CSI 3450 Airline Project

Team members: Noah Giles, John Robertson, Brianna Kearney

I. Introduction

For this project, our goal was to create an airline database management system for the fictitious international airline company, Regal Airways. We approached this task first by constructing an entity relationship diagram (ERD) that would give us some idea as to how the database would be structured. The ERD modelled the entities, their attributes, and the relationships between different entities in a visual format. It proved useful for the next step of constructing the relational schema and the SQL table creation code.

Our project tasks diverged after the relational schema and SQL tables were created. Noah and John took on the task of programming the project's Java interface and the code to make it work. Brianna worked on SQL code to populate the database's various tables with dummy data, as well as programming parts of the SQL queries that would be added to the Java interface code by Noah and John. We were successfully able to integrate the SQL code and the Java code into a fully functional interface. One of the features of our interface was the ability to populate the database with buttons on one of the interface's tabs, minimizing the need to run a separate SQL script when testing the interface's functionality.

The completed project interface performs adequately when interacting with the database. However, due to time constraints, we did not have the opportunity to see how it performs with a significantly larger amount of data entered into the database. Future developments with the project might involve the incorporation of additional attributes to include data that was not specified in the assignment's business rules, such as airline employee salary, and customer rebate programs such as frequent flier miles that are based on distance traveled instead of a percentage of the ticket price. It would also be possible to incorporate much more complex schedules for fees and discounts. Lastly, our method of implementing multi-way and round-trip flights could be improved with additional tables that correlate customer information with all flights reserved, allowing for better tracking of the customer's entire trip.

II. Requirements

The project consists of two components. First is a MySQL database responsible for hosting the demo data. The second is a Java application which is responsible for handling user interaction. It is assumed that the MySQL database and Java front-end are running on the same host. This application should be able to run on any system that is capable of running MySQL and Oracle's Java runtime.

III. Assumptions

While many "micro-assumptions" were involved while designing the database, some of the key assumptions can be broken down as follows:

- There was no anticipated need to remove entities such as airports, planes, and employees.
- Reservations cannot be updated or edited - only created or deleted
- A multi-way or round-way trip can be represented by simply chaining two flights together, as opposed to having to account for a special mechanism.
- The Java UI does not attempt to track anything in real time. All time values are entered by a user. For example, the user must specify the date a reservation is created, as opposed to the program inferring that from the actual time at which the application is running.
- A customer who makes a reservation is not necessarily a passenger on the scheduled flight(s).
- There is no restriction on who can be a member, or when

- Employees are assigned to flights, not aircraft
- Life insurance costs a fixed amount

IV ERD

The ERD Diagram has been included as a separate image file ERD.jpg in the submitted zip file, as it is too large to fit conveniently into this report.

V Relational Schemes

The ERD Diagram was eventually resolved into the following relational schema

CSI3450 AIRLINE PROJECT - RELATIONAL SCHEMA

(*underlined indicates primary key, bold indicates foreign key*)

Customer(customerID, firstName, lastName, birthDate, member, wheelchair, oxygen)

Purchase(**reservationID**, **customerID,** paymentMethod, date)

Charge(**reservationID, customerID,** memberDiscount, childDiscount, multiWayDiscount, refund, weightFee, insuranceFee, ticketPrice)

Passenger(**customerID**, **reservationID**, carryWeight, class)

Reservation(reservationID, **flightID,** cancelled)

Flight(**flightID, aircraftID, sourceAirportID, destAirportID,** liftOffTime, landTime)

FlightDeparted(**flightID**, departTime)

FlightArrived(**flightID**, arriveTime)

ClassPrices(**class**, **flightID**, price)

FlightAssignment(**flightID, empID)**

Employee(**<u>empID</u>, prevFlightID, positionID**, firstName, lastName)

EmployeePosition(**<u>positionID</u>**, dressCode)

AirportAssignment(**airportID, empID**)

Airport(**<u>airportID</u>,** name, latitude, longitude)

Aircraft(**<u>aircraftID</u>,** mileage, routingRange, firstClassSeats, businessSeats, familySeats, premiumSeats, econSeats)

Service(**aircraftID**, servicetType)

VI Description of Code

The project code was done in Java, and managed using the Gradle build tool. This means that, provided the host computer has a recent version of Java installed, the Java application can be quickly launched using the gradle build script.

1. `$ ./gradlew build`
2. `$ ./gradlew run`

Or, `gradlew.bat` on Windows systems.

The Gradle tool should automatically download and install necessary dependencies, including the JDBC jar. The UI is coded using JavaFX, which is present in most desktop Java installations.

The Java application expects that there will be a local MySQL installation with an accessible root user with password ‘rootpassword’. The database can be setup by executing the files `sql/tables.sql` and `sql/populate.sql`. Note that the file sql/application_queries is purely for observational purposes, and is not actually used directly in any code. Additionally note that the created database *must* be name “flights”.

Finally, in general the Java files can be broadly broken down into three categories, as noted below:

1. Dialog Classes - these classes typically extended the JavaFX Stage class, and represent window that host controls

2. Control classes - these classes typically extend some form of JavaFX element, and are responsible for interacting with the database in response to user input

3. Object Mapping Classes - these are classes that parallel tables in the database and represent a single record.

VII Conclusion

The development of the project has utilized large swaths of the material covered in lectures and homework. The top-down approach, running from ERD diagram to direct a direct Java-and-SQL implementation served to guide the development of the application in an orderly manner. While the application accomplishes the majority of the tasks laid out in the specification, it is certainly possible to expand and improve upon what has been developed thus far. We believe the process has been able to provide valuable insight about the importance of measured and carefully considered design practices.

VIII Future Work

We judge that there are a number of areas that would especially benefit from further development, should this have been an actual business enterprise:

1. Network Distribution - currently the assumption is that the database resides on the localhost, but this is clearly not a universal rule. Adopting the program to target a database over a network would provide additional flexibility
2. UI “bulletproofing” - the UI is generally functional, and *should* not crash due to “garbage” input, but this was certainly not the focal point of the development. Therefore, it is possible that certain input fields could provide vectors for “messing up” the application or the database through manipulative inputs.
3. General feature expansion - while we consider business features developed in the project as being generally sufficient for exploring database uses, the project would certainly stand to benefit from a more nuanced approach to how a business might handle both the presentation of a user interface and the modelling of a broad variety of datasets.
4. Optimization - in general, the application doesn’t pay much heed to the efficient usage of resources such as database connections. While this is not a problem at the simple scope the project was intended to cover, it certainly would need to be adjusted were it to be scaled up to a production model.

IX References

References:

1. https://docs.oracle.com/javase/8/docs/
2. https://docs.oracle.com/javase/8/javafx/api/toc.htm
3. https://gradle.org/docs/
4. https://dev.mysql.com/doc/

X Appendix

Code to populate the database can be found in the project zip under the `sql` folder.

Contents of README.txt

CSI3450 AIRLINE PROJECT

Copyright (2017) Noah Giles, John Robertson, Brianna Kearney

Number of files:

java: 29

sql: 3

Q&A Testing - System Configuration

--Noah--

OS: Windows 10 Homee

Java: Java SE Development Kit 8 Update 151

MySQL: 5.7 MySQL Workbench 6.3 CE

--John--

OS: Mac OSX High Sierra

Java: Java(TM) SE Runtime Environment (build 1.8.0_131-b11)

MySQL: mysql-5.7.20-macos10.12-x86_64

--Brianna--

OS: Windows 10 Home

Java: Java SE Development Kit 8 Update 151

MySQL: 5.7 MySQL Workbench 6.3 CE

How To Run Code

Windows:

-Navigate file explorer to the project directory. Open a terminal in the directory.

-Compile the program by entering the following on the command line and pressing Enter:

./gradlew build (NOTE: invoke./gradlew.bat on Windows)

-Once the program has compiled, run the program by entering the following on the command line and pressing Enter: ./gradlew run

-Before performing any commands using the UI, populate the database using the buttons on the Database tab.