CIS016-1 Assignment 2

Global Music Concert / Festival Booking Agency System

Group Report G15

List of Group Members:

Muhammed Ahmed, ID 1813735

Alexandra Ghetau, ID 1811800

Alin-Ionel Bogatu, ID 1712124

Andreas Progkri, ID 1813966

Michael Songer, ID 1814927

**Introduction**

In this group project we have created a software linking to a global concert/festival booking system. Within this task we have included and created various UML diagrams (use case and class). This was used as a main base in order to guide to coding aspect of the project. After this, we also created the SQL codes in order to create a database, which will then later be implemented into the software, allowing the database to constantly be updated as new accounts are created and changes are added from anyone who has an admin account.

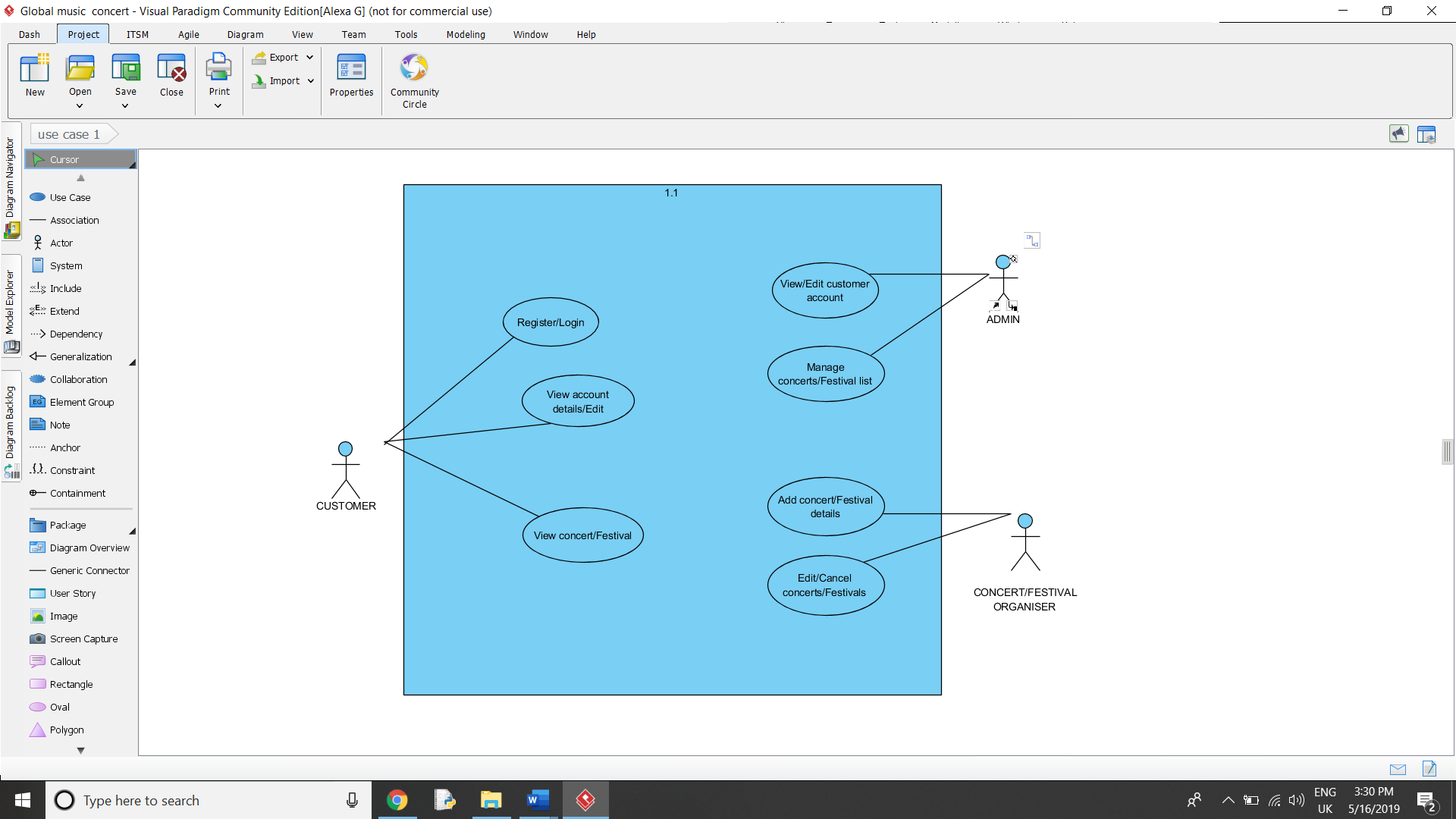
**Use case diagram**

The use case diagrams illustrated the main base of the project and created an outline for the coding. We created various actors which includes: Customer, Administrator, Corporate organization and Corporate/Festival organizer. We have created two use case diagrams, Fish and Clam level.

In the Fish diagram this portrays the beginning layout of the whole structure which starts with the customer actor making an account and be able to login in. Furthermore, they are also allowed to view their own account details and edit them if they need to. And finally, they are able to view the concerts and festivals available.

After the customer, the admin actor is then involved, and has more access in the software, being able to edit and view the customer accounts, but also allowed to manage the concert and festivals list.

The final actor in the fish diagram is called the Concert/Festival Organizer. This actor is able to add, edit and cancel the festivals and concerts.



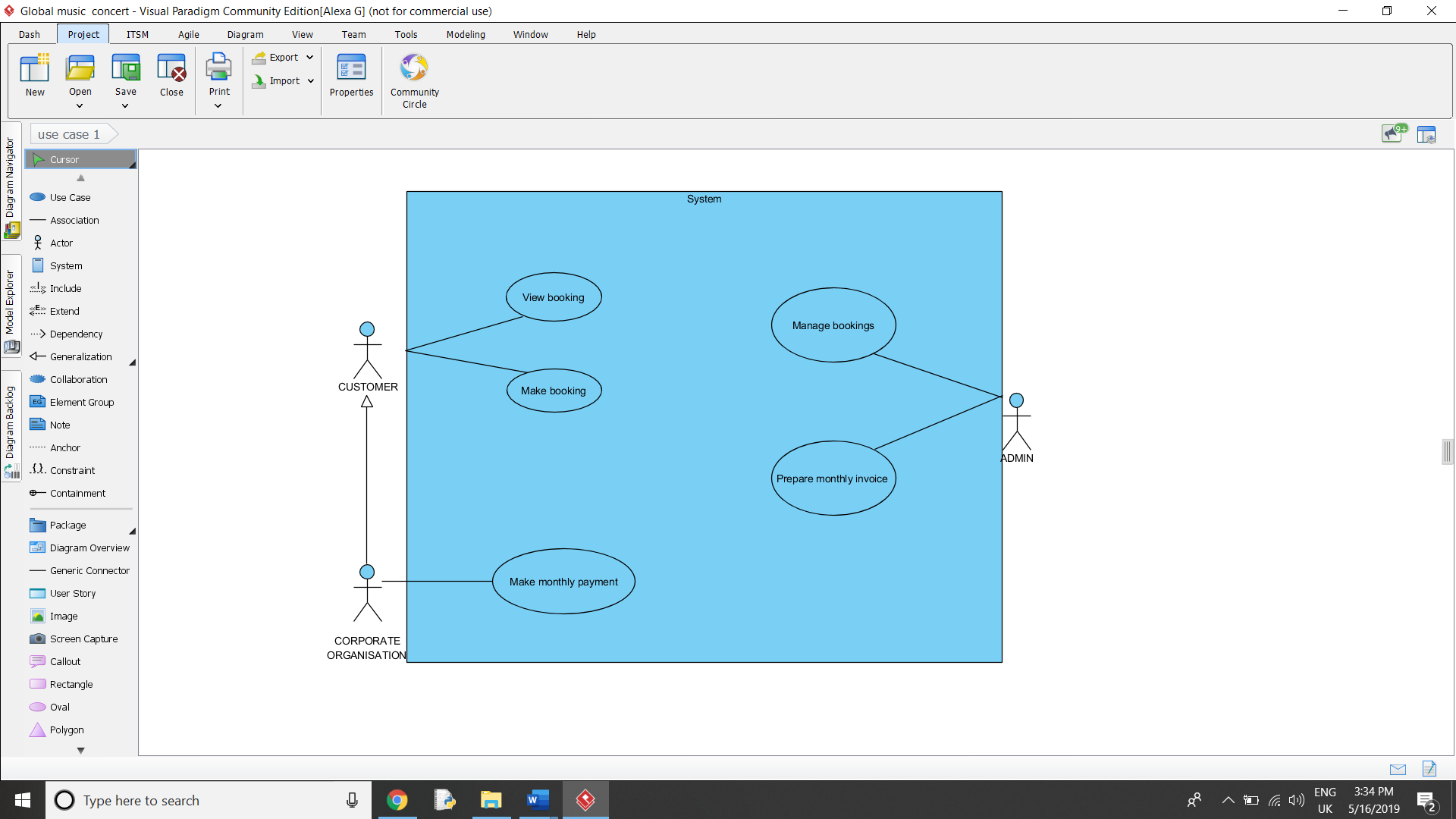
**1.1**

The second use case diagram we have created is the clam diagram. This includes, the customer, corporate organization and admin.

The customer is able to view their bookings and make bookings to whichever concert they demand subject to availability.

The admin actor is able to manage the bookings and prepare the monthly invoices for the corporate organisations.

The corporate organization actor is able to perform everything the customer can do; however, they have the ability to create and make monthly payments. In the diagram below, there is a generalization arrow linking the corporate organizer to the customer, showing that the organizer can carry out the same tasks but have the ability to do extra.

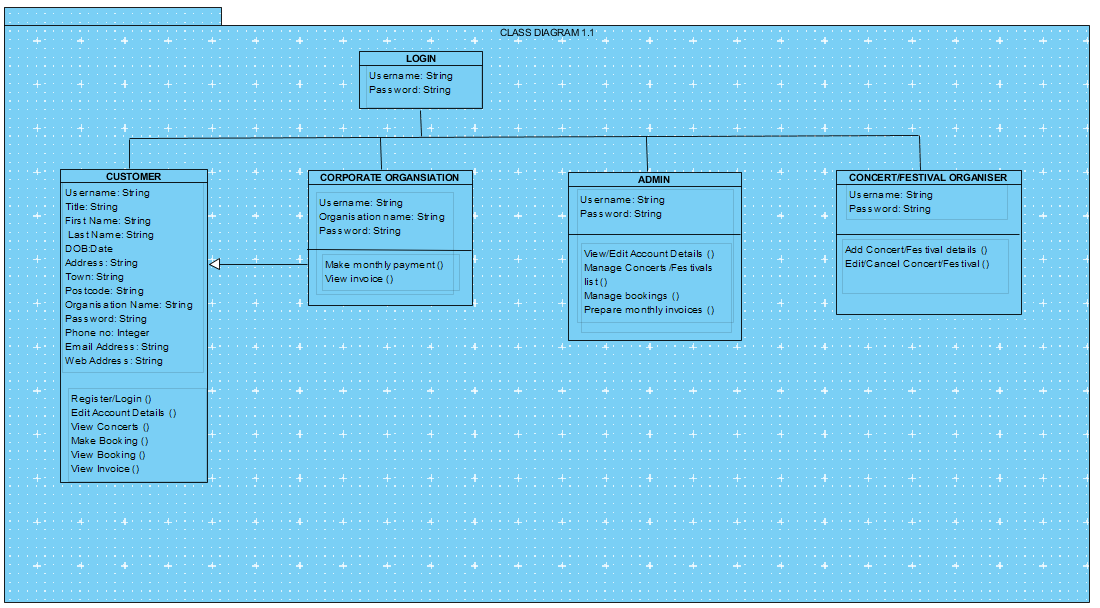


**1.2**

**UML Class diagram**

In this class diagram we have created a brief layout which is constructed by the main classes which will be included within the code. This is so we have a good understanding of what we need to include and focus on, in order to allow different levels of access towards different individuals, weather they are a customer, an organizer, or admin for example. Similarly, to the Use case diagrams we have created the generalization arrow from the corporate organizer towards the customer as they have similar aspects, however the corporate organizer can do more actions. Within this Class diagram, it presents what each attribute consists of; depending on the different variables each aspect consists of. In the diagram below (1.3) you can see that each class has many variables attached to the attributes and actions, all of them have different abilities and every class links to the main class at the top of the diagram which is labelled as the login.

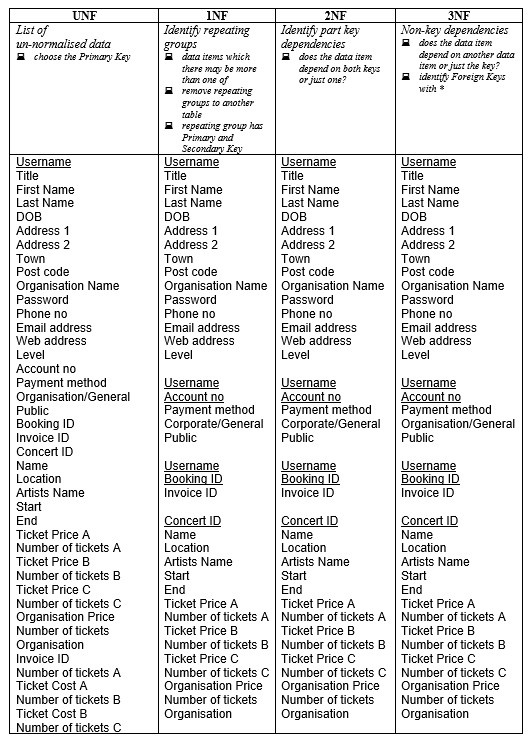
**1.3**

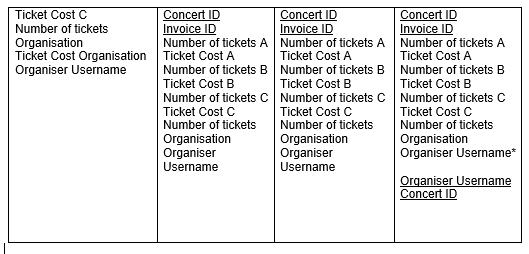


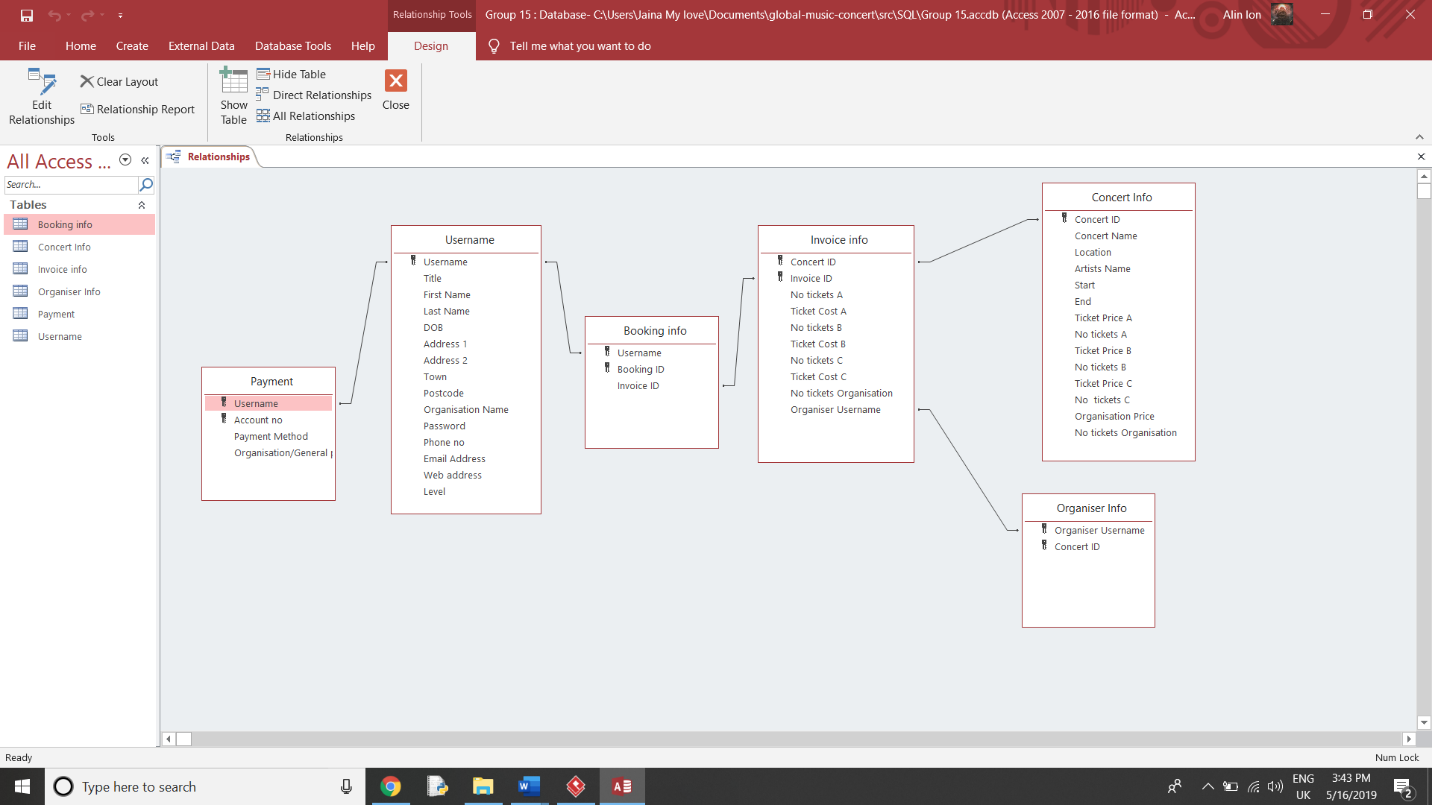
**Database Design/Entity-relationship model**

In order to create a database straight away using Notepad, we have created a Normalization form. This contains all the data needed for our system. It has four stages: first one is the UNF which is the unnormalized form, so all the information is in one table. At this step is required to define the Primary Key which is the unique identifier for the records in a table. Secondly, we have the 1NF – first normal form which allows us to identity repeating groups and the primary keys. The third stage is related to the 2NF – second normal form which eliminate part key dependencies. Finally, the 3NF – third normal form excludes inter-data dependencies. In this case the third normal form has a foreign key marked with \* as we can see below. (1.4)

**1.4**







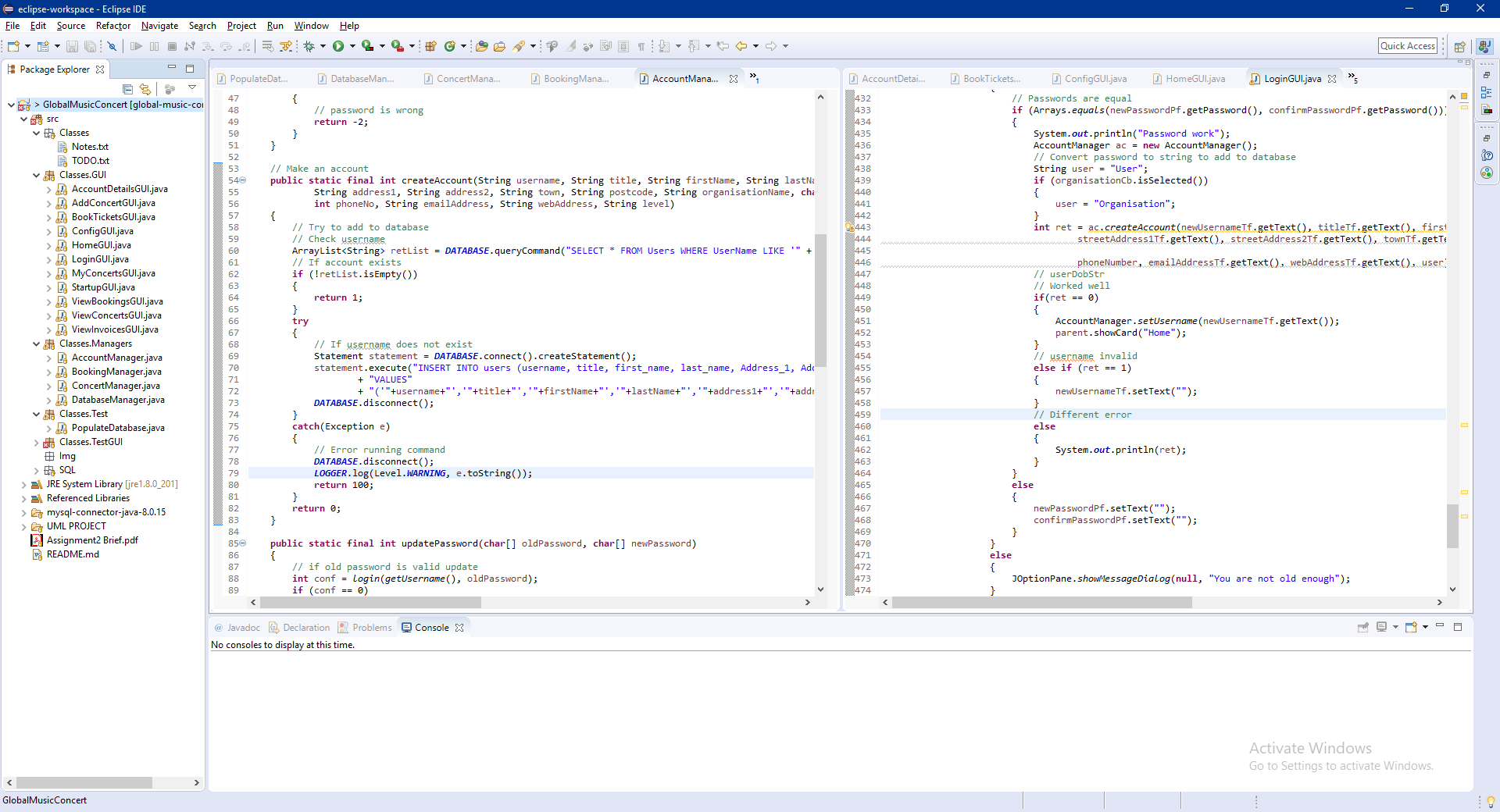
**1.5**

**Entity-relationship model** is a structure of data represented graphically with the following elements: entity sets, attributes and relationships. Based on our Normalization form, we have created the tables using 3NF in Microsoft Access. Furthermore, we built relationships between the tables as we can see above (1.5). Every table has a specific set of data. Starting with Username we are able to figure out the Payment info and Booking info. Accessing Booking info, we can open the Invoice info followed by Concert and Organizer info.

We chose to create a database in Notepad because it can be modified easily in here and we wanted to keep it neat and more organized. After that, all the data has been imported to our local host PhpMyAdmin where we kept all the information. Moreover, to create a local web server for database we have used XAMPP Platform.

**Implementation**

We decided to have Eclipse as our IDE because it is free and very reliable. Initially it can be difficult to learn but it has a lot of useful features and shortcuts that make programming more efficient and a better experience.

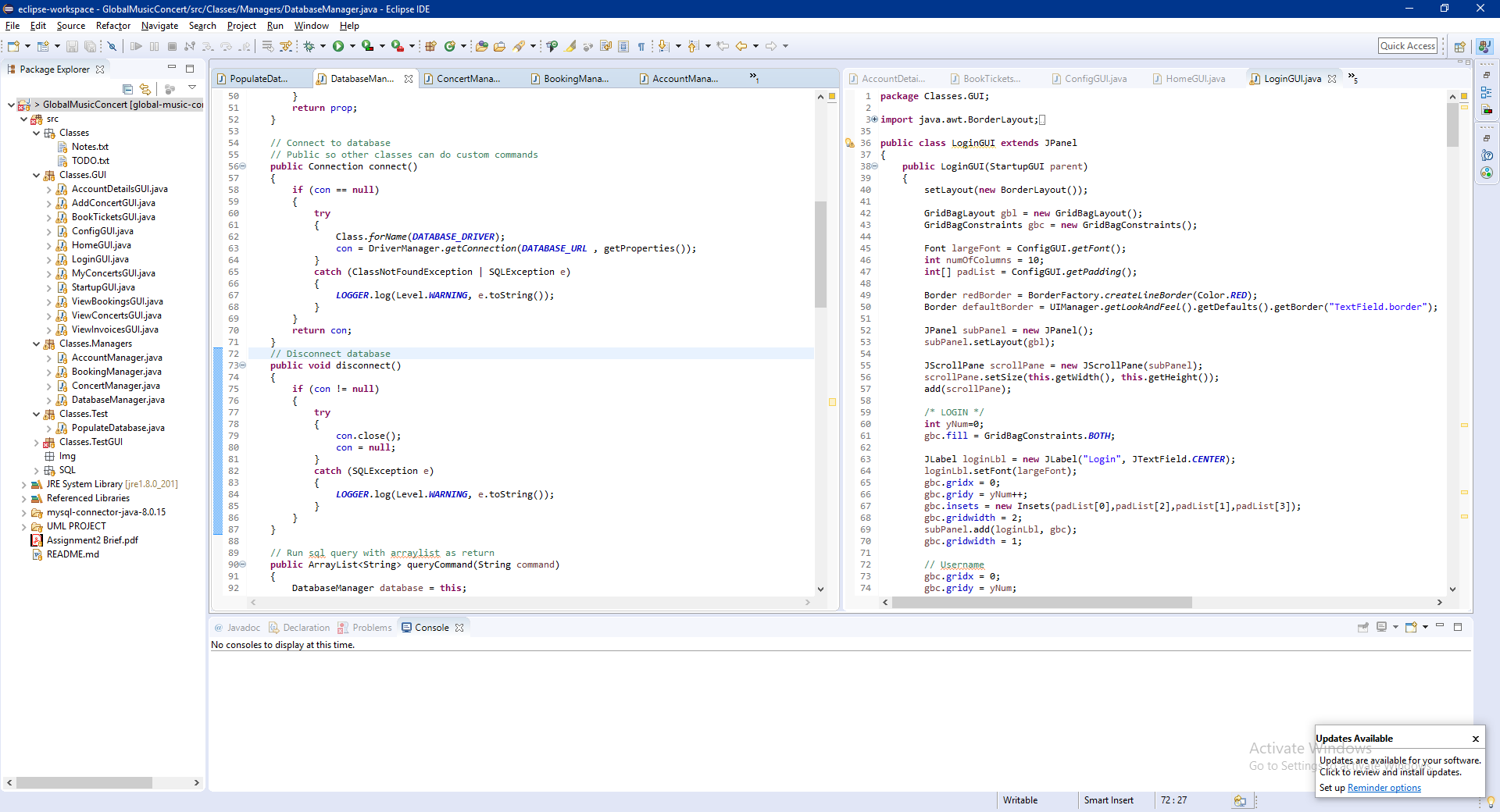


**1.6**

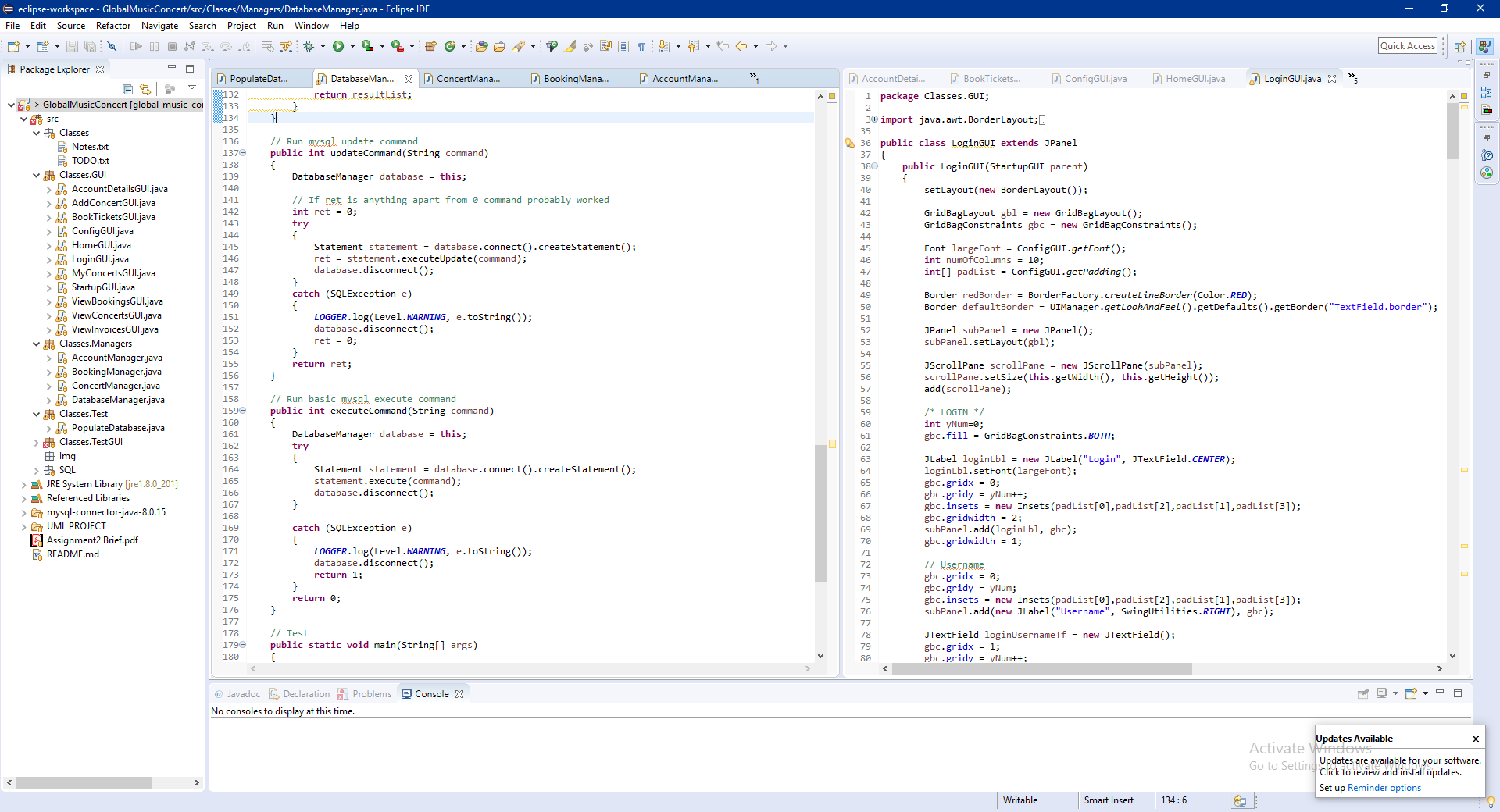
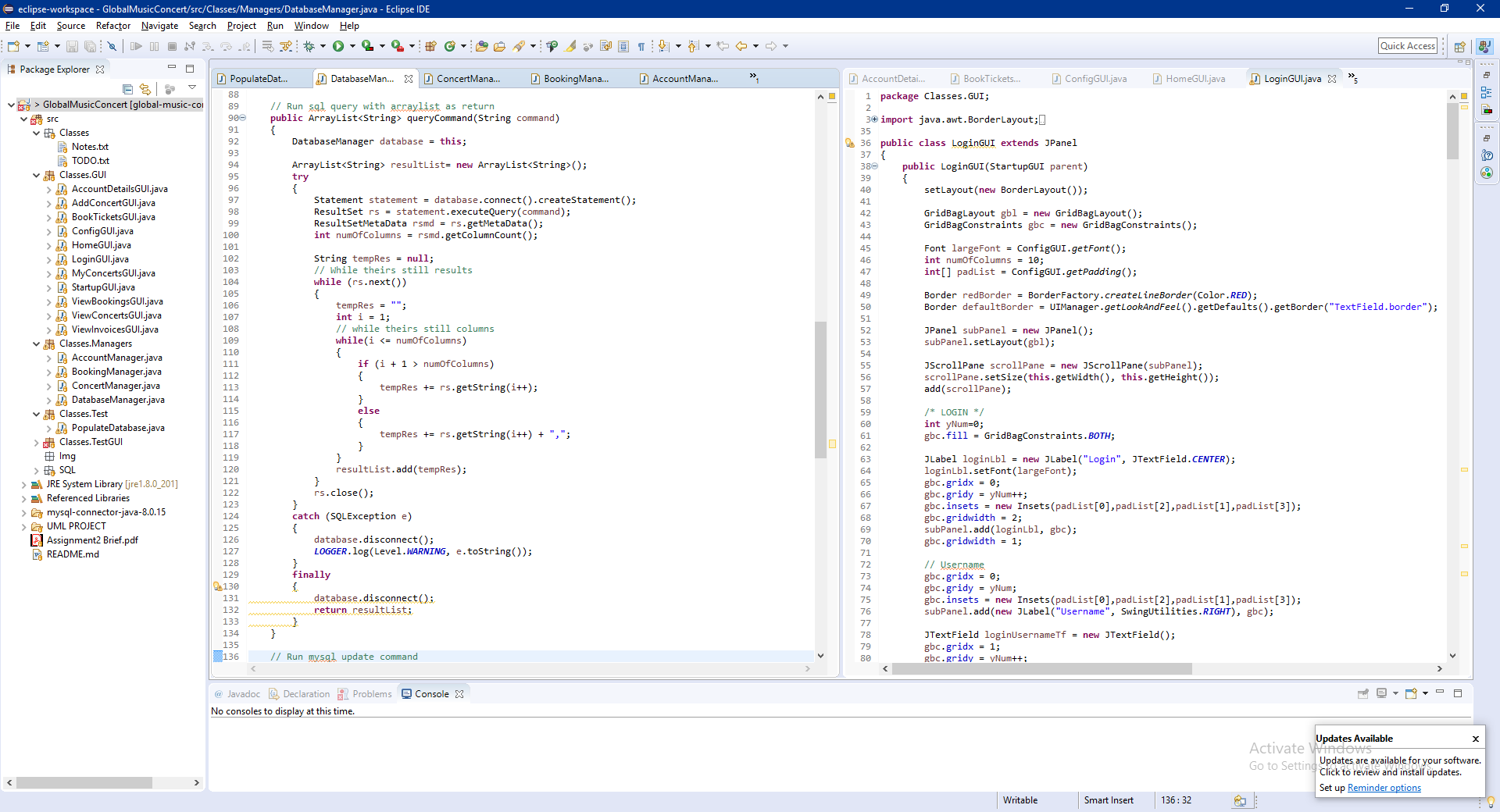
The software was split up into GUI classes and Manager classes. This was done for multiple reason, one being ease of use. Splitting the software up into smaller classes makes it a lot easier for different people to understand where everything is and makes modification a lot simpler. The class names are designed to be self-explanatory but can be a bit long making programming a small but harder.

The overall layout is the grid bag layout because it is relatively easy to control but can be very effective at making a nice organized layout. One thing that was found useful when making the GUI was storing the current y axis as an integer and incrementing it after use. Doing this made it so that if something was to be moved up or down, or something new added you would not have to redo all the numbers because the y number will change automatically.

All the database controls are in the manager classes. The database manager class has generic database methods that can be used by other managers. It also adds the capability of having the other classes connect to the database and run customized commands.

**1.7**

The image above shows the connect and disconnect methods from the database manager class. These can be used by the other managers in order to connect to the database that one may run any custom commands that are needed.

**1.8**

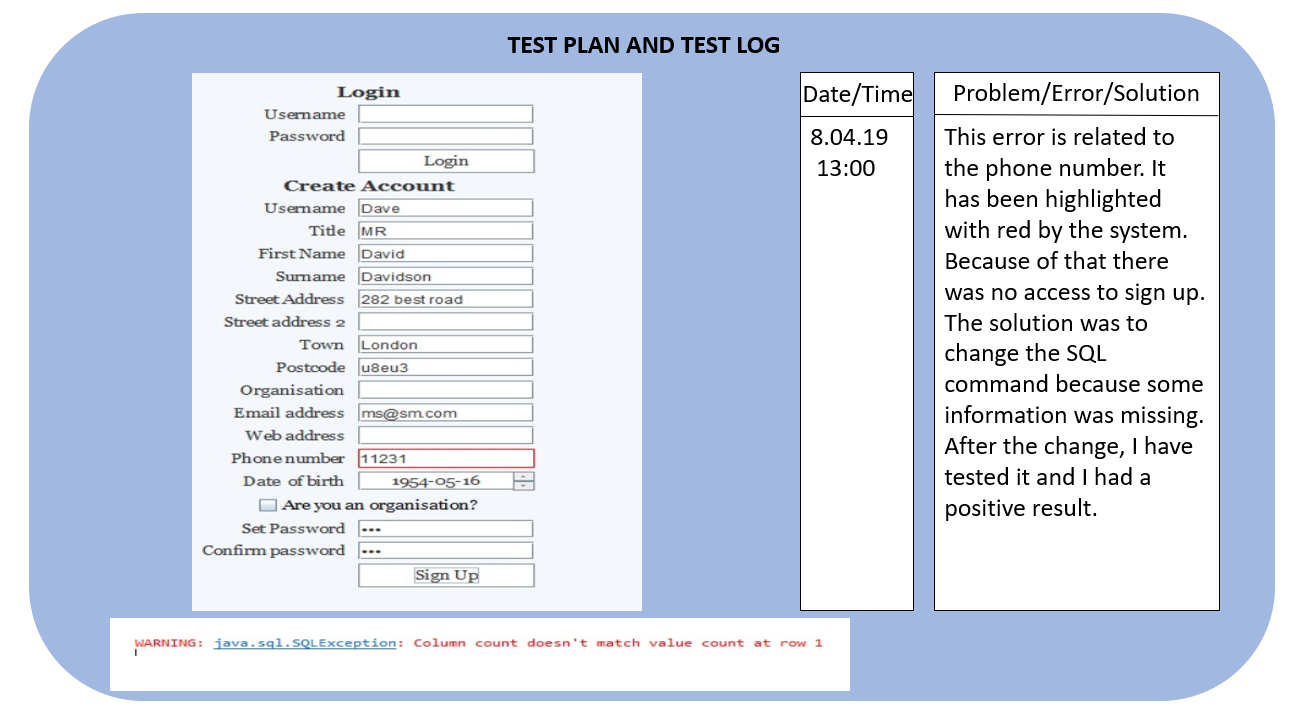
The image above (1.8) shows the different methods in the database manager that make it so that the other manager classes can run generic commands easily without having to working about dealing with possible errors.

In order to be able to get the multiple GUI pages to work together Michael found something called a card layout. The card layout basically holds multiple different JPanels together and can switch between them when necessary. Having this made the process of switching pages a lot easier and more fluid. So in order to start the software you run StartupGUI and just use the interface to go between the pages. Michael also set it up so that when going between different pages they get refreshed so they are always up to date.

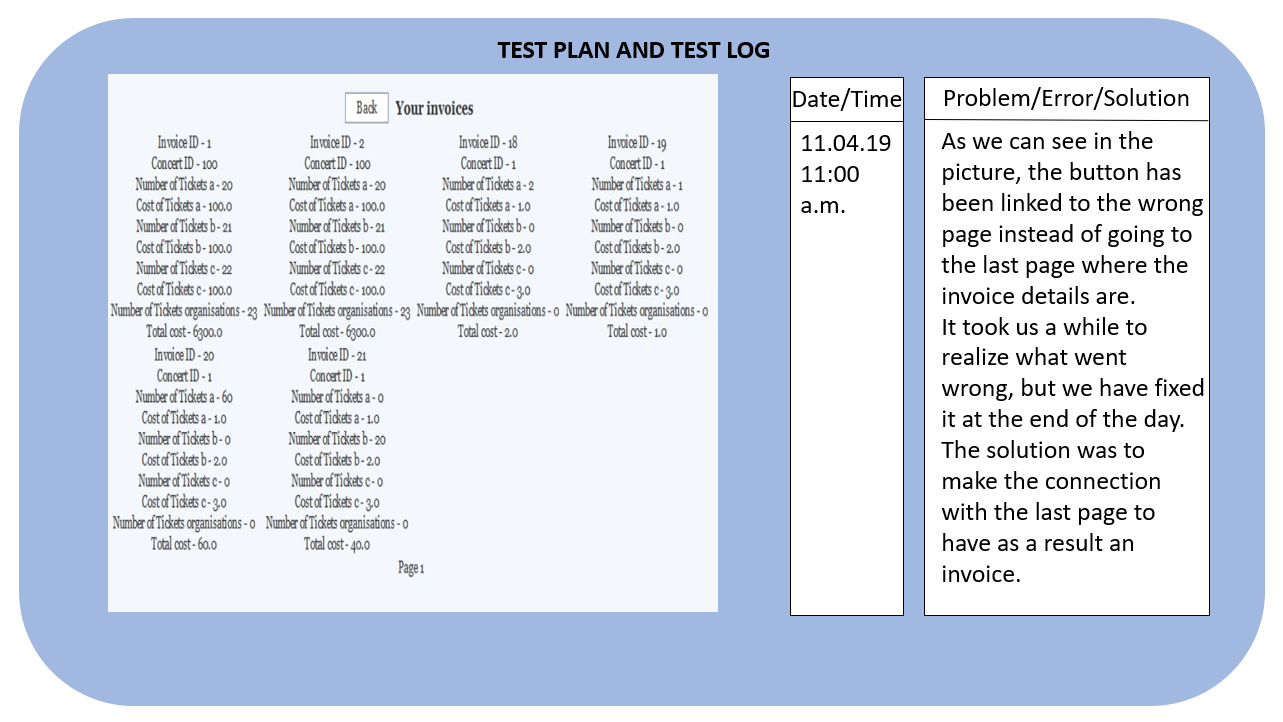
**Testing**

We have begun by checking every button on every screen to make sure it works. Next step was to do a code review after we have checked everybody’s work. As a developer you can find mistakes during the coding so that will be changed immediately if you have realized where the problem is coming from. Below there are few examples of testing with explanation (1.9, 1.10, 1.11, 1.12).

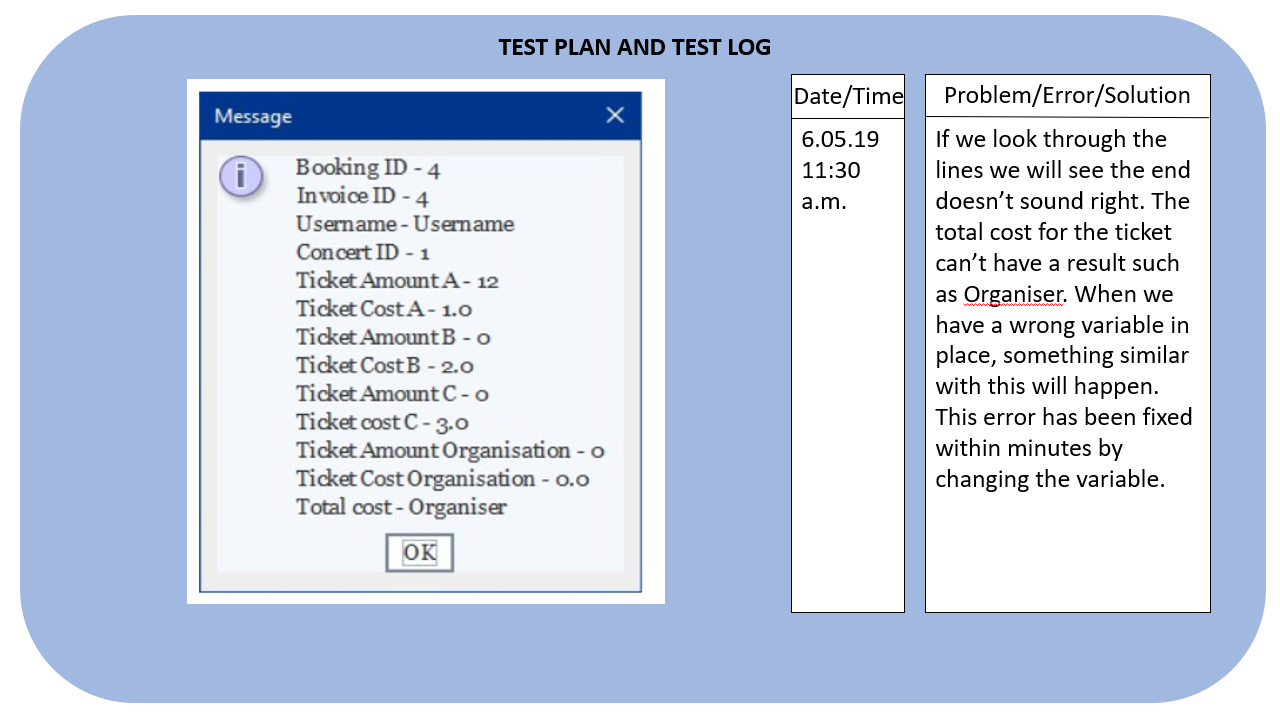
**1.9**



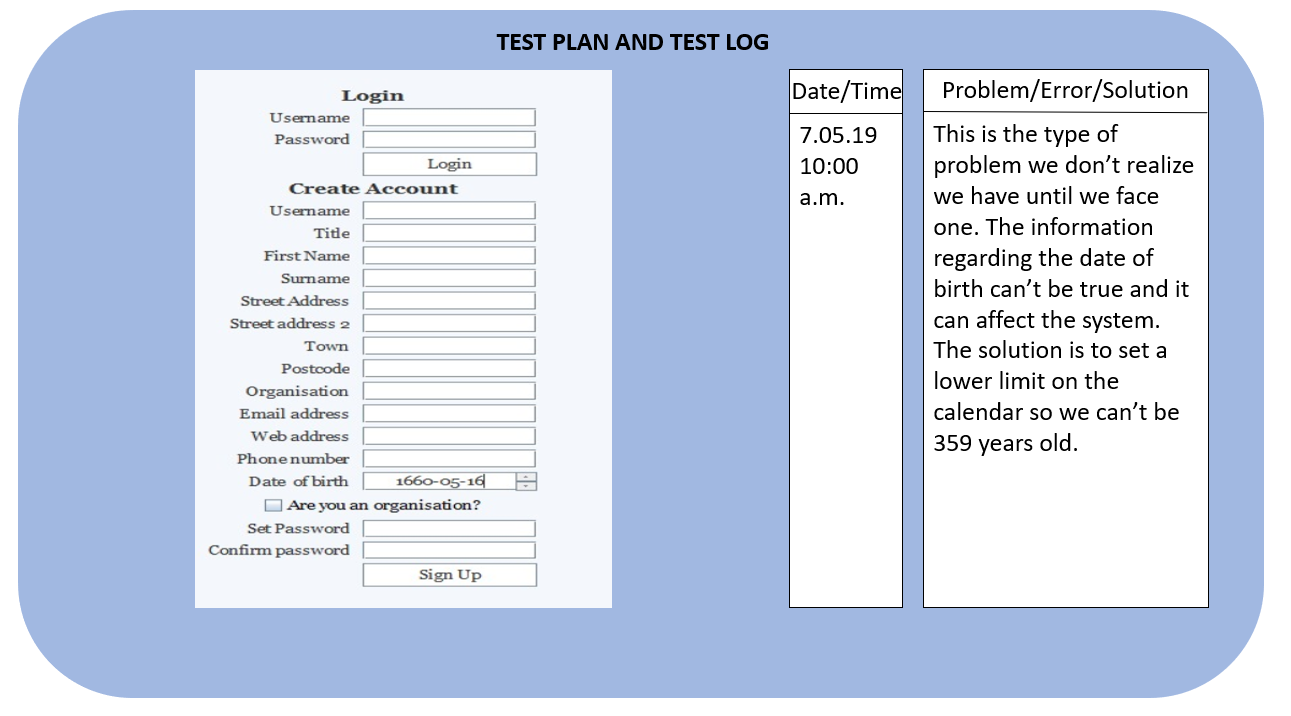
**1.10**



**1.11**



**1.12**



**Reflection and Responsibilities**

Starting from the beginning of the project we had created a plan for the UML diagrams in order to allow it to run smoothly. The planning aspect in the group went well as we were prepared and organized, assigning everyone different tasks in order to work more efficiently and co-operate in a professional manner. During the database part of the project, it took a while to manage to finish the normalization accurately but got achieved a while after. Once the normalization was completed, we then started to the SQL coding, allowing us to create our tables and different sub section which suits the software. The java coding went well as we were all doing different parts of the coding then pieces them all together when we were done. Some of the code were added and changed by different people in case it didn’t suit the software or if we were able to make the software look neater and more organized. During the entity relationship model, it didn’t work very well at the beginning as we had less knowledge of the model, however, after some research and experimentation with the entity relationship model, we finally picked up on it and were able to complete it to the best of our ability. Similarly, to the UML class diagram, we had less knowledge how to organize the diagram, but after research and looking at our past notes, in order to gain a deeper understanding of the diagram, we were able to complete the diagram to the bests of our ability.

Table listing the roles and responsibilities of all group members (See the example in Table 1.)

Table 1. Roles and Responsibilities

|  |  |
| --- | --- |
| Name and ID | Roles |
| Muhammed Ahmed, ID 1813735 | UML, SQL, main documentation,  Coding – account details, view bookings, view invoices |
| Alexandra Ghetau, ID 1811800 | UML, SQL, ERM ACCES, main documentation,  presentation  Coding - Testing |
| Alin-Ionel Bogatu, ID 1712124 | Coding – GUI’s login page, view concerts |
| Andreas Progkri, ID 1813966 | Coding – add concerts |
| Michael Songer, ID 1814927 | Main coding (managers and overall work to make sure the program works), |

**Conclusion**

In this report we have discussed the overall layout of the entire project from start to finish. This includes our organization and planning to be able to allow us to make this software work and presentable. We have included various UML diagrams as well as our SQL coding, normalization forms and an entity relationship model. Furthermore, we have discussed and presented our code, including little snippets for the reader to gain a better and deeper understanding. In addition to this, we have included the testing table which portrays how we have tested the software and kept a specific log on some of our problems and errors we have within the process of making the software. This highlighted key dates and screenshots necessary to show evidence of our test log and show the reader an insight of the process of creating the software. Finally, we have given our overall self-reflection and responsibilities within everyone in the group.

**References**

[www.visual-paradigm.com](http://www.visual-paradigm.com)

<https://stackoverflow.com/questions/2839321/connect-java-to-a-mysql-database>  
<https://coderwall.com/p/609ppa/printing-the-result-of-resultset>  
<https://thebadprogrammer.com/swing-uimanager-keys/>  
<https://programming.guide/java/create-a-custom-event.html>

Dr. Anthony Brown Lectures and Practical

**Appendix**

Diagram 1.1 – page 2

Diagram 1.2 – page 3

Diagram 1.3 – page 4

Diagram 1.4 – page 5

Diagram 1.5 – page 6

Diagram 1.6 – page 7

Diagram 1.7 – page 8

Diagram 1.8 – page 9

Diagram 1.9 – page 9

Diagram 1.10 – page 10

Diagram 1.11 – page 11

Diagram 1.12 – page 11

Table 1 – page 12