Project Report

Data Storage Paradigms, IV1351

William Axbrink waxbrink@kth.se

January 5, 2021

1 Introduction

This projects purpose is to design an application that will handle all the data and transactions of the Soundgood music school company. The project deals with conceptualizing, designing and building a database and the necessary tools around it. It is divided into 5 different tasks consisting of first modelling the reality around the application area, then modelling the database and its structure, developing the queries needed to gather data based on the requirements, creating an application to deal with the rental system and finally optimizing the database accesses.

The project requirements are based on the written Project Description under the course page on Canvas and as per the instructors instructions, many requirements have been for the student to interpret.

2 Literature Study

2.1 Task 1 Conceptual Model

Before starting properly with the first task, I gathered information by watching the videos posted on Canvas and reading more about Inheritance in Chapter three of the book Fundamentals of Database Systems. Since creating a conceptual model is similar to creating a domain model, we students part of TIDAB had a head start thanks to the course IV1350, Object-oriented Development. Then when starting to create the conceptual model, I had the instructional videos on the side and paused to add and make changes. I also compared to other database diagrams to get an idea of what it might look like.

2.2 Task 2 Logical and Physical Model

I worked similarly for task 2 by starting to watch the videos on Logical and Physical models and then read in the chapters 14 and 15, mostly skimming to find the topics I had harder time understanding, like really grasping the idea of Primary Keys and Foreign keys and the Normalization forms.

2.3 Task 3 Queries and Data Serialization

I started with the IMDB tutorial and quizzes which I felt was an appropriate amount of beginner learning to start with the queries and used W3Schools to check SQL syntax while working on the queries. I followed the Docker setup tutorial to setup docker, PostgreSQL and Adminer. I used the course book to learn more about OLAP, Views and OLTP.

2.4 Task 4 Programmatic Access

I started the process with watching the videos under Database Applications to learn more about the Java Database Connectivity(JDBC) API and MVC Architecture. After that the focus on learning how to work with the JDBC and MVC was to experiment with the JDBC-Intro and JDBC-Bank resources before starting the process of creating the Soundgood application.

2.5 Task 5 Performance

First I watched the recommended videos and continued with reading about the different indices closer in the book. Due to problems grasping the ideas of indices, I also watched other videos recommended by course students and read explanations on Stackoverflow. After that I felt comfortable to start the process of creating indices and continued meanwhile to re-read the book after getting a bit of experience.

3 Method

3.1 Task 1 Conceptual Model

The first step of creating a database is to make a model that illustrates the data that exists in reality. It works as a simplified version of the data that exists and will help to create a database, which will be similar to this conceptual model.

3.1.1 Domain Modelling Method

The first step is to identify all nouns based on the requirement specification that we have received. This information will provide candidates to become entities in the conceptual model. Then the next step is to use a category list to find more entities, followed by cleaning up unnecessary classes from the model. For the entities created and kept, the next step is to find attributes and then create associations between the classes. Lastly the model is reviewed and refined.

3.1.2 Conceptual Model

But since this is a conceptual model we have data instead of entities, and therefore we should not have an entity without attributes and it is important to find all relevant attributes. Then the cardinality for each attribute shall be defined or noted if the attribute can be without value. It is especially important to find all relations between the data and specify the cardinality between the relations.

3.2 Task 2 Logical and Physical Model

3.2.1 Logical Model

This model is similar to the conceptual model as it defines what to store, even if it does not define how to store it or describe the reality. It is adapted to the type of data storage that is used but not to any particular choice of Database Management System(DBMS). When converting a conceptual model to a logical model you first have to choose what relations/tables to use. The conceptual model contains more than is necessary for the logical model so parts have to be removed that is redundant. Another important part is to normalize the model based on the data being stored. The types of attribute can be specified to note if data is a date, a text or a number for example.

3.2.2 Physical Model

This model relates to how the data is stored and defined in the logical model. This is more adapted to specific DBMS, and in this example PostgreSQL is used. Then all storage details like column type, index and views and included. This is a complete model that can be used in real databases.

3.2.3 Logical/Physical Model

For this project the requirement is to create a single model that is mainly a logical model but with aspects of a physical model. The first steps are to create tables based on the entities in the conceptual model and create columns for all columns which have a cardinality of 0 to 1 or 1 to 1 and specify the type, limit and constraints. The attributes with higher cardinality will be made into new tables. Then every entity that can exist on its own will receive a Primary Key(PK) or a Surrogate key.

Afterwards we can start creating the relations between the tables. The primary key is what will connect them and on the weak end the strong ends PK will be a Foreign Key(FK). For the Many to Many relationships we will have to create a Cross-Reference table containing both tables Primary Keys together as 2 Foreign Keys.

Finally we will verify that the model is normalized and that all planned operations are possible to perform on the data.

3.3 Task 3 Queries and Data Serialization

For this project the database is run with PostgreSQL via Docker with Adminer as the Database Management. The data is randomly generated with a python script that creates SQL code to insert. I manually verified the queries by comparing the results to the data tables.

3.4 Task 4 Programmatic Access

The projects programmatic access was done with Java and Maven. Before getting started the first goal was to understand the JDBC-Intro and JDBC-Bank programs, which followed with rewriting the JDBC-Intro since it had the settings configured and a clean slate to work on. First step was to create the MVC architecture and then rewrite the main parts from JDBC-Bank to the SoundGood application.

At the start, no actual logic was added, just the necessary methods, classes and folders. Then when the foundation of the application was done, the actual logic and likes was written. More methods etc was added during the course of writing the application depending on what missed details required it. But this application was intended to follow the structure of the JDBC-Bank application closely and work with similar actions.

4 Result

4.1 Task 1 Conceptual Model

See attachment 7.1 for the conceptual model.

4.1.1 Rental Instrument

First we have an class named RentalInstrument which contains the attributes instrumentID, returnDate, studentID, monthlyPrice and instrumentBrand. The purpose of this class is to store the data about each instrument being rented or being available for renting. Each instrument has an unique ID, a set price per month, and the brand of the instrument. Then a student can rent it and their studentID is saved, including the date which the student shall return it.

4.1.2 Contact Details

Every person involved in Sound Good Music School has their contact details saved in the class named ContactDetails which stores firstName, lastName, personNumber, email, age, phonenumber, street, zipcode, city and parents names. The students, staff and applications are connected to this information via the person number, and the amount which shall be filled out is connected to their position, I.E staff does not have to fill out their parents names.

4.1.3 Staff

The staff has employment ID and personNumber which connects to the contact details information.

4.1.4 Students

The data on students is kept with their own class which contains personNumber, familyID, skillLevel and studentID. Each student has an unique ID and a person number which connects to the contact details stored in a different class. Each student also have a skill level which is either beginner, intermediate or advanced. And they also have a family ID which siblings share.

4.1.5 Applications

The application stores personNumber, sibling and auditionVideo. It is connected to Contact Details to store the applicants information and the sibling is to connect it to the correct family, and the audition video is for applicants applying for advanced courses.

4.1.6 Salaries

Salaries contain amount, dueDate and employmentID. It is the monthly salary for an employment with the amount based on the lessons and their details. dueDate is the date which it will be paid out.

4.1.7 Payments

Payments contain amount, dueDate, studentID and siblingDiscount. The amount is based on lessons taken, dueDate is when the payment shall be paid and siblingDiscount is based on the amount of people in their families.

4.1.8 Lesson

Lesson contains lessonID, skillLevel, location, teacher and receipt. Each lesson have their own unique ID. Each lesson also has a skill level; beginner, intermediate or advanced. There is also a location and an allocated teacher for the lesson. And there is a receipt. Lesson is the parent entity to 3 sub types: Individual Lesson, Group Lesson and Ensembles

4.1.9 Individual Lesson

IndidivualLesson contains lessonType, time, instrument, studentID and price. lessonType specifies the type of lesson it is, in this case an individual lesson. Time specifies the time and date of the lesson. Instrument specifies the instrument being taught and the price is the amount which the student will have to pay later.

4.1.10 Group Lesson

IndidivualLesson contains lessonType, timeSlot, instrument, studentID and price. lessonType specifies the type of lesson it is, in this case a group lesson.

timeSlot specifies the time and date of the lesson. Instrument specifies the instrument being taught and the price is the amount which the student will have to pay later.

4.1.11 Ensembles

Ensembles contains lessonType, timeSlot, genre, studentID and price. lessonType specifies the type of lesson it is, in this case an individual lesson. Time specifies the time and date of the lesson. Genre specifies the genre being played and the price is the amount which the student will have to pay later.

4.2 Task 2 Logical and Physical Model

See attachment 7.2 for the model.
Link to the git repository: https://github.com/Willgar/IV1351

4.2.1 Rental Instrument

The table rental_instrument contains the PK instrument_id, the FK student_id, instrument_type, return_date, monthly_price and the instrument_brand.

Each instrument has its own ID and a FK student ID to the student renting the instrument. The student_id and return_date can be null to indicate that the instrument is not being rented.

4.2.2 Contact Details

The table contact_details contains the primary key person_number, first_name, last_name, age, home_number, parent_number, mobile_number, street, zip, city, parent_first_name and parent_last_name. Most of the names are self-explanatory and it is connected to the other tables student, application and staff.

4.2.3 Payment

The table payment contains the FK student_id, amount, due_date and sibling_discount. The names are self-explanatory and it is connected to the student table.

4.2.4 Student

The table student contains the PK student_id, email, skill_level, the FK person_number and family_id. It is connected to payment, rental_instrument, lesson_attendees, contact_details and application. Every unique family has their own family_id to identify them and check for the sibling discount. The students skill level has to match the lessons to attend them.

4.2.5 Salary

The table salary contains the FK employment_id, amount and due_date.

4.2.6 Staff

The table staff contains the PK employment_id, email and the FK person_number. It is connected to contact_details, application, salary and lesson.

4.2.7 Application

The table application contains the PK application_id, email, audition_video, the FK employment_id, the FK student_id, the FK person_number. The employment_id is for the employee that handles the application. It is connected to student, contact_details and staff.

4.2.8 Lesson

The table lesson contains the PK lesson_id, location, instrument, the FK employment_id, the FK receipt. It contains information about the lesson and is connected to receipt which contains more information about details relevant to the pricing. The students attending the lesson is stored in the table lesson_attendees.

4.2.9 Receipt

The table receipt contains the PK receipt, time, genre, price, skill_level, lesson_type, min_students and max_students. It is connected to lesson and provides information about costs and details about the lesson.

4.2.10 Lesson Attendees

The table lesson_attendees is the cross-reference table for students and lesson containing the PK attendance_id, the FK lesson_id and the FK student_id.

4.2.11 SQL Script

The script was created by using Astah's built in function to export diagrams to SQL.

4.3 Task 3 Queries and Data Serialization

4.3.1 First bullet point

The instructions of the first bullet point was: *"Show the number of instruments rented per month during a specified year. It shall be possible to retrieve the total number of rented instruments (just one number) and rented instruments of each kind (one number per kind, guitar, trumpet, etc). The latter list shall be sorted by number of rentals. This query is expected to be performed a few times per week."*

The SQL query can be found under attachment 7.3 and the result can be found under attachment 7.4. The query starts with listing the total amount of instruments rented during the year, followed by the total amount of instruments rented per month, and then the total amount of each type of instrument rented during the year. And lastly the separate types of instruments rented per month.

The query was tested by manually checking that the amount of rentals for the separate sections give the same amount and that it matches the database.

4.3.2 Second bullet point

The instructions of the second bullet point was: *"The same as above, but retrieve the average number of rentals per month during the entire year, instead of the total for each month."*

The SQL query can be found under attachment 7.5 and the result can be found under attachment 7.6, which gives an average of 5 instruments per month being rented. The results were compared to the database and that it matches the numbers of the first query.

4.3.3 Third bullet point

The instructions of the third bullet point was: *"Show the number of lessons given per month during a specified year. It shall be possible to retrieve the total number of lessons (just one number) and the specific number of individual lessons, group lessons and ensembles. This query is expected to be performed a few times per week."*

The SQL query can be found under attachment 7.7 and the result can be found under attachment 7.8. The query first lists the total count of the amount of unique lesson_id occurrences in the table lesson and then lists the separate types of lessons afterwards. Then it lists total amount of lessons per month. The testing was done manually by comparing the results to the database and seeing how the sum of the different lesson types equaled the total amount of lessons.

4.3.4 Fourth bullet point

The instructions of the fourth bullet point 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."*

The SQL query can be found under attachment 7.9 and the result can be found under attachment 7.10. This query is a similar version to the third except that it calculates the average based on the results and gives an average of 42 lessons per month which averages up correctly based on the amount of lessons and months.

4.3.5 Fifth bullet point

The instructions of the fifth bullet point was: *"List all instructors who has given more than a specific number of lessons during the current month. Sum all lessons, independent of type. Also list the three instructors having given most lessons (independent of lesson type) during the last month, sorted by number of given lessons. This query will be used to find instructors risking to work too much, and will be executed daily."*

The SQL query can be found under attachment 7.11 and the result can be found under attachment 7.12. The query starts with listing the three employees who have worked the most amount of lessons and then lists all the instructors who have given more than 2 lessons during the month of December. The testing is done manually by comparing the results to the database.

4.3.6 Sixth bullet point

The instructions of the sixth bullet point 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 SQL query can be found under attachment 7.13 and the result can be found under attachment 7.14. The query lists all lessons until a set date, which is in this test example larger than a week forward, and writes out the date and the amount of spots left and the genre. It then sorts everything by date and then by the genre. The testing is done manually by comparing the results to the database.

4.3.7 Seventh bullet point

The instructions of the sixth bullet point was: *"List the three instruments with the lowest monthly rental fee. For each instrument tell whether it is rented or available to rent. Also tell when the next group lesson for each listed instrument is scheduled."*

The SQL query can be found under attachment 7.15 and the result can be found under attachment 7.16. The query lists the 3 cheapest instruments, lists the availability of them, lists the monthly price and lists the next group lesson with the type of instrument. The testing is done manually by comparing the results to the database.

4.4 Task 4 Programmatic Access

Link to the git repository: https://github.com/Willgar/IV1351

4.4.1 Startup

The program starts in Main where the Controller and View is created, passing down the Controller with the View. Like everything else, we have database exceptions ready to catch any errors.

4.4.2 View

The view consists of 3 files based on the JDBC-Bank, BlockingIntepreter, CmdLine and Command. This is the part which interacts with the user in the terminal, where all the commands are stored in Command, CmdLine contains functions for the program to use the terminal and get the input of the user and BlockingInterpreter which interprets the commands and with the Controller acts upon the users interactions.

4.4.3 Controller

The controller receives the user input from the View and either retrieves information to the user from the Model or changes information in the Model. Any information that is retrieved from the Model is gathered from Integration which accesses the database and in this application creates objects named instrument.

The Controller has 3 different actions it can perform based on the project requirements and it is to rent an instrument, list all instruments that can be rented and search for specific types of instruments to rent and lastly to terminate an ongoing rental.

4.4.4 Model

The model is what manages the data and logic. All the instruments are stored in the model which contains Instrument, InstrumentDTO and InstrumentException. The Instrument class keeps the information of each instrument and with the InstrumentDTO we can pass the means of receiving the data secured from the other layers by not giving out the data straight away and instead giving the other layer like Integration a way to gather the information via the DTO.

4.4.5 Integration

This is the layer which accesses the database with SoundGoodDAO as the Data Access Object. At creation of the object, it attempts to connect to the database and then prepares statements of SQL code using the established connection. Using prepared statements saves time when accessing the database compared to normal statements.

When the database needs to be accessed, the Controller calls for the SoundGoodDAO to use certain functions to perform certain actions with the database. The transaction begins with the functions in SoundGoodDAO making changes to the statements and inserting data to the SQL code. If everything goes as planned, the connection finishes and commits the transaction which ends it. If an error occurs during the transaction, the SQLException which catch it and handle the exception by attempting to rollback the connection. If the rollback also fails then then an error will simply be written out.

4.5 Task 5 Performance

4.5.1 First query

The first query searches the data first based on month and then groups them by instrument type, so naturally the solution to increasing performance would be to create a composite index with return_date and instrument_type. For this I would choose to have a non-clustered index since they seem to go well with composite keys and take up less space.

4.5.2 Second query

The second query searches data based on the date so a single index with return_date would be sufficient. This would mean that the rental_instrument relation could have a return_date as one index and return_date and instrument_type as a single composite index. I would choose a non-clustered index for this too if going with the composite key and a clustered index if going with just time as a key.

4.5.3 Third query

Similar to the first query, the query searches based on the date and the type of lesson, so a composite index with time and lesson_type could be a solution. Clustered index would fit here too, especially since there aren't too many inserts and updates done in the lesson.

4.5.4 Fourth query

Since the type of lesson is not relevant in this query, a single index with time would be good. This means that the receipt relation could have a single index time and a composite index with time and lesson_type. For the single index I would choose a clustered index and for the composite index I would go with the non-clustered index.

4.5.5 Fifth query

The fifth query searches data based on how many occurrences there are of an employment_id in lesson and the date from receipt. We would potentially already have a index based on time from the fourth query so it is just necessary for a index on employment_id to sort them and potentially ease the counting. We could also have an index on the receipt in lesson since that is a foreign key and it would ease connecting with the join function. Since we would have a single index, I would choose a clustered index.

4.5.6 Sixth query

A majority of the columns in the WHERE clause consists of Primary Keys and the few non PK would be from the relation receipt with lesson_type and time which we already have a index for. Therefore I believe that there is no need to make a new index for the sixth query and we can use a non-clustered index.

4.5.7 Seventh query

A composite key consisting of monthly_price and instrument_type for the rental_instrument relation and time and lesson_type for the receipt would fit for the query since we are getting the 3 lowest monthly_price and the minimum time based on the lesson_type. And then a non-clustered index would fit the composite key.

5 Discussion

5.1 Task 1 Conceptual Model

The naming conventions are retrieved from the Conceptual Model video from the course and is as following: Entities/Classes are upper camel case like IndidivudalLesson or Lesson and attribute names are lower camel case like lessonType and instrument. All entities and attributes in the conceptual model follow these naming conventions.

For the current purposes and needs, I believe that there is a reasonable number of entities for the conceptual model. Some flaws have been found at a later point and adjusted during the course of the project, but it remains in structure similar to the earlier version.

Some attributes have special cardinality attributed to them but most have a standard cardinality of 0 to 1.

All relations have cardinality at both ends and name at at least one end. The exception is the sub types of Lesson since I thought it seemed a bit redundant.

I believe that most business rules are written out in notes unless it is part of the diagram.

For the sake of trying to learn more about the Conceptual Model I attempted the higher grade part and split up the lesson and had Inheritance to specify its sub types Individual lesson, group lesson and ensemble. They are all very similar to each other with a few differences. One advantage is that looking at the model makes it clearer for the viewer to understand that there are more types of lessons than just one. One advantage of not using it is that it is easier to write a model not using Inheritance, especially since it is more relevant in Object-oriented situations and less in databases. In the logical/physical model I made the lesson to be of 2 parts, Lesson and Receipt which together define what type of lesson it is. I consider this an example of how it could have been modeled without using Inheritance.

Based on a few other course students examples of their own models I've seen I believe that mine might appear a bit simple, but I also can't motivate myself as to why I should make a vast over complicated conceptual model when it will be easier for another user to view and understand the model if it is kept simple but still satisfying the requirements. And after working on the logical/physical model I believe that my model worked quite well to convey the reality of a Music School Database.

The only downside I realise in hindsight is that it might appear to be a bit programmatic model which is a common mistake when making domain models, which the conceptual model is related to considering both are created with similar instructions. I have however managed to stay away from a spider-in-the-web type of model and the other common mistakes brought up in the Object-oriented development course.

5.2 Task 2 Logical and Physical Model

The naming convention for this model is snake case. This implies that everything is in lower case and words are combined with underscore. The naming convention is followed in this model. I've attempted to make all the names self-explanatory.

As far as I know, the model is in 3NF. I attempted to use a third party modelling software to confirm that it was in 3NF and it proved helpful, even if the main work was done beforehand.

For the project requirements, all the tables are relevant. However based on work done in task 3 and task 4, the application table seems redundant since it is never used.

There are columns for all data that shall be stored and other information can be retrieved based on the data that is stored. For example, there is no column for if an instrument is rented, but knowing that the return_date column is null we also know that no one is renting the instrument. There are some small constraints that could be improved but the benefit is minor and have been left as is. The column types are primarily varchar and timestamp with a few integer since the SQL queries did not want to to arithmetic with varchar in task 3. The limits are dependant on what we are storing, for example a limit 12 for the person_number and limit 50 for names and words. The name/word limit could be lowered but I wouldn't increase it since 50 letters are more than enough for most purposes.

Most Primary Keys can be motivated but I realise in hindsight some might be a bit redundant. For example could the lesson_attendees PK be a Surrogate Key of lesson_id and student_id, but I've left it since there is no harm done. And person_number could be an attribute to a Primary Key person_id but the person_number works like an ID in reality already.

I believe that the cardinality of the relations are correctly specified and believe most of the types are correct too. I believe that all of the tasks can be performed, some changes have been done during the work of task 3 to satisfy those needs. The few business rules and constraints that are not specified in the diagram is instead explained in notes in the diagram.

5.3 Task 3 Queries and Data Serialization

I made all the queries to be either a view or a materialized view. There was none that was not either one of those. When choosing which type of view it was supposed I reasoned that the first 4 queries where only expected to be performed a few times per week so the urgency to have updated information wasn't that high and therefore we could have it as a view. But for the the 2 queries that was performed programmatically and displayed it seemed more reasonable to have them be a materialized view so that they always had the relevant data. The fifth query that lists the instructor seemed to be in between being performed programmatically and just a few times a week so I chose to make it a materialized view since it is still being performed once a day.

Another issue I faced was whether I should make the bullet points as multiple queries or one single query that might not look as neat. I started by making them as multiple smaller queries before changing my mind and merging the query results together and making a single query for each bullet point.

5.4 Task 4 Programmatic Access

All code is to follow the java naming conventions, and any instance where it does not follow is not intentional. Methods use lower camel case like rentInstrument and classes have upper camel case like InstrumentException.

The code uses try catch statements in a majority of tasks, and catching an error while the application is dealing with transactions causes the connection to try to perform a rollback instead of continuing and committing. When no error happens, the connection commits the transaction. The application performs transaction management correctly to my knowledge and is heavily inspired by the JDBC-Bank application and how they deal with the transaction management.

The program works as I expect it to work and can run the necessary commands as the project specifies. It is however not made to be usable to the average user since any person can change any students rentals without any safety measure at the moment.

To ease any verifying, I've placed the SQL code to create the database and the SQL code to insert random data into the database in the resource folder located in the soundgood application folder. I've focused on trying to follow the MVC architecture, having split up the program into Controller, Integration, Model, Startup and View. And then also putting each part of the program into their respective folder to perform their part.

I've tried to make the code easy to understand. All methods are named after what they do and smaller logic parts in methods are made into their own private methods that the previous method calls upon. All public methods is commented upon but no private methods are commented, as was taught in the Object-oriented Development course.

5.5 Task 5 Performance

The focus on indices for the queries and relations was for columns used in the WHERE clauses, like time and return_date were used in the first to sixth query. For example in the first query, we retrieve instrument information based on checking in a WHERE clause if the return_date equals a certain month, and the same for time. We do the same type of comparison in the first, second, third and fourth query and compare time/return_date.

For the queries which only have the date in the WHERE clause, I chose to make some of the index suggestions of being a composite index with the date and another column. For example in the first query I chose the instrument_type since it is the next most important part of the query since that is the other part which is retrieved and sorted.

The third one could be argued the same as the first one, with time since it is part of the WHERE clause and lesson_type since it is what the query selects.

The fifth query however was more focused on utilizing the foreign key receipt in lesson to ease the search for the matching receipt in the relation receipt. We would also potentially be able to reuse an index on time from previous queries.

Just the same as for the sixth query and the foreign key lesson_id in lesson_attendees, and a the receipt index from the fifth query, but due to being from multiple tables all the indices would have to be separate.

So to summarize the focus of indices was primarily on the items in the WHERE clause which was most of the times either a date or a foreign key being compared to the primary key of another table. Where there were no other important parts except for the dates, a suggestion was to create composite indices with the other column from the SELECT statement was added.

For the few queries that have both an index for the time/return_date and an index for e.g. lesson_type/instrument_type I would choose the time/return_date index as the primary index since I believe it is better suited as a clustered index since most of the dates are more or less unique with only a few occurrences of lessons being on the same date in this database, so the ordering would be more specific rather than having lesson_type which can only sort the table in 3 sections. The time/return_date columns are also used in the WHERE clause and should therefore also be prioritized over

6 Comments about the course

The topics brought in this course were interesting and the Task 3 and Task 4 problems were fun to solve. Due to the other course I read parallel to this course, I felt that there wasn't enough time to spend at the assignment at the first half period, and only just enough time to prepare for the quizzes, but this was mostly due to my own lack of time management. But thanks to that course finishing up early, IV1351 got my undivided attention the last 3 weeks of the course to put full time into it.

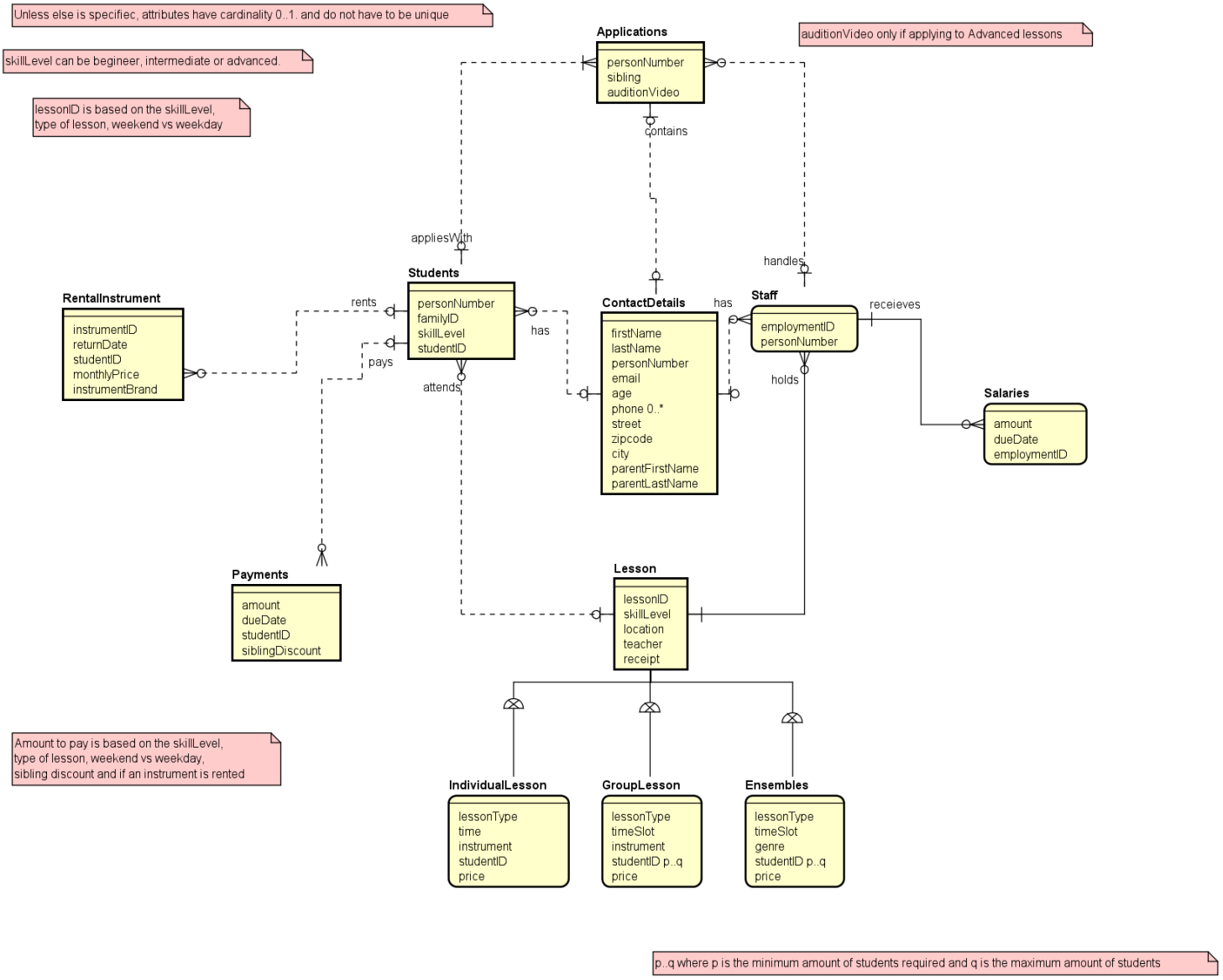
It is difficult to comment too much on the aspects since this will probably be the only time the course is run on distance due to Covid-19, but the only major part I felt was missing was the ability to get feedback on the work. Even if we could send in questions for Piazza, there was still an urge to be able to discuss face to face with an assistant(or webcam to webcam). I liked how the course IV1350 was held with biweekly seminars for the tasks, even if this freedom to work on the project whenever also was nice.

However for short and small questions, I think that Piazza is great. Especially being able to read through other peoples questions and finding problems I had yet to encounter and not having to worry about that. By just roughly estimating the time spent on the project itself, I would guess around 80-100 hours.

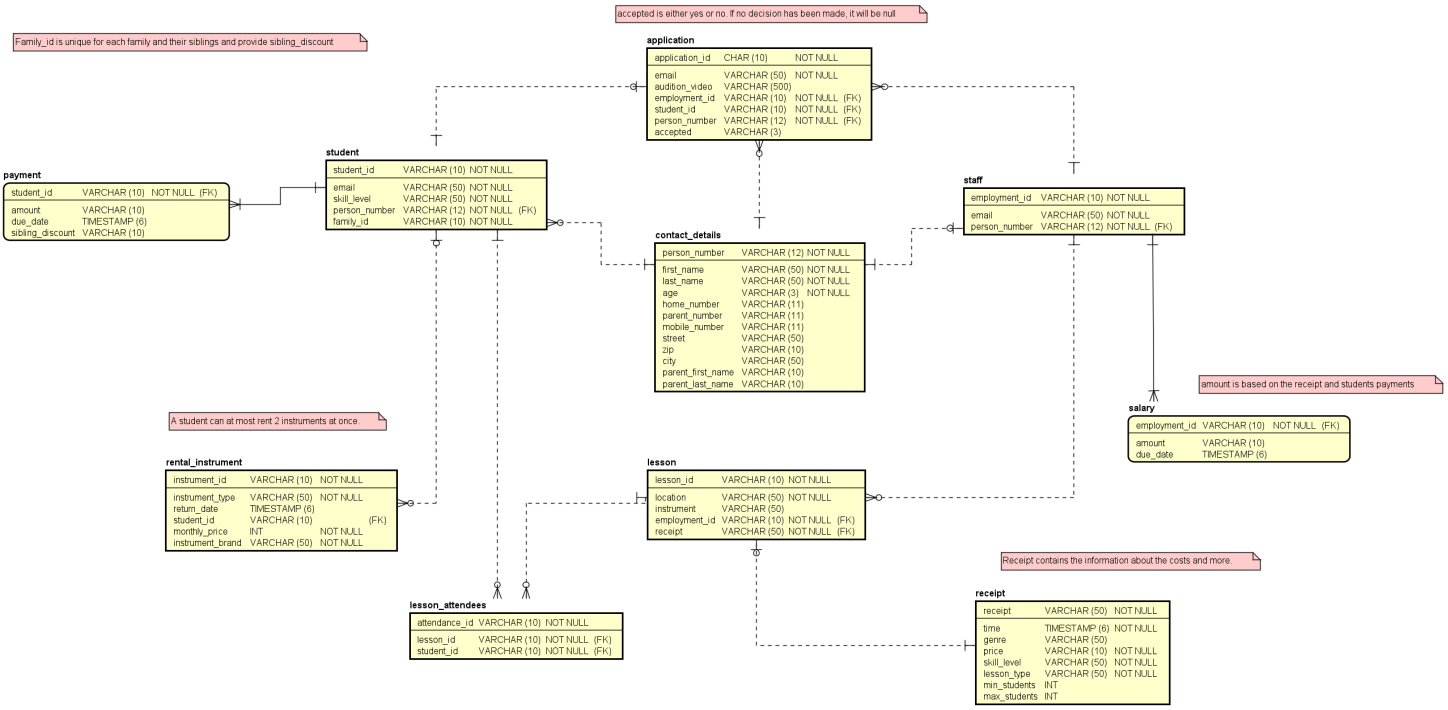
7 Attachments

7.1 Conceptual Model
7.2 Logical/Physical Model
7.3 First query
7.4 Result of first query
7.5 Second query
7.6 Result of second query
7.7 Third query
7.8 Result of third query
7.9 Fourth query
7.10 Result of fourth query
7.11 Fifth query
7.12 Result of fifth query
7.13 Sixth query
7.14 Result of sixth query
7.15 Seventh query
7.16 Result of seventh query
7.17 Printout of a sample run

7.1 Conceptual Model



7.2 Logical/Physical Model



7.3 First query

```
CREATE VIEW RentedInstruments AS
SELECT 'Total' as month, count(instrument_id) AS rentals,
'All instruments' as instrument
FROM rental_instrument
WHERE return_date IS NOT NULL
AND (EXTRACT(YEAR FROM return_date))=2021
UNION ALL
(SELECT 'Jan', COUNT(instrument_id), 'All instruments'
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 1
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT 'Feb',COUNT(instrument_id), 'All instruments'
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 2
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT 'Mars',COUNT(instrument_id), 'All instruments'
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 3
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
( SELECT 'April',COUNT(instrument_id), 'All instruments'
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 4
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT 'May',COUNT(instrument_id), 'All instruments'
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 5
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT 'June',COUNT(instrument_id), 'All instruments'
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 6
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT 'July',COUNT(instrument_id), 'All instruments'
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 7
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT 'Aug',COUNT(instrument_id), 'All instruments'
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 8
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT 'Sep',COUNT(instrument_id), 'All instruments'
```

```
49 FROM rental_instrument
50 WHERE (EXTRACT(MONTH FROM return_date)) = 9
51 AND (EXTRACT(YEAR FROM return_date))=2021)
52 UNION ALL
53 (SELECT 'Oct',COUNT(instrument_id), 'All␣instruments'
54 FROM rental_instrument
55 WHERE (EXTRACT(MONTH FROM return_date)) = 10
56 AND (EXTRACT(YEAR FROM return_date))=2021)
57 UNION ALL
58 (SELECT 'Nov',COUNT(instrument_id), 'All␣instruments'
59 FROM rental_instrument
60 WHERE (EXTRACT(MONTH FROM return_date)) = 11
61 AND (EXTRACT(YEAR FROM return_date))=2021)
62 UNION ALL
63 (SELECT 'Dec',COUNT(instrument_id), 'All␣instruments'
64 FROM rental_instrument
65 WHERE (EXTRACT(MONTH FROM return_date)) = 12
66 AND (EXTRACT(YEAR FROM return_date))=2021)
67
68 UNION ALL
69 SELECT 'Total',COUNT(instrument_id), instrument_type
70 FROM rental_instrument
71 WHERE return_date IS NOT NULL
72 AND (EXTRACT(YEAR FROM return_date))=2021
73 GROUP BY instrument_type
74 UNION ALL
75 SELECT 'Jan', COUNT(instrument_id), instrument_type
76 FROM rental_instrument
77 WHERE (EXTRACT(MONTH FROM return_date)) = 1
78 AND (EXTRACT(YEAR FROM return_date))=2021
79 GROUP BY instrument_type
80 UNION ALL
81 SELECT 'Feb',COUNT(instrument_id), instrument_type
82 FROM rental_instrument
83 WHERE (EXTRACT(MONTH FROM return_date)) = 2
84 AND (EXTRACT(YEAR FROM return_date))=2021
85 GROUP BY instrument_type
86 UNION ALL
87 SELECT 'Mars',COUNT(instrument_id), instrument_type
88 FROM rental_instrument
89 WHERE (EXTRACT(MONTH FROM return_date)) = 3
90 AND (EXTRACT(YEAR FROM return_date))=2021
91 GROUP BY instrument_type
92 UNION ALL
93 SELECT 'April',COUNT(instrument_id), instrument_type
94 FROM rental_instrument
95 WHERE (EXTRACT(MONTH FROM return_date)) = 4
96 AND (EXTRACT(YEAR FROM return_date))=2021
97 GROUP BY instrument_type
98 UNION ALL
```

```
99  SELECT 'May',COUNT(instrument_id), instrument_type
100 FROM rental_instrument
101 WHERE (EXTRACT(MONTH FROM return_date)) = 5
102 AND (EXTRACT(YEAR FROM return_date))=2021
103 GROUP BY instrument_type
104 UNION ALL
105 SELECT 'June',COUNT(instrument_id), instrument_type
106 FROM rental_instrument
107 WHERE (EXTRACT(MONTH FROM return_date)) = 6
108 AND (EXTRACT(YEAR FROM return_date))=2021
109 GROUP BY instrument_type
110 UNION ALL
111 SELECT 'July',COUNT(instrument_id), instrument_type
112 FROM rental_instrument
113 WHERE (EXTRACT(MONTH FROM return_date)) = 7
114 AND (EXTRACT(YEAR FROM return_date))=2021
115 GROUP BY instrument_type
116 UNION ALL
117 SELECT 'Aug',COUNT(instrument_id), instrument_type
118 FROM rental_instrument
119 WHERE (EXTRACT(MONTH FROM return_date)) = 8
120 AND (EXTRACT(YEAR FROM return_date))=2021
121 GROUP BY instrument_type
122 UNION ALL
123 SELECT 'Sep',COUNT(instrument_id), instrument_type
124 FROM rental_instrument
125 WHERE (EXTRACT(MONTH FROM return_date)) = 9
126 AND (EXTRACT(YEAR FROM return_date))=2021
127 GROUP BY instrument_type
128 UNION ALL
129 SELECT 'Oct',COUNT(instrument_id), instrument_type
130 FROM rental_instrument
131 WHERE (EXTRACT(MONTH FROM return_date)) = 10
132 AND (EXTRACT(YEAR FROM return_date))=2021
133 GROUP BY instrument_type
134 UNION ALL
135 SELECT 'Nov',COUNT(instrument_id), instrument_type
136 FROM rental_instrument
137 WHERE (EXTRACT(MONTH FROM return_date)) = 11
138 AND (EXTRACT(YEAR FROM return_date))=2021
139 GROUP BY instrument_type
140 UNION ALL
141 SELECT 'Dec',COUNT(instrument_id), instrument_type
142 FROM rental_instrument
143 WHERE (EXTRACT(MONTH FROM return_date)) = 12
144 AND (EXTRACT(YEAR FROM return_date))=2021
145 GROUP BY instrument_type
```

7.4 Result of first query

month	rentals	instrument
Total	60	All instruments
Jan	4	All instruments
Feb	7	All instruments
Mars	2	All instruments
April	4	All instruments
May	6	All instruments
June	7	All instruments
July	3	All instruments
Aug	4	All instruments
Sep	5	All instruments
Oct	6	All instruments
Nov	3	All instruments
Dec	9	All instruments
Total	6	bass
Total	5	drums
Total	10	flute
Total	9	guitar
Total	8	piano
Total	3	saxophone
Total	9	trumpet
Total	10	violin
Jan	1	flute
Jan	1	piano
Jan	1	trumpet
Jan	1	violin
Feb	3	bass
Feb	1	flute
Feb	1	saxophone
Feb	1	trumpet
Feb	1	violin
Mars	1	piano
Mars	1	violin
April	1	bass
April	1	flute
April	1	guitar
April	1	trumpet
May	2	guitar
May	2	piano
May	1	saxophone
May	1	trumpet
June	3	flute
June	1	guitar
June	1	piano
June	1	trumpet
June	1	violin
July	1	flute
July	1	guitar
July	1	trumpet
Aug	2	drums
Aug	1	flute
Aug	1	violin
Sep	1	bass
Sep	1	drums
Sep	1	saxophone
Sep	1	trumpet
Sep	1	violin
Oct	1	drums
Oct	1	flute
Oct	2	guitar
Oct	1	piano
Oct	1	violin
Nov	1	piano
Nov	1	trumpet
Nov	1	violin
Dec	1	bass
Dec	1	drums
Dec	1	flute
Dec	2	guitar
Dec	1	piano
Dec	1	trumpet
Dec	2	violin

7.5 Second query

```
CREATE VIEW AverageInstrumentsPerMonth AS
SELECT AVG(count)
FROM ((SELECT COUNT(instrument_id)
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 1
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT COUNT(instrument_id)
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 2
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT COUNT(instrument_id)
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 3
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
( SELECT COUNT(instrument_id)
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 4
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT COUNT(instrument_id)
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 5
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT COUNT(instrument_id)
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 6
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT COUNT(instrument_id)
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 7
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT COUNT(instrument_id)
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 8
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT COUNT(instrument_id)
FROM rental_instrument
WHERE (EXTRACT(MONTH FROM return_date)) = 9
AND (EXTRACT(YEAR FROM return_date))=2021)
UNION ALL
(SELECT COUNT(instrument_id)
```

```
49 FROM rental_instrument
50 WHERE (EXTRACT(MONTH FROM return_date)) = 10
51 AND (EXTRACT(YEAR FROM return_date))=2021)
52 UNION ALL
53 (SELECT COUNT(instrument_id)
54 FROM rental_instrument
55 WHERE (EXTRACT(MONTH FROM return_date)) = 11
56 AND (EXTRACT(YEAR FROM return_date))=2021)
57 UNION ALL
58 (SELECT COUNT(instrument_id)
59 FROM rental_instrument
60 WHERE (EXTRACT(MONTH FROM return_date)) = 12
61 AND (EXTRACT(YEAR FROM return_date))=2021)) AS total_rental
```

7.6 Result of second query

average_rentals_per_month
5.0000000000000000

7.7 Third query

```
CREATE VIEW AmountOfLessons AS
SELECT 'total' as lesson_type, 'All year' as date,
COUNT(receipt) as amount_of_lessons
FROM receipt
WHERE (EXTRACT(YEAR FROM time))=2021
UNION ALL
SELECT lesson_type,'All Year', COUNT(lesson_type)
FROM receipt
GROUP BY lesson_type
UNION ALL
(SELECT 'total','Jan',COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 1
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT 'total','Feb',COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 2
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT 'total','Mars',COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 3
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
( SELECT 'total','April',COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 4
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT 'total','May',COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 5
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT 'total','June',COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 6
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT 'total','July',COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 7
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT 'total','Aug',COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 8
```

```
49 AND (EXTRACT(YEAR FROM time))=2021)
50 UNION ALL
51 (SELECT 'total','Sep',COUNT(receipt)
52 FROM receipt
53 WHERE (EXTRACT(MONTH FROM time)) = 9
54 AND (EXTRACT(YEAR FROM time))=2021)
55 UNION ALL
56 (SELECT 'total','Oct',COUNT(receipt)
57 FROM receipt
58 WHERE (EXTRACT(MONTH FROM time)) = 10
59 AND (EXTRACT(YEAR FROM time))=2021)
60 UNION ALL
61 (SELECT 'total','Nov',COUNT(receipt)
62 FROM receipt
63 WHERE (EXTRACT(MONTH FROM time)) = 11
64 AND (EXTRACT(YEAR FROM time))=2021)
65 UNION ALL
66 (SELECT 'total','Dec',COUNT(receipt)
67 FROM receipt
68 WHERE (EXTRACT(MONTH FROM time)) = 12
69 AND (EXTRACT(YEAR FROM time))=2021)
```

7.8 Result of third query

lesson_type	date	amount_of_lessons
total	All year	500
individual	All Year	167
ensemble	All Year	169
group	All Year	164
total	Jan	38
total	Feb	45
total	Mars	47
total	April	33
total	May	35
total	June	41
total	July	51
total	Aug	47
total	Sep	43
total	Oct	36
total	Nov	38
total	Dec	46

7.9 Fourth query

```
CREATE VIEW AverageLessonsPerMonth AS
SELECT AVG(count)
FROM((SELECT COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 1
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 2
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 3
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
( SELECT COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 4
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 5
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 6
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 7
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 8
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT COUNT(receipt)
FROM receipt
WHERE (EXTRACT(MONTH FROM time)) = 9
AND (EXTRACT(YEAR FROM time))=2021)
UNION ALL
(SELECT COUNT(receipt)
```

```
49 FROM receipt
50 WHERE (EXTRACT(MONTH FROM time)) = 10
51 AND (EXTRACT(YEAR FROM time))=2021)
52 UNION ALL
53 (SELECT COUNT(receipt)
54 FROM receipt
55 WHERE (EXTRACT(MONTH FROM time)) = 11
56 AND (EXTRACT(YEAR FROM time))=2021)
57 UNION ALL
58 (SELECT COUNT(receipt)
59 FROM receipt
60 WHERE (EXTRACT(MONTH FROM time)) = 12
61 AND (EXTRACT(YEAR FROM time))=2021)) AS total_lesson
```

7.10 Result of fourth query

avg
41.66666666666667

7.11 Fifth query

```
CREATE MATERIALIZED VIEW LessonsByInstructors AS
(SELECT 'Top three Dec' AS section,foo.employment_id,
COUNT(foo.employment_id) AS number_of_lessons
FROM (SELECT lesson.employment_id, receipt.time
FROM lesson
LEFT JOIN  receipt ON receipt.receipt= lesson.receipt
WHERE (EXTRACT(MONTH FROM time)) = 12
GROUP BY lesson.employment_id, receipt.time) AS foo
GROUP BY foo.employment_id
ORDER BY COUNT(*) DESC
LIMIT 3)
UNION ALL
(SELECT 'Dec', foo.employment_id, COUNT(foo.employment_id)
FROM (SELECT lesson.employment_id, receipt.time
FROM lesson
LEFT JOIN  receipt ON receipt.receipt= lesson.receipt
WHERE (EXTRACT(MONTH FROM time)) = 12
GROUP BY lesson.employment_id, receipt.time) AS foo
GROUP BY foo.employment_id
HAVING COUNT(foo.employment_id) > 2)
```

7.12 Result of fifth query

section	employment_id	number_of_lessons
Top three Dec	EI1007	6
Top three Dec	EI1000	5
Top three Dec	EI1001	5
Dec	EI1000	5
Dec	EI1001	5
Dec	EI1002	3
Dec	EI1003	5
Dec	EI1004	4
Dec	EI1006	4
Dec	EI1007	6
Dec	EI1008	3
Dec	EI1009	5

7.13 Sixth query

```
CREATE MATERIALIZED VIEW UpcomingEnsembles AS
SELECT lesson_attendees.lesson_id,
CASE
WHEN (receipt.max_students-COUNT(lesson_attendees.lesson_id)) < 1
THEN 'Fully␣Booked'
WHEN receipt.max_students-COUNT(lesson_attendees.lesson_id) > 3
THEN 'Many␣seats␣left'
ELSE CAST(receipt.max_students-COUNT(lesson_attendees.lesson_id) AS varchar(255))
END AS spots_left, receipt.time, receipt.genre
FROM lesson_attendees, lesson, receipt
WHERE lesson.lesson_id = lesson_attendees.lesson_id
AND receipt.receipt = lesson.receipt
AND receipt.lesson_type = 'ensemble'
AND receipt.time < TO_TIMESTAMP('2021-02-28','YYYY-MM-DD␣HH24:MI:SS')
GROUP BY lesson_attendees.lesson_id, receipt.receipt, receipt.genre
ORDER BY time ASC, receipt.genre
```

7.14 Result of sixth query

lesson_id	spots_left	time	genre
LI1123	Many seats left	2021-01-01 00:00:00	country
LI1436	Many seats left	2021-01-03 00:00:00	metal
LI1281	Many seats left	2021-01-06 00:00:00	jazz
LI1492	2	2021-01-07 00:00:00	rock
LI1498	2	2021-01-07 00:00:00	rock
LI1396	Many seats left	2021-01-13 00:00:00	classic
LI1023	Many seats left	2021-01-15 00:00:00	metal
LI1418	Many seats left	2021-01-19 00:00:00	rock
LI1021	Many seats left	2021-01-24 00:00:00	rock
LI1223	Many seats left	2021-01-25 00:00:00	country
LI1421	2	2021-01-27 00:00:00	metal
LI1369	Many seats left	2021-02-01 00:00:00	country
LI1255	Many seats left	2021-02-03 00:00:00	rock
LI1075	Many seats left	2021-02-06 00:00:00	classic
LI1091	3	2021-02-12 00:00:00	rock
LI1303	3	2021-02-13 00:00:00	country
LI1128	1	2021-02-16 00:00:00	country
LI1301	Many seats left	2021-02-21 00:00:00	rock
LI1072	Many seats left	2021-02-22 00:00:00	jazz
LI1109	Many seats left	2021-02-23 00:00:00	metal
LI1405	Fully Booked	2021-02-27 00:00:00	metal

7.15 Seventh query

```
CREATE MATERIALIZED VIEW TopThreeCheapInstrument AS
SELECT DISTINCT instrument_id, instrument_type,
monthly_price, MIN(time) as next_lesson,
CASE
WHEN rental_instrument.return_date IS NOT NULL THEN 'Rented'
ELSE 'Available'
END AS availability
FROM rental_instrument, receipt, lesson
WHERE instrument = instrument_type
AND lesson.receipt = receipt.receipt AND lesson_type = 'group'
GROUP BY instrument_id, instrument_type, monthly_price, instrument
ORDER BY monthly_price ASC
LIMIT 3
```

7.16 Result of seventh query

instrument_id	instrument_type	monthly_price	next_lesson	availability
II1019	drums	65	2021-01-13 00:00:00	Available
II1029	violin	70	2021-01-05 00:00:00	Rented
II1091	bass	70	2021-01-12 00:00:00	Rented

7.17 Printout of a sample run

```
> help
list
help
quit
rent
end
> list piano
Instrument ID II1028, Instrument Type: piano, Monthly Price: 110, Instrument Brand: roland
Instrument ID II1046, Instrument Type: piano, Monthly Price: 90, Instrument Brand: steinway musical instruments
Instrument ID II1047, Instrument Type: piano, Monthly Price: 90, Instrument Brand: sennheiser
Instrument ID II1052, Instrument Type: piano, Monthly Price: 150, Instrument Brand: sennheiser
Instrument ID II1061, Instrument Type: piano, Monthly Price: 70, Instrument Brand: kawai
Instrument ID II1068, Instrument Type: piano, Monthly Price: 130, Instrument Brand: yamaha
Instrument ID II1088, Instrument Type: piano, Monthly Price: 90, Instrument Brand: fender musical instruments
Instrument ID II1090, Instrument Type: piano, Monthly Price: 110, Instrument Brand: steinway musical instruments
Instrument ID II1016, Instrument Type: piano, Monthly Price: 90, Instrument Brand: steinway musical instruments
Instrument ID II1025, Instrument Type: piano, Monthly Price: 130, Instrument Brand: roland
Instrument ID II1022, Instrument Type: piano, Monthly Price: 110, Instrument Brand: steinway musical instruments
> rent II1022 SI1011
Instrument II1022 is now being rented to student SI1011
> rent II1028 SI1011
Instrument II1028 is now being rented to student SI1011
> list piano
Instrument ID II1046, Instrument Type: piano, Monthly Price: 90, Instrument Brand: steinway musical instruments
Instrument ID II1047, Instrument Type: piano, Monthly Price: 90, Instrument Brand: sennheiser
Instrument ID II1052, Instrument Type: piano, Monthly Price: 150, Instrument Brand: sennheiser
Instrument ID II1061, Instrument Type: piano, Monthly Price: 70, Instrument Brand: kawai
Instrument ID II1068, Instrument Type: piano, Monthly Price: 130, Instrument Brand: yamaha
Instrument ID II1088, Instrument Type: piano, Monthly Price: 90, Instrument Brand: fender musical instruments
Instrument ID II1090, Instrument Type: piano, Monthly Price: 110, Instrument Brand: steinway musical instruments
Instrument ID II1016, Instrument Type: piano, Monthly Price: 90, Instrument Brand: steinway musical instruments
Instrument ID II1025, Instrument Type: piano, Monthly Price: 130, Instrument Brand: roland
> rent II1025 SI1011
User already has 2 instruments
> list piano
Instrument ID II1046, Instrument Type: piano, Monthly Price: 90, Instrument Brand: steinway musical instruments
Instrument ID II1047, Instrument Type: piano, Monthly Price: 90, Instrument Brand: sennheiser
Instrument ID II1052, Instrument Type: piano, Monthly Price: 150, Instrument Brand: sennheiser
Instrument ID II1061, Instrument Type: piano, Monthly Price: 70, Instrument Brand: kawai
Instrument ID II1068, Instrument Type: piano, Monthly Price: 130, Instrument Brand: yamaha
Instrument ID II1088, Instrument Type: piano, Monthly Price: 90, Instrument Brand: fender musical instruments
Instrument ID II1090, Instrument Type: piano, Monthly Price: 110, Instrument Brand: steinway musical instruments
Instrument ID II1016, Instrument Type: piano, Monthly Price: 90, Instrument Brand: steinway musical instruments
Instrument ID II1025, Instrument Type: piano, Monthly Price: 130, Instrument Brand: roland
> end II1022
Rental for instrument II1022 is now terminated.
> end II1028
Rental for instrument II1028 is now terminated.
> list piano
Instrument ID II1046, Instrument Type: piano, Monthly Price: 90, Instrument Brand: steinway musical instruments
Instrument ID II1047, Instrument Type: piano, Monthly Price: 90, Instrument Brand: sennheiser
Instrument ID II1052, Instrument Type: piano, Monthly Price: 150, Instrument Brand: sennheiser
Instrument ID II1061, Instrument Type: piano, Monthly Price: 70, Instrument Brand: kawai
Instrument ID II1068, Instrument Type: piano, Monthly Price: 130, Instrument Brand: yamaha
Instrument ID II1088, Instrument Type: piano, Monthly Price: 90, Instrument Brand: fender musical instruments
Instrument ID II1090, Instrument Type: piano, Monthly Price: 110, Instrument Brand: steinway musical instruments
Instrument ID II1016, Instrument Type: piano, Monthly Price: 90, Instrument Brand: steinway musical instruments
Instrument ID II1025, Instrument Type: piano, Monthly Price: 130, Instrument Brand: roland
Instrument ID II1028, Instrument Type: piano, Monthly Price: 110, Instrument Brand: roland
Instrument ID II1022, Instrument Type: piano, Monthly Price: 110, Instrument Brand: steinway musical instruments
> rent II1025 SI1011
Instrument II1025 is now being rented to student SI1011
> quit
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  04:32 min
[INFO] Finished at: 2021-01-03T19:09:47+01:00
[INFO] ------------------------------------------------------------------------
PS C:\Users\William\github\IV1351\soundgood>
```