Crash Airlines Data Analysis

Ian Rishell, Kyle Wear, Ethan Hardy

Tranq Consulting Group

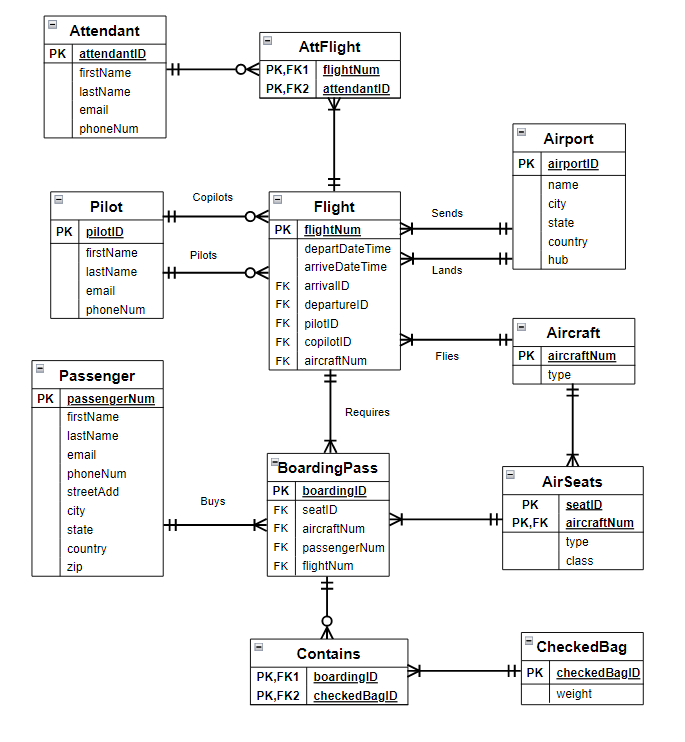
1. **Introduction - Overview**

The purpose of this report is to design and implement a relational database for Crash Airlines, and to help the management of Crash Airlines understand and utilize the database.

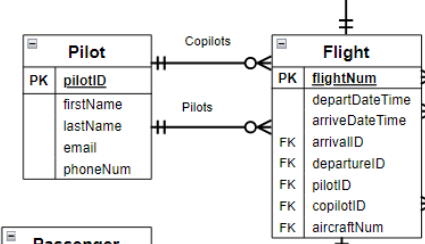
Crash Airlines is an American company that serves their customers by providing national and international flights at a competitive price. Crash is based in Miami, Florida, with hubs around the world. Crash focuses on providing the most reliable flight arrival time on the market, making Since their founding in 1991, they have worked to create a culture of efficiency and reliability in their employees. Crash Airlines has also consistently donated to international relief efforts following natural disasters.

The sequence of this report is as follows: In the Conceptual Model, we will show how we plan to organize Crash’s data using a drawing (Entity Relationship Diagram). We will explain the ERD and the relationships between data that it portrays. Next, in Database Construction and Population, we will show the data we took from Crash Airlines to demo our database, and explain the code we used to implement it. In Queries, we will demonstrate how Crash Airlines management could use our database to make business decisions using a few sample queries. In Data Visualization, we will take two of the queries mentioned above, and show how Crash Airlines managers could create useful visual aids from the database using Python. Finally, we will summarize our report in the Conclusion.

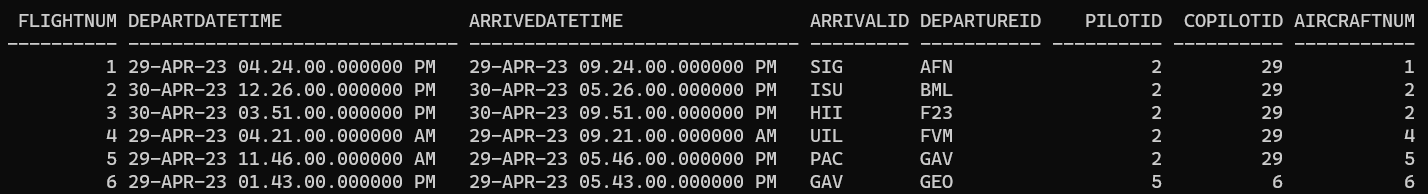
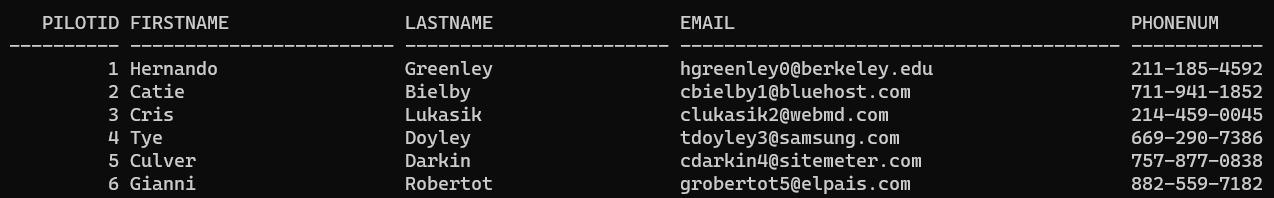
1. **Conceptual Model**



The purpose of the Entity Relationship Diagram is to give the viewer a visual depiction of the relationships between the data. We designed the ERD with the most integral data entities (the entities most widely related to other entities) in the center. To explain the rules of ERDs and the information they convey, we will focus on the relationship between the Flight and Pilot entities.



In ERDs, relationships with circles mean 0 and three lines mean many. From top down, the first relationship between Pilot and Flight shows that a Pilot copilots 0 to many flights. Some of Crash Airlines’ Pilots only pilot Flights and copilot 0 Flights, hence the circle. Additionally, some pilots employed by Crash Airlines copilot many Flights, hence the three lines. The same relationship applies to Pilots piloting Flights. Relationships with two vertical lines denote a mandatory one relationship. Moving from right to left on the topmost relationship between Flight and Pilot, the first vertical line shows that a flight must be copiloted by a pilot. The second vertical line shows that a flight can only be copiloted by 1 pilot. If the first vertical line was switched to a circle, the relationship would show that a Flight does not need a copilot, a Flight could be copiloted by 0 to 1 pilots. The “FKs” next to pilotID and copilotID in the Flight table show that the pilotID attribute of the Flight table will be linked to a pilotID in the Pilot table. This is how the data become related to each other in the construction of the database, which we will explain below.

1. **Database construction and population**

In our file where we constructed the tables, the flight table works by taking values from Airport, Aircraft, and Pilot as “foreign keys” which are ways to relate data from one table to the other using a singular value. You can see from the tables that the first flight, flightNum 1, is piloted by pilotID 2, and copiloted by copilotID 29. The flight table also uses the “primary key” flightNum to uniquely identify each row and pairs that primary key with the attributes departDateTime and arriveDateTime, and arrivalID, departureID, pilotID, copilotID, and aircraftNum which are all foreign keys.

In the following code, flightNum is established as the unique identifier by the code “PRIMARY KEY.” Additionally, pilotIDs from Pilot are linked to each flight by the code

“ REFERENCES Pilot(pilotID).” The same code is used to link pilotIDs from Pilot to the copilotID attribute of Flight. The code “NOT NULL” before the “REFERENCES” establishes that each Flight must have a pilot and a copilot.

CREATE TABLE Flight (

flightNum INTEGER PRIMARY KEY NOT NULL,

departDateTime TIMESTAMP NOT NULL,

arriveDateTime TIMESTAMP NOT NULL,

arrivalID VARCHAR(3) NOT NULL REFERENCES Airport(airportID),

departureID VARCHAR(3) NOT NULL REFERENCES Airport(airportID),

pilotID INTEGER NOT NULL REFERENCES Pilot(pilotID),

copilotID INTEGER NOT NULL REFERENCES Pilot(pilotID),

aircraftNum INTEGER NOT NULL REFERENCES Aircraft(aircraftNum));

CREATE TABLE Pilot(

pilotID INTEGER NOT NULL PRIMARY KEY,

firstName VARCHAR(24) NOT NULL,

lastName VARCHAR(24) NOT NULL,

email VARCHAR(40) NOT NULL,

phoneNum VARCHAR(12) NOT NULL);

**4. Queries**

**Query 1(S1) - Simple Join:** Counts the number of first-class seats on any given aircraft associated with Crash Airlines.

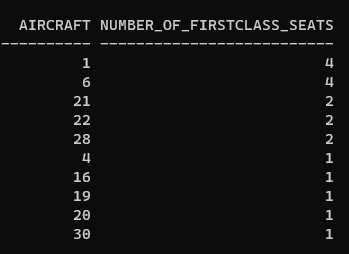
SELECT aircraftNum Aircraft, COUNT(seatID) Number\_Of\_Firstclass\_Seats

FROM AirSeats

WHERE UPPER(class) = 'FIRST CLASS'

GROUP BY aircraftNum

ORDER BY Number\_Of\_Firstclass\_Seats DESC, Aircraft;



With this query, you are able to see the distribution of first class seats of each aircraft with them. Using this query, you are able to allocate a certain aircraft if you were to have more first class passengers on a flight.

**Query 2(J2) - Join using 2 Tables:** Airports with more than 5 departing flights.

COLUMN Airport FORMAT A7

SELECT AP.airportID Airport, AP.country, AP.city, COUNT(F.flightNum) Num\_Flights

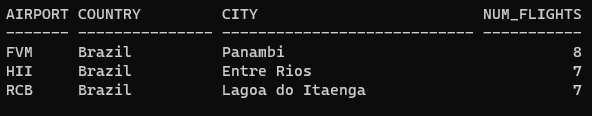
FROM Airport AP, Flight F

WHERE AP.airportID = F.arrivalID

HAVING COUNT(F.flightNum) > 5

GROUP BY AP.airportID, AP.country, AP.city

ORDER BY Num\_Flights DESC;



These airports have a higher number of flights compared to other airports and with this you will be able to determine that there needs to be a hub in Brazil.

**Query 3(J2) - Join using 2 Tables:** Lists pilots and copilots who have flown together 5 or more times.

SELECT P.pilotID Pilot\_ID, F.copilotID Copilot\_ID, COUNT(F.flightNum) Times\_Partnered

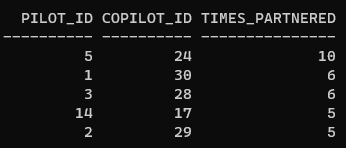
FROM Pilot P, Flight F

WHERE F.pilotID = P.pilotID

HAVING COUNT(F.flightNum) >= 5

GROUP BY P.pilotID, F.copilotID

ORDER BY Times\_Partnered DESC;



The importance of this query is to show the experience that pilots and copilots have together so that they should grouped together more.

**Query 4(J3) - Join using 3 or more Tables:** Lists the passengers flying out of the United States with bag weights heavier than 10lbs.

SELECT (P.firstName || ' ' || P.lastName) Passenger\_Name, CB.weight Bag\_Weight

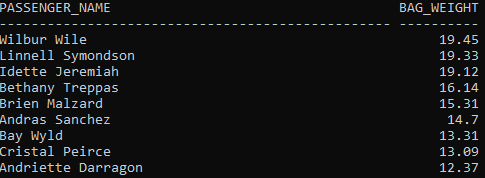
FROM Passenger P, BoardingPass BP, Contains C, CheckedBag CB

WHERE UPPER(P.country) = 'UNITED STATES' AND CB.weight > 10

AND P.passengerNum = BP.passengerNum AND BP.boardingID = C.boardingID

AND C.checkedBagID = CB.checkedBagID

ORDER BY CB.weight DESC;



Using this query, you may be able to charge more for heavier bags if you see the majority of customers bags over 10 pounds.

**Query 5(J3) - Join using 3 or more Tables:** Associates a pilot with an aircraft and shows the number of times that particular pilot has flown that aircraft.

SELECT P.lastName, AC.type, COUNT(F.flightNum) Num\_Flights

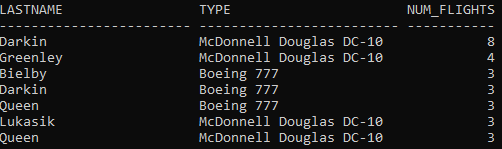
FROM Pilot P, Aircraft AC, Flight F

WHERE P.pilotID = F.pilotID AND F.aircraftNum = AC.aircraftNum

HAVING COUNT(F.flightNum) > 2

GROUP BY P.lastName, AC.type

ORDER BY Num\_Flights DESC, AC.type;



This query is important because it shows the familiarity that pilots have with certain aircrafts. Using this, you should place pilots in their aircraft of choice.

**Query 6(OJ) - Outer Join:** Lists the attendants that have not participated in any flights.

SELECT A.attendantID, (A.firstName || ' ' || A.lastName) Attendant\_Name

FROM Attendant A, AttFlight AF

WHERE A.attendantID = AF.attendantID(+)

HAVING COUNT(flightNum) = 0

GROUP BY A.attendantID, A.firstName, A.lastName;



This query is helpful because it shows which attendants are not going on any flights and that Crash Airlines should consider their employment status.

**Query 7(SUB) - Normal Subquery:** Shows the boarding passes that are associated with people flying in business class seats.

SELECT COUNT(boardingID) Business\_Boarding\_Passes

FROM BoardingPass

WHERE (seatID,aircraftNum) IN

(SELECT seatID, aircraftNum

FROM AirSeats

WHERE UPPER(class) LIKE 'B%')

ORDER BY Business\_Boarding\_Passes;



Using this query, you are able to distinguish the number of people flying in business class. This will be helpful because you can decide to add or remove seats if they are/ are not being bought.

**Query 8(NEST) - Nested Subquery:** Finds passengers riding first class aisle and center seats.

SELECT passengerNum passengerID, (firstName || ' ' || lastName)

Riding\_First\_Class\_Aisle\_Seats

FROM Passenger

WHERE passengerNum IN

(SELECT passengerNum

FROM BoardingPass

WHERE seatID IN

(SELECT seatID

FROM AirSeats

WHERE UPPER(class) LIKE 'F%' AND (UPPER(type) LIKE 'A%' OR UPPER(type) LIKE 'C%' )))

ORDER BY passengerID;



This query is helpful in making the decision to lower the prices for under-utilized first class seats. For instance, there are 23 people who bought first class window seats and 17 people combined bought first class aisle and center seats.

**Query 9(CSUB) - Correlated Subquery:** Shows the number passengers that are flying from each country.

SELECT country, COUNT(\*) AS Num\_Passengers\_Flying

FROM Passenger P1

WHERE EXISTS

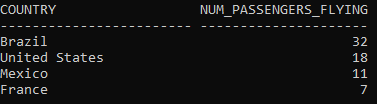
(SELECT \*

FROM Passenger P2

WHERE P1.country = P2.country AND P1.passengerNum != P2.passengerNum)

GROUP BY country

ORDER BY Num\_Passengers\_Flying DESC;



This query is important because you are able to decide which country would need accommodation over other countries that may not use Crash Airlines as often.

**Query 10(ASUB) - Alternative Subquery Structure:** Lists the Maximum number of passengers flying from a country

SELECT Country, Number\_Of\_Passengers Max\_Num\_Passengers

FROM

(SELECT country Country, COUNT(passengerNum) Number\_Of\_Passengers

FROM Passenger

GROUP BY country)

WHERE Number\_Of\_Passengers =

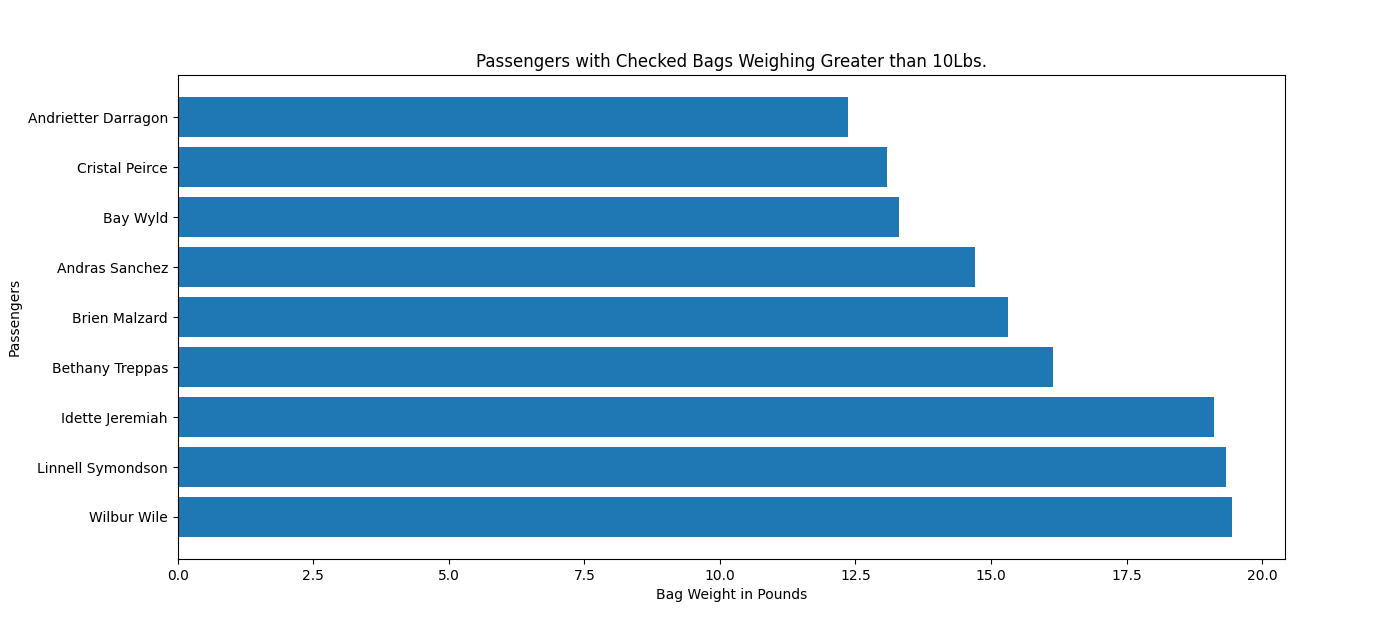
(SELECT MAX(COUNT(passengerNum))

FROM Passenger

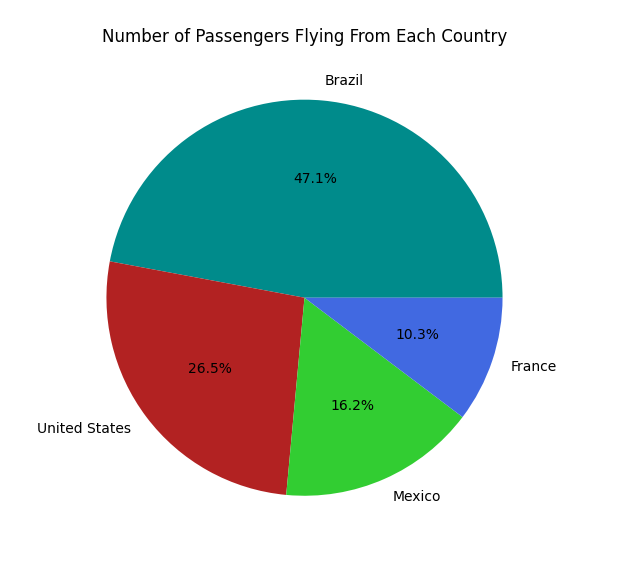
GROUP BY country);



This query ties into queries 2 and 9 to show that Brazil needs a hub because it has the most passengers flying out of it.

**6. Data Visualization** This bar chart represents passengers who have flown using your airline with checked luggage weighing greater than ten pounds. This bar chart shows the top nine passengers in terms of their bag weight. It was created using Python and implemented features from the Matplotlib library. We utilized a dictionary which is similar to a list or array, but it contains a collection of key-value pairs. The keys, in this case, are the passenger names which were derived from a CSV file. The values that were paired with each name were obtained from the same CSV file. In order to match these together, we utilized a for loop that went through each row of data, assigned the found values to temporary variables, and then added that pair to the dictionary. Finally, we used some Matplotlib functionality to form a bar chart displaying each pair. 

Knowing the weight of the luggage which your passengers check can be used to determine certain rules or policies that you may want to implement at Crash Airlines. For example, a nearly twenty pound bag may cause undue problems on your luggage system and could warrant having an additional fee charged to a customer. You could also consider implementing bag weight limits to prevent customers from attempting to check in luggage exceeding a certain limit.



The pie chart above breaks down your passengers who have flown from the countries you service over the past two days. We created it in a similar fashion to the bar chart with a couple key differences. Firstly, the labels for each slice were able to be hard coded in due to the likelihood of Crash Airlines beginning to service another country being low enough to not warrant a method to derive said countries from a datafile. The other primary difference is what functions of Matplotlib that we used. The pie chart methods were relatively simpler to implement than the bar chart with a lot of the customization being added in one line of code.

Crash Airlines can use this information on what countries its passengers fly from most to guide business decisions when it comes to things like expansion. Brazil is the most popular country to fly from by far, so an expansion of more airports that Crash Airlines flies from may be reasonable. On the opposite side is France with less than a quarter of passengers when compared to Brazil. To remedy this, Crash Airlines could consider raising marketing efforts or offering some sort of incentive to passengers in France.