mp.introdb.io/cebu\_pacific\_air

A Requirements Specification Document

For the course on

Introduction to Databases

(INTRODB)

Submitted by:

Cometa, Jet

del Carmen, Paolo

Que, Marcus Jonathan

Wong, Michael Edmund

Ms. Joanna Pauline C. Rivera

Teacher

December 15, 2017

Table of Contents

[**Introduction**](#_z0lmvsfjceuv) **3**

[**Conceptual Model**](#_ws1lhoqqxmcd) **5**

[Figure](#_vi5j14i89yww) 5

[Business Rule](#_6yiroe2fy865) 6

[**Software Features**](#_c9j0jbnyuujy) **7**

[**Query Statements**](#_bvanqf943dso) **20**

# Introduction

This application serves as an interactive flight booking simulator with minimal complexity. The software is built around the end-user’s perspective in mind, providing a ingenuitive suitability. Administrators or certain users that may be acquainted to them may access the software’s administrator mode, giving further depth of the system’s capabilities, such as generating analytical records.

The system requires minimal effort to use, with the least amount of data inquired. Features for end-users are namely the following: flight booking, flight searching, account creation, account view, account flight history view, and account configuration. Administrators may generate analytical records, modify the database’s settings, and toggle certain hidden functionalities of the application. There are different kinds of functionalities dedicated for the administrators, and some are ‘Enable Flight Limiter (stop users from accessing old flights)’, and ‘Enable Account Deprecation (deletes all non-registered accounts that have no future booked flights)’. The remaining are irrelevant and are considered accessories - these are ‘Flight Generator (generates a random flight from January 1, 1970 to December 31, 9999 every 2 seconds)’ and ‘Random Flight Chance (when the given specifications in flight searching does not exist, there is a 17% chance of generating a flight for it, for only 1 PHP seat/bag/meal cost, 1 PHP tax cost, and 1 PHP flight charge cost)’.

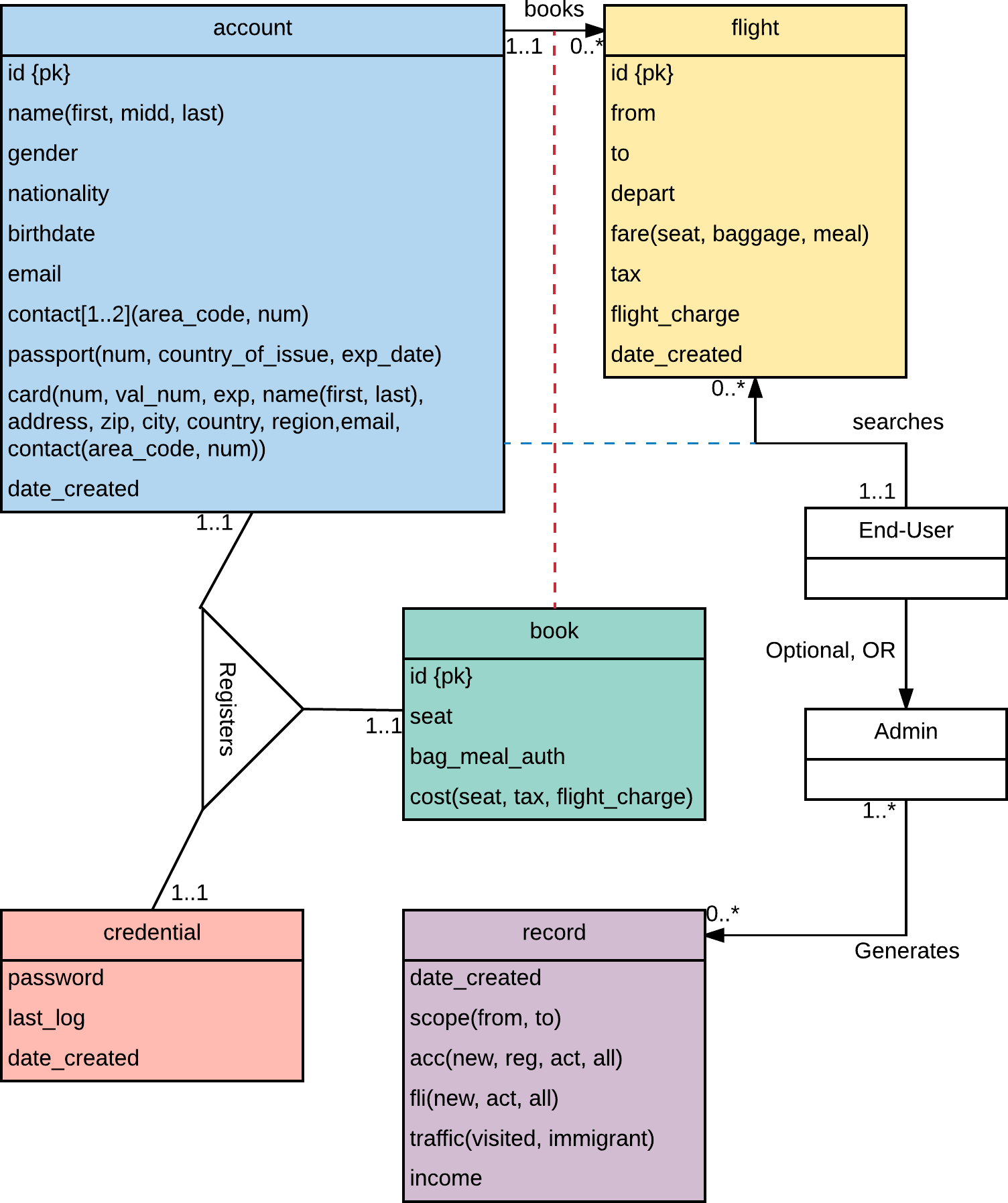
The system is limited to that of inexistent input, meaning there is no way to pay the costs. As such, the system will always assume that all booked flights are paid.

Certain restrictions are implemented, but mostly are not. This is to maintain the system’s integrity to prevent data haywires and malfunctions. Data input does not require a real working design, such as a real credit card number. In addition, the users are guided throughout the session to help guide them.

To clarify, NO AIRCRAFTS or REAL FLIGHTS are involved in the making of this software. It’s purpose is merely to simulate a flight booking software.

# Conceptual Model

## Figure



## Business Rule

* End-Users may search multiple Flights, generating 1 Account per session.
* An Account may book multiple Flights, generating 1 Book per Flight.
* When an Account books a round-trip or multi-city, 2 Books are generated.
* A round-trip booking has 2 Flights and 2 Books, where the second flight is the reciprocal of the first flight, having the first Book has the first Flight, and the second Book has the second flight.
* A multi-city booking has 2 Flights and 2 Books, where the first Book has the first Flight, and the second Book has the second Flight.
* An Account may also register a Credential with a Book once.
* The End-User may become an Admin.
* An Admin may generate multiple Records.
* If enabled, Accounts without any future booked Flights will be deleted if not registered.
* If enabled, Flights are generated randomly every 2 seconds.
* If enabled, Flights have a 17% chance of being generated an End-User searches.
* If enabled, End-Users may search past flights and Accounts may book past flights.
* Credential password require 6 alphanumeric/symbol/white-space characters or more.
* Account.email are all unique, but not a primary key.
* All attributes in Account are required.
* All attributes in Flight except Flight.fare[bagagge, meal] are required.
* All attributes in Credential are required.
* All attributes in Book are required.
* All attributes in Record are required.
* Admins may revert from being an admin anytime.
* End-Users may become an admin anytime.

# Software Features

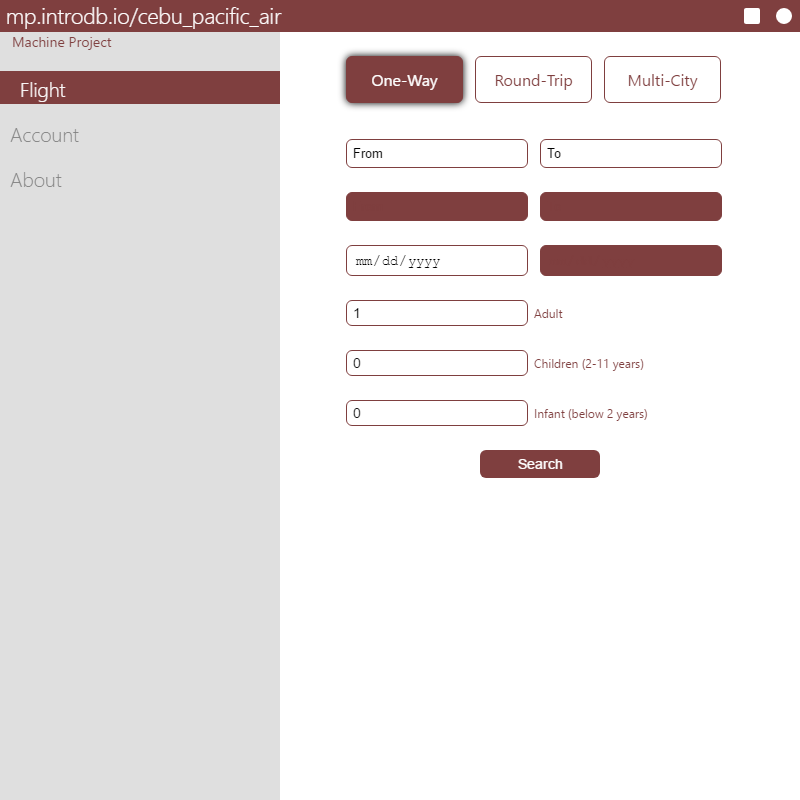


Figure 1.a; Flight Search View

The first view that will be shown is the *Flight Search View*. This sends a query to the database with the designated specification.

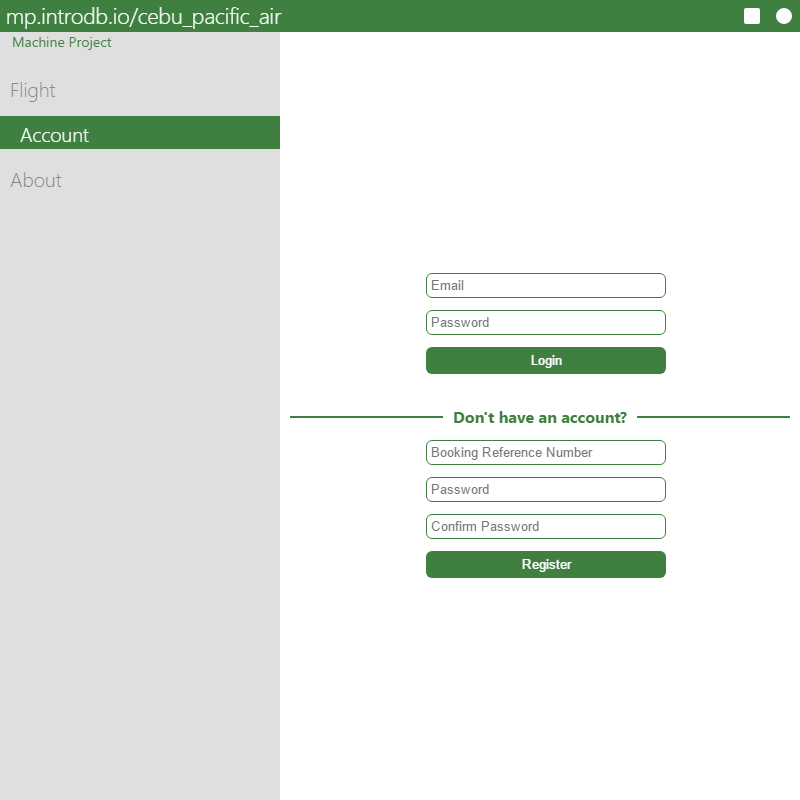


Figure 1.b; Account Login View

The *Account Login View* allows the users to send a query to the database to access their registered accounts. This view also allows users to register their accounts for a credential to unlock access to their accounts.

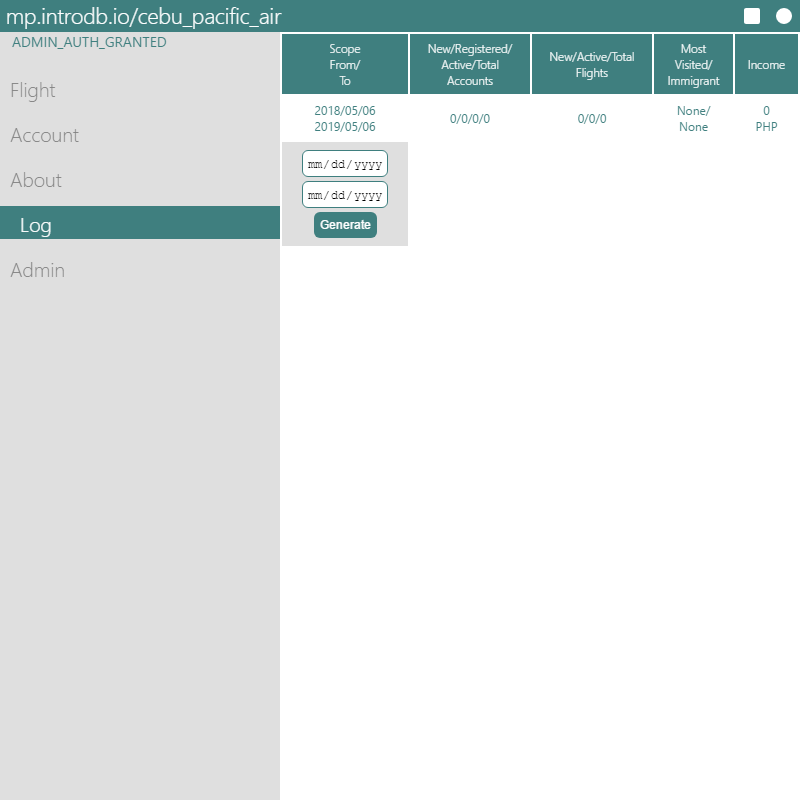


Figure 1.c; Log Generation View

The *Log Generation View* is an Admin-only feature, where one may generate records of the simulator’s condition. The figure above shows a record with a clean condition, meaning there are no inputted data.

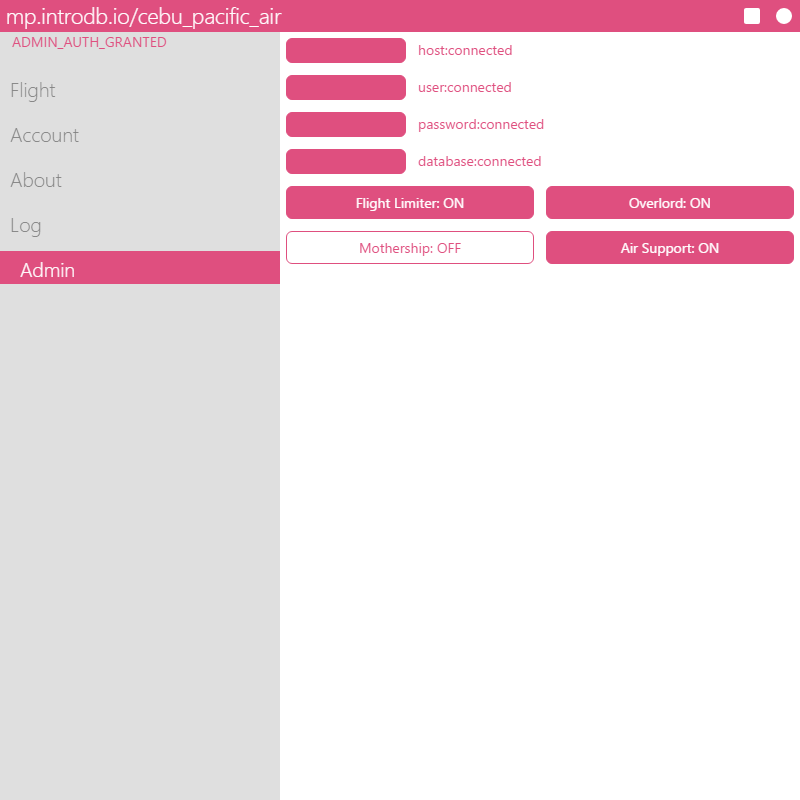


Figure 1.d; Admin View

The *Admin View* allows the user to manipulate the simulator’s condition. The figure above shows that the database is connected properly, and this cannot be changed anymore. Additional options are also available for the user to change.

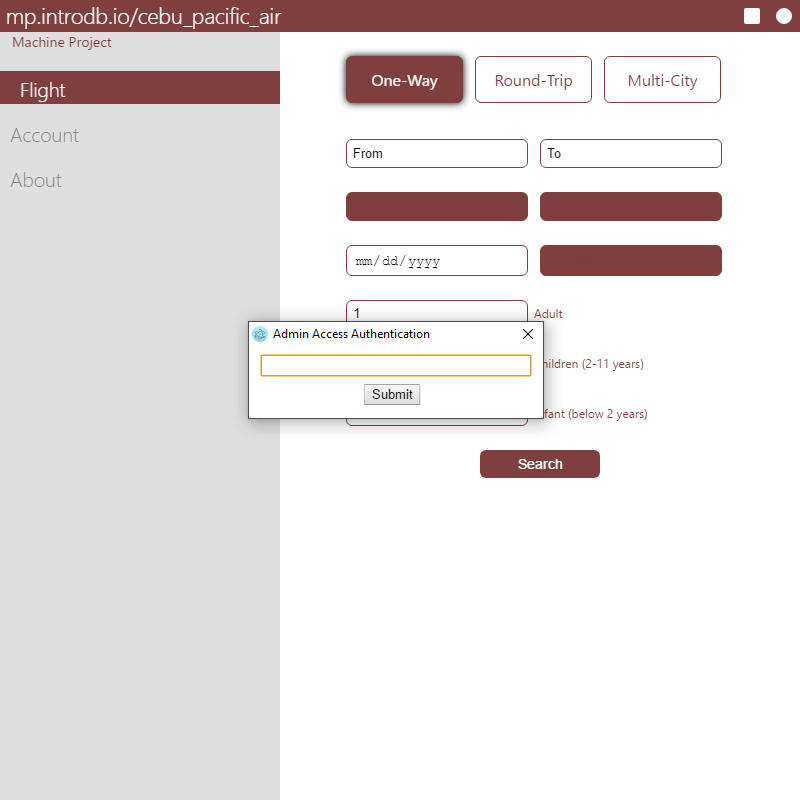


Figure 1.e; Admin Access Authentication Prompt

The *Admin Access Authentication Prompt* is the entry of the administrator mode. It requires the password in order to unlock additional features of the software. There is a hidden button in the software that will show this prompt.

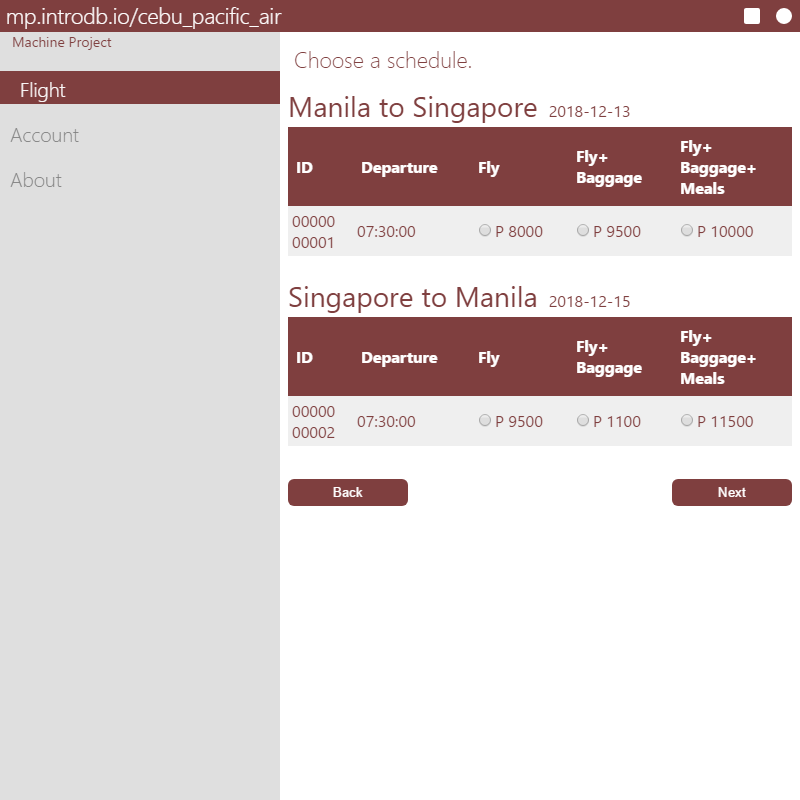


Figure 1.f; Flight List View

The *Flight List View* shows the available flights given in the search view. Each passenger will have the same specification taken in this view.

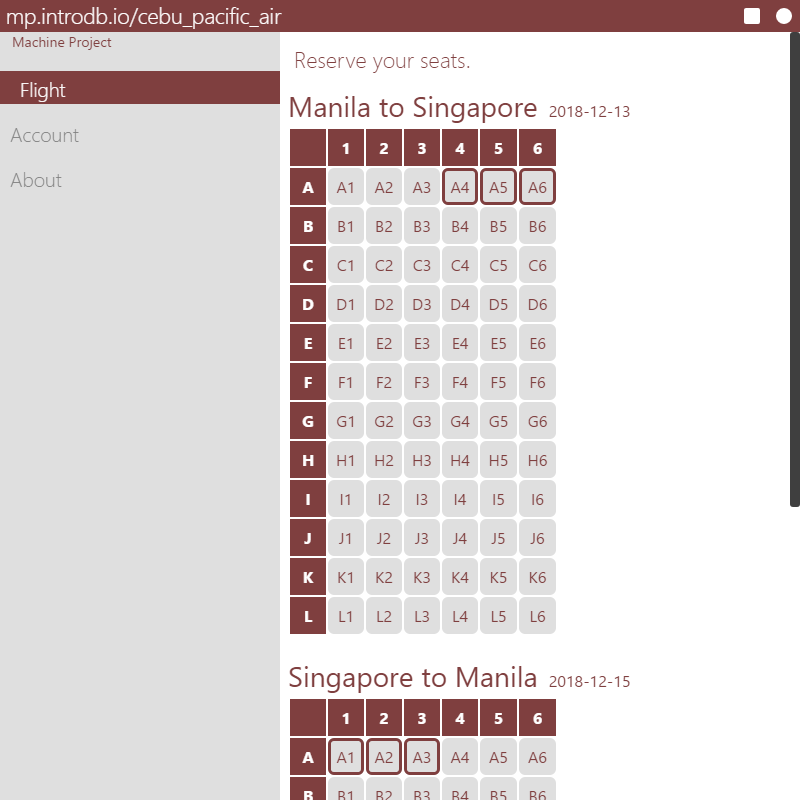


Figure 1.g; Flight Seat View

The *Flight Seat View* allows the user to select their seat in the aircraft, as well as showing the available and taken seats. The figure above shows that the departure flight has A4 through A6 taken, while the return flight has A1 through A3 taken. Each flight must have the same amount of seats designated for each passengers.

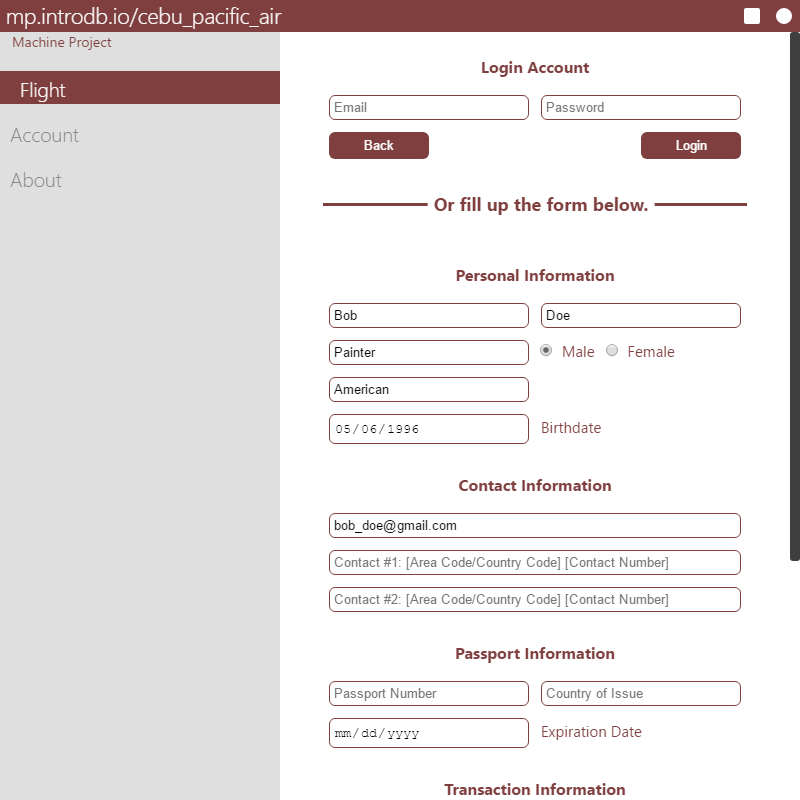


Figure 1.h; Account Details Form

The *Accounts Details Form* requires the user to fill in order to book a flight. If the user has a registered account, they may login their account as seen in the figure above.

\*THE PERSONAL INFORMATION SHOWN ABOVE IS PURELY FICTIONAL\*

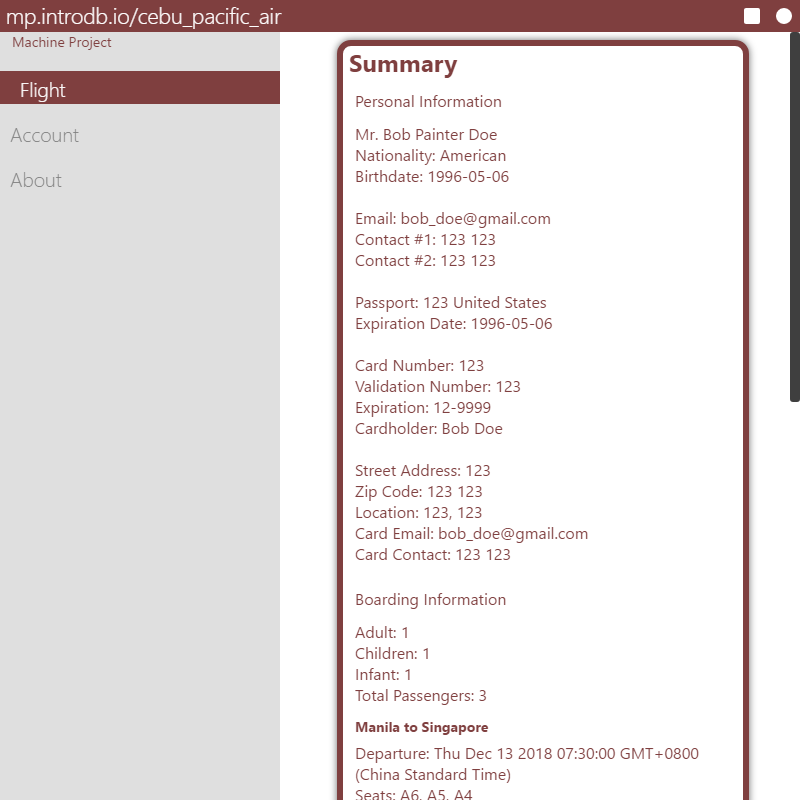


Figure 1.i; Summary View

The *Summary View* lists down all of what the user has inputted earlier, along with the total price.

\*THE PERSONAL INFORMATION SHOWN ABOVE IS PURELY FICTIONAL\*

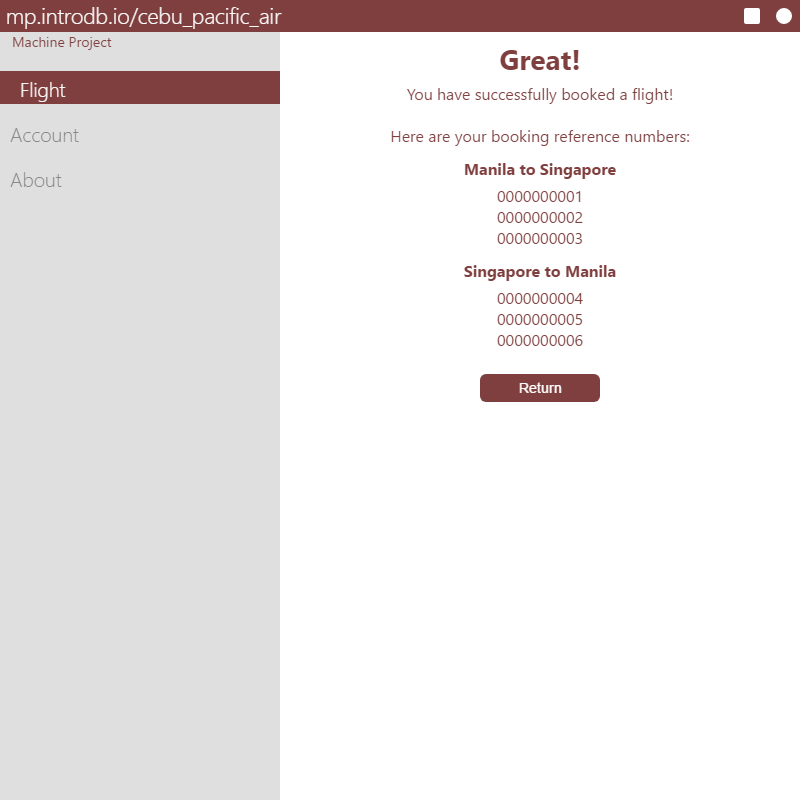


Figure 1.j; The ‘Great’ View

*The ‘Great’ View* shows the user’s hard work and dedication, rewarding them with their booking reference numbers. These numbers are proportional to the passengers, and each may be swapped. Each number is tied to a seat number. The user may also register their account with one of the given reference numbers. Although not implemented, the user will receive a message via email to verify their registration.

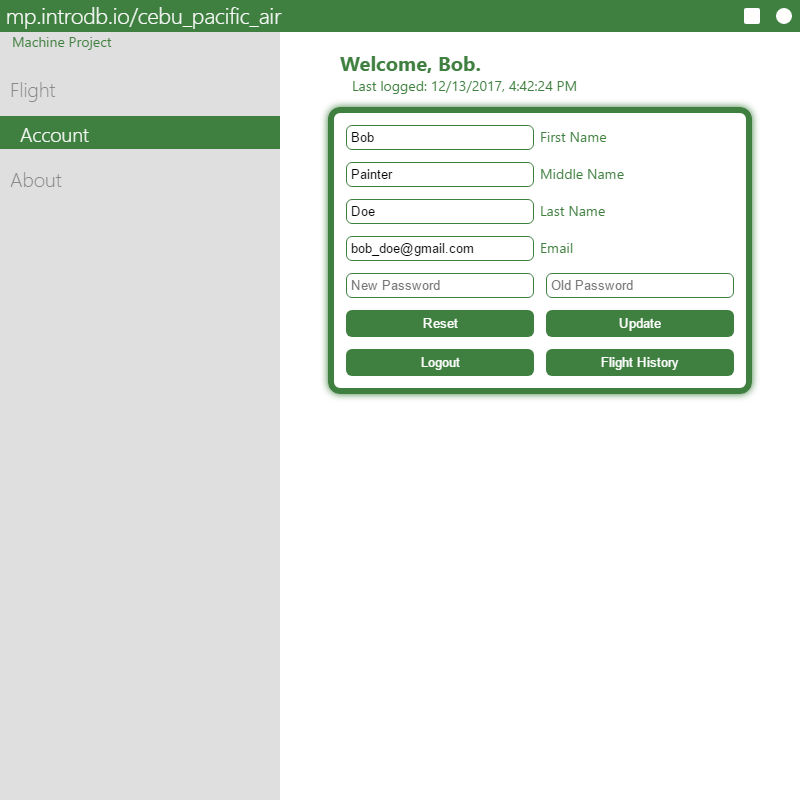


Figure 1.k; Account View

The *Account View* shows the user’s information and also allows them to change it. The software will greet them promptly after each update.

\*THE PERSONAL INFORMATION SHOWN ABOVE IS PURELY FICTIONAL\*

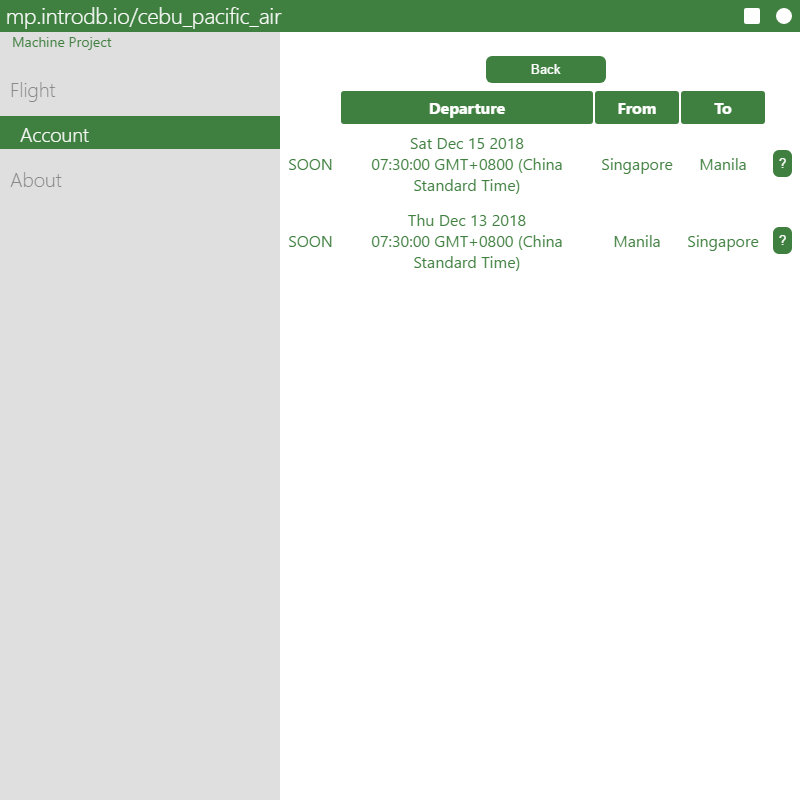


Figure 1.l; Flight History View

The *Flight History View* shows the user’s booked flights. If the flights are in the future, they will be tagged with ‘SOON’, while the flights happening today will be tagged as ‘NOW’.

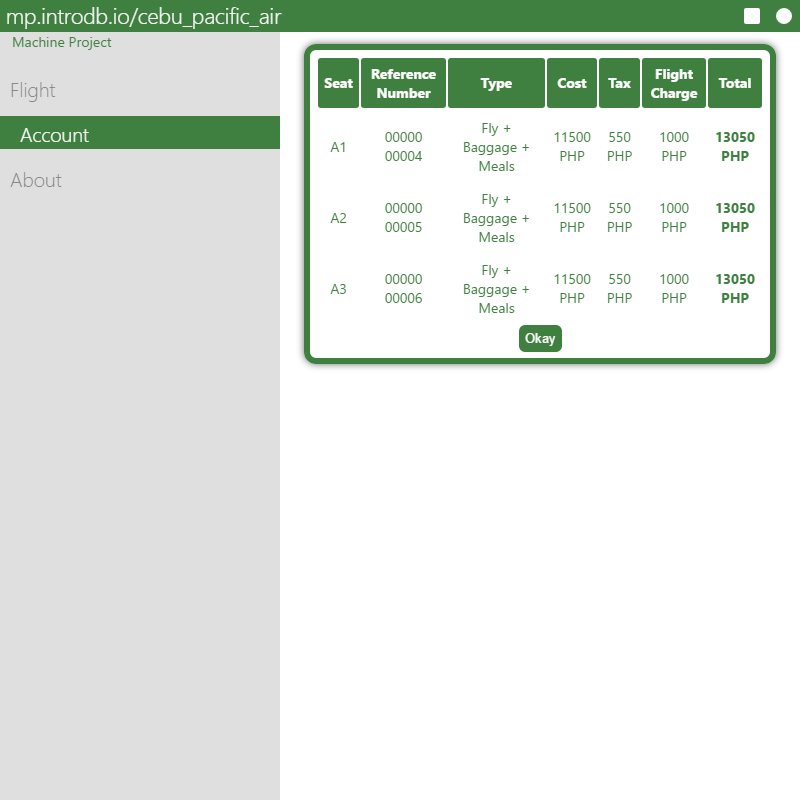


Figure 1.m; Flight History Information View

The *Flight History Information View* or the *Flight HIV* shows the flights detailed information. This allows the user to reassess their budget.

# Query Statements

The following queries are written in JavaScript.

|  |
| --- |
| con.query(  "INSERT INTO account (name\_first, name\_last, name\_midd, gender, nationality, birthdate, email, contact0\_area\_code, contact0\_num, contact1\_area\_code, contact1\_num, passport\_num, passport\_country\_of\_issue, passport\_exp\_date, card\_num, card\_val\_num, card\_exp, card\_name\_first, card\_name\_last, card\_address, card\_zip, card\_city, card\_country, card\_region, card\_email, card\_area\_code, card\_contact\_num) VALUES (" + v + ")")  // Where ‘v’ is the values inputted by the user. |

Figure 2.a; Account Creation Query

This query inputs all the required attributes for the account.

|  |
| --- |
| con.query("INSERT INTO book (flight\_id, account\_id, seat, bag\_meal\_auth, cost, cost\_tax, cost\_flight\_charge) VALUES (" + selected\_fare[i].id + "," + id + ",\"" + k + "\"," + selected\_fare[i].selected + "," + selected\_fare[i][fare\_index[selected\_fare[i].selected]] + "," + selected\_fare[i].tax + "," + selected\_fare[i].flight\_charge + ")"  /\* Where ‘i’ designates if the book is departure (0) or return (1).  Where ‘k’ is the seat location.  Where ‘selected\_fare[i].selected’ tells if the flight is a fly only, baggage, or baggage+meals.  \*/ |

Figure 2.b; Book Creation Query

This query creates a book for each seat location. Two books are generated if there is a return flight.

|  |
| --- |
| con.query("SELECT id, depart, fare\_seat, fare\_baggage, fare\_meal, tax, flight\_charge FROM flight WHERE flight.from = \"" + from + "\" AND flight.to = \"" + to + "\" AND DATE(flight.depart) = \"" + depart + "\"" + (parent.admin\_config[0] ? " AND DATE(flight.depart) >= NOW()" : "")  // Don’t mind the parent.admin\_config[0]. It's a secret. |

Figure 2.c; Flight Search Query

This query lists all the available flights designated by the user.

|  |
| --- |
| con.query(" SELECT id FROM book WHERE book.flight\_id = " + Number(selected\_fare[i].id) + " AND book.seat = \"" + seat + "\"")  // Where ‘i’ designates if the book is departure (0) or return (1). |

Figure 2.d; Flight Seat Query

This query disables any seats that are already reserved.

|  |
| --- |
| con.query("SELECT id FROM account WHERE email = \"" + elm[11].value.toLowerCase() + "\"")  // Where ‘elm[11]’ is the email. |

Figure 2.e; Email Anti-Duplication Query

This query makes sure that all emails are unique.

|  |
| --- |
| con.query("SELECT a.\* FROM account a, credential b WHERE a.id = b.account\_id AND a.email = \"" + elm[0].value + "\"\ AND b.password = \"" + elm[1].value + "\"")  /\* Where ‘elm[0]’ is the email.  Where ‘elm[1]’ is the password.  \*/ |

Figure 2.f; Book Login Query

This query allows the user to login to their account when booking.

|  |
| --- |
| con.query("SELECT a.\* FROM book a WHERE a.account\_id = " + account\_dat.id + " AND a.flight\_id = " + tag.id)  /\* Where ‘account\_dat’ is the user’s account information.  Where ‘tag’ is the current selected flight.  \*/ |

Figure 2.g; Flight History Information View Query

This query shows how the *Flight HIV* shows the information of the flight in the user’s flight history.

|  |
| --- |
| con.query("SELECT a.\* FROM flight a, book b WHERE a.id = b.flight\_id AND b.account\_id = " + account\_dat.id + " GROUP BY a.id ORDER BY a.depart DESC, a.id")  // Where ‘account\_dat’ is the user’s account information. |

Figure 2.h; Flight History Query

This query shows how the flight history generates its list.

|  |
| --- |
| con.query("UPDATE credential SET last\_log = \"" + t.getFullYear() + "-" + (t.getMonth()+1) + "-" + t.getDate() + " " + t.getHours() + ":" + t.getMinutes() + ":" + t.getSeconds() + "\" WHERE account\_id = " + account\_dat.id)  // Where ‘t’ is the current date and time. |

Figure 2.i; Account Last Login Query

This query sets the account’s last login date and time every time the account is logged in.

|  |
| --- |
| con.query("UPDATE account a SET name\_first = \"" + input[0].value + "\", name\_midd = \"" + input[1].value + "\", name\_last = \"" + input[2].value + "\", email = \"" + input[3].value.toLowerCase() + "\" WHERE a.id = " + account\_dat.id)  /\* Where ‘input[0]’ is the first name.  Where ‘input[1]’ is the middle name.  Where ‘input[2]’ is the last name.  Where ‘input[3]’ is the email.  Where ‘account\_dat’ is the user’s account information.  \*/ |

Figure 2.j; Account Changer Query

This query allows the user to change some of their account details.

|  |
| --- |
| con.query("UPDATE credential a SET password = \"" +input[4].value + "\" WHERE a.account\_id = " + account\_dat.id)  /\* Where ‘input[4]’ is the password.  Where ‘account\_dat’ is the user’s account information.  \*/ |

Figure 2.k; Credential Changer Query

This query allows the user to change their password.

|  |
| --- |
| con.query("SELECT \* FROM record") |

Figure 2.l; Analytical Record Log Query

This query shows the analytical records.

|  |
| --- |
| con.query("INSERT INTO record (scope\_from, scope\_to, acc\_new, acc\_reg, acc\_act, acc\_all, fli\_new, fli\_act, fli\_all, visited, immigrant, income ) VALUES (\"" + input[0].value + "\",\"" + input[1].value + "\"," + res[0][0].res + "," + res[1][0].res + "," + res[2][0].res + "," + res[3][0].res + "," + res[4][0].res + "," + res[5][0].res + "," + res[6][0].res + ",\"" + (!res[7][0] ? "None" : res[7][0].to) + "\",\"" + (!res[8][0] ? "None" : res[8][0].from) + "\"," + (res[9][0].res || 0) + ")") |

Figure 2.m; Record Generation Query

This query creates an entry in the record table. The ambiguous variables are explained at ‘Figure 2.p’.

|  |
| --- |
| con.query("SELECT \*FROM flight a WHERE a.from = \"" + dat[0] + "\" AND a.to = \"" + + dat[1] + "\" AND a.depart = \"" + + dat[2] + "\"") |

Figure 2.n; Flight Anti-Duplication Query

This query prevents flights that are exactly the same.

|  |
| --- |
| con.query("INSERT INTO flight (flight.from, flight.to, depart, fare\_seat, fare\_baggage, fare\_meal, tax, flight\_charge ) VALUES (\"" + dat[0] + "\",\"" + dat[1] + "\",\"" + dat[2] + "\"," + dat[3] + "," + dat[4] + "," + dat[5] + "," + dat[6] + "," + dat[7] + ")") |

Figure 2.o; Flight Random Generator Query

This query creates random flight entries.

|  |
| --- |
| consecutive\_query(0,[  // acc\_new 0  "SELECT COUNT(\*) AS 'res' FROM account a WHERE a.create >= \"" + input[0].value + "\" AND a.create <= \"" + input[1].value + "\"",  // acc\_reg 1  "SELECT COUNT(\*) AS 'res' FROM credential a WHERE a.create >= \"" + input[0].value + "\" AND a.create <= \"" + input[1].value + "\"",  // acc\_act 2  "SELECT COUNT(\*) AS 'res' FROM credential a WHERE a.last\_log >= \"" + input[0].value + "\" AND a.last\_log <= \"" + input[1].value + "\"",  // act\_all 3  "SELECT COUNT(\*) AS 'res' FROM account a WHERE a.create <= \"" + input[1].value + "\"",  // fli\_new 4  "SELECT COUNT(\*) AS 'res' FROM flight a WHERE a.create >= \"" + input[0].value + "\" AND a.create <= \"" + input[1].value + "\"",  // fli\_act 5  "SELECT COUNT(\*) AS 'res' FROM (SELECT COUNT(\*) FROM flight a, book b WHERE a.id = b.flight\_id AND a.create >= \"" + input[0].value + "\" AND a.create <= \"" + input[1].value + "\" GROUP BY a.id) a",  // fli\_all 6  "SELECT COUNT(\*) AS 'res' FROM flight a WHERE a.create <= \"" + input[1].value + "\"",  // visited 7  "SELECT COUNT(\*), a.to FROM flight a, book b WHERE a.id = b.flight\_id AND a.create >= \"" + input[0].value + "\" AND a.create <= \"" + input[1].value + "\" GROUP BY a.id, a.to ORDER BY 1 DESC LIMIT 1",  // immigrant 8  "SELECT COUNT(\*), a.from FROM flight a, book b WHERE a.id = b.flight\_id AND a.create >= \"" + input[0].value + "\" AND a.create <= \"" + input[1].value + "\" GROUP BY a.id, a.from ORDER BY 1 DESC LIMIT 1",  // income 9  "SELECT SUM(b.cost) + SUM(b.cost\_tax) + SUM(b.cost\_flight\_charge) AS 'res' FROM flight a, book b WHERE a.id = b.flight\_id AND a.depart >= \"" + input[0].value + "\" AND a.depart <= \"" + input[1].value + "\""  ]) |

Figure 2.p; Spooky Creepy Scary Lengthy Query

This list of query generates all the required attributes to create an entry in the record table. And it’s also spooky, and creepy, and scary. And lengthy.

|  |
| --- |
| con.query("DELETE FROM account WHERE id NOT IN (SELECT account\_id FROM credential ) AND id NOT IN (SELECT account\_id FROM book a, flight b WHERE a.flight\_id = b.id AND b.depart > NOW())") |

Figure 2.q; Overlord’s Query

This query devours (deletes) any accounts that are not registered and has no future booked flights.