_renting\_history ADD CONSTRAINT FK\_instrument\_renting\_history\_0 FOREIGN KEY (instrument\_renting\_id) REFERENCES instrument\_renting(id);

CREATE TABLE lease\_time (

 instrument\_renting\_id INT NOT NULL,

 rented\_time TIME(6)

);

ALTER TABLE lease\_time ADD CONSTRAINT PK\_lease\_time PRIMARY KEY (instrument\_renting\_id);

CREATE TABLE Lesson (

 id INT NOT NULL UNIQUE,

 pay\_per\_lesson INT,

 undisclosed\_timeframe TIME(6),

 unlimited VARCHAR(10),

 instrument VARCHAR(100),

 student\_id INT

)INHERITS(lestot);

ALTER TABLE Lesson ADD CONSTRAINT PK\_Lesson PRIMARY KEY (id);

CREATE TABLE max\_student\_enrollment (

 ensemble\_id INT NOT NULL,

 max\_students\_in\_en INT NOT NULL

);

ALTER TABLE max\_student\_enrollment ADD CONSTRAINT PK\_max\_student\_enrollment PRIMARY KEY (ensemble\_id);

CREATE TABLE ensemble\_students (

 student\_id INT NOT NULL,

 ensemble\_id INT NOT NULL

);

ALTER TABLE ensemble\_students ADD CONSTRAINT FK\_ensemble\_student\_0 FOREIGN KEY (student\_id) REFERENCES student (id);

ALTER TABLE ensemble\_students ADD CONSTRAINT FK\_ensemble\_student\_1 FOREIGN KEY (ensemble\_id) REFERENCES ensemble(id);

CREATE TABLE group\_students (

 student\_id INT NOT NULL,

 group\_lesson\_id INT NOT NULL

);

ALTER TABLE group\_students ADD CONSTRAINT FK\_group\_student\_0 FOREIGN KEY (student\_id) REFERENCES student(id);

ALTER TABLE group\_students ADD CONSTRAINT FK\_group\_student\_1 FOREIGN KEY (group\_lesson\_id) REFERENCES group\_lesson(id);

CREATE TABLE min\_student\_enrollment\_ensemble (

 ensemble\_id INT NOT NULL,

 min\_etudent\_enrollment\_en INT

);

ALTER TABLE min\_student\_enrollment\_ensemble ADD CONSTRAINT PK\_min\_student\_enrollment\_ensemble PRIMARY KEY (ensemble\_id);

CREATE TABLE min\_student\_enrollment\_group\_lesson (

 group\_lesson\_id INT NOT NULL,

 min\_student\_enrollment\_gp INT NOT NULL

);

ALTER TABLE min\_student\_enrollment\_group\_lesson ADD CONSTRAINT PK\_min\_student\_enrollment\_group\_lesson PRIMARY KEY (group\_lesson\_id);

CREATE TABLE siblings (

 student\_id INT NOT NULL,

 nr\_siblings INT

);

ALTER TABLE siblings ADD CONSTRAINT PK\_siblings PRIMARY KEY (student\_id);

CREATE TABLE differentInstrument (

 ensemble\_id INT NOT NULL,

 instrument VARCHAR(300)

);

ALTER TABLE differentInstrument ADD CONSTRAINT PK\_differentInstrument PRIMARY KEY (ensemble\_id);

CREATE TABLE discount\_amount (

 student\_id INT NOT NULL,

 discount INT NOT NULL

);

ALTER TABLE discount\_amount ADD CONSTRAINT PK\_discount\_amount PRIMARY KEY (student\_id);

CREATE TABLE fee (

 instrument\_renting\_id INT NOT NULL,

 price INT

);

ALTER TABLE fee ADD CONSTRAINT PK\_fee PRIMARY KEY (instrument\_renting\_id);

CREATE TABLE name\_of\_siblings (

 student\_id INT NOT NULL

)INHERITS (person);

ALTER TABLE name\_of\_siblings ADD CONSTRAINT PK\_name\_of\_siblings PRIMARY KEY (student\_id);

CREATE VIEW booked\_students AS

SELECT ensemble\_id, count(\*) as boked

FROM ensemble\_students

group by ensemble\_id

order by ensemble\_id;

/\*ALTER TABLE ContactDetails ADD CONSTRAINT FK\_ContactDetails\_0 FOREIGN KEY (id) REFERENCES aplication (id);\*/

ALTER TABLE instructor\_payments ADD CONSTRAINT FK\_instructor\_payments\_0 FOREIGN KEY (instructor\_id) REFERENCES Instructor (id);

/\*ALTER TABLE room ADD CONSTRAINT FK\_room\_0 FOREIGN KEY (aplication\_id) REFERENCES aplication (id);\*/

/\*ALTER TABLE student ADD CONSTRAINT FK\_student\_0 FOREIGN KEY (aplication\_id) REFERENCES aplication (id);\*/

ALTER TABLE student\_payments ADD CONSTRAINT FK\_student\_payments\_0 FOREIGN KEY (student\_id) REFERENCES student (id);

ALTER TABLE amount\_of\_lessons\_per\_month ADD CONSTRAINT FK\_amount\_of\_lessons\_per\_month\_0 FOREIGN KEY (instructor\_id) REFERENCES Instructor (id);

ALTER TABLE amount\_of\_lessons\_per\_month ADD CONSTRAINT FK\_amount\_of\_lessons\_per\_month\_1 FOREIGN KEY (instructor\_payment\_id) REFERENCES instructor\_payments (id);

ALTER TABLE available\_time ADD CONSTRAINT FK\_available\_time\_0 FOREIGN KEY (instructor\_id) REFERENCES Instructor (id);

ALTER TABLE ensemble ADD CONSTRAINT FK\_ensemble\_0 FOREIGN KEY (instructor\_id) REFERENCES Instructor (id);

--ALTER TABLE ensemble ADD CONSTRAINT FK\_ensemble\_1 FOREIGN KEY (student\_id) REFERENCES student (id);

ALTER TABLE group\_lesson ADD CONSTRAINT FK\_group\_lesson\_0 FOREIGN KEY (instructor\_id) REFERENCES Instructor (id);

--ALTER TABLE group\_lesson ADD CONSTRAINT FK\_group\_lesson\_1 FOREIGN KEY (student\_id) REFERENCES student (id);

ALTER TABLE instrument\_renting ADD CONSTRAINT FK\_instrument\_renting\_0 FOREIGN KEY (student\_id) REFERENCES student (id);

ALTER TABLE lease\_time ADD CONSTRAINT FK\_lease\_time\_0 FOREIGN KEY (instrument\_renting\_id) REFERENCES instrument\_renting (id);

ALTER TABLE Lesson ADD CONSTRAINT FK\_Lesson\_0 FOREIGN KEY (instructor\_id) REFERENCES Instructor (id);

ALTER TABLE Lesson ADD CONSTRAINT FK\_Lesson\_1 FOREIGN KEY (student\_id) REFERENCES student (id);

ALTER TABLE max\_student\_enrollment ADD CONSTRAINT FK\_max\_student\_enrollment\_0 FOREIGN KEY (ensemble\_id) REFERENCES ensemble (id);

ALTER TABLE min\_student\_enrollment\_ensemble ADD CONSTRAINT FK\_min\_student\_enrollment\_ensemble\_0 FOREIGN KEY (ensemble\_id) REFERENCES ensemble (id);

ALTER TABLE min\_student\_enrollment\_group\_lesson ADD CONSTRAINT FK\_min\_student\_enrollment\_group\_lesson\_0 FOREIGN KEY (group\_lesson\_id) REFERENCES group\_lesson (id);

ALTER TABLE siblings ADD CONSTRAINT FK\_siblings\_0 FOREIGN KEY (student\_id) REFERENCES student (id);

ALTER TABLE differentInstrument ADD CONSTRAINT FK\_differentInstrument\_0 FOREIGN KEY (ensemble\_id) REFERENCES ensemble (id);

ALTER TABLE discount\_amount ADD CONSTRAINT FK\_discount\_amount\_0 FOREIGN KEY (student\_id) REFERENCES siblings (student\_id);

ALTER TABLE fee ADD CONSTRAINT FK\_fee\_0 FOREIGN KEY (instrument\_renting\_id) REFERENCES instrument\_renting (id);

ALTER TABLE name\_of\_siblings ADD CONSTRAINT FK\_name\_of\_siblings\_0 FOREIGN KEY (student\_id) REFERENCES siblings (student\_id);

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 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:

        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

## 4.4 Task 4, Programmatic Access

# 5 Discussion

## 5.1 Task 1 Conceptual model

## 5.2 Task 2, Logical and Physical Model

## 5.3 Task 3, SQL

## 5.4 Task 4, Programmatic Access