# FLIGHT RESERVATION DATABASE PROJECT

Using MySQL in phpMyAdmin

Wiscovitch – DB Project
Informatics Department Spring B 2023

# Table of Contents

# Introduction

## Project Overview

The following document contains technical information for the database design and implementation for a travel agency specializing in flight reservations. The fictitious business, FlyRes, needed a database containing information on flights, airlines, airports, flight schedules, pricing, reservations, passengers, customers, and employees. This document details the design, implementation, and testing of this database. The project was completed using Lucid Chart for diagramming the design plans and phpMyAdmin for the implementation of MySQL architecture and database administration. The phpMyAdmin program was chosen for ease of use. The GUI provided by phpMyAdmin allows for non-technical administration of the database that would theoretically be passed on to the manager of the FlyRes business. The goal of this project was to learn the basics of database design, implementation, and administration using the MySQL language.

# Database Design

## Business Rules

Each flight departs and arrives from/to only one airport.
Each airport can be the origin or destination of many flights.

Each flight is operated by one airline.
Each airline operates many flights.

Each scheduled flight is related to only one flight.
Each flight may result in many scheduled flights.

Each reservation record is for a scheduled flight.
Each scheduled flight has many reservation records.

Each reservation record is created by or assigned to one customer representative/employee.
Each employee/customer representative can make/be assigned many reservations.

Each reservation record belongs to one passenger.
Each passenger can have many reservation records.

Each customer is a passenger, but not every passenger is a customer.

Each passenger that is also a customer can have many accounts.
Each account belongs to only one passenger that is also a customer.

## Entity Relationship Diagram

The entity relationship diagram in figure 1 provides a visual representation of the entities, their attributes, and the entity relationships as defined by the business rules.

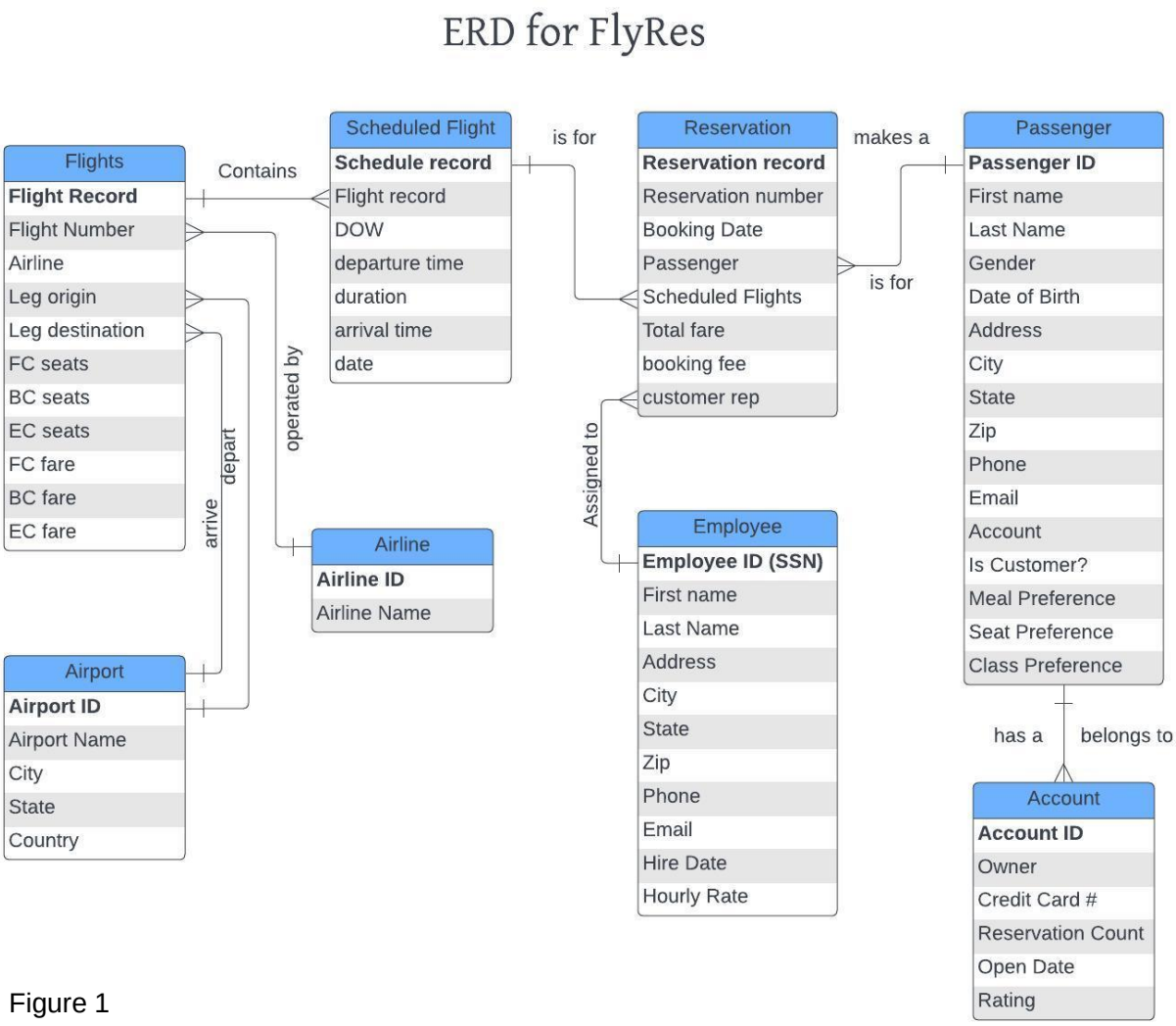



Figure 1

## Data Dictionary

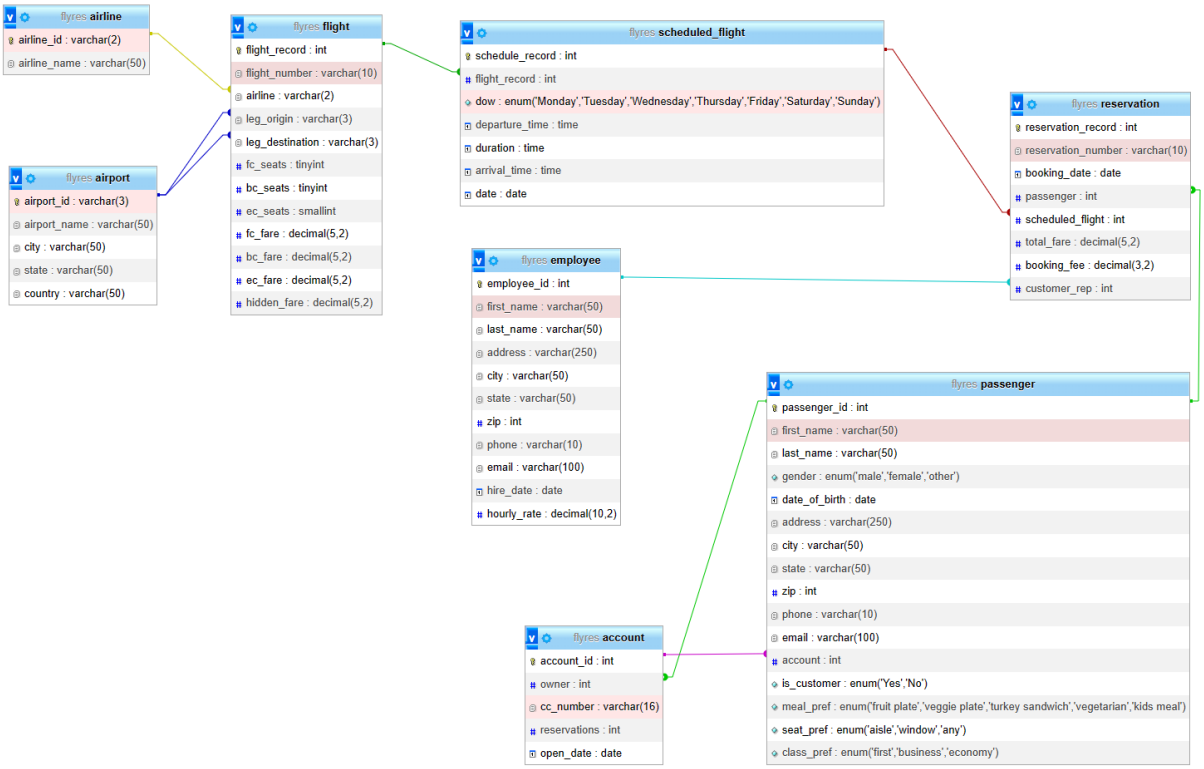
| Entity | Attribute | Data Type | Description |
|---|---|---|---|
| **Flight** | Flight Record | Int | This primary key of the flight table is the record instance of a specified flight number and the associated leg/route. A single flight may be made up of multiple flight records. |
| **Flight** | Flight Number | Varchar(10) | The flight number is a flight operated by a specified airline and is comprised of 1 to many legs. |
| **Flight** | Airline | Varchar(2) | This foreign key is the airline which operates the specified flight. |
| **Flight** | Leg Origin | Varchar(3) | This foreign key references the airport code of the origin location of the flight and associated flight leg. |
| **Flight** | Leg Destination | Varchar(3) | This foreign key references the airport code of the destination of the specified flight and associated flight leg. |
| **Flight** | FC Seats | Int | This number represents the number of first-class seats available for the associated flight. (may be NULL) |
| **Flight** | BC Seats | Int | This number represents the number of business class seats available for the associated flight. (may be NULL) |
| **Flight** | EC Seats | Int | This number represents the number of economy class seats available for the associated flight. |
| **Flight** | FC Fare | Decimal(5,2) | This decimal number represents the cost to purchase a first-class seat for the associated flight. (may be NULL) |
| **Flight** | BC Fare | Decimal(5,2) | This decimal number represents the cost to purchase a business class ticket for the associated flight. (may be NULL) |
| **Flight** | EC Fare | Decimal(5,2) | This decimal number represents the cost to purchase an economy class ticket for the associated flight. |
| **Flight** | Hidden Fare | Decimal(5,2) | This is the lowest fare that would be accepted for this flight in customer bidding method of purchase. |
| **Airline** | Airline ID | Varchar(2) | This primary key is a unique two-character identification for an airline. (e.g. AA = American Airlines.) |
| **Airline** | Airline Name | Varchar(50) | This is the long hand name of the airline. (e.g. American Airlines) |
| **Airport** | Airport ID | Varchar(3) | This primary key is the 3-character airport code. (e.g. SFO = San Francisco International Airport) |
| **Airport** | Airport Name | Varchar(50) | This is the long hand name of the airport. (e.g. San Francisco International Airport) |
| **Airport** | City | varchar (50) | This is the city in which the airport is located. |
| **Airport** | State | Varchar(50) | If applicable, this is the state where the airport is located. |
| **Airport** | Country | Varchar(50) | This is the country of the airport location. |

| Entity | Attribute | Data Type | Description |
|---|---|---|---|
| **Scheduled Flight** | Schedule Record | Int | This primary key uniquely identifies an occurrence of a flight number and the associated flight leg for a specific date. |
| **Scheduled Flight** | Flight Record | Int | This foreign key identifies which flight number and flight leg the schedule is for. |
| **Scheduled Flight** | DOW | Enum | This predefined list will contain the days of the week indicating which days the flight is scheduled to operate on. |
| **Scheduled Flight** | Departure Time | Time | This is the time for which the flight is scheduled to depart. |
| **Scheduled Flight** | Duration | Time | This is the length of time it takes for the flight to go from origin to destination. |
| **Scheduled Flight** | Arrival Time | Time | This calculated field results from taking the departure time adding the duration which equals out to the scheduled arrival time for the flight. |
| **Scheduled Flight** | Date | Date | The scheduled date for the flight populated from the day of the week's association with the current year's calendar. |
| **Reservation** | Reservation Record | Int | This primary key uniquely identifies a reservation instance for a specified passenger. |
| **Reservation** | Reservation Number | Varchar(10) | This variable character field is the representation of a reservation created by a customer or customer representative. |
| **Reservation** | Booking Date | Date | This date field represents when the reservation was created. |
| **Reservation** | Passenger | Int | This foreign key identifies the passengers associated with a reservation. |
| **Reservation** | Scheduled Flights | Int | This foreign key identifies which flight/leg the reservation is for. |
| **Reservation** | Total Fare | Decimal(5,2) | This decimal number represents the total cost of the flight reservation. |
| **Reservation** | Booking Fee | Decimal(3,2) | This decimal number represents the booking fee associated with the reservation. |
| **Reservation** | Customer Rep | Int(9) | This foreign key identifies the customer representative that created the reservation or who was assigned to the associated reservation. |
| **Passenger** | Passenger ID | Int | This primary key uniquely identifies each passenger. |
| **Passenger** | First Name | Varchar(25) | This is the first name of the passenger. |
| **Passenger** | Last Name | Varchar(50) | This is the last name of the passenger. |
| **Passenger** | Gender | Enum | This is a predefined list of genders. (i.e. male (M), female (F), other/nonbinary (X). |
| **Passenger** | Date of Birth | Date | This is the passenger's date of birth. |
| **Passenger** | Address | Varchar(250) | This is the street address of the passenger. |
| **Passenger** | City | Varchar(50) | This is the city associated with the passenger's address. |
| **Passenger** | State | Varchar(50) | This is the state associated with the passenger's address. |
| **Passenger** | Zip | Int(10) | This is the zip code associated with the passenger's address. |

| Entity | Attribute | Data Type | Description |
|---|---|---|---|
| **Passenger** | Phone | Varchar(10) | This is the phone number of the passenger. (may be NULL) |
| **Passenger** | Email | Varchar(100) | This is the email address associated with the passenger. (may be NULL) |
| **Passenger** | Account | Int | This foreign key belongs to the associated customer account if the passenger is also a customer. (may be NULL) |
| **Passenger** | Is Customer | Boolean | This Boolean identifies if the passenger is also the purchasing customer/account holder. |
| **Passenger** | Meal Preference | Enum | Identifies the passenger's preferred meal for the flight from a predefined list. (may be NULL) |
| **Passenger** | Seat Preference | Enum | Identifies the passenger's preferred seating from a predefined list. (i.e. aisle or window) (may be NULL) |
| **Passenger** | Class Preference | Enum | Identifies the passenger's preferred flight class from a predefined list. (i.e. first, business, economy) (may be NULL) |
| **Account** | Account ID | Int | This primary key uniquely identifies each account in the reservation system. |
| **Account** | Owner | Int | This foreign key belongs to an associated passenger who is also a customer. |
| **Account** | Credit Card # | Varchar(16) | This 16-digit number is the credit card number on file for the associated account after the first purchase. (may be NULL) |
| **Account** | Reservations | Int | This is a running count of the number of reservations created by this account. (may be NULL) |
| **Account** | Open Date | Date | This is the date that the account was created. |
| **Employee** | Employee ID | Int(9) | This primary key uniquely identifies the employee using their social security number. |
| **Employee** | First name | Varchar(25) | This is the first name of the employee. |
| **Employee** | Last name | Varchar(50) | This is the last name of the employee. |
| **Employee** | Address | Varchar(250) | This is the street address of the employee. |
| **Employee** | City | Varchar(50) | This is the city of the employee's address. |
| **Employee** | State | Varchar(50) | This is the state of the employee's address. |
| **Employee** | Zip | Int(10) | This is the zip code of the employee's address. |
| **Employee** | Phone | varchar(10) | This is the phone number of the employee. |
| **Employee** | Email | Varchar(50) | This is the employee's email address. |
| **Employee** | Hire Date | Date | This is the date the employee was hired. |
| **Employee** | Hourly Rate | Decimal(2,2) | This is the hourly rate the employee is paid. |

# Relational Schema

A relational schema provides a high-level overview of each table and the relationships between tables. Primary keys become foreign keys in the related tables. For example, airline_id is a primary key of the airline table, but a foreign key in the flight table identifying which airline operates each flight.



# Database Implementation

## Creating Tables Using SQL

```
CREATE TABLE airline (
    airline_id varchar(2) NOT NULL PRIMARY KEY,
    airline_name varchar(50) NOT NULL
    );
```

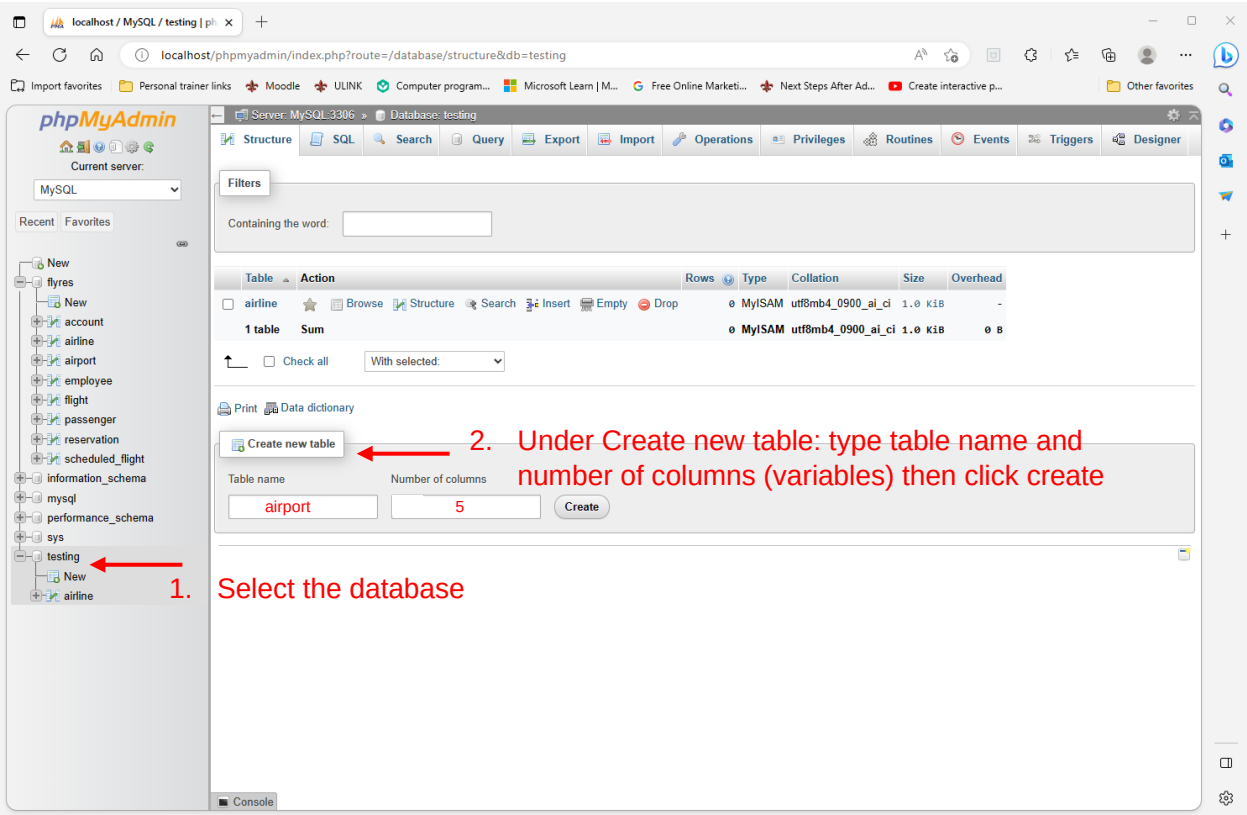
```
SELECT * FROM `airline`
```

Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

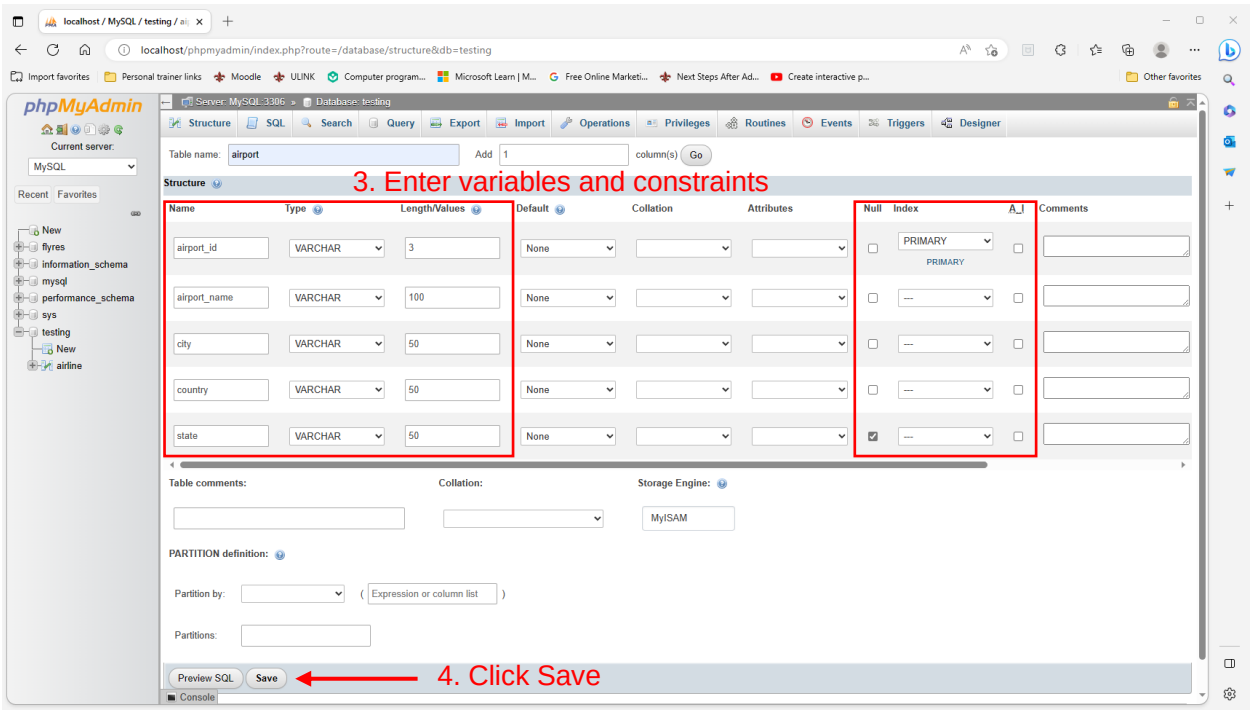
| airline_id | airline_name |
|---|---|

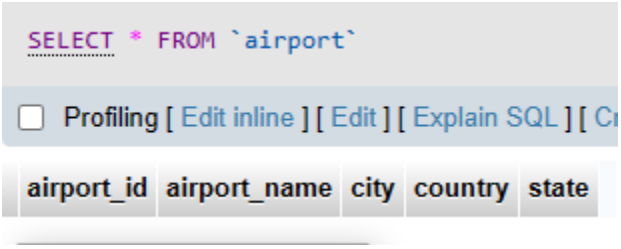
## Creating Tables Using phpMyAdmin GUI

PhpMyAdmin's GUI offers user friendly methods for creating new tables. The following screenshots detail how the GUI can be used to create new database tables. These are for example purposes in a test database to demonstrate the GUI.




3. Input variables and information. Name of variable, type, length (if applicable), index if needed (for primary and foreign key variables, options to select NULL or AI (auto-increment), and set PRIMARY KEYS.

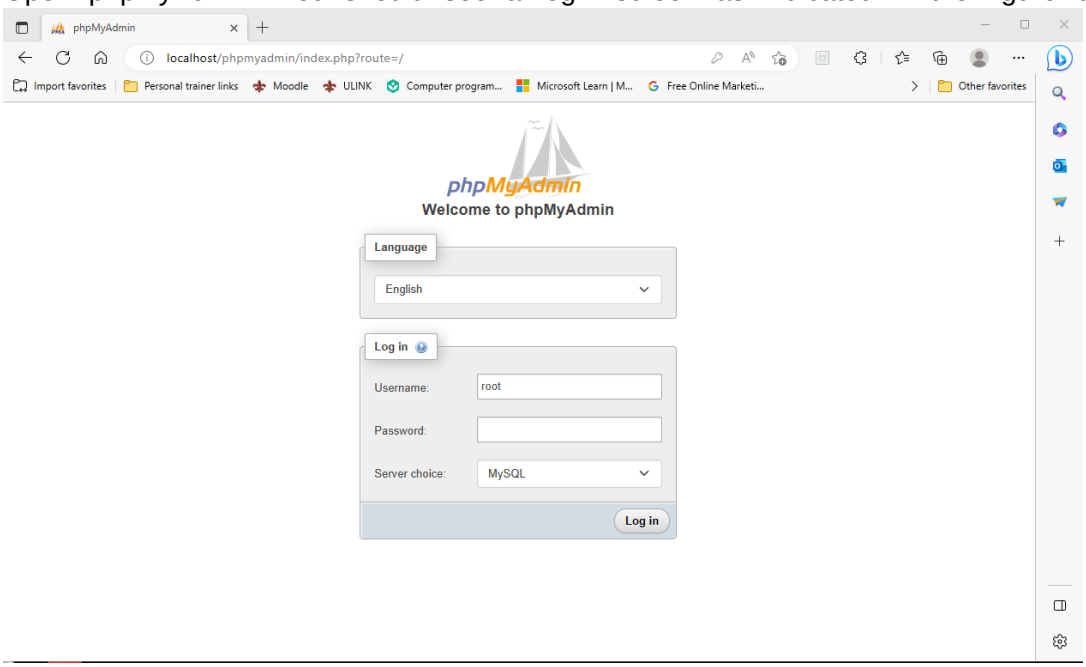

3. Enter variables and constraints
4. Click Save


SELECT * FROM `airport`
Profiling [ Edit inline ] [ Edit ] [ Explain SQL ]
airport_id
airport_name
city
country
state

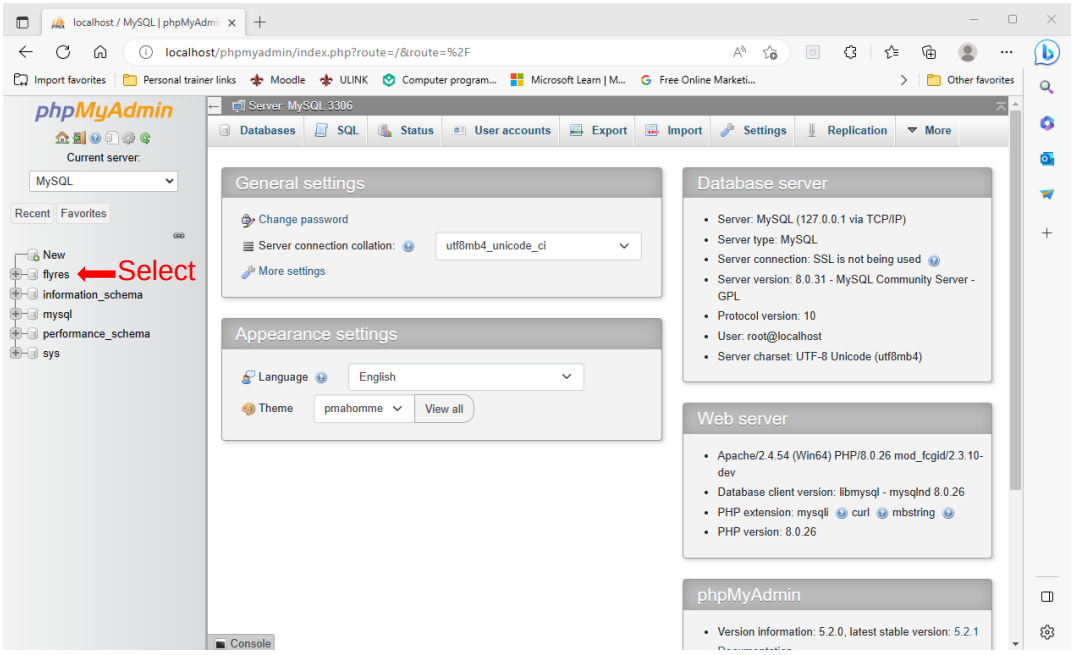
# Entering Table Information Using GUI

## Entering a New Employee into the Employee Table

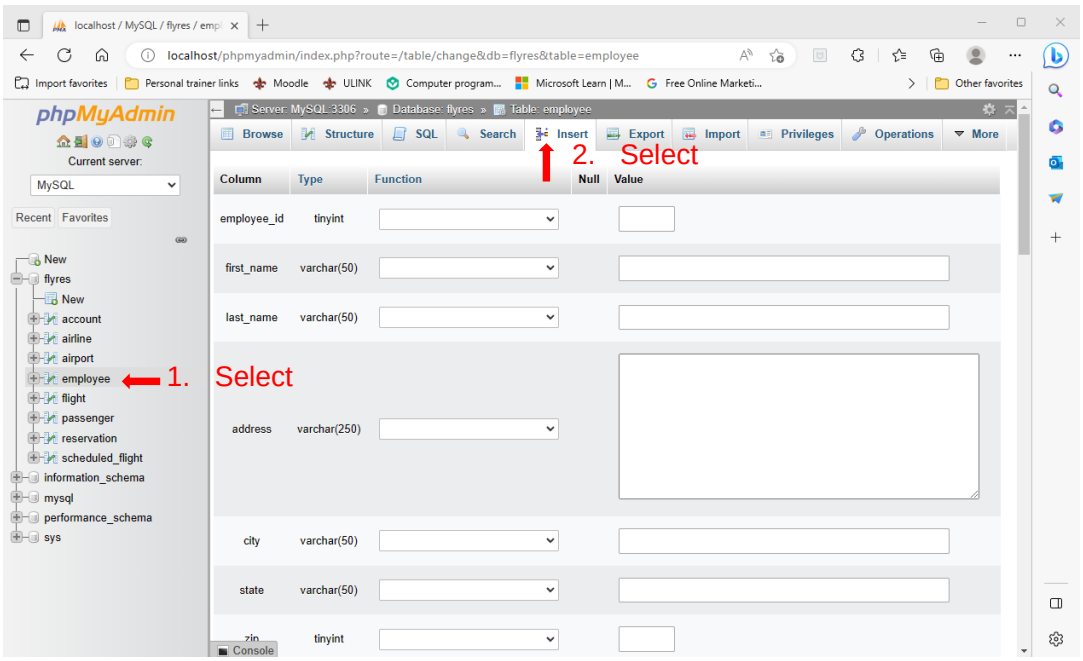
1. Open phpMyAdmin. You should see a login screen as indicated in the figure below:



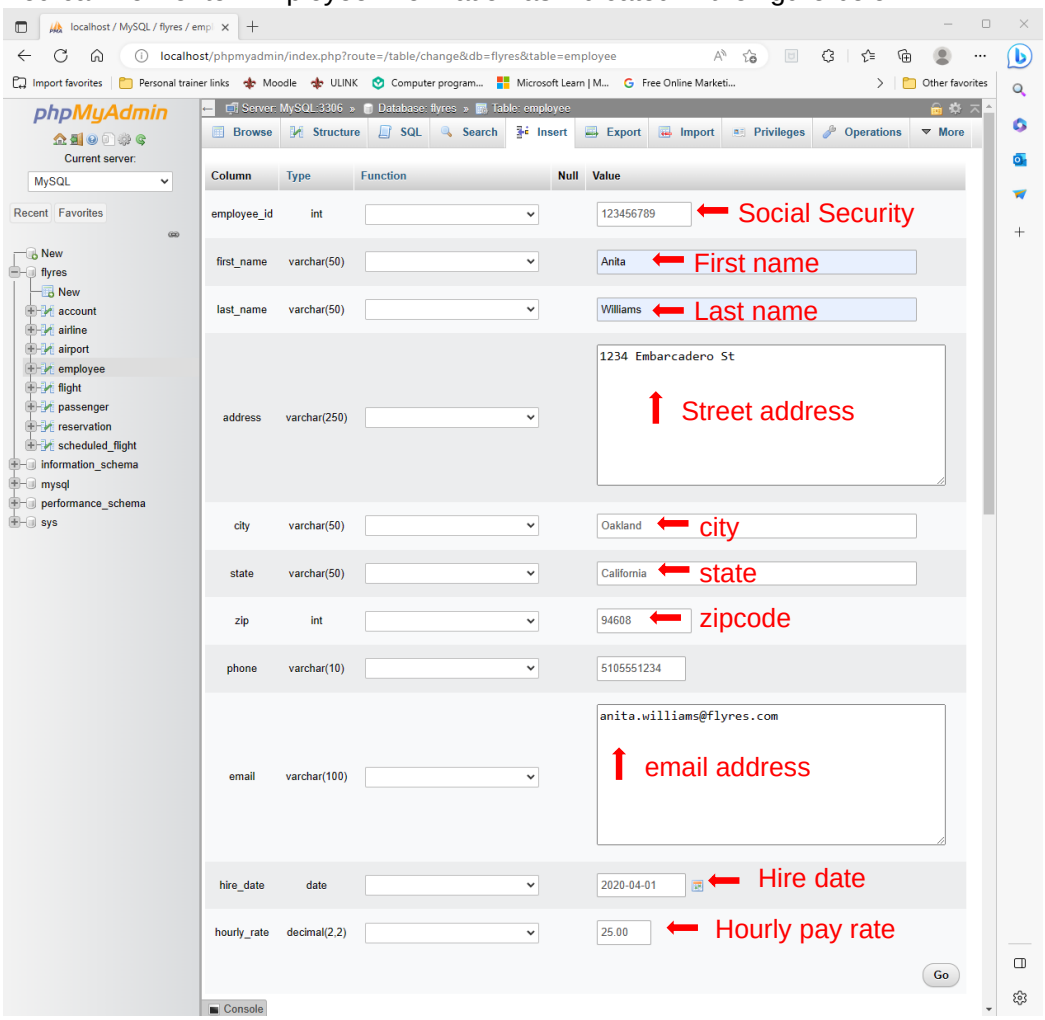

2. Log in username: manager
   Password: flyresmanager
3. Select the flyres database on the lefthand menu to open the flyres database




4. You will now see the list of tables on the left-hand menu. Click on employee as indicated in the figure below. Then, select Insert tab along the top toolbar.




5. You can now enter Employee information as indicated in the figure below.



6. After entering in the required information click the “Go” button to save.

## Entering Table Information Using SQL

### Entering a new Airline into the Airline Table

```
INSERT INTO airline (airline_id, airline_name)
VALUES ('AA', 'American Airlines');
```

# Generating Flight Schedule Information

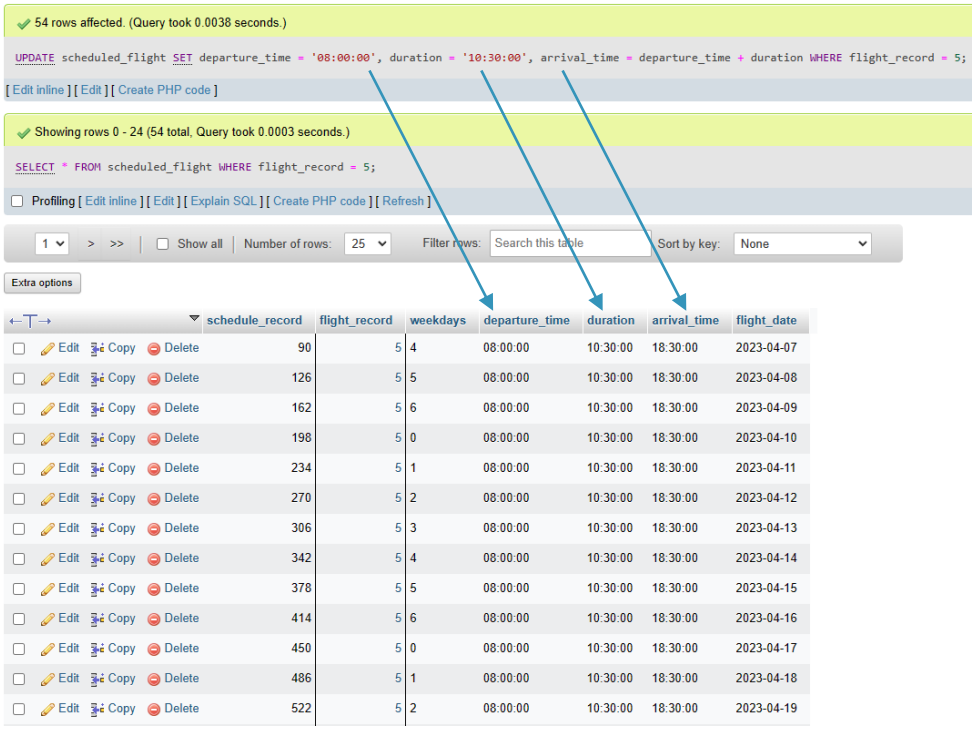
## Scheduled Days and Dates:

For simplification and time constraints for completion of this project, I will set all flights to run daily and generate a flight schedule for all dates/days from 2023-04-07 to 2023-05-31.

```
INSERT INTO scheduled_flights (flight_number, flight_date, weekdays)
SELECT flights.flight_number, date_list.flight_date, WEEKDAY(date_list.flight_date)
FROM (
  SELECT
    DATE_ADD('2023-04-07', INTERVAL days.number DAY) AS flight_date,
    WEEKDAY(DATE_ADD('2023-04-07', INTERVAL days.number DAY)) AS day_of_week
  FROM
    (
      SELECT 0 AS number UNION SELECT 1 UNION SELECT 2 UNION SELECT 3
      UNION SELECT 4 UNION SELECT 5 UNION SELECT 6
      UNION SELECT 7 UNION SELECT 8 UNION SELECT 9 UNION SELECT 10
      UNION SELECT 11 UNION SELECT 12 UNION SELECT 13 UNION SELECT 14
      UNION SELECT 15 UNION SELECT 16 UNION SELECT 17 UNION SELECT 18
      UNION SELECT 19 UNION SELECT 20 UNION SELECT 21 UNION SELECT 22
      UNION SELECT 23 UNION SELECT 24 UNION SELECT 25 UNION SELECT 26
      UNION SELECT 27 UNION SELECT 28 UNION SELECT 29 UNION SELECT 30
      UNION SELECT 31 UNION SELECT 32 UNION SELECT 33 UNION SELECT 34
      UNION SELECT 35 UNION SELECT 36 UNION SELECT 37 UNION SELECT 38
      UNION SELECT 39 UNION SELECT 40 UNION SELECT 41 UNION SELECT 42
      UNION SELECT 43 UNION SELECT 44 UNION SELECT 45 UNION SELECT 46
      UNION SELECT 47 UNION SELECT 48 UNION SELECT 49 UNION SELECT 50
      UNION SELECT 51 UNION SELECT 52 UNION SELECT 53
    ) AS days
  WHERE
    DATE_ADD('2023-04-07', INTERVAL days.number DAY) <= '2023-05-31'
) AS date_list
CROSS JOIN (
  SELECT DISTINCT flight_number FROM flights
) AS flights;
```

## Departure, Duration, and Arrival Times:

I will UPDATE the departure_time and duration for each flight into the scheduled_flight table. I will then calculate the arrival time using the departure_time + the duration to = arrival_time. For the purposes of this project all times will be UTC. No time conversions will be made for different time zones.




# Creating Customer Reservations

## Scenario

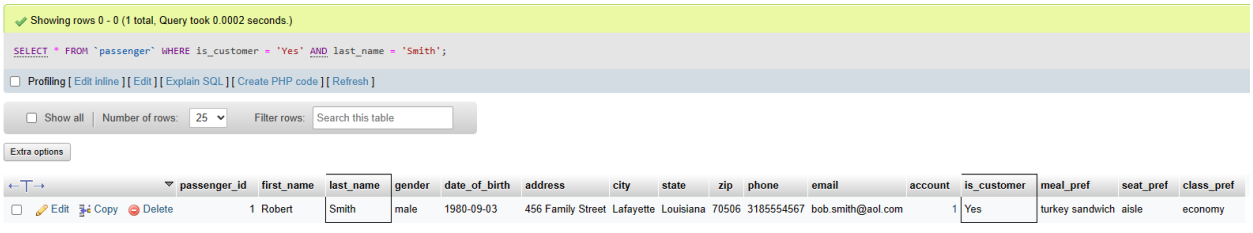
A customer has called requesting to book roundtrip flights from New Orleans to Orlando. His last name is Smith and he has booked with FlyRes before. He will be traveling with his wife Susan and their daughter Annie. They would like to depart between 4/20/23 and 4/23/23 and return 5 days later.

## Customer/Passenger Information Lookup

We need to confirm that Mr. Smith is a customer so that we can book the reservation and associate it with his existing account. We can search our passenger table as follows:

```
SELECT * FROM `passenger` WHERE is_customer = 'Yes' AND last_name = 'Smith'
```

We can search the passenger table using a SELECT statement for the ‘passenger’ table and specify that we are looking for instances where the passenger is a customer, and the last name is Smith.

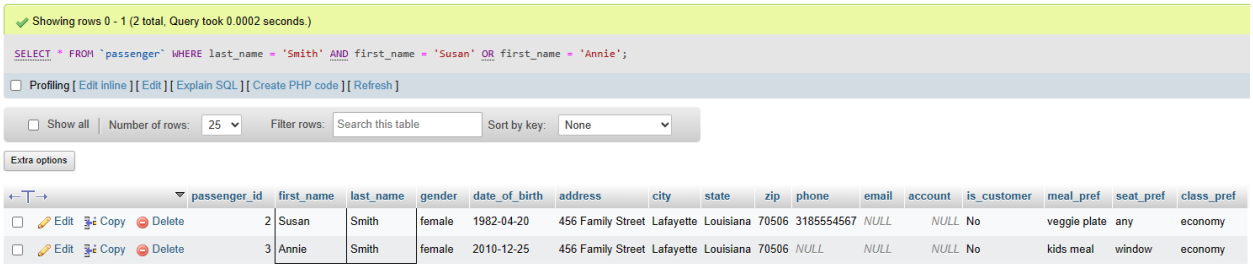



The results of our query show that we have one customer meeting this criterion. He is associated with account #1 and his passenger ID is 1. Now, let’s look at his family members so that we can make

reservations for them as well. We know their last name and first name. So, we will use this information to look them up in the passenger table.

```
SELECT * FROM `passenger` WHERE last_name = 'Smith' AND first_name = 'Susan' OR first_name = 'Annie';
```

We use the OR clause because we are looking for two different people. If we were to use the AND clause here the query would be looking for someone with two first names. We get the following result from the query:




| | passenger_id | first_name | last_name | gender | date_of_birth | address | city | state | zip | phone | email | account | is_customer | meal_pref | seat_pref | class_pref |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Edit Copy Delete | 2 | Susan | Smith | female | 1982-04-20 | 456 Family Street | Lafayette | Louisiana | 70506 | 3185554567 | NULL | NULL | No | veggie plate | any | economy |
| Edit Copy Delete | 3 | Annie | Smith | female | 2010-12-25 | 456 Family Street | Lafayette | Louisiana | 70506 | NULL | NULL | NULL | No | kids meal | window | economy |

We now know that we will be booking flights for Passenger IDs 1, 2, and 3. Robert Smith, Susan Smith, and Annie Smith. We can also see that their class preference is economy, their seat preference, and the meals they prefer, if available. All reservations will be booked under customer account # 1 which is associated with Robert Smith.

## Gathering Flight Information

Now that we have all the information we need about our passengers, we need to search for flights meeting their specified criteria. Flights from New Orleans to Orlando that depart between 4/20/23 and 4/23/23. Then return flights from Orlando to New Orleans 5 days after departure which would be between 4/25/23 and 4/28/23.

Let's assume that our customer representative is relatively new and can't remember the airport codes for New Orleans and Orlando. We will start with a simple query to acquire the airport codes first so that we can look up the necessary flight information.

```
SELECT *
FROM airport
WHERE city = 'New Orleans' OR city = 'Orlando';
```

| airport_id | airport_name | city | state | country |
|---|---|---|---|---|
| MCO | Orlando International Airport | Orlando | Florida | United States |
| MSY | Louis Armstrong New Orleans International Airport | New Orleans | Louisiana | United States |

Now we know our leg_origin and leg_destination airport codes are New Orleans = MSY and Orlando = MCO. We can perform a join on our flight table with our scheduled_flight table to look up flights from MSY to MCO and then return flights from MCO to MSY with our specified dates.

```
SELECT *
FROM flight
JOIN scheduled_flight ON flight.flight_record = scheduled_flight.flight_record
WHERE scheduled_flight.flight_date BETWEEN '2023-04-20' AND '2023-04-23'
AND flight.leg_origin = 'MSY' AND flight.leg_destination = 'MCO';
```

The result of the above query is as follows:

| flight_record | flight_number | airline | leg_origin | leg_destination | fc_seats | bc_seats | ec_seats | fc_fare | bc_fare | ec_fare | hidden_fare | schedule_record | flight_record | weekdays | departure_time | duration | arrival_time | flight_date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | AA001 | AA | MSY | MCO | 12 | 36 | 38 | 800.00 | 500.00 | 300.00 | 250.00 | 37 | 1 | 3 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-20 |
| 1 | AA001 | AA | MSY | MCO | 12 | 36 | 38 | 800.00 | 500.00 | 300.00 | 250.00 | 38 | 1 | 4 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-21 |
| 1 | AA001 | AA | MSY | MCO | 12 | 36 | 38 | 800.00 | 500.00 | 300.00 | 250.00 | 39 | 1 | 5 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-22 |
| 1 | AA001 | AA | MSY | MCO | 12 | 36 | 38 | 800.00 | 500.00 | 300.00 | 250.00 | 40 | 1 | 6 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-23 |

We can repeat the above query changing the dates, origin, and destination for the return flight 5 days later.

```
SELECT *
FROM flight
JOIN scheduled_flight ON flight.flight_record = scheduled_flight.flight_record
WHERE scheduled_flight.flight_date BETWEEN '2023-04-25' AND '2023-04-28'
AND flight.leg_origin = 'MCO' AND flight.leg_destination = 'MSY';
```

| flight_record | flight_number | airline | leg_origin | leg_destination | fc_seats | bc_seats | ec_seats | fc_fare | bc_fare | ec_fare | hidden_fare | schedule_record | flight_record | weekdays | departure_time | duration | arrival_time | flight_date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | AA002 | AA | MCO | MSY | 12 | 36 | 38 | 800.00 | 600.00 | 300.00 | 250.00 | 739 | 4 | 1 | 17:30:00 | 01:21:00 | 18:51:00 | 2023-04-25 |
| 4 | AA002 | AA | MCO | MSY | 12 | 36 | 38 | 800.00 | 600.00 | 300.00 | 250.00 | 775 | 4 | 2 | 17:30:00 | 01:21:00 | 18:51:00 | 2023-04-26 |
| 4 | AA002 | AA | MCO | MSY | 12 | 36 | 38 | 800.00 | 600.00 | 300.00 | 250.00 | 811 | 4 | 3 | 17:30:00 | 01:21:00 | 18:51:00 | 2023-04-27 |
| 4 | AA002 | AA | MCO | MSY | 12 | 36 | 38 | 800.00 | 600.00 | 300.00 | 250.00 | 847 | 4 | 4 | 17:30:00 | 01:21:00 | 18:51:00 | 2023-04-28 |

We could now either call our customer, Robert Smith, or email him the options. We will assume that Mr. Smith has responded and decided he would like to book the Friday flight out of New Orleans on 2023-04-21 and return 5 days later, on 2023-04-26. We will need to create the reservations for schedule_record 38 and 775 with economy class tickets.

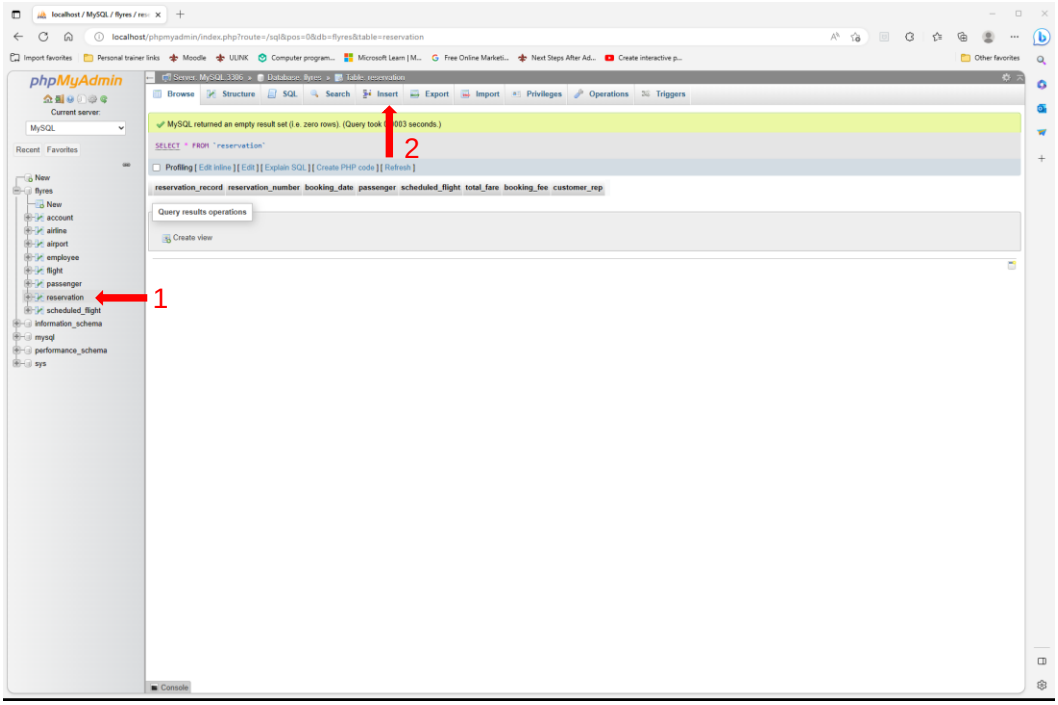
Based on the information from the queries above we know that they will be traveling on American Airlines flight AA001 from MSY to MCO. Each ticket will cost $300. The flight departs New Orleans at 06:00 is 1 hr and 21 mins and should arrive in Orlando at 07:21 on 4/21/23. The return flight is American Airlines Flight AA002 from MCO to MSY. Each economy ticket will cost $300. The flight departs Orlando at 17:30 (5:30pm) is 1 hr and 21 mins in duration and will arrive in New Orleans at 18:51 on 4/26/23. Now that we have all the necessary information, we can make reservations for the Smith family vacation.

## Creating Reservations

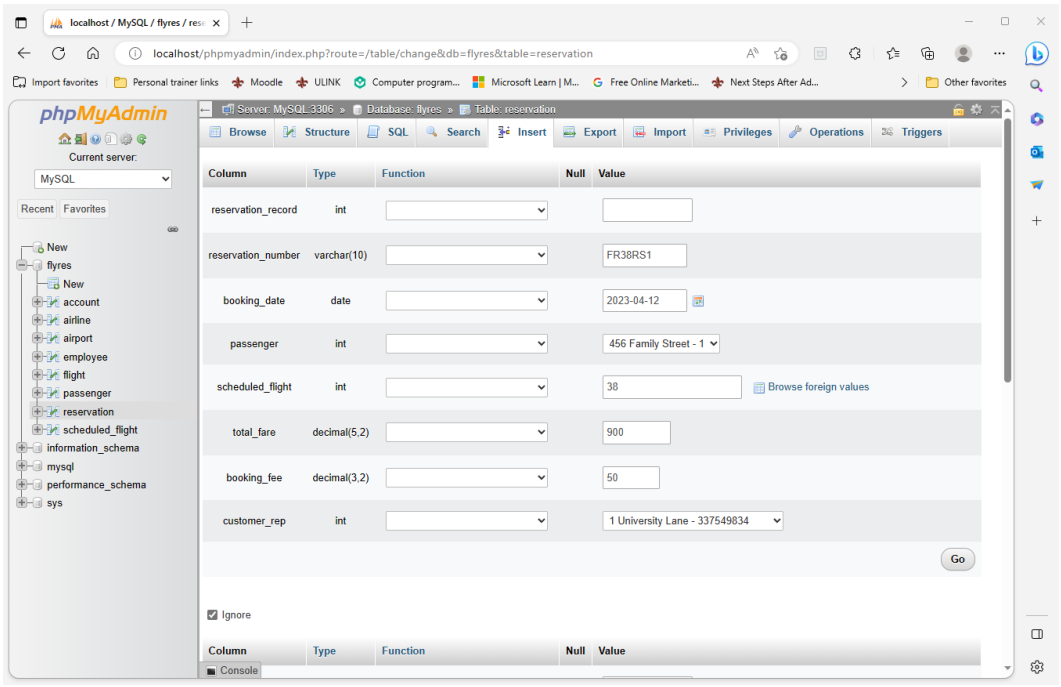
We can easily add reservations using the phpMyAdmin user interface like we did with adding new employees. We can also create reservations using INSERT. The instructions for each are as follows:

## phpMyAdmin GUI Reservation Creation

1. Select the reservation table from the left-hand menu.




2. Select the Insert tab.
3. Enter the required reservation information as follows.
    - Reservation_record: N/A this value is automatically populated.
    - Reservation_number: FR(flight_record)RS(Customer Initials)1(#of reservation) In this case FR38RS1 (longhand: flightrecord38robertsmith1)
    - Booking date: Select Current Date
    - Passenger: select the passenger_id # Robert Smith = 1
    - Scheduled_flight: schedule_record # identified earlier = 38
    - Total_fare = 3 economy tickets at 300 each = $900
    - Customer_Rep = select your employee ID
4. When finished entering information click ‘Go’.




5. We can now review the reservation by calling a simple SELECT * FROM reservation.

```
SELECT * FROM `reservation`
```

| reservation_record | reservation_number | booking_date | passenger | scheduled_flight | total_fare | booking_fee | customer_rep |
|---|---|---|---|---|---|---|---|
| 1 | FR38RS1 | 2023-04-12 | 1 | 38 | 900.00 | 50.00 | 337549834 |

## Reservation Creation using SQL

We will now create the reservation for the child Annie Smith (passenger_id = 3) for the initial scheduled_flight 38. There will not be a total_fare or booking fee as those values are only associated with the customer and not the passenger. All passengers for this trip are listed under the same reservation number FR38RS1.

```
INSERT INTO `reservation` (`reservation_record`, `reservation_number`, `booking_date`, `passenger`, `scheduled_flight`, `total_fare`, `booking_fee`,
`customer_rep`) VALUES (NULL, 'FR38RS1', '2023-04-12', '3', '38', '', '', '337549834');
```

This results in the reservation record for Annie Smith.

| reservation_record | reservation_number | booking_date | passenger | scheduled_flight | total_fare | booking_fee | customer_rep |
|---|---|---|---|---|---|---|---|
| 1 | FR38RS1 | 2023-04-12 | 1 | 38 | 900.00 | 50.00 | 337549834 |
| 2 | FR38RS1 | 2023-04-12 | 2 | 38 | 0.00 | 0.00 | 337549834 |
| 3 | FR38RS1 | 2023-04-12 | 3 | 38 | 0.00 | 0.00 | 337549834 |

## Altering a Table

After creating some reservations, I realized that I needed to add a column to calculate the revenue generated by each reservation. The calculation will be revenue = (fare*.15) + booking fee. All flights booked through FlyRes have a 15% up charge to generate revenue. The remainder is paid to the airline hosting the flight. The booking fee is 100% revenue. I will ALTER TABLE and ADD a column named revenue.

```
ALTER TABLE reservation
ADD revenue decimal(10,2);
```

| reservation_record | reservation_number | booking_date | passenger | scheduled_flight | fare | booking_fee | customer_rep | revenue |
|---|---|---|---|---|---|---|---|---|
| 1 | FR38RS1 | 2023-04-12 | 1 | 38 | 900.00 | 50.00 | 337549834 | NULL |
| 2 | FR38RS1 | 2023-04-12 | 2 | 38 | 0.00 | 0.00 | 337549834 | NULL |
| 3 | FR38RS1 | 2023-04-12 | 3 | 38 | 0.00 | 0.00 | 337549834 | NULL |
| 4 | FR775RS2 | 2023-04-12 | 1 | 775 | 900.00 | 50.00 | 337549834 | NULL |
| 5 | FR775RS2 | 2023-04-12 | 2 | 775 | 0.00 | 0.00 | 337549834 | NULL |
| 6 | FR775RS2 | 2023-04-12 | 3 | 775 | 0.00 | 0.00 | 337549834 | NULL |

Above is the reservation table with the new revenue column added. I now need to be able to calculate the revenue using the equation I defined above. In this scenario for reservation_record 1 and 4 we have (900*.15)+50 = 135+50 = 185.

```
UPDATE reservation SET revenue = (fare*.15)+booking_fee;
```

| reservation_record | reservation_number | booking_date | passenger | scheduled_flight | fare | booking_fee | customer_rep | revenue |
|---|---|---|---|---|---|---|---|---|
| 1 | FR38RS1 | 2023-04-12 | 1 | 38 | 900.00 | 50.00 | 337549834 | 185.00 |
| 2 | FR38RS1 | 2023-04-12 | 2 | 38 | 0.00 | 0.00 | 337549834 | 0.00 |
| 3 | FR38RS1 | 2023-04-12 | 3 | 38 | 0.00 | 0.00 | 337549834 | 0.00 |
| 4 | FR775RS2 | 2023-04-12 | 1 | 775 | 900.00 | 50.00 | 337549834 | 185.00 |
| 5 | FR775RS2 | 2023-04-12 | 2 | 775 | 0.00 | 0.00 | 337549834 | 0.00 |
| 6 | FR775RS2 | 2023-04-12 | 3 | 775 | 0.00 | 0.00 | 337549834 | 0.00 |

# Testing the Database

Now that I have demonstrated how the database was populated, I will test the database to ensure it functions by executing several functions.

## SELECT *

The SELECT * FROM <table> function selects all entries from the specified table. I will select all from each of the eight tables included in the FlyRes database system. Screenshots below may not show all records from each table, but a small selection of the first few rows in each table.

```
SELECT * FROM airline;
```

| airline_id | airline_name |
|---|---|
| AA | American Airlines |
| AF | Air France |
| BA | British Airways |
| DL | Delta Airlines |
| QF | Qantas Airways |
| SW | Southwest |
| UA | United Airlines |
| VS | Virgin Atlantic Airways |

```
SELECT * FROM airport;
```

| airport_id | airport_name | city | state | country |
|---|---|---|---|---|
| ABQ | Albuquerque International Sunport | Albuquerque | New Mexico | United States |
| ANC | Ted Stevens Anchorage International Airport | Anchorage | Alaska | United States |
| ATH | Athens International Airport | Athens | *NULL* | Greece |
| ATL | Hartsfield-Jackson Atlanta International Airport | Atlanta | Georgia | United States |
| BCN | Barcelona International Airport | Barcelona | *NULL* | Spain |
| BDL | Bradley International Airport | Hartford | Connecticut | United States |
| BHM | Birmingham-Shuttlesworth International Airport | Birmingham | Alabama | United States |

```
SELECT * FROM flight;
```

| flight_record | flight_number | airline | leg_origin | leg_destination | fc_seats | bc_seats | ec_seats | fc_fare | bc_fare | ec_fare | hidden_fare |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | AA001 | AA | MSY | MCO | 12 | 36 | 38 | 800.00 | 500.00 | 300.00 | 250.00 |
| 2 | AA001 | AA | MCO | MAD | 33 | 48 | 185 | 1000.00 | 800.00 | 600.00 | 500.00 |
| 3 | AA002 | AA | MAD | MCO | 33 | 48 | 185 | 1000.00 | 800.00 | 600.00 | 500.00 |
| 4 | AA002 | AA | MCO | MSY | 12 | 36 | 38 | 800.00 | 600.00 | 300.00 | 250.00 |
| 5 | AF001 | AF | LAX | LHR | 44 | 56 | 255 | 2000.00 | 1500.00 | 1000.00 | 850.00 |
| 6 | AF002 | AF | LHR | LAX | 44 | 56 | 255 | 2000.00 | 1500.00 | 1000.00 | 850.00 |
| 7 | AF003 | AF | SEA | LHR | 33 | 48 | 185 | 1800.00 | 1400.00 | 900.00 | 800.00 |
| 8 | AF004 | AF | LHR | SEA | 33 | 48 | 185 | 1800.00 | 1400.00 | 900.00 | 800.00 |
| 9 | DL001 | DL | SEA | ATL | 12 | 36 | 38 | 400.00 | 300.00 | 200.00 | 150.00 |

```
SELECT * FROM scheduled_flight;
```

| schedule_record | flight_record | weekdays | departure_time | duration | arrival_time | flight_date |
|---|---|---|---|---|---|---|
| 24 | 1 | 4 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-07 |
| 25 | 1 | 5 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-08 |
| 26 | 1 | 6 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-09 |
| 27 | 1 | 0 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-10 |
| 28 | 1 | 1 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-11 |
| 29 | 1 | 2 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-12 |
| 30 | 1 | 3 | 06:00:00 | 01:21:00 | 07:21:00 | 2023-04-13 |

```
SELECT * FROM passenger;
```

| passenger_id | first_name | last_name | gender | date_of_birth | address | city | state | zip | phone | email | account | is_customer | meal_pref | seat_pref | class_pref |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Robert | Smith | male | 1980-09-03 | 456 Family Street | Lafayette | Louisiana | 70506 | 3185554567 | bob.smith@aol.com | 1 | Yes | turkey sandwich | aisle | economy |
| 2 | Susan | Smith | female | 1982-04-20 | 456 Family Street | Lafayette | Louisiana | 70506 | 3185554567 | NULL | NULL | No | veggie plate | any | economy |
| 3 | Annie | Smith | female | 2010-12-25 | 456 Family Street | Lafayette | Louisiana | 70506 | NULL | NULL | NULL | No | kids meal | window | economy |
| 4 | Donald | Duck | male | 1976-03-03 | 100 Cartoon Lane | Hollywood | California | 90210 | 9095551000 | donnytheduck@quack.com | 2 | Yes | fruit plate | window | first |
| 5 | Tweety | Bird | other | 2020-07-04 | 424 Twitter Rd | Denver | Colorado | 80123 | 3037204242 | ogtwitter@aol.com | 3 | Yes | vegetarian | aisle | economy |
| 6 | Roger | Rabbit | male | 1959-09-21 | 8 Bunnyhop Lane | Seattle | Washington | 98101 | 2065551959 | r.rabbit@verizon.net | 4 | Yes | NULL | NULL | business |

```
SELECT * FROM employee;
```

| employee_id | first_name | last_name | address | city | state | zip | phone | email | hire_date | hourly_rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 111223333 | John | Wick | 337 Doggy Lane | Lafayette | Louisiana | 70506 | 3375551234 | john.wick@flyres.com | 2020-04-01 | 25.00 |
| 123456789 | Anita | Williams | 1234 Embarcadero St | Oakland | California | 94608 | 5105551234 | anita.williams@flyres.com | 2020-04-01 | 25.00 |
| 337549834 | Michael | Totaro | 1 University Lane | Lafayette | Louisiana | 70503 | 3375555540 | michael.totaro@flyres.com | 2020-01-01 | 30.00 |
| 459082345 | Wonder | Woman | 911 Hero St | Lafayette | Louisiana | 70506 | 3375559911 | wonder.woman@flyres.com | 2021-01-04 | 22.00 |
| 987654321 | Bob | Builder | 540 Construction Rd | Oakland | California | 94607 | 5105559876 | bob.builder@flyres.com | 2020-07-03 | 20.00 |

```
SELECT * FROM reservation;
```

| reservation_record | reservation_number | booking_date | passenger | scheduled_flight | fare | booking_fee | customer_rep | revenue |
|---|---|---|---|---|---|---|---|---|
| 1 | FR38RS1 | 2023-04-12 | 1 | 38 | 900.00 | 50.00 | 337549834 | 185.00 |
| 2 | FR38RS1 | 2023-04-12 | 2 | 38 | 0.00 | 0.00 | 337549834 | 0.00 |
| 3 | FR38RS1 | 2023-04-12 | 3 | 38 | 0.00 | 0.00 | 337549834 | 0.00 |
| 4 | FR775RS2 | 2023-04-12 | 1 | 775 | 900.00 | 50.00 | 337549834 | 185.00 |
| 5 | FR775RS2 | 2023-04-12 | 2 | 775 | 0.00 | 0.00 | 337549834 | 0.00 |
| 6 | FR775RS2 | 2023-04-12 | 3 | 775 | 0.00 | 0.00 | 337549834 | 0.00 |

```
SELECT * FROM account;
```

| account_id | owner | cc_number | open_date |
|---|---|---|---|
| 1 | 1 | 4347123443219876 | 2023-01-02 |
| 2 | 4 | 3456738249089831 | 2023-02-08 |
| 3 | 5 | 3902491247894487 | 2023-04-03 |
| 4 | 6 | 4347091234987401 | 2023-02-07 |

# JOINs

## Joining Two Tables

Joining two tables allows for seeing all the information pertaining to multiple tables based on the foreign key/primary key relationships. For example, with the FlyRes database, you can join the account table with the passenger table to see both the customer information and account information. This is a simple JOIN of two tables. The 'owner' variable of the account table is the 'passenger_id' of the passenger table. So, we will JOIN the two tables on these variables.

```
SELECT * FROM account JOIN passenger ON account.owner = passenger.passenger_id;
```

This query results in the following combined table:

| account_id | owner | cc_number | open_date | passenger_id | first_name | last_name | gender | date_of_birth | address | city | state | zip | phone | email | account | is_customer | meal_pref | seat_pref | class_pref |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 4347123443219876 | 2023-01-02 | 1 | Robert | Smith | male | 1980-09-03 | 456 Family Street | Lafayette | Louisiana | 70506 | 3185554567 | bob.smith@aol.com | 1 | Yes | turkey sandwich | aisle | economy |
| 2 | 4 | 3456738249089831 | 2023-02-08 | 4 | Donald | Duck | male | 1976-03-03 | 100 Cartoon Lane | Hollywood | California | 90210 | 9095551000 | donnytheduck@quack.com | 2 | Yes | fruit plate | window | first |
| 3 | 5 | 3902491247894487 | 2023-04-03 | 5 | Tweety | Bird | other | 2020-07-04 | 424 Twitter Rd | Denver | Colorado | 80123 | 3037204242 | ogtwitter@aol.com | 3 | Yes | vegetarian | aisle | economy |
| 4 | 6 | 4347091234987401 | 2023-02-07 | 6 | Roger | Rabbit | male | 1959-09-21 | 8 Bunnyhop Lane | Seattle | Washington | 98101 | 2065551959 | r.rabbit@verizon.net | 4 | Yes | NULL | NULL | business |

## Joining Three Tables

The reservation table is a junction table with multiple relationships to different tables. Let's say we wanted to see information from the reservation table, passenger table, and employee table all together. So, if we wanted to see a reservation, the passenger the reservation is for and the employee who created the reservation. We could JOIN three tables to get the relevant information as follows.

```
SELECT
    reservation.reservation_number,
    passenger.passenger_id,
    passenger.first_name,
    passenger.last_name,
    employee.employee_id,
    employee.first_name,
    employee.last_name
FROM passenger
JOIN reservation
ON passenger_id = reservation.passenger
JOIN employee
ON employee_id = reservation.customer_rep;
```

We get the following result with the desired information from three different tables.

| reservation_number | passenger_id | first_name | last_name | employee_id | first_name | last_name |
|---|---|---|---|---|---|---|
| FR38RS1 | 1 | Robert | Smith | 337549834 | Michael | Totaro |
| FR775RS2 | 1 | Robert | Smith | 337549834 | Michael | Totaro |
| FR38RS1 | 2 | Susan | Smith | 337549834 | Michael | Totaro |
| FR775RS2 | 2 | Susan | Smith | 337549834 | Michael | Totaro |
| FR38RS1 | 3 | Annie | Smith | 337549834 | Michael | Totaro |
| FR775RS2 | 3 | Annie | Smith | 337549834 | Michael | Totaro |
| FR414DD1 | 4 | Donald | Duck | 459082345 | Wonder | Woman |
| FR557DD2 | 4 | Donald | Duck | 459082345 | Wonder | Woman |
| FR650DD3 | 4 | Donald | Duck | 111223333 | John | Wick |
| FR662DD3 | 4 | Donald | Duck | 111223333 | John | Wick |
| FR841DD4 | 4 | Donald | Duck | 111223333 | John | Wick |
| FR865DD4 | 4 | Donald | Duck | 111223333 | John | Wick |
| FR827TB1 | 5 | Tweety | Bird | 337549834 | Michael | Totaro |
| FR936TB2 | 5 | Tweety | Bird | 337549834 | Michael | Totaro |
| FR334TB3 | 5 | Tweety | Bird | 337549834 | Michael | Totaro |
| FR1435TB4 | 5 | Tweety | Bird | 337549834 | Michael | Totaro |
| FR334RR1 | 6 | Roger | Rabbit | 987654321 | Bob | Builder |
| FR333RR2 | 6 | Roger | Rabbit | 987654321 | Bob | Builder |
| FR512RR3 | 6 | Roger | Rabbit | 987654321 | Bob | Builder |
| FR511RR4 | 6 | Roger | Rabbit | 987654321 | Bob | Builder |
| FR678RR5 | 6 | Roger | Rabbit | 987654321 | Bob | Builder |
| FR929RR6 | 6 | Roger | Rabbit | 987654321 | Bob | Builder |
| FR1092RR7 | 6 | Roger | Rabbit | 337549834 | Michael | Totaro |
| FR1343RR8 | 6 | Roger | Rabbit | 337549834 | Michael | Totaro |

## INSERT New Information

We can add new information by using the INSERT command. Let's create a new customer. It is important to know what information goes into adding a new customer. So, we look back at the SELECT * FROM passenger; to see the information in each passenger/customer instance.

```
SELECT * FROM `passenger`
```

| passenger_id | first_name | last_name | gender | date_of_birth | address | city | state | zip | phone | email | account | is_customer | meal_pref | seat_pref | class_pref |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Robert | Smith | male | 1980-09-03 | 456 Family Street | Lafayette | Louisiana | 70506 | 3185554567 | bob.smith@aol.com | 1 | Yes | turkey sandwich | aisle | economy |

By looking at an existing instance in our passenger table we learn what information is needed and the format to enter it. Now, we can add a new customer to the passenger table.

```
INSERT INTO `passenger` (`passenger_id`, `first_name`, `last_name`, `gender`, `date_of_birth`, `address`, `city`, `state`, `zip`, `phone`, `email`, `account`, `is_customer`, `meal_pref`, `seat_pref`, `class_pref`) VALUES (NULL, 'Payton',
'Wiscovitch', 'male', '1986-12-23', '1614 Campbell St, Apt 424', 'Oakland', 'California', '94607', '9099421495', 'C00122694@louisiana.edu', NULL, 'Yes', 'turkey sandwich', 'aisle', 'economy');
```

| passenger_id | first_name | last_name | gender | date_of_birth | address | city | state | zip | phone | email | account | is_customer | meal_pref | seat_pref | class_pref |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Robert | Smith | male | 1980-09-03 | 456 Family Street | Lafayette | Louisiana | 70506 | 3185554567 | bob.smith@aol.com | 1 | Yes | turkey sandwich | aisle | economy |
| 2 | Susan | Smith | female | 1982-04-20 | 456 Family Street | Lafayette | Louisiana | 70506 | 3185554567 | NULL | NULL | No | veggie plate | any | economy |
| 3 | Annie | Smith | female | 2010-12-25 | 456 Family Street | Lafayette | Louisiana | 70506 | NULL | NULL | NULL | No | kids meal | window | economy |
| 4 | Donald | Duck | male | 1976-03-03 | 100 Cartoon Lane | Hollywood | California | 90210 | 9095551000 | donnytheduck@quack.com | 2 | Yes | fruit plate | window | first |
| 5 | Tweety | Bird | other | 2020-07-04 | 424 Twitter Rd | Denver | Colorado | 80123 | 3037204242 | ogtwitter@aol.com | 3 | Yes | vegetarian | aisle | economy |
| 6 | Roger | Rabbit | male | 1959-09-21 | 8 Bunnyhop Lane | Seattle | Washington | 98101 | 2065551959 | r.rabbit@verizon.net | 4 | Yes | NULL | NULL | business |
| 7 | Payton | Wiscovitch | male | 1986-12-23 | 1614 Campbell St, Apt 424 | Oakland | California | 94607 | 9099421495 | C00122694@louisiana.edu | NULL | Yes | turkey sandwich | aisle | economy |

## UPDATE Existing Entries

There are times when information will need to be edited. For example, if a customer moves and we need to update their address information. We can easily do this using the UPDATE command. Using the entry just created for myself, Payton Wiscovitch, I will change the address information.

```
UPDATE `passenger` SET `address`='123 I Moved Street',`city`='San Francisco',`state`='California',`zip`='94117' WHERE passenger_id = '7'
```

| passenger_id | first_name | last_name | gender | date_of_birth | address | city | state | zip | phone | email | account | is_customer | meal_pref | seat_pref | class_pref |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Robert | Smith | male | 1980-09-03 | 456 Family Street | Lafayette | Louisiana | 70506 | 3185554567 | bob.smith@aol.com | 1 | Yes | turkey sandwich | aisle | economy |
| 2 | Susan | Smith | female | 1982-04-20 | 456 Family Street | Lafayette | Louisiana | 70506 | 3185554567 | NULL | NULL | No | veggie plate | any | economy |
| 3 | Annie | Smith | female | 2010-12-25 | 456 Family Street | Lafayette | Louisiana | 70506 | NULL | NULL | NULL | No | kids meal | window | economy |
| 4 | Donald | Duck | male | 1976-03-03 | 100 Cartoon Lane | Hollywood | California | 90210 | 9095551000 | donnytheduck@quack.com | 2 | Yes | fruit plate | window | first |
| 5 | Tweety | Bird | other | 2020-07-04 | 424 Twitter Rd | Denver | Colorado | 80123 | 3037204242 | ogtwitter@aol.com | 3 | Yes | vegetarian | aisle | economy |
| 6 | Roger | Rabbit | male | 1959-09-21 | 8 Bunnyhop Lane | Seattle | Washington | 98101 | 2065551959 | r.rabbit@verizon.net | 4 | Yes | NULL | NULL | business |
| 7 | Payton | Wiscovitch | male | 1986-12-23 | 123 I Moved Street | San Francisco | California | 94117 | 9099421495 | C00122694@louisiana.edu | NULL | Yes | turkey sandwich | aisle | economy |

Here, I use the conditional WHERE to direct my UPDATE of information to passenger_id = '7' to update the address for this specific instance.

## DELETE Unwanted Information

Removing unwanted information from a table can be done using the DELETE command. For instance, if we have an employee that leaves the company, we may want to remove that employee's information from the database. First, we need to look at the employee table to reference the entries. We are checking to see that the employee we want to remove exists and what information we can use to remove the instance from the employee table. Employee Anita Williams no longer works for FlyRes and we will be looking for the information needed to delete the employee record.

```
SELECT * FROM `employee`
```

| employee_id | first_name | last_name | address | city | state | zip | phone | email | hire_date | hourly_rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 111223333 | John | Wick | 337 Doggy Lane | Lafayette | Louisiana | 70506 | 3375551234 | john.wick@flyres.com | 2020-04-01 | 25.00 |
| 123456789 | Anita | Williams | 1234 Embarcadero St | Oakland | California | 94608 | 5105551234 | anita.williams@flyres.com | 2020-04-01 | 25.00 |
| 337549834 | Michael | Totaro | 1 University Lane | Lafayette | Louisiana | 70503 | 3375555540 | michael.totaro@flyres.com | 2020-01-01 | 30.00 |
| 459082345 | Wonder | Woman | 911 Hero St | Lafayette | Louisiana | 70506 | 3375559911 | wonder.woman@flyres.com | 2021-01-04 | 22.00 |
| 987654321 | Bob | Builder | 540 Construction Rd | Oakland | California | 94607 | 5105559876 | bob.builder@flyres.com | 2020-07-03 | 20.00 |

The best information to use to delete an entire row is the primary key as it is unique to that specific instance. In this case it is the employee_id for Anita Williams, 123456789.

```
DELETE FROM employee WHERE employee_id = '123456789';
```

| employee_id | first_name | last_name | address | city | state | zip | phone | email | hire_date | hourly_rate |
|---|---|---|---|---|---|---|---|---|---|---|
| 111223333 | John | Wick | 337 Doggy Lane | Lafayette | Louisiana | 70506 | 3375551234 | john.wick@flyres.com | 2020-04-01 | 25.00 |
| 337549834 | Michael | Totaro | 1 University Lane | Lafayette | Louisiana | 70503 | 3375555540 | michael.totaro@flyres.com | 2020-01-01 | 30.00 |
| 459082345 | Wonder | Woman | 911 Hero St | Lafayette | Louisiana | 70506 | 3375559911 | wonder.woman@flyres.com | 2021-01-04 | 22.00 |
| 987654321 | Bob | Builder | 540 Construction Rd | Oakland | California | 94607 | 5105559876 | bob.builder@flyres.com | 2020-07-03 | 20.00 |

The new employee table no longer has a record for Anita Williams.

# Project Conclusion

Throughout this project I learned a great deal about database design, implementation, and administration. I initially struggled with the design of this database as there are several relationships among the entities. My initial design was much more in depth and had at least double the number of tables. However, due to my inexperience and time frame for the project, I ultimately decided to simplify the database as much as possible. The result is what I have presented here. The basic design allowed me to explore the SQL language, learn how to implement my designs, set up relationships, and work with the data I created. Overall, I am pleased with the results and believe I have gained a solid foundation of SQL to build on and develop skills in the future. Given more time, I would have created a front-end webpage for the FlyRes business allowing for simple user interaction by customer representatives and customers to book flights.