Kamran Djourshari, Animesh Nandan, Iskander Lou

Team 2: "Spotify, Billboard, NBA"

INST733-IM01

Project Final Report: 9 May 2023

Vacation in Europe SQL Database: Final Report

Introduction

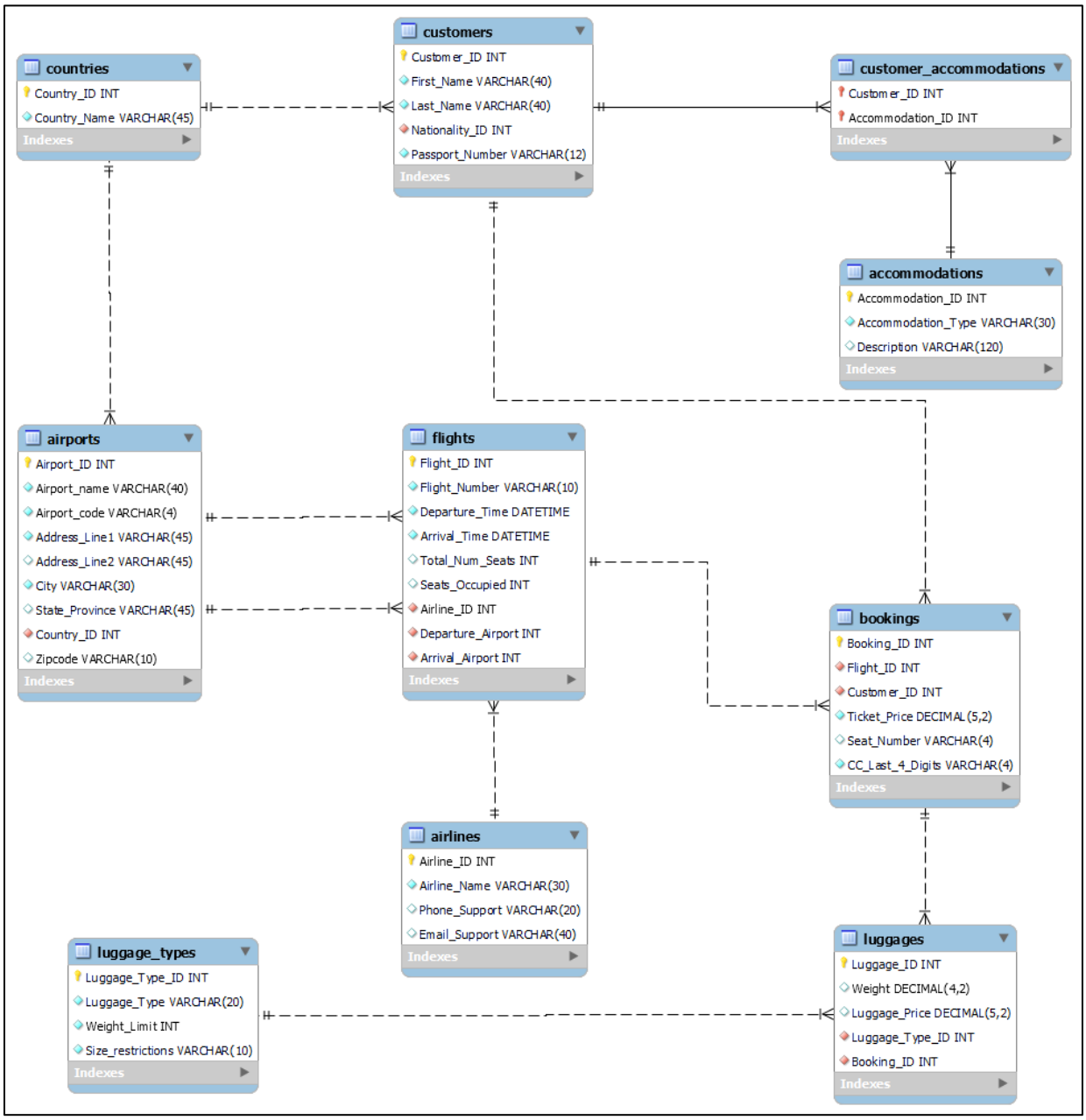
The Flight Scheduling Database System aims to develop a simplified version of the databases used in commercial aviation for booking agents and airport staff handling flight reservations within Europe. The project focused on designing an efficient and user-friendly system to provide reliable information on flight schedules, availability, and pricing for a selected group of airlines and routes.

In this final report, we will discuss the objectives and deliverables accomplished throughout the project. We will also present the methodology, including the design of the Entity-Relationship Diagram (ERD), normalization, data generation, and the creation of queries to answer the specified questions outlined in the proposal. Additionally, we will highlight the assumptions made and the challenges encountered during the project.

Furthermore, we will provide an overview of the system's scalability and performance testing, as well as the lessons learned and future recommendations for improving the flight scheduling database system. By examining the project's achievements and areas for further development, this report will serve as a comprehensive summary and analysis of the Flight Scheduling Database System project, showcasing the team's efforts in creating a valuable resource for European booking agents and airport staff.

Database Design and Implementation

Logical Design



From our posted logical design, we can see we have 10 tables total describing our vacation air travel in Europe. Our database has various elements including the different European airports, airlines, flights, luggages, luggage_types, customers, and accommodations. Each of these tables is connected via one-to-many relationships allowing for us to define many instances. For example, with many countries for different airports and customers or many luggage types for different luggages in bookings.

There are two different linking tables, which are the customer_accomadations table (linking the customers and accommodations table together) and the bookings table (linking the flights table with the customers table). With this database on European air travel, here are a quick summary of each of these tables:

- **Countries:** A table summarizing all the European countries that will be used for travel.
- **Customers:** Element summarizing all the air-travel customers (living in Europe) who will be traveling throughout Europe.
- **Accommodations:** Listing all the possible accommodations the Airports must account for.
- **Customer_Accomodations (linking table):** Holding attributes for the customer and accommodation they have.
    - As many customers could have multiple accommodations, this table is needed.
- **Airports:** The defining table of all the European airports that will be used to depart and travel to for customers.
- **Flights:** Table defining all the possible flights with the arrival time, a departing and arrival airport, and the number of seats available.
- **Airlines:** Defining table of all the airlines that will be used with their contact information listed.
- **Luggage_Types:** Table listing all the possible luggage types with the maximum amount of weight allowed.
- **Luggages:** The luggage table describes the luggage a part of the booking with the luggage type, price, and weight. Although it has foreign keys for booking and luggage_type tables, it is not a linking table because we decided to give it its own primary key, luggage_id.
- **Bookings:** Linkage table between flights and customers highlighting every booking made for any given flight by the customer. It describes all aspects of the booking including ticket price, seat number, and the credit card's last 4 digits used for the purchase.

- As it is possible for many customers to have made multiple bookings, we made it the linkage table.

As listed above, these tables are our final normalized version of the flights in the Europe database. We included as much information as possible that we found relevant for any travel booked with the customers, booking, flight, and airports.

Design Choices

We made several design choices to ensure our database is robust, usable, and easy to implement. When determining the data types, we carefully considered what real-world values would look like in our database. For instance, we used a varchar character limit of 10 for zip codes and a decimal format for prices. We also specified that each customer could have only one nationality and multiple pieces of luggage. Additionally, all flight times are represented in the GMT-0 time zone, and only direct flights are included in the database. The prices for tickets and luggage are listed in US Dollars, although in retrospect, it would have been better to use Euros instead. All flights in the database take place on May 9th, 2023. Weight limits are listed in kilograms, while size restrictions are in centimeters. Finally, we accounted for the fact that some countries do not have states, like the US, and may display provinces instead.

Physical Design

For the physical design of our database, we included a SQL creation script that can be run to present all the aspects of our database including the structure with the 10 tables (and the constraints and indexes as well), the physical data (which were imported in the tables using the import wizard), and the views. We made sure to follow the specific order of import with the tables with foreign keys not being imported until the table with the referenced foreign key was imported first.

Specifically for the physical design script, we gained this file by doing a data export on MySQL Workbench and selecting the option for Dump Structure and Data for our given database. This created the creation script with these aspects, which we also tested between our group to make sure it exported properly.

Sample Data

For the sample data aspect, we gained our sample using the mock data generator website titled Mockaroo (Mockaroo). This site allowed us to generate realistic data for each of these tables with the specific restraints we needed (i.e., NOT NULL, specific datatype, etc.). Mockaroo allowed us to formulate our own formulas as well (with REGEX) which made it possible for us to generate different attributes data that were rather unique/constrained. This made it possible for us to generate rows into a CSV file to ensure that not only did we have enough data, but it was realistic and made sense.

Each of these tables hit the minimum requirement of 20 rows except for the ones we deemed to have less in nature. Specifically, the Accommodations table only had 6 rows as we only deemed these 6 to be relevant for airports and flights to consider for customers. As for the Airlines table, we only wanted to use the most popular European airlines since we wanted to cover only the major flights that happen in Europe for a single day, hence the reason for only using 15 of the biggest (and busiest) airlines in Europe. Even though there are 20 airlines listed, we prioritized using 15 of these airlines that we viewed to be the busiest to make it as realistic as possible. Also, the Luggages_Type table only has 3 rows as we know there are only 3 options available for luggage types (personal item, carry-on bag, and checked-in bag) in major airlines, from our research and personal experiences from traveling. These tables in general didn't allow us to hit the minimum requirement to ensure that we keep our database as realistic as possible in hopes of applying it in any work situation where an air-travel database would be required.

However, the other tables all hit the minimum requirement of 20 rows for non-linking tables and 50 records for linking tables. For example, in the Countries table, we generated 33 different European countries that we deemed to be the most popular for travel. Also, for the linking table of Bookings, we generated 100 different bookings, where customers booked flights, with some having multiple bookings set in place. The other linking table, Customer_Accomdations, also has 55 different customer accommodations with some customers needing accommodations (and some needing multiple accommodations) when traveling.

View/Queries

View Name	JOINS	Filtering	Aggregation	Subquery	Linking Table
Popular_Destinations	✅		✅		✅
Flight_Ticket_Price_London	✅	✅	✅	✅	✅
Passenger_Moving_Assistance	✅	✅			
German_Passenger_Glasgow_Airport	✅	✅	✅		
Passenegers_Zurich_to_Rome	✅	✅	✅	✅	
Arriving_Country_AVG_Price	✅	✅	✅	✅	✅
Customer_Aggregate_Flight_Info	✅	✅	✅		✅

We have written seven queries that can satisfy potential inquiries from our database users:

- What are the most popular destinations in Europe? Return the number of flights arriving in each country. (Popular_Destinations)
- What is the price range of flying from London to other European countries? Return the airport, airline, flight info, destination, and ticket price. (Flight_Ticket_Price_London)
- Which passengers require moving assistance? Return a list of passenger names. (Passanger_Moving_Assistance)
- Are there any German citizens that are flying from Glasgow? Return passenger names, airline, airport, flight info, and luggage info. (German_Passenger_Glasgow_Airport)
- Who are the passengers flying from Zurich to Rome? Return the same information as above and include seat info, accommodation info, and luggage price. (Pssenegers_Zurich_to_Rome)
- What is the average price to fly to a particular country? Return the average ticket price for each destination. (Arriving_Country_AVG_Price)

- How many flights have passengers taken and how much money have they spent on flights? Return passenger names, the count of flights, and the total price paid for all flights. (Customer_Aggregate_Flight_Info)

Changes from Original Design

From our original design we have made some changes that we prioritized to enhance our database. Firstly, in the original design we had listed the concept of a time zone table to list the variety of time zones that each airport went by (with it being a foreign key in the airports table), however, in the end, we decided against this as it would complicate our database further. We decided to use just the GMT time zone for all flights as mentioned earlier.

Also in the flights table, we removed the flight_duration column/attribute as we deemed it to be redundant with the columns of depature_time and arrival_time already existing which we could use to calculate in queries if needed.

In the booking table, we had previously included the luggage id as a part of the booking as a foreign key, however, we moved away from this as we decided that it could be its own separate entity. In the end we decided to have a luggage table where we could list our every unique luggage (with a reference to the luggage type table to identify what type of luggage it is) for any booking.

Lastly, we had previously placed the accommodation id as a foreign key in the customers table, but also decided that it was most likely not the best method. We thought that some customers may have multiple accommodations (and some with none at all) so we decided to make a linking table of customer_accomdations to make it possible to fit these listed conditions. These changes in the end made it possible for us to improve our database to better fit our narrative based on feedback given to us by both the Professors and also our TA.

Issues Experienced During Development

The process of generating the flight data went smoothly, nevertheless, we faced a couple of challenges that required our attention. One issue we encountered was during the import of one of the tables, Airports, where we got an error due to an unexpected character present in the dataset.

The reason for this was that the Workbench uses the ASCII table, which only accepts English characters. To overcome this problem, we substituted the unexpected characters with their corresponding English equivalents. For example, we replaced the French letter "Â" with "A".

Another issue that we encountered was the repetition of values taking place in several columns, including in primary keys, which went unnoticed during the initial data generation process. We had to revisit the data and modify some of the values to avoid errors during import.

Upon successfully importing the dataset into our database, we, unfortunately, had to face yet another obstacle. Our database name was not properly formatted, which prevented us from creating a data dump. We were unable to change the name of the existing database in the Workbench, which meant we had to go back to our ERD, rename our database, redo the forward engineering, and import all of the data again. This setback caused a considerable delay, but we were able to overcome it in time for the due date.

Lessons Learned

Collaboration and Communication

Effective collaboration and open communication among team members proved to be essential in this project. Regular meetings, sharing ideas, and keeping everyone informed about progress allowed us to identify potential issues early on and resolve them in a timely manner. This experience reinforced the importance of fostering a collaborative environment for successful project execution.

Planning and setting realistic expectations

At the beginning of the project, we carefully defined the scope, objectives, and timeline to set realistic expectations. This process helped us prioritize tasks and allocate resources efficiently. However, we also learned to be adaptable and prepared to adjust our plans when unexpected challenges arose.

Data management and normalization

Ensuring data integrity and minimizing redundancy were critical aspects of this project. We learned the importance of normalization and the need to create a well-structured and efficient database. This process required a thorough understanding of the data and relationships between tables, as well as making informed assumptions.

Overcoming challenges and problem-solving

Throughout the project, we encountered various challenges, such as designing an effective ERD and generating realistic sample data. These experiences taught us the value of perseverance and creative problem-solving. We learned to think critically, explore alternative solutions, and seek feedback from our peers to overcome obstacles.

Scalability and performance

One of the key lessons learned was the importance of considering scalability and performance during the design and development phases. We had to ensure that our database could handle large volumes of data and concurrent users without compromising its performance. This taught us the value of thorough testing and evaluation to identify and address potential bottlenecks.

Learning and improvement

The project highlighted the importance of staying up to date with the latest technologies, tools, and best practices in the industry. We recognized that the rapid evolution of technology requires continuous learning to maintain and improve our skills. This lesson will motivate us to pursue ongoing professional development to stay competitive in the field.

In conclusion, the lessons learned from the Flight Scheduling Database System project have significantly enriched our knowledge and skills in database design, teamwork, problem-solving, and adaptability. These insights will be invaluable in guiding our future endeavors and helping us become more effective professionals.

Potential Future Work

While the Flight Scheduling Database System has been successful in providing a streamlined platform for European booking agents and airport staff, there is room for improvement and expansion. Several potential extensions could be integrated into the current technology and design to enhance its functionality and cater to a broader range of user needs.

Extensions within the current technology and design

Firstly, incorporating real-time flight status updates, such as delays and cancellations, would provide users with more accurate and up-to-date information. This could be achieved by integrating the database with an external flight data provider.

Another extension could be the inclusion of data on employees and their work schedules. This would allow airport staff to manage human resources more effectively, ensuring that the right personnel are available at the right times.

Additionally, expanding the database to include information on aircraft manufacturers, total emissions, hotel bookings, and car rentals would offer a more comprehensive range of services to users. This expansion would also increase the database's appeal to travel agencies and customers looking for a one-stop solution for their travel needs.

Feasibility of alternative implementations

Exploring alternative implementations for the Flight Scheduling Database System can provide new perspectives on how to optimize and further improve the system. These alternative implementations may offer better performance, scalability, or ease of maintenance.

One possible alternative is utilizing a NoSQL database instead of the current SQL-based database. NoSQL databases are known for their flexibility and scalability, which may prove beneficial in handling vast amounts of data and adapting to changing user requirements.

However, the switch to a NoSQL database would require redesigning the database schema and may pose a challenge in maintaining data consistency and integrity.

Another potential alternative is adopting cloud-based database solutions, such as Amazon Web Services (AWS) or Google Cloud Platform (GCP), which can offer cost-effective and scalable infrastructure. These services can handle large volumes of data, improve accessibility, and reduce the need for maintaining physical servers. However, data security and privacy concerns, as well as the potential for vendor lock-in, must be taken into consideration.

Overall, the consideration of extensions and alternative implementations for the Flight Scheduling Database System will ensure its continued evolution, meeting the growing needs of users and maintaining its relevance in the rapidly changing aviation industry.

References

Barnhart, C. (2008). Airline Scheduling: Accomplishments, Opportunities and Challenges. *Integration of AI and or Techniques in Constraint Programming for Combinatorial Optimization Problems*, *5015*, 1–1. https://doi.org/10.1007/978-3-540-68155-7_1

Mockaroo. (n.d.). *Mockaroo - Random Data Generator and API Mocking Tool*. www.mockaroo.com. https://www.mockaroo.com/