

SoundGood Music School Report

Data Storage Paradigms, IV1351

Garó Malko, garom@kth.se

1/4/2021

Contents

SoundGood Music School Report	1
1. Introduction	3
2. Literature Study	4
3. Method	5
4. Result	8
5. Discussion	19

1. Introduction

Git repository: <https://gits-15.sys.kth.se/garom/soundgoodmusicschool>

The application and the database are about a music school which rent different types of instruments to students, they can also participate to learn playing on instruments in individual lessons, group lessons or in ensembles. When the student applies to start learning, he/she should also choose the level of the lessons. In case they choose to participate in the advanced level, they have first to audition before their place is granted. If they have not been accepted, they can keep their application and wait to be contacted when there is a free place. For the rental part, each student is not allowed to rent more than two instruments at the same time. The database is not responsible for processing the rental fee and student payments and instructor salary, but it will only make the mathematical calculations and send them later to the financial system.

By designing the conceptual model and the logical model will help to create the database and will make it clearer which tables are needed to be added to the database and the relations between these tables. The relations between the tables will make it easier to transfer data between the tables. By using the command line user interface will help the administrative staff or the student to execute the queries for different tasks, and get the results shown in the command line. And as improvement some suggestions will be made for how to improve the tables and database to make it easier to receive the needed data.

2. Literature Study

Task 1:

I have learned about conceptual model and how it can be designed from Leif Lindbäck's YouTube videos "Conceptual Model Part 1" and "Conceptual Model Part 2", where he showed an example of how to design the conceptual model. The [pdf file](#) he published I have also learned about IE Notation from Leif's video "IE Notation".

Task 2:

I have learned about the logical and physical models by watching Leif's videos on YouTube "Logical and physical models", part 1, part 2 and part3. I have also read the pdf file of the presentation which has been used in the video in the [link](#):

Task 3:

I have learned about query processing and optimization from the chapters 18 and 19 in the book "FUNDAMENTALS OF DATABASE SYSTEMS. 7th edition" written by "Ramez Elmasri & Shamkant B. Navathe". I have also searched for some tutorials on YouTube which helped a lot to understand and use the queries.

Task 4:

I have learned about how programmatically execute queries using the programming language Java, and by reading the chapters 20, 21 in the "FUNDAMENTALS OF DATABASE SYSTEMS" book. I have also watched Leif's videos: "(Introduction to JDBC), (A JDBC URL), (architecture) and (Architecture and Design of a Database Application)" which are published on the course page (Database Applications). I used some YouTube materials to find more information

Task 5:

I have learned about how indexing happens and what is the purpose of the indexing from Paris Carbon video "[IV1351-2020- Database indices](#)" and from the lab 5 which was presented by Max Meldrum Coding "[Lab 5: Transactions and Indexes](#)".

3. Method

Task 1:

The task is about creating a conceptual model that describes the main idea of the project in real life and how each entity can have relations with other entities. These entities describe different parts of the project and in the music school application. The application called Soundgood Music School Application which lets the students and the administrative staff make different tasks. Some of these tasks are booking lessons or ensembles, process students rentals, their payments, and instructor's payments. The creation starts with listing all possible and related objects to the project, like Instructor, Student, GroupLesson, etc. Then list the categories and start removing the unnecessary entities and replace them with attributes or remove them permanently. Each entity includes its data like id, name, personalNumber (for Student, Parent, etc.) and the number of the attribute (like enrolled Students can be more than 1). The model shows attributes' types and if they can be null or not.

After adding all needed entities, a backup of the .asta file has been taken for easy data restore. The relations between the entities are drawn, Parent requirements are checked, and verb phrase is added to each relation. The verb phrase explains what each relation is for, and each relation's cardinality is chosen carefully. The conceptual model has been uploaded on the Git repository.

Task 2:

The second task is about translating the conceptual model into logical model with enough physical aspects to be able to create the database according to it. The logical model design starts with turning the entities from the conceptual model into tables in the logical model and change some of the entities or attributes in the conceptual model into columns in the logical model, and if some columns return more than one value, they have been replaced to another table and connected with a relation between them and their main table. Each column shows its' type and the length (Char) or the precision of that type if needed. The relations are by passing or calling the primary key of the main table and returns the needed data. For example, the table student instead of storing the contact info in the same table, it is better if we store the info in another table called student_contact_info. Some of the table columns get their data from another table through a relation using the PKs and FKs for example in the relation between instrument and instrument_rental, where the column called instrument_id in instrument_rental gets its' data from the column id in the table "instrument". By looking on the table and the column and the relations between them and how the data transferred between the table, it makes it easier to create the database than creating it from zero directly.

Task 3:

The third task is about to start writing the queries which will show the results of each relation and the transferred data between the tables in the database. First, some tests can be run on PostgreSQL DBMS before start using and filling the database with data. To make the usage of PostgreSQL DBMS easier, pgAdmin 4 can be used as administrative platform for the PostgreSQL to simply working on the database (GUI). To verify that the queries work, some data must be inserted into the tables, and to avoid the duplication of the inserted data, it has been generated randomly and inserted into the tables.

Another application/tool that called PostgreSQL Maestro which has been used for creating and improving the queries. PostgreSQL Maestro helps a lot to learn how to create queries by opening the table in the Visual Query Builder and choose the needed data, connect the table to each other and add criteria if needed, selection, group criteria or sorting. Generating the SQL queries script can be executed to preview the result before using it as a final query.

To make it easier to understand how the queries are being executed in the database, it is better to write each part of the query by its' own, and not try to write it all by once. For example, first a total_rentals view being created every time the query is being executed. The data counts and calculates the total number of the instruments' rentals in a specific year and list all the monthly rental for that year. Different instruments' rentals are being viewed in different views and then use LEFT JOIN to collect them and show them in a single table. The same method is being used for other queries with different requirements. In the queries CASE, ELSE which is the same as IF statement in other programming languages as JAVA. CASE, ELSE usage is for switching the null value with a 0 value in the column. Different JOIN types are used to fulfill each task's requirements.

Task 4:

The 4th task is about writing some java code which will give access for the user to access the different type of operations in the database, but for the mandatory part. It is enough to make the code do these three tasks for the student usage. The tasks are: List all id:s of the available instruments in the stock, their kind, their brand and their rental fee. The second task is about student rental process, that is why it could be divided to two parts, one for showing student's current rentals which would make it easier for the student's to know if he/she is eligible for another rental or the student reached the limit of rentals and the second part of the second task is to make a rental. The third task is to give the student the ability to terminate an ongoing rental before the rental date ends.

First thing to do is to build the project structure following the MVC model. An external library is being used along with the DB to enable the usage of the connection to the database and its name is "[postgresql-42.2.18](#)". By using this library, a connection could be created with the database in the main.java which is in the startup package and which will run when the user starts the application. This connection will be passed to the Controller which will pass it forward to the Model package where most of the calculations, data insertion and queries execution occurs in different methods. The DTO package will store the data which has been imported by executing the queries by creating a statement from the connection and use the method connection.executeUpdate to and commit the query to print the result. Then this data can be saved in RAM and use it until the connection is closed and that happens when the user chooses the fifth

category/alternative in the list. The user's different category selections' input will be entered to the View file in the View package and which will run in loop until the user break that loop by choosing the 5th alternative. A new Exception has been created for printing a message to let the user that something went wrong without breaking the loop of the application.

4. Result

Task 1:

The link of the conceptual model result as .png is uploaded on the [Git repository](#):

The link of the conceptual model asta file is uploaded on the [Git repository](#):

The relation in Figure 1 makes the model connected even if we remove some of these entities, the model will be connected and the data will be able to be transferred through the entities but will make harder and take longer way to do it.

Each relation has its own verb phrase which explains the purpose of that relations existence. Which can make it easier for the reader or user to understand the conceptual model of this project.

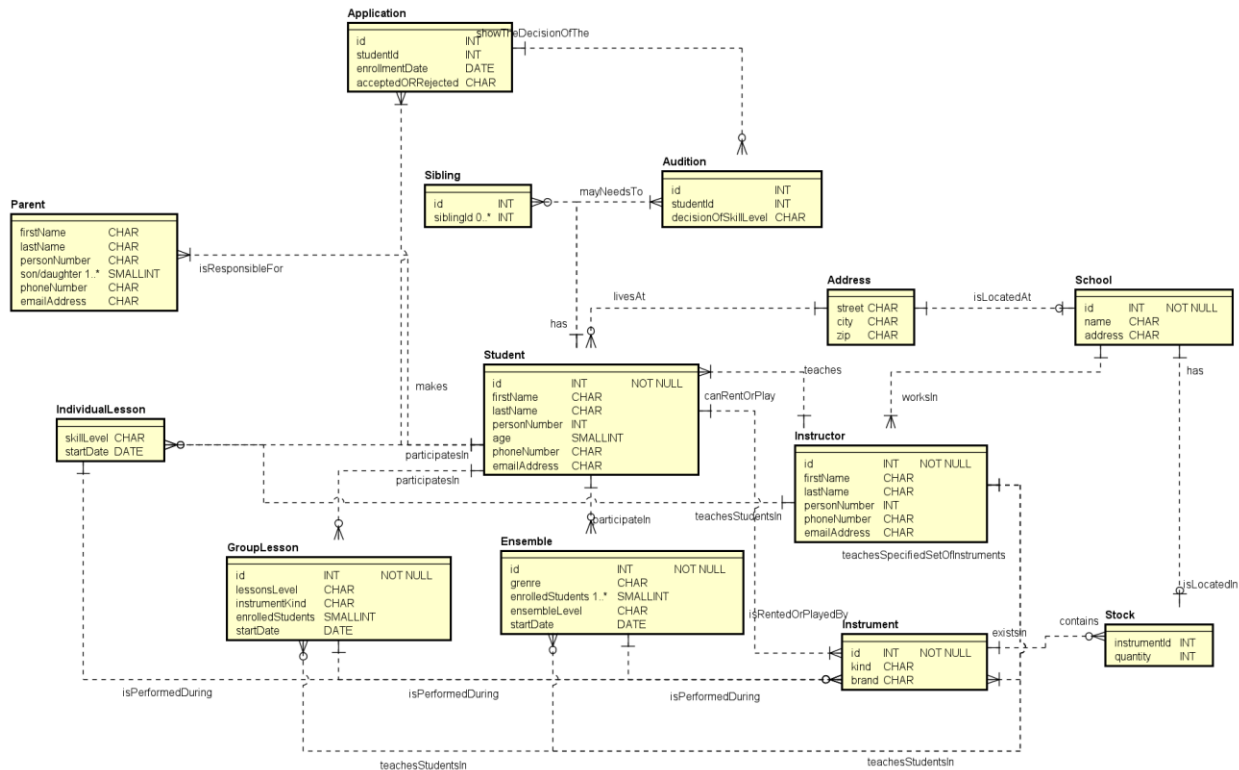


Figure 1: The conceptual model of the Soundgood Music School Application

Task 2:

The link of the logical model result as .png is uploaded on the [Git repository](#):

The link of the logical model asta file is uploaded on the [Git repository](#).

The link of the SQL script for creating the database is uploaded on the [Git repository](#).

It can happen sometime and the query tool in the pgAdmin 4 does not support adding all rows by once for creating the schema of the database. That's why I have used PostgreSQL Maestro which is a tool for generating, executing, and creating queries. The tool used for creating the database with the data in the task 3.

The link of the mydbSchema (my database schema) include the script for creating the view and materialized view as well. It was where I have taken a backup of the database after creating them. But the uploaded code is only for creating the database and not inserting data as wanted in the requirements. The data will be inserted later into the tables.

As it can be seen in figure 4, the `ensemble_id` will be connected to the `id` column in the `ensemble` table. To transfer the data through the tables.

id	first_name	last_name	person_number	street	city	zip_code
[PK] integer	character varying (100)	character varying (100)	bigint	character varying (100)	character varying (100)	character varying (100)

Figure 2: The administrator table and all columns are included

id	genre	instructor_id	minimum_enrolled_students	maximum_enrolled_students	enrolled_students	number_of_instruments	ensemble_level
[PK] integer	character varying (100)	integer	smallint	smallint	smallint	smallint	character varying (15)

Figure 3: the ensemble table and all columns are included.

id	ensemble_id	start_date	week_day
[PK] integer	integer	timestamp without time zone	character varying (15)

Figure 4: The ensemble_schedule table and all columns are included.

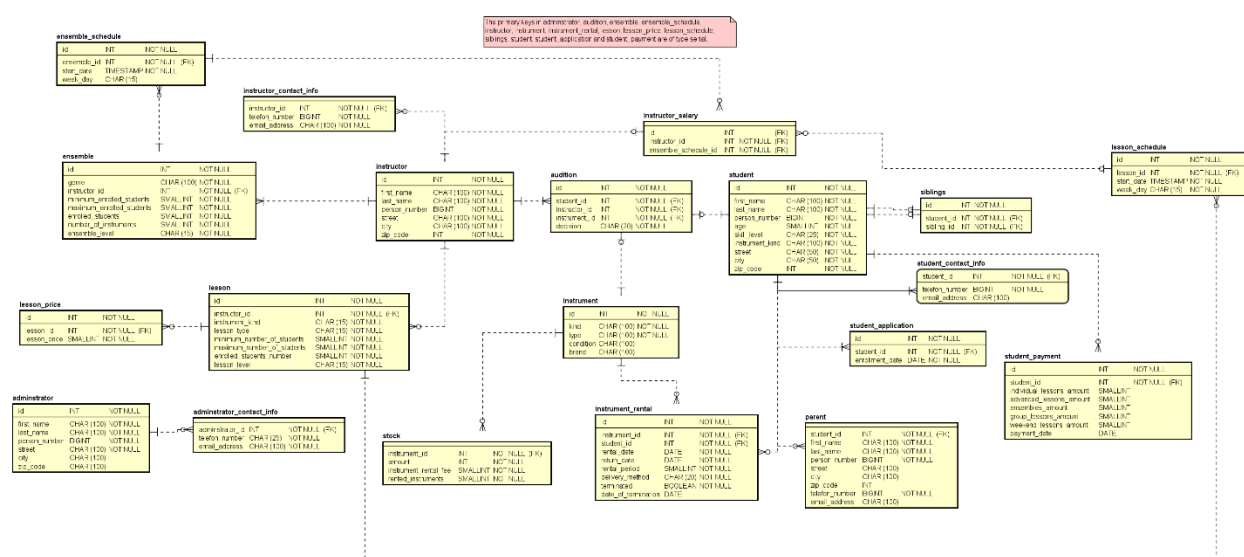


Figure 5: The logical model and all its tables and columns

Task 3:

The queries: 1, 2, 3, 4, 5, 6 and 7 and the creation of the views can be found [here](#).

The queries 1, 2, 3, 4, 5, 6 and 7 without their view creation can be found [here](#).

Figure 7 shows the result of executing the first query in pgAdmin 4 on the database which has been created and filled with needed data. The query itself could be a View (not materialized) or could be executed as a query. The first column of the table is the month's number of the year 2020 (it can be changed in the query where it can be found in the link above). The second column was about the total number of rentals in that year for each month. And the rest of columns show which kind of the instruments have been rented. LEFT JOIN has been used because of that the total_rentals column's rows are the same number as other columns, and if there is something null value, they have been replaced with 0. All imported data about saxophone, clarinet, piano rentals, etc. (figure 9) are imported from views which have been created later. But added to the mysql schema in the Git repository (figure 8 shows the view that collects all rentals for all previous years). The purpose of creating these views is making it easier for the query to import all needed data and list them in table's columns from the executed query (in figure 7). All tests have been verified manually and checked more than once either by counting the rentals in every month in the specific year or used PostgreSQL Maestro to verify that the result is correct. The same method has been used for verifying all the other queries results.

Figure 10 shows the result of executing the second query. The query could be created as a View (not materialized) or could be executed as a query. The query was about showing the average of rentals in every year for each month (like in figure 10 and the average rental of 2014 is 4.17/month which could be rounded to 4 rentals per month but is shown as decimal to notice the difference). From the third column contain the average rental of each instrument per month. The same thing as in query 1. New views have been created for storing the average of each kind of instruments rental. The subqueries are the same as in the first query. But with different data. The data imported data from the result of the query and showing them as unique view made it possible to list the result of each rentals kind of instrument and divide them with the number of months in each year and the result of the query 2 are shown in figure 10.

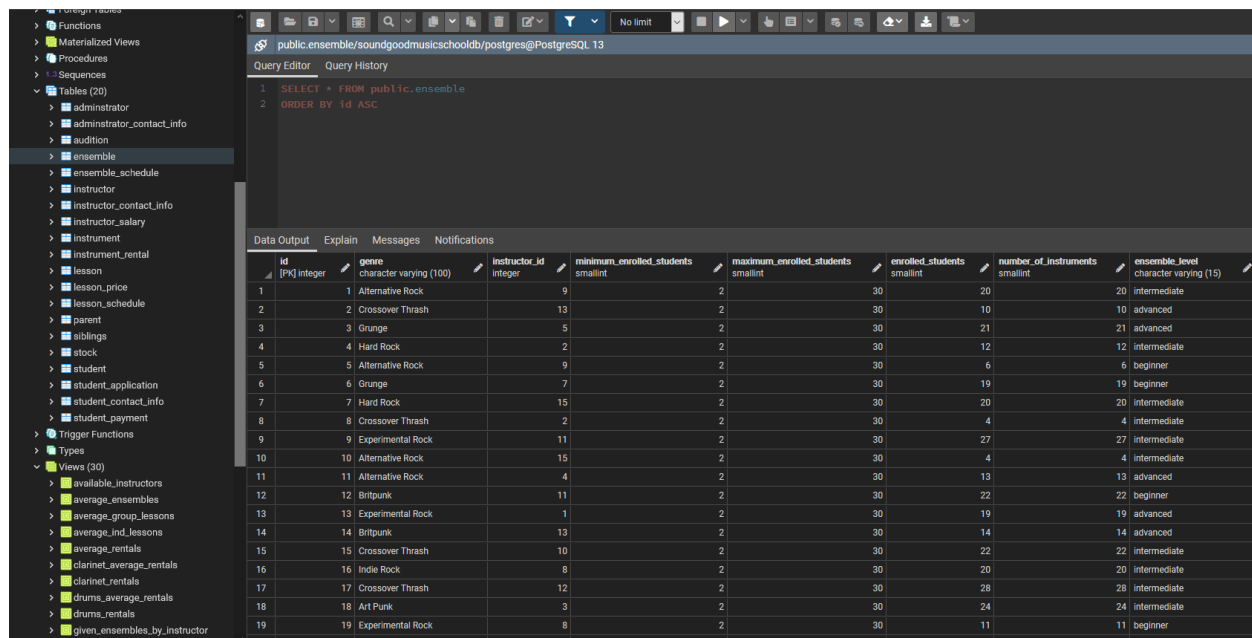
Figure 11 shows the result of query 3 which require to show the total number of lessons in every month in a specific year. The table shows detailed data about the number of lessons for different types such as individual lessons, group lessons and ensembles. For importing the data about the individual, group lessons and the ensembles, new views have been created and each view stored different type of data. Figure 12 is about the individual lessons, but the same method has been followed for the other lessons. The result is placed in the columns 3rd, 4th and 5th. The sum of the values in the columns 3, 4 and 5 should be equal to the values in the 2nd column.

In figure 13, the table shows the result of the query 4. The average of total number of given lessons and ensembles for the entire year. For example, in figure 13 in the 2020, the average of given lessons in the school is 134 lessons and ensembles per month. The average of the individual lessons which is divided by the number of months are placed in the 3rd column, the average of group lessons is placed in the 4th column and the average of the ensembles count is placed in the 5th column of the table.

Figure 15 shows the result of the executed query number 5. It is about showing the instructors who have given more than a specific number of lessons in the current month. And list the three instructors who have given highest number of lessons in the previous month. It has been solved by creating different views, one for collecting the ensembles for each instructor and the same thing for the individual lessons and group lessons for the current month (figure 16). The same process and similar view for these types of lessons for the previous month (figure 17). The query 5 imported these data and added these data to the available_instructors view by using FULL OUTER JOIN which append the new data to the previous data regardless the number of the rows. The result was ordered descending to show the three most given lessons number by instructor at the top of the table.

Figure 18 shows the result of the executed query number 6. The query will sort ensembles ordered by the genre and the date for the next week and to show the user if there are available seats in the ensemble, and when will the ensemble start (date and time). It was easier to execute the query and save it in a single view than the previous queries. The type of the query is Materialized View where it stores the data in it and no need for executing the query every time the query is executed but will only show the saved data.

Figure 19 show the Materialized View which called instrument_lowest_monthly_fee which list the three lowest monthly rental fee instruments in the stock and show the type of lesson where these instruments will be performed and the date and time as well. The view will also show if the instrument is available to be rented or out of stock.



id	genre	instructor_id	minimum_enrolled_students	maximum_enrolled_students	enrolled_students	number_of_instruments	ensemble_level
1	Alternative Rock	9	2	2	30	20	intermediate
2	Crossover Thrash	13	2	2	30	10	advanced
3	Grunge	5	2	2	30	21	advanced
4	Hard Rock	2	2	2	30	12	intermediate
5	Alternative Rock	9	2	2	30	6	beginner
6	Grunge	7	2	2	30	19	beginner
7	Hard Rock	15	2	2	30	20	intermediate
8	Crossover Thrash	2	2	2	30	4	intermediate
9	Experimental Rock	11	2	2	30	27	intermediate
10	Alternative Rock	15	2	2	30	4	intermediate
11	Alternative Rock	4	2	2	30	13	advanced
12	Britpunk	11	2	2	30	22	beginner
13	Experimental Rock	1	2	2	30	19	advanced
14	Britpunk	13	2	2	30	14	advanced
15	Crossover Thrash	10	2	2	30	22	intermediate
16	Indie Rock	8	2	2	30	20	intermediate
17	Crossover Thrash	12	2	2	30	28	intermediate
18	Art Punk	3	2	2	30	24	intermediate
19	Experimental Rock	8	2	2	30	11	beginner
20	Experimental Rock	6	2	2	30	14	advanced

Figure 6: The layout of the pgAdmin 4 that has been used to SELECT, INSERT and DELETE data into and from the database.

	month double precision	total_rentals bigint	saxophone_rental_count bigint	piano_rental_count bigint	violin_rental_count bigint	drums_rental_count bigint	clarinet_rental_count bigint	trumpet_rental_count bigint
1	1	5	2	1	0	0	2	0
2	2	2	1	1	0	0	0	0
3	3	3	0	0	0	0	2	1
4	4	6	1	0	2	0	1	2
5	5	5	1	2	0	2	0	0
6	6	3	0	0	0	0	2	1
7	7	4	0	1	0	1	0	2
8	8	4	0	0	0	2	1	1
9	9	2	0	0	0	0	2	0
10	10	6	0	0	2	1	1	2
11	11	4	1	1	1	0	0	1
12	12	3	1	0	0	1	1	0

Figure 7: The result of the query number 1.

	year double precision	month double precision	total_rentals bigint
64	2019	8	2
65	2019	5	4
66	2019	11	2
67	2019	12	2
68	2019	2	7
69	2019	3	11
70	2019	1	4
71	2019	6	7
72	2019	4	5
73	2020	12	3
74	2020	2	2
75	2020	5	5
76	2020	3	3
77	2020	8	4
78	2020	9	2
79	2020	7	4
80	2020	1	5
81	2020	6	3
82	2020	4	6
83	2020	10	6
84	2020	11	4
85	2021	1	1

Figure 8: The subquery of collecting total number of rentals

	year double precision	month double precision	clarinet_rental_kind bigint	instrument_kind character varying (100)
32	2019	1	1	Clarinet
33	2019	2	3	Clarinet
34	2019	3	3	Clarinet
35	2019	5	2	Clarinet
36	2019	6	2	Clarinet
37	2019	8	1	Clarinet
38	2019	11	1	Clarinet
39	2020	1	2	Clarinet
40	2020	3	2	Clarinet
41	2020	4	1	Clarinet
42	2020	6	2	Clarinet
43	2020	8	1	Clarinet
44	2020	9	2	Clarinet
45	2020	10	1	Clarinet
46	2020	12	1	Clarinet

Figure 9: The subquery of collecting clarinet rentals

	year double precision	average_rental numeric (1000,2)	clarinet_average_rentals_count numeric	trumpet_average_rentals_count numeric	saxophone_average_rental_count numeric	piano_average_rental_count numeric	violin_average_rental_count numeric	drums_average_rental_count numeric
1	2014	4.17	1.17	1.00	0.33	0.75	0.42	0.50
2	2015	5.00	0.92	1.58	0.25	1.33	0.33	0.58
3	2016	3.75	0.83	1.33	0.33	0.50	0.25	0.50
4	2017	4.42	0.58	1.50	0.17	1.17	0.50	0.50
5	2018	4.00	0.75	1.42	0.83	0.75	0.08	0.17
6	2019	4.25	1.08	1.75	0.08	0.58	0.42	0.33
7	2020	3.92	1.00	0.83	0.58	0.50	0.42	0.58
8	2021	0.08	0	0	0.08	0	0	0

Figure 10: The result of the query number 2.

	month double precision	total_lessons bigint	individual_total_given_lessons_count bigint	group_total_given_lessons_count bigint	total_given_ensembles_count bigint
1	1	145	35	53	57
2	2	153	37	50	66
3	3	141	35	47	59
4	4	130	33	33	64
5	5	142	37	46	59
6	6	137	38	46	53
7	7	155	50	51	54
8	8	129	36	43	50
9	9	152	48	42	62
10	10	122	30	49	43
11	11	161	40	53	68
12	12	133	33	35	65

Figure 11: The result of the query number 3.

	year double precision	month double precision	given_lessons bigint	lesson_type character varying (15)
1	2014	1	35	individual
2	2014	2	37	individual
3	2014	3	35	individual
4	2014	4	33	individual
5	2014	5	37	individual
6	2014	6	38	individual
7	2014	7	50	individual
8	2014	8	36	individual
9	2014	9	48	individual
10	2014	10	30	individual
11	2014	11	40	individual
12	2014	12	33	individual
13	2015	1	12	individual
14	2015	2	32	individual
15	2015	3	20	individual

Figure 12: The subquery of query 3 for collecting the individual lessons.

	year double precision	average_year_lessons numeric	average_ind_lessons_count numeric	average_group_lessons_count numeric	average_ensembles_count numeric
1	2014	142	38	46	58
2	2015	96	23	31	42
3	2016	116	35	48	33
4	2017	150	38	45	67
5	2018	158	37	46	75
6	2019	167	38	46	83
7	2020	134	38	46	50
8	2021	109	23	32	54

Figure 13: The result of the query number 4.

	first_name character varying (100)	last_name character varying (100)	total_number_of_lessons_for_current_month bigint	total_number_of_lessons_for_previous_month bigint
1	Kalle	Holmgren	10	27
2	Noelia	Lindqvist	18	17
3	Susanna	Eliasson	6	17
4	Dani	Carlsson	16	0
5	Sanja	Gunnarsson	9	0
6	Alwin	Söderberg	8	0
7	Jack	Göransson	7	0
8	Elvin	Karlsson	7	0

Figure 14: The result of the query number 5.

	id integer	first_name character varying (100)	last_name character varying (100)	person_number bigint
1	5	Alexia	Berggren	197511219581
2	1	Moltas	Blom	199108260317
3	3	Dani	Carlsson	197206179892
4	11	Susanna	Eliasson	198207099121
5	8	Jack	Göransson	197006182773
6	15	Anastasija	Gunnarsson	195607194601
7	2	Norea	Gunnarsson	196211079162
8	4	Sanja	Gunnarsson	197303269968
9	14	Alex	Hellström	199106277636
10	9	Kalle	Holmgren	198712053951
11	13	Elvin	Karlsson	198505221518
12	7	Noelia	Lindqvist	196907107764
13	6	Alwin	Nyström	199011255958
14	12	Petrus	Pettersson	198809199691
15	10	Alwin	Söderberg	198709085412

Figure 15: The available instructors in the school are collected in one view

	first_name character varying (100)	last_name character varying (100)	person_number bigint	id integer	total_lessons_for_current_month bigint
1	Noelia	Lindqvist	196907107764	7	18
2	Dani	Carlsson	197206179892	3	16
3	Kalle	Holmgren	198712053951	9	10
4	Sanja	Gunnarsson	197303269968	4	9
5	Alwin	Söderberg	198709085412	10	8
6	Elvin	Karlsson	198505221518	13	7
7	Jack	Göransson	197006182773	8	7
8	Anastasija	Gunnarsson	195607194601	15	6
9	Susanna	Eliasson	198207099121	11	6
10	Moltas	Blom	199108260317	1	5
11	Norea	Gunnarsson	196211079162	2	5
12	Alexia	Berggren	197511219581	5	4

Figure 16: The total given lesson by instructors for the current month.

	id integer	first_name character varying (100)	last_name character varying (100)	person_number bigint	total_lessons_for_previous_month bigint
1	9	Kalle	Holmgren	198712053951	27
2	7	Noelia	Lindqvist	196907107764	17
3	11	Susanna	Eliasson	198207099121	17

Figure 17: The total given lesson by instructors for the previous month.

	id integer	week_day character varying (15)	start_date timestamp without time zone	genre character varying (100)	available_seats text
1	31	tuesday	2021-01-12 12:00:00	Alternative Rock	has more seats left
2	11	saturday	2021-01-16 09:45:00	Alternative Rock	has more seats left
3	12	sunday	2021-01-10 14:15:00	Britpunk	has more seats left
4	12	friday	2021-01-15 14:30:00	Britpunk	has more seats left
5	22	friday	2021-01-15 16:45:00	College Rock	has more seats left
6	24	monday	2021-01-11 08:45:00	Crossover Thrash	has more seats left
7	24	monday	2021-01-11 13:15:00	Crossover Thrash	has more seats left
8	28	tuesday	2021-01-12 16:45:00	Crossover Thrash	has more seats left
9	27	tuesday	2021-01-12 09:30:00	Experimental Rock	has more seats left
10	13	tuesday	2021-01-12 17:45:00	Experimental Rock	has more seats left
11	3	wednesday	2021-01-13 15:15:00	Grunge	has more seats left
12	4	sunday	2021-01-10 13:00:00	Hard Rock	has more seats left
13	39	friday	2021-01-15 11:15:00	Indie Rock	has more seats left

Figure 18: The result of the query number 6.

	instrument_kind character varying (15)	instrument_fee smallint	lesson_type character varying (15)	start_date timestamp without time zone	available_instruments text
1	Trumpet	110	group	2021-01-13 17:15:00	Available
2	Clarinet	201	group	2021-01-29 14:45:00	Available
3	Drums	218	group	2021-01-07 09:30:00	Available

Figure 19: The result of the query number 7.

Task 4:

The java code is uploaded on this [Git repository](#).

First the Main java class creates a connection between the application and the database. The connection will let the application execute queries. After creating the connection, it is passed to the controller which will pass it forward to the model classes like Stock.java, StudentRental.java. Stock.java execute the query that select the data about instruments from the public.instrument table in the database, to collect the instrument id, instrument kind and instrument brand, and to compare the id with the instruments in the stock and return the rental fee from public.stock in the database. After that, the data will be saved in the StockInstrumnet in the DTO package. Same process with the StudentRental.java in the model package which collects all data about the instrumnet_rental and save the returned data in StudentRentalsInfo.java in DTO package. The termination category will be able to change the status of the rental from Terminated=false into Terminated = true and set the date_of_termination as the Current date.

All entered data which by student is entered in the View.java (figure 20), where the student chooses one of the five categories depending on the number the student enters. 1 for showing the available instruments (figure 21), 2 for showing the student's current rentals by writing the student ID (figure 22). 3 for adding a new rental by writing the student id, instrument id, rental period (number of months) and the delivery method. The student can verify that the rental is submitted by rechecking his/her current rentals after finished the rental process (figure 23). 4 for terminating a current rental and that can be done by entering the student id and rental's id which can be found in the first column when the student shows his/her current rentals and verify that the termination is submitted (figure 24, 25).

After each select, update or delete, a connction.commit() has to be called for committing the query and show the result. That will be used when the auto commit is turned off in the ConnectionConfig.java. The code will still be running until the student enter number 5 which is the exit command, and which will break the switch case and break the while loop.

```

Welcome to the SoundGood Music school:
Enter the number of the category you want to enter:
1: Show the available instruments in the stock.
2: Show your current rentals.
3: Add a new rental to your account.
4: Terminate an ongoing rental.
5: To exit.

```

Figure 20: the available categories to choose between


```
1
```

ID	Kind	Brand	Rental Price
1	Piano	Bechstein	292
2	Piano	FAZIOLI	300
3	Violin	Stentor	418
4	Drums	YAMAHA	218
5	Saxophone	Selmer Paris	425
6	Clarinet	Yamaha	306
7	Trumpet	Rossetti	202
8	Clarinet	Mendini	201
9	Trumpet	Cecilio	110

Figure 21: The available instrumnets in the stock

```
2
Enter the Student id to show your current rentals
26
```

Rental ID	Student ID	Instrument ID	Rental Date	Return Date	Rental Period	Delivery Method
337	26	4	2020-08-16	2021-07-16	11	Pick up

Figure 22: Show the student's current rentals

```
3
Enter the student id:
26
Enter the instrument id:
2
Enter the rental period:
11
Choose the delivery method: 1: for Delivery to house  2: for Pick up
1
```

Figure 23: The process of renting a new instrument

```
2
Enter the Student id to show your current rentals
26
```

Rental ID	Student ID	Instrument ID	Rental Date	Return Date	Rental Period	Delivery Method
337	26	4	2020-08-16	2021-07-16	11	Pick up
356	26	2	2021-01-06	2021-12-06	11	Deliver to house

Figure 24: Verify the rentals after the rentals has been added to the student's id

```

4
Enter the student id:
26
Enter The the Rental Id: (You can check it by choosing the 2 category to show your current rentals).
337
Done
Make sure to check your current rentals to confirm the termination
2
Enter the Student id to show your current rentals
26
Rental ID      Student ID      Instrument ID      Rental Date      Return Date      Rental Period      Delivery Method
356            26              2                 2021-01-06       2021-12-06       11                Deliver to house

```

Figure 25: The termination process and verify that the termination is succeeded

Task 5:

The first index can be created for the column “lesson_id” in the table “lesson_schedule”. The usage of the column occurs in the WHERE clause and INNER JOIN. The column is used in four views by executing the 3rd, 4th, 5th, and 7th queries. For example:

```
public.lesson_schedule.lesson_id = public.lesson.id
```

```
INNER JOIN public.lesson_schedule ON (public.lesson.id =
public.lesson_schedule.lesson_id)
```

The second index can be created for the extracted “month” from “start_date” column in the table “lesson_schedule”. The extraction occurs in WHERE clause in query 5 execution. for example:

```
EXTRACT(month FROM public.lesson_schedule.start_date) = EXTRACT(month FROM
CURRENT_DATE)
```

```
EXTRACT(month FROM public.ensemble_schedule.start_date) = EXTRACT(month FROM
CURRENT_DATE - interval'1 month')
```

The third index can also be created for the extracted “month” from the column “start_date” but in the table “ensemble_schedule” and the column is called “ensemble_month”. The extraction occurs in LEFT JOIN in FROM clause occurs in 3rd query:

```
LEFT JOIN public.total_given_ensembles ON ( public.given_lessons.month =
public.total_given_ensembles.ensemble_month AND public.given_lessons.year =
public.total_given_ensembles.ensemble_year)
```

It is possible to use more indexes in the database for better performance, but the most used columns in the executed queries and views are the ones I have chosen above.

5. Discussion

Task 1:

The entities name and its attributes are following the naming convention by starting with an uppercase for each word, and with a lowercase in the first word in every attribute. Which is the naming convention in the conceptual model. The names of the entities explain the purpose of their existence, for example. The entity Student explain the student and where will store all data that belongs to the student, like id, firstName, lastName, etc. All included entities in the conceptual diagram are important to complete and fulfill the project's requirements. Some more entities could be added such as InstrumentRental entity but that would make it more complicated to connect the Instrument and Student entity to describe the rental process. Every relation has the cardinality its' cardinality, if it 0-1, 0 or more and 1 or more, and it depends on the entity and its' relations. Some of the relations have cardinalities at the both end for example, the relation School and Stock and the relation between Student and Instrument.

Task 2:

The tables names and its columns are following the naming convention in the logical model and in the database itself. The naming convention is by writing all names with lowercase and leave an underscore between the words in the tables and columns name for example in the table name: instructor_contact_info or in the column start_date. The model is following the 3NF as much as it could be. Each table cell contains only one value of the same type with different columns names, there is a primary key and they do not have dependencies on each other. The tables are relevant to the application and meets the requirements for 3NF. It could be have missing some tables for extra criteria, but for the part I have done, the current tables do their job and are relevant for the queries in the task 3. Every table contains number of columns where the relevant data to that table will be stored. Some columns types could be discussed, but it depends on the requirements, for example person_number or telefon_number could be stored either as BigInt (personal number and telefon_number could be greater than the limit of int) or CHAR with decided length. Most of the primary keys are int to avoid the possibility of being repeated or like another value in the primary key column, and most of the table have ids except the ones related to another table. For example, the table lesson has its own id, which is the primary key. But lesson_schedule does not have primary key, and the reason why that is because the lesson id could be repeated more than once. As it is obvious in the logical model in figure 5 each foreign key (FK) is displayed on the right side of the column in tables, which make it easier to understand each column's value where they are imported from. There is a note in the logical model which describe that the primary keys with type Integer are serial type but displayed as Integer.

Task 3:

I have chosen to let the query 1 executes every time it is needed, that is because in the query commands is being decided for which year the query will sort, sum the instrument rentals. Also, if when the year has ended or not. All that will be affected if we would create another view for the query itself. That would force the PostgreSQL run and calculate the data for all years if we would show the result of all previous years. The same thing goes for second query execution. Where the other subqueries were created as views, I thought it would be better if the administrator executes the SQL query since it will be used mostly for the statistics or could be another usage as well. The third query is as the first query but for showing the number of total lessons and ensembles for specific year. That means that the user or administrator may want to execute the query for different year. He/she will choose the year in the that in the SQL query before the execution. And the same thing goes with the 4th query where it calculates the average of the lessons for individual, group lessons and ensembles. The query number 5 is displaying the number of lessons given by instructors and that query will be executed daily. That is why the best choice would be the View and not the Materialized View because the result changes daily and there would be meaningless save the data that will change daily. And for not executing it as a View is because of the query requirements where the administrator or the user choose which instructor have given more than a specific number of lessons. That number can change daily or not. So not taking risks, chose to execute the query as SQL query and not a View.

The query 6 is being updated and executed weekly or even monthly. That is why it would fit more to be a Materialized View and save its data which is the purpose of the Materialized Views. The same thing with the query 7 where the monthly fee is rarely updated and changed. For the availability part, while the student rent an instrument, he chooses at the time when he will return the instrument, and it would be shown in the database. In case he terminates the rental and return it before the end time, that will increase the quantity in the stock. That is why it is ok to create Materialized View for this query as well.

Task 4:

The java code follows the naming convention in Java language (variables' and methods' names following the camelCase convention, and classes follow PascalCase). And that is clear in classes' functions', and variables' names. Every method name, class name and variable name explains its data. For example. StockInstruments.java class contains information about the stock's instruments and their amount, StudentRentalsInfo.java contains information about student's id, rentalId, rentalDate and InstrumentID. terminateRental(studentId, rentalId) is a function be called when a student with the id (studentId) wants to terminate a current rental with the id (rentalId). And The variable availableInstrumentInStock StringBuilder is filled with the available instruments in the stock when it is passed as parameter in the function Stock.getAvailableInstruments().

In some methods where the application execute SQL queries, it uses connection.createStatement() to create an instance of Statement which executes the query, and after saving the result of the executed query in ResultSet, use connection.commit() for making connection.commit() in the try statement for updating the database or connection.rollback() to restore the database if anything wrong happens. When the user chooses to exit the application. The application closes the connection to the database. That save the

database of unnecessary and unexpected SQL queries execution. Especially when for example some student rent an instrument or pay his payment, unexpected query execution exceptions can occur some issues. That is why the `connection.commit()` after completing some task or task query, and `connection.rollback()` to rollback to the most recent committed query where the query execution has been successful.

The functionality works as expected, for example. To choose what to do, there is a list of categories to choose between (see figure 20), functions calling occurs in switch case to handle the chosen alternative. If the student wants to rent an instrument chooses third category (see figure 20). That call let the user enter his/her id, instrument id, and rental date (see figure 23). These information are being sends to the function that handles adding new rentals to the student's account. To make sure that the student has successfully rent the instrument, can choose the 2nd category (see figure 20) and check rental's info (see figure 24). If something unexpected occurs, it will roll back to the most recent successful commit.

The classes are mapped in different packages (tried to follow the MVC as much as I could). DTO package include the classes which describe StockInstruments and their information. The Model package includes the Stock class and StudentRental. That makes it easier for the access modifiers to control the data flow.

Task 5:

Queries in the databases are sets and searching for data might take some time and especially for large amount of data, such as IMDB data and other databases. There is a data structure called indexing which works as a pair of <Key, value>, where the key is a primary key and value contains the value of the chosen/connected column in a table. That makes it easier for searching for these data rather than checking every executed query and compare the value if it is used in WHERE clause, nested SELECT statements (which may contain WHERE clause in it), etc. The index will have either time complexity $O(1)$ as average or $O(n)$ as worst case in HASH_TABLE and $O(\log n)$ in B-Trees.

The indexing algorithms are used in queries include WHERE and INNER JOIN. The HASH_TABLE indexing algorithm is used when two types of data (variable, number, columns, etc.) are compared using equal (for example: `variable1 = variable2`). The B-Tree indexing algorithm is mostly used when two types of data are compared using range (for example: `variable1 > variable2` "or" `variable1 < variable2`)

The first index in my database was chosen to be created for "lesson_id" column in "lesson_schedule" table. The column is used in four views by executing the 3rd and 4th queries and in INNER JOIN in queries 5 and 7. The views do not store data. That means they will be created or replaced every time these queries be executed. The index for the "lesson_id" column will make it easier for filtering and searching for data in the result of the executed query. By connecting each value in the column lesson_id to a key in the index can be chosen to be a HASH_TABLE, for example:

In query 3 and 4 "WHERE public.lesson_schedule.lesson_id = public.lesson.id").

In query 7 "INNER JOIN public.lesson_schedule ON (public.lesson_schedule.lesson_id = public.lesson.id)".

The second index can be created for the extracted "month" from "start_date" column in the table "lesson_schedule". The extraction occurs in WHERE clause in query 5 execution. Indexes are used in WHERE clause which filters the result before grouping them. It may take some time if the database is big.

By using index for the column “month from start_date”. That will save time and make it easier for filtering and searching for data (will work as search tree different algorithms). As a best alternative to choose between, the index can be of type HASH_TABLE.

(i.e. “**EXTRACT (month FROM public.lesson_schedule.start_date) =
EXTRACT (month FROM CURRENT_DATE - interval'1 month')**”).

The third index can also be created for the extracted “month” from the column “start_date” but in the table “ensemble_schedule”. The extraction occurs in WHERE clause in the 5th and 6th queries. As said before, indexing a column used in WHERE clause will help a lot in searching and filtering data. By using an index for the months from the column “start_date” in the table “ensemble_schedule”. Using index for this column will save time while comparing the months in “start_date” with the previous month or the current month. The index can be chosen to be a HASH_TABLE.

**EXTRACT (month FROM public.ensemble_schedule.start_date) =
EXTRACT (month FROM CURRENT_DATE - interval'1 month')**