Team 12 - Target 1

Branch, Rasean - dbs009
Wang, Zi Long (Andy) - dbs120
Luu, Nghia Nghiep - dbs065
Ngo, Chi - dbs082

Link to Demo

https://www.youtube.com/watch?v=7KrHgKtXmCc

Airline Summary

The following Conceptual ER Diagram, and Physical ER Diagram is formulated based on the needs of target 1's topic. We have decided to use these relations based on our research of how most airline databases function. Our webapp is mainly designed to handle flight bookings, and the transactions that follow. The ER diagrams are further explained below. As for normalizing the database, it is done up to bcnf, because we understand the benefits of a normalized database. Furthermore, not only does a normalized database allow the user to understand their queries better, it also minimizes redundant data, and ensures only related data is stored into their respective tables.

2 ER Diagrams

Conceptual ER Diagram

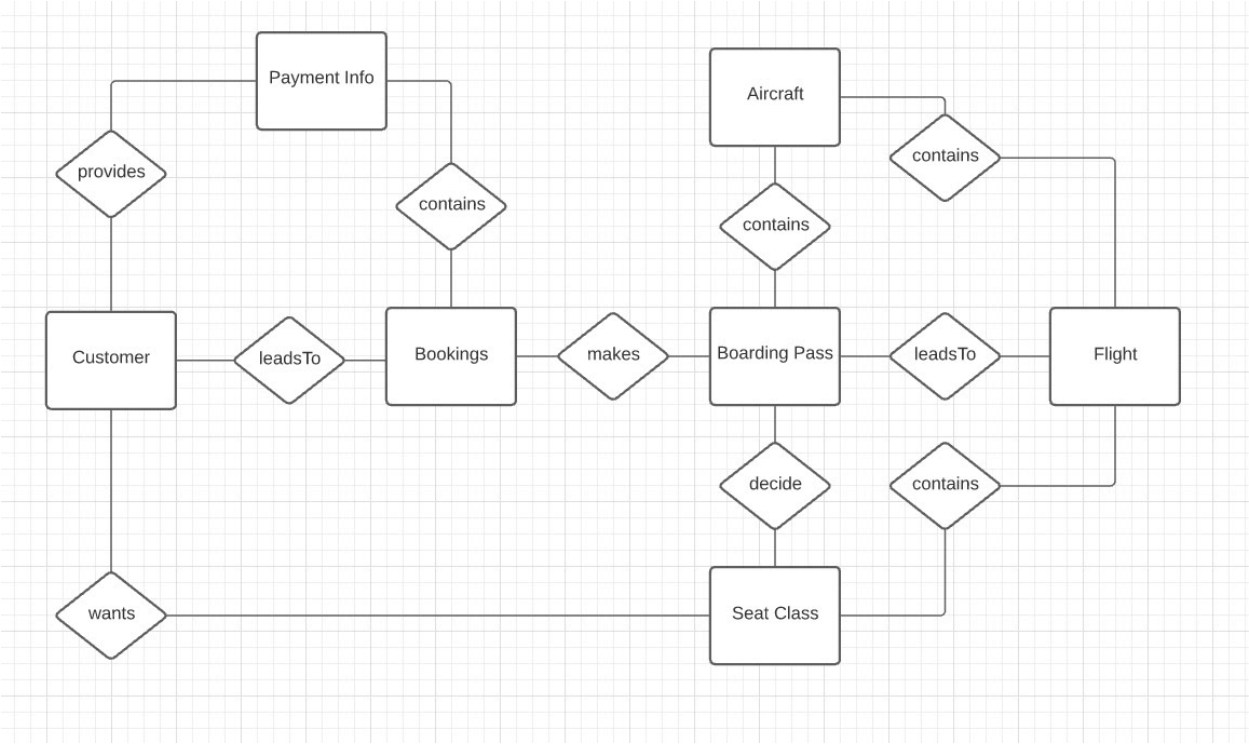

Figure 1. Conceptual ER Diagram

Team 12 - Target 1

Branch, Rasean - dbs009
Wang, Zi Long (Andy) - dbs120
Luu, Nghia Nghiep - dbs065
Ngo, Chi - dbs082

Our team started hw3-airline, with the Conceptual ER Diagram shown above. This conceptual diagram allowed us to design and implement the most basic functionalities to our airline-web app by organizing the entities and their respective relationships. To further elaborate this Conceptual ER Diagram, a customer first makes a booking for a flight. At this stage, the customer provides their payment info. Then, the bookings entity makes the boarding pass which contains the specific aircraft, flight and seat. This serves as the basis of our airline design.

Physical ER Diagram

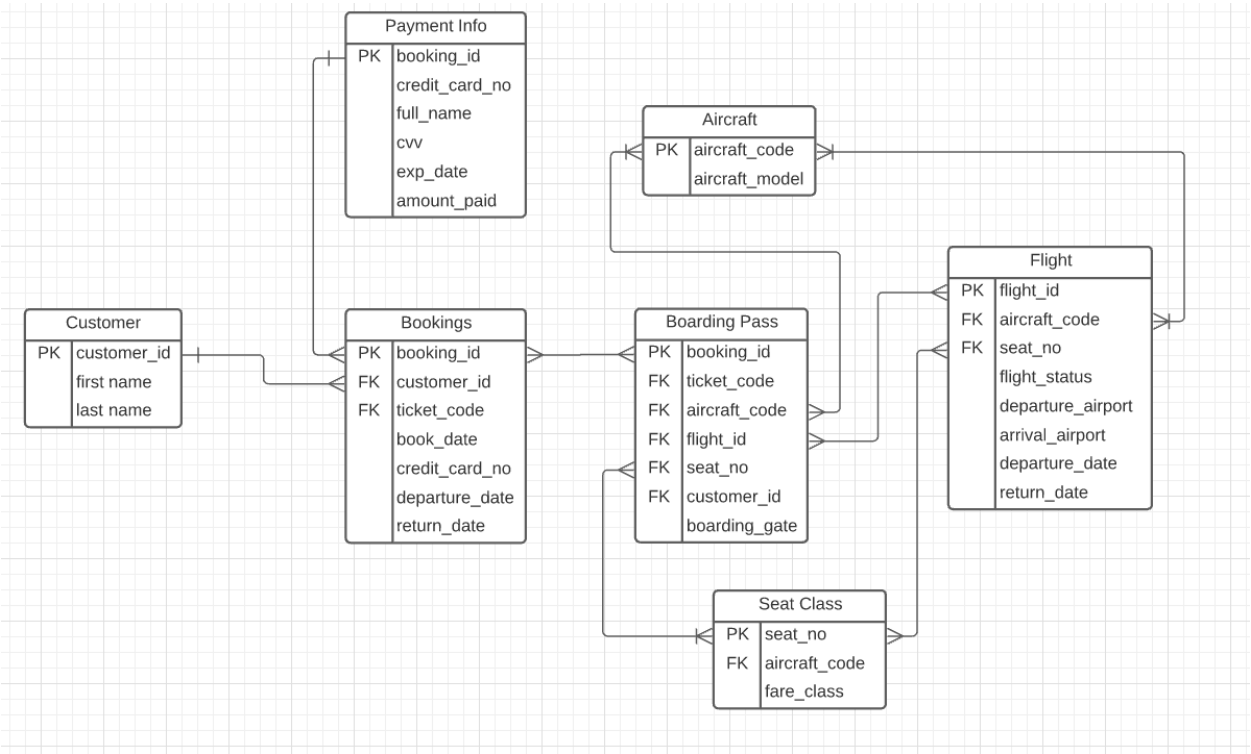

Figure 2. Physical ER Diagram

Team 12 - Target 1

Branch, Rasean - dbs009
Wang, Zi Long (Andy) - dbs120
Luu, Nghia Nghiep - dbs065
Ngo, Chi - dbs082

After the creation of the Conceptual ER Diagram, a Physical ER Diagram allows us to further expand on the entities. In our database design, we have chosen to have a customer_id to identify the customers that are booking a flight, and can be related to the customer_id in the bookings entity. The bookings entity shares the payment info, and the boarding pass entity via a booking_id. Furthermore, the boarding pass contains the boarding gate, and is related to the specific aircraft, seat, and the flight entity which has all the flight details. All of this allows the specific customer to book their flights, and see their flight details. Lastly, each relation is connected by either one-to-many, or many-to-many.

Database transactions

The source codes are divided into 3 parts: .html files handle the Graphical User Interface (or web client), main.js file is web server, and index.js connects to the

database. Whenever there is a transaction request entered from the web client, it calls an intermediate function in the web server, and then the server connects to the database to do SQL queries. Transaction requests include booking (post), retrieving (get), and canceling (delete) that a user or a customer can leverage.

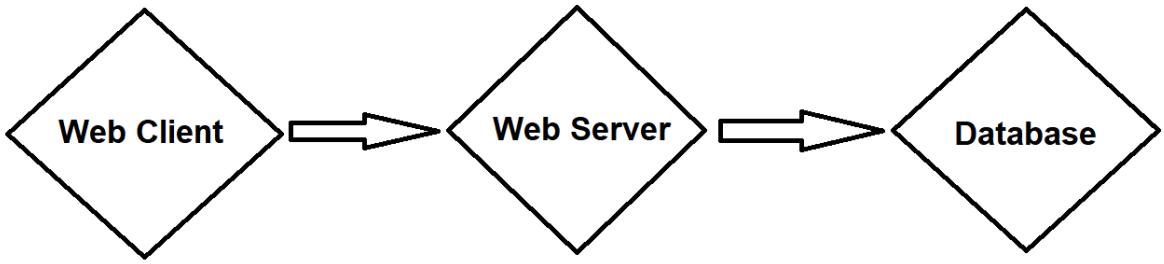

Figure 3. Transaction Diagram

Web app features

Connect to the server (node index.js), then open the index.html file, and choose a feature to use. Instructions on how to use web app features are below:

+ Bookings:

Team 12 - Target 1

Branch, Rasean - dbs009
Wang, Zi Long (Andy) - dbs120
Luu, Nghia Nghiep - dbs065
Ngo, Chi - dbs082

- Enter all required information on the web client browser.
- 'Clear' button is used to reset all entered information. 'Book' is for booking a ticket with the given information.
- After clicking 'Book', the app prompts users to either get a ticket code or book another flight. Ticket code is used to track or cancel a personal flight, the user should save it.

+ Manage Trips:

- Enter all required information on the web client browser.
- If the user does not remember their ticket code, they can retrieve it using 'Forgot your ticket code?'. If they click on that button, they will be asked to input first name, last name, and driver license ID.
- After entering all the required information, click 'Find your trip', a detailed list of information of their flight appears.

4. They have the option to choose to find another trip.

+ Flight Status:

- This feature is used for retrieving all flights with the given constraints.
- For example: if a user wants to look for all flights from Miami with departure date between 07/01/2021 and 09/10/2021. They can leave the 'Arrival' box empty.
- For example: if a user wants to look for all flights to Houston before 05/01/2021. They can leave the 'Departure' and 'From' box empty.
- A user can enter as many (up to 4) constraints as they want to retrieve all flights.

+ Cancel Flights:

- Enter all required information on the web client browser.
- Click 'Find your trip' to show the information of your flight. If a user has made sure that it is their flight, click the button 'Cancel your flight'.
- A new window pops up saying your flight has been canceled. The user has the option to book another flight if they want.
- The user can check that their flight has been canceled by going to the 'Manage Trips' tab and search for their flight, it will then say the flight is invalid.

Input Validation Checking

Team 12 - Target 1

Branch, Rasean - dbs009
Wang, Zi Long (Andy) - dbs120
Luu, Nghia Nghiep - dbs065
Ngo, Chi - dbs082

For inputs that only accept numbers such as Driver License ID, Credit Card Number, and Card Verification Value, we use format type=”text” and pattern=”\d*” with maxlength=”xx”, this way the user is only allowed to enter numbers with a limit of maximum length depending on the variables (maxlength(ID) = 8, maxlength(CCN) = 16, maxlength(CVV) = 3).

For inputs that only accept date types such as Departure Date, Return Date, Date Expiration, we use <input type=”date”>, in which the user can only type in a valid format of date (mm/dd/yyyy).

For other inputs, since they are strings, we can use type=”text”, and maxlength=”xx” to limit the numbers of characters that the user can type in each box.

Since all the inputs are needed for inserting into the database, we make them all required. If a user leaves one of the boxes blank, it will prompt them to enter something into the box before moving into processing.

Storage/Indexing

Storage: every time a new data is inserted into the database, it will be added to the bottom of the table (or last row of table). We can keep track of the number of rows of a table by using table.rows.length. We increase the number of rows, and then do the query “INSERT INTO table (...) VALUES (...)” to insert data into the table.

Indexing: everytime there is a request to retrieve a specific data in a table, the program will do the query “SELECT … FROM table WHERE…” to look for data from that table. Since SQL select statements use the primary key for searching, and because the primary key has an index on it, which is typically a b-tree, the time complexity would be O(log(n)) where "n" is the size of the table.

Transaction.sql and query.sql files

Team 12 - Target 1

Branch, Rasean - dbs009
Wang, Zi Long (Andy) - dbs120
Luu, Nghia Nghiep - dbs065
Ngo, Chi - dbs082
The query.sql file stores sql codes to generate all the relations based on the built ER diagram and store it into a given database.

The transaction.sql file stores all the sql codes that are automatically generated by the source codes whenever there is a transaction request from a user.

How transactions are packaged and sent to the DBMS

A transaction begins with:
“BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;”

and ends with:
“COMMIT;
END TRANSACTION;”
If successfully processed, otherwise, it will end with “ROLL BACK;”

Booking transaction (or .post method):

+ This transaction is performing an adding process of new data into an existing table in the database.
+ When the button “Book” is clicked from the Web Client, after checking for input validation, the function bookingAFlight() is called from “booking.html”, the program will then go to “main.js” to look for the function definition. Within the function bookingAFlight(), the program will call the fetch() function with method: POST, now it will go to “index.js” to look for the POST method. Here, we perform SQL SELECT and INSERT statements to insert the data into the table.

Retrieving transaction (or .get method):

+ This transaction is performing a retrieving data process from an existing table in the database.
+ When the button “Find your trip” is clicked from the Web Client, after checking for input validation, the function findAFlight() is called from “manage-trips.html”, the

program will then go to “main.js” to look for the function definition. Within the function findAFlight(), the program will call the fetch() function with GET method,

now it will go to “index.js” to look for method GET. Here, we perform SQL SELECT statements to retrieve the data from the table.

Removing transaction (or .delete method):

+ This transaction is performing a deleting process of an existing data from a table in the database.
+ When the button “Cancel your flight” is clicked from the Web Client, after checking for input validation, the function cancelFlight() is called from “cancel-flight.html”, the program will then go to “main.js” to look for the function definition. Within the function cancelFlight(), the program will call the fetch() function with DELETE method, now it will go to “index.js” to look for method DELETE. Here, we perform SQL SELECT and DELETE statements to delete the data from the table.