

Introduction

The overall aim for this project was to reverse engineer a database system that would mirror aspects of Easyjet.com, with a particular focus on the storage of data required to make a flight booking. The database was designed firstly with an ERD, then implemented with SQL through PhpMyAdmin.

Due to time and resource constraints, I set out a specified scope of the areas of Easyjet.com I wanted to replicate in my database. As previously stated, my focus was creating a database which could facilitate the booking of flights. Areas of the EasyJet.com site such as travel insurance, car hire and hotel booking were therefore left out of scope.

Core Relationships in the Flight Booking System

The core relationships which make up the Easyjet.com booking system and are reflected in my database are.

1. A **customer** can make a **booking**
2. This **booking** can contain a **flight** and a **passenger** who is flying
3. The **passenger** can be attributed to a **seat** on the **flight**
4. The **flight** must take a specified **route**

At its absolute basic, this principle is shown in the below diagram, figure 1.

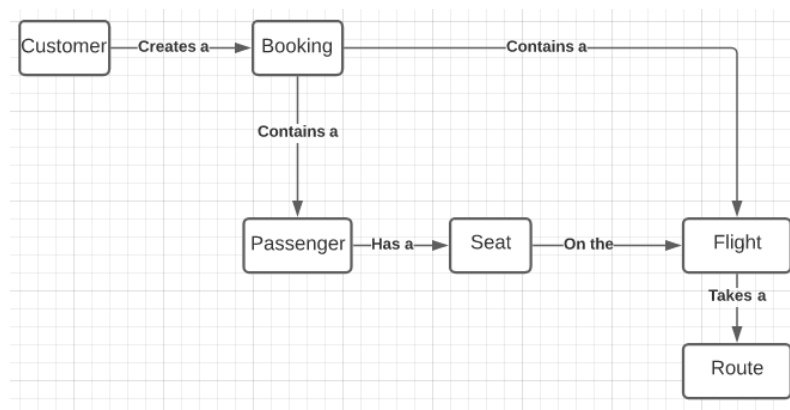


Figure 1 - Basic core relationships in a flight booking system

From the outset, this diagram shows the importance of both the **booking** and **flight** tables to the structure of the database.

The booking table is central as it works as an intermediary between the customer, the passenger, and the flight. Having a separate booking table allows for flexibility to replicate real life situations. For example, the customer might not necessarily be the passenger flying, they could also be one of multiple passengers on a particular booking. The customer could also have multiple flights on a single booking, such as return flight on a specific route.

The flight table links together the route with a schedule on which flights will run. The flight table also provides a standard price for the flight and allows the user to record the number of seats remaining.

Expanding on the Core Relationships

As part of the group-based portion of this project, we developed on these central ideas, adding further tables of importance, and including useful columns in which key data could be stored.

Figure 2 below shows some of the most important tables and columns conceived while working as a group. Many of these tables and columns have become crucial aspects of my final design. Our full ERD can be viewed in my appendix named *Figure 3*.

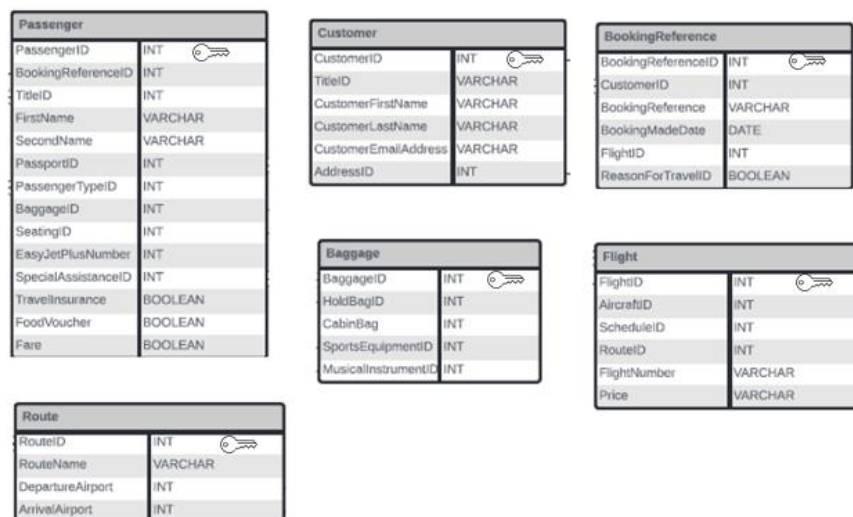


Figure 2 – Further key tables

As a group we decided that there needed to be a way to distinguish data entries in each table as there was likely to be repetition of data. We concluded the best way to do this was to create a column that would act as the primary key in each table. I have included key icons in *Figure 3* to highlight the columns we created for this purpose. We decided an int would be the best data type for these primary keys as it would be easy to reference in other tables that we wanted to link together.

In *Figure 3*, the chosen primary key for our *Customer* table was “*CustomerID*”. A column called “*CustomerID*” also exists in the *BookingReference* table to link these tables together through a foreign key constraint. It is an important feature of the database structure to be able to attribute the information held about a booking, to a particular customer.

The *Customer* table however does not contain a *BookingReference* column. The reason for this is that we wanted to create a database with the flexibility that a single customer could have many bookings. The link between these tables is therefore a one to many relationship.

The Final Design

Below is the final design of my database, taken from the designer view of PHPMyAdmin, titled *Figure 4*.

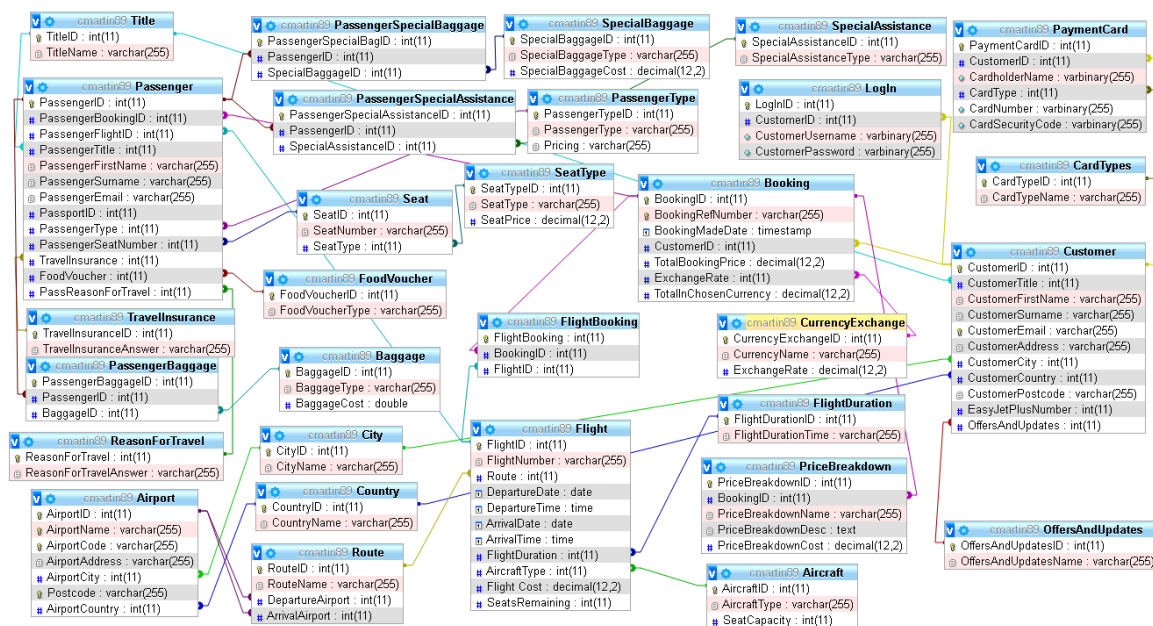


Figure 4 – Final database design

Customer

From the first very basic diagram in *Figure 1*, the idea of a *Customer* table has been a foundation for the architecture of this database. The front end of EasyJet.com allows a new user to enter their personal information and create an account through which they can book flights. I have mirrored this idea in my own database and have included a copy of my *Customer* table below in *Figure 5*.

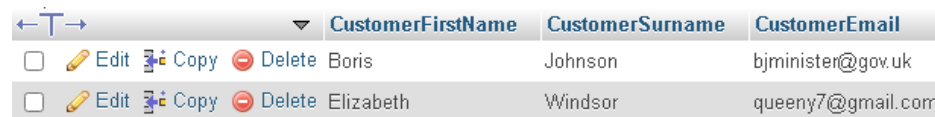
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	CustomerID	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	CustomerTitle	int(11)			No	None			Change Drop More
3	CustomerFirstName	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
4	CustomerSurname	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
5	CustomerEmail	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
6	CustomerPhoneNo	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
7	CustomerAddress	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
8	CustomerCity	int(11)			No	None			Change Drop More
9	CustomerCountry	int(11)			No	None			Change Drop More
10	CustomerPostcode	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
11	EasyJetPlusNumber	int(11)			Yes	NULL			Change Drop More
12	OffersAndUpdates	int(11)			No	None			Change Drop More

Figure 5 – Customer Table

My *Customer* table is made up of 12 columns, which when filled, gives an overview of the customer's basic personal information. I have used the *Unique* key in the *CustomerEmail* column to prevent a repetition of data and to avoid the confusion of having several customers with the same email address. Along with personal information, I have included an option for customers to choose whether they would like to be contacted with marketing information from the airline named "*OffersAndUpdates*".

I thought that it was important to include this marketing preference column as customers have the right to opt out of being contacted by a company.

The airline could use this “*OffersAndUpdates*” column through a process similar to *Query 1*. This query selects the first name, surname and email address of customers who live in London and have signed up to receive offers and updates from the airline. This information could then be used for targeted marketing, contacting relevant customers with deals on flights leaving from London based airports. *Figure 6* shows the results of this query.



	CustomerFirstName	CustomerSurname	CustomerEmail
<input type="checkbox"/> Edit Copy Delete	Boris	Johnson	bjminister@gov.uk
<input type="checkbox"/> Edit Copy Delete	Elizabeth	Windsor	queen7@gmail.com

Figure 6 – Results of Query 1

I chose to normalise the *OffersAndUpdates* column to a separate table also named *OffersAndUpdates*, as shown in *Figure 4*. I could have chosen a Boolean data type for this as there were only 2 options “Yes” or “No”, but decided normalisation gave flexibility for the options to be updated in the future. Example of options which the company may wish to also include could be “Yes, but only contact me by email” or “Yes, but only contact me by phone”.

Normalisation is something I chose to use extensively within my database. I found it to be the best option to control as much of the data entry as possible, to prevent repetition and human error.

In the *Customer* table shown in *Figure 5*, I chose to normalise *CustomerTitle*, *CustomerCity*, *CustomerCountry* and *OffersAndUpdates*. I was also able to reuse this normalised tables to input data to the *Passenger* and *Airport* tables.

I chose *CustomerID* as the primary key for the *Customer* table and used the *Auto Increment* feature of PHPMyAdmin to absolutely ensure each row had an individual reference and again, prevent the potential for human error.

CustomerID became an important linking point across my database, appearing in my tables *LogIn*, *PaymentCard* and *Booking* as a foreign key constraint. This allowed data entered into these tables to be attributed to a particular customer in the database.

LogIn and PaymentCard

My *LogIn* and *PaymentCard* tables contain customer information which I did not deem as appropriate to include in the more general *Customer* table. *LogIn* contains a username and password for each customer which would be used to access their account online.

PaymentCard contains bank card information for each customer which would be used to pay for services rendered.

Due to the sensitive nature of the data in these tables I decided it would be best to encrypt the majority of information entered using *AES Encrypt*. In order to view the encrypted data in a human readable form it is necessary to decrypt using the queries *Query 4* and *Query5*.

Figure 7 below shows both the encrypted and decrypted data in the *PaymentCard* table after using *Query 5*. The results of *Query 4* can be viewed in the appendix, titled *Figure 8*.

CardholderName	CardNumber	CardSecurityCode	AES_DECRYPT (CardholderName, 'mySecretKey')	AES_DECRYPT (CardNumber, 'mySecretKey')	AES_DECRYPT (CardSecurityCode, 'mySecretKey')
0xd7a60472f703a9edfe167586dddc39c	0xdb47e6cea3dd516a5990bcfce6deb419eade3faa6dcd8a9a...	0x11ec300fe59f3ccf6e2e67bb3981fc	RianWatson	8008 1546 4659 1546	596
0x96170c18fd1cbaf2ac692c464c5e7037	0xc7d27e08df2cc58ce4a0e5802fa4c547393870bc79b8c79...	0x0b7f6022b85ae9cc92f654a709246a96	PaulMW	1659 2516 1548 1265	594
0x7b6358cd185cb1bf0a174ee49fd2ffb	0x4c6b8c2e6f1b96a1cabe0635a9964524393870bc79b8c79...	0xa8f691afeec35c051b9e938016221020	BJMinister	4565 1256 9854 3265	456
0x8daee56c99770113193867b0d7928600	0x6cb000ccbcae871955a462a8bd41cd48316dc38f92d7b8f5...	0x007b062a38299db2d4cb78a25550149	ShivPL	4595 6532 1546 8974	125
0xdab9b261a3114d5f6ce3c7fe3186500f	0x985250336e5605c34f725ef326748b6b426ad8c4aa9202c...	0xafae8ba112dc35b8f140a54dcec59f04	AntoRHCP	5698 4587 4695 12365	175
0xfda4b303071915cdbc0ce1d1de99cfb28	0x6c3e8861ac5f8e0b1fbfd61c314ef9ff68eb783b695cc9...	0x89b30958492ad9a264f1e3d9c3f944f3	Prince	5698 4565 5989 4563	156
0xb7e29bb34cbac8ba5b494389f07b1190	0x665f3cbf73f537204032e49bdf11ecc8317b272358e6cc...	0x75b81cd854b470eb3292be4bb74e063	Tamelmpala	5698 4565 1254 5698	695
0x38a69f9274461d9392679f58f9f317	0x292c5962d985191fe45d90719fa0a09da8f1c3dcd2829d...	0x1a1df6264dab79b0bbe934e658aaf7aa	EWindsor	5645 7898 5623 6548	465
0xd49b55bdee29e7f59a304f081e1249ba	0x4c90ee67905db7b16a11c7f2d2b21cbb4856953c772b6ed1...	0x3bc81519e1696b1f1fa60c76c5c0b63	Mr Pirlo	6598 1254 2365 6598	659
0xfc2905565148ea3ae207bace15ac40af	0x96b316eb26eb9191999c5d11a6f4bbf34856953c772b6ed1...	0x1a1df6264dab79b0bbe934e658aaf7aa	Diego Maradona	6598 4565 1254 6598	465

Figure 7 – Decrypted data in the *LogIn* table

I did not deem *CardType* in the *LogIn* table to be sensitive enough to warrant encryption and instead normalised out the column to the *CardTypes* table, as displayed below in *Figure 9*.

CardTypeID	CardTypeName
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	1 Visa
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2 VisaDebit
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	3 Mastercard
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	4 AmericanExpress

Figure 9 – The *CardTypes* table

Booking

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
<input type="checkbox"/> 1	BookingID	int(11)			No	None		AUTO_INCREMENT
<input type="checkbox"/> 2	BookingRefNumber	varchar(255)	latin1_swedish_ci		No	None		
<input type="checkbox"/> 3	BookingMadeDate	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()
<input type="checkbox"/> 4	CustomerID	int(11)			No	None		
<input type="checkbox"/> 5	TotalBookingPrice	decimal(12,2)			Yes	NULL		
<input type="checkbox"/> 6	ExchangeRate	int(11)			Yes	NULL		
<input type="checkbox"/> 7	TotalInChosenCurrency	decimal(12,2)			Yes	NULL		

Figure 10 – The *Booking* table

Figure 10 above shows the *Booking* table. Much like the *Customer* table, I have used an autoincremented primary key to ensure there is a unique reference for each booking. In this case I have named the primary key *BookingID*. I have also included a booking reference number which could be used by the customer to retrieve information on their booking without giving out the primary key. This varchar values helps to disguise the inner workings of the database and prevent the customer from easily seeing how many bookings are held in the system. I have used a Unique key to ensure the booking reference numbers are exclusive to one booking only.

BookingID is used as a foreign key constraint in the *Passenger*, *FlightBooking* and *PriceBreakdown* tables. The *Booking* table also contains the primary key from the *Customer* table “*CustomerID*”. The use of these foreign key constraints allows the *Customer* table to be indirectly linked to *Passenger*, *FlightBooking* and *PriceBreakdown* without the restrictions of being directly linked.

In this design a customer can have multiple bookings, multiple flights on each booking and multiple passengers on each flight. I have tried to make the design as flexible as possible to prevent the need for restructuring the database should an unusual circumstance arise.

To demonstrate the *Booking* table’s linking capabilities ,I have used *Query 6* to select the first and surname of a specific customer, display all bookings under their name, the flights within these bookings and the passengers flying on each flight. I have ordered the results by the *BookingID*.

CustomerFirstName	CustomerSurname	BookingID ▲ 1	BookingRefNumber	PassengerFirstName	PassengerSurname	FlightNumber
Boris	Johnson	18	EJZ1015	Boris	Johnson	BELAUCK4587
Boris	Johnson	18	EJZ1015	Baby	Johnson	BELAUCK4587
Boris	Johnson	18	EJZ1015	Boris	Johnson	BELAUCK4587
Boris	Johnson	18	EJZ1015	Baby	Johnson	BELAUCK4587
Boris	Johnson	18	EJZ1015	Boris	Johnson	AUCKBEL6598
Boris	Johnson	18	EJZ1015	Baby	Johnson	AUCKBEL6598
Boris	Johnson	18	EJZ1015	Boris	Johnson	AUCKBEL6598
Boris	Johnson	18	EJZ1015	Baby	Johnson	AUCKBEL6598
Boris	Johnson	19	EJZ1016	Boris	Johnson	HEABEL6148
Boris	Johnson	19	EJZ1016	Boris	Johnson	HEABEL6148
Boris	Johnson	19	EJZ1016	Boris	Johnson	BELHEA5698
Boris	Johnson	19	EJZ1016	Boris	Johnson	BELHEA5698

Figure 11 – Results displayed from Query 6

The *Booking* table also contains a *BookingMadeDate* column, recording the date and time each customer made their booking. I have used the *Current Timestamp* feature to ensure the accuracy of data recorded here, this is a change from our group design where we chose the data type as *Date*. This change reduces the chance of human error and lessens the data input work involved. *Query 7* can be used to compile a list of customer bookings ordered by when they were made. The results are shown in *Figure 12* in the appendix.

Total Booking Price/ Price Breakdown

Figure 10 shows a *TotalBookingPrice* column in the *Booking* table. I have chosen a default *NULL* value as I wanted the option to be able to filter the data for this column from another table.

I have created a table specifically to house all the items purchased on a particular booking and the costs for these items. I have named this table *PriceBreakdown*.

PriceBreakdown includes a primary key *PriceBreakdownID*, *BookingID* as a foreign key constraint (so the items can be organised by booking) and columns to detail the name, description and cost of each item included in the booking. I have included the structure of this table in *Figure 13* in the appendix. An example of this table holding data from one booking can be viewed using *Query 8* with the results viewable in *Figure 14*.

To utilise this table to update *TotalBookingPrice*, I have used the SELECT SUM function to add all the costs relating to *BookingID* '3', saved this total as a variable and updated the *Booking* table. The result is shown below. I have recorded this query as *Query 9*.

BookingID	BookingRefNumber	BookingMadeDate	CustomerID	TotalBookingPrice	ExchangeRate	TotalInChosenCurrency
3	EZJ1001	2020-12-01 11:57:21	10	651.96	NULL	NULL

Figure 15 – Updated *TotalBookingPrice* for *BookingID* '3'

Taking inspiration from EasyJet.com, I thought it would be a good feature to be able to display this *TotalBookingPrice* in a currency of the customer's choosing.

To do this, I included some extra columns in my *Booking* table; *ExchangeRate* and *TotalInChosenCurrency*. I chose to normalise *ExchangeRate* to a separate table named *CurrencyExchange* which I have included as *Figure 16* in the appendix. This normalisation allows for the exchange rate to be updated when necessary and further currencies to be added with ease.

I decided to make the default value of these columns *NULL*, as mirroring EasyJet.com, I chose to keep the base currency of my database in GBP and use these columns as an optional extra for customers wanting to see their *TotalBookingPrice* in another currency.

I used *Query 10* to choose the EUR exchange rate, save it as a variable and multiply the *TotalBookingPrice* for *BookingID* 3, by this exchange rate, then update *TotalInChosenCurrency* to this converted amount. The updated row in my *Booking* table can be viewed in *Figure 17* below.

BookingID	BookingRefNumber	BookingMadeDate	CustomerID	TotalBookingPrice	ExchangeRate	TotalInChosenCurrency
3	EZJ1001	2020-12-01 12:50:56	10	651.96	2	730.20

Figure 17 – Updated *Booking* row for *BookingID* '3'

Passenger

Similar to the *Customer* table, the *Passenger* table contains personal information on each passenger who is taking a flight. The structure of this table is available in *Figure 18*.

PassengerID is used as the autoincrement primary key to ensure uniqueness and gives the ability for it to be referenced as a foreign key constraint in other tables and thus link them.

The *Passenger* table contains 8 foreign key constraints, these are used to normalise columns from this table out to 8 further tables. These foreign key constraints connect each passenger to a booking and a flight, which ultimately connects them to a customer and a route.

The passenger is given the option to enter their personal information, including picking a title from the normalised list. They also notify the airline whether they have travel insurance and their reason for travel, this information could you used by the airline to cross sell further products.

The passenger is asked to provide their type, whether they would like to pay extra for a better seat and whether they would like to buy a food voucher for the flight. Their decisions

in each of these, will potentially add items to *PriceBreakdown* table and change the total cost of the booking. I have included the options that can be picked from each of these table below.

PassengerTypeID	PassengerType	Pricing	FoodVoucherID	FoodVoucherType
1	Adult(16+)	Standard Flight Price	1	£5FoodVoucher
2	Child(2-15)	20% Discount	2	£10FoodVoucher
3	Infant(<2)	50% Discount	3	£20FoodVoucher

SeatTypeID	SeatType	SeatPrice
1	NoneSelected	0.00
2	Standard	5.00
3	UpFront	10.00
4	ExtraLegRoom	20.00

Figure 19 – Passenger options

The options for seating come from the *Seat* table, displayed in *Figure 20* in the appendix. This table is made up of a primary key, a given seat number and a seat type. *SeatType* is normalised to the *SeatType* table, which includes a cost for each type, as displayed in *Figure 19*.

Extra costs chosen in the passenger table can be added to the *PriceBreakdown* table, this can then be totalled and added to the *Booking* table, as shown in *Figure 15* with the use of *Query 9*.

Baggage

The *Passenger* table does not contain any columns in relation to baggage. It is structured in this way so as not to restrict a passenger to having only one bag. Instead I chose to create a baggage table, including the types and costs of baggage items, along with a primary key. This is shown in *Figure 21*.

To link the *Baggage* table with the *Passenger* table, I created a further table called *PassengerBaggage*. This table is made up its own primary key and the primary keys from *Baggage* and *Passenger* as foreign key constraints. This structure allows flexibility for each passenger to have many items of baggage linked to them. The JOIN query named *Query 11* can be used to demonstrate a single customer having multiple items of baggage.

PassengerFirstName	PassengerSurname	PassengerBaggageID	PassengerID	BaggageID
Rian	Watson	5	7	3
Rian	Watson	6	7	3

Figure 22 – Results from Query 11

I made a separate table for baggage which I deemed as “special”, included in this category are sports equipment and musical instruments. I differed from the group design here as we

had 2 separate tables, one for musical instruments and one for sports equipment. I decided that I thought it was cleaner to have these combined into one table.

Similar to the *Baggage* table, this *SpecialBaggage* table required an intermediary table between it and the *Passenger* table, to allow each passenger the flexibility to have multiple items of special baggage. I named this table *PassengerSpecialBaggage*.

Special Assistance

I chose to include an option for passengers to disclose any disabilities they might have and would potentially require assistance with when boarding the flight. The options I included can be seen in *Figure 23*.

I used the same method as my baggage tables to allow a passenger to disclose multiple areas where they may require assistance. The intermediary table in this case was named *PassengerSpecialAssistance*.

Flight

I have included the structure of my *Flight* table as *Figure 24* in the appendix. As with all the tables in this database, it includes a primary key using auto increment. The primary key is named *FlightID* in this case. I also included a *varchar FlightNumber* which could be used as a reference to each flight outside of the database. *FlightNumber* has a Unique key to prevent repetition.

The *Flight* table contains a *Route* column which has been normalised to a separate *Route* table, as I expected many flights to take the same route.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	RouteID	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	RouteName	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
3	DepartureAirport	int(11)			No	None			Change Drop More
4	ArrivalAirport	int(11)			No	None			Change Drop More

Figure 24 – Route table structure

The *Route* table contains a primary key, a name, a departure airport, and an arrival airport. The departure and arrival airport are further normalised to the *Airport* table, as this data was likely to be repeated.

In creating the *Flight* table, I differed from the group design by including the departure date/time and the arrival date/time of the flight as columns, as opposed to normalising this to a separate schedule table. I did this because I decided the use of *date* and *time* data types for these columns was sufficient enough to control the data entered without the need for normalising.

I chose to normalise the column *FlightDuration* as I expected some flights to take the same length of time. This table, also named *FlightDuration*, had a primary key and a flight duration time in hours and minutes. The primary key is then referenced in the *Flight* table when entering flight information.

I also normalised the *AircraftType* column, to control the data that was entered here. The *Aircraft* table contains a primary key, the aircraft type, and the seat capacity of that aircraft. This is useful as it allows the user to cross reference the amount of bookings on a flight, with the seat capacity of the aircraft used for that flight, and so determine the seats remaining. I have used *Query 12* to demonstrate how this would work. I firstly set a variable to the seat capacity of the aircraft used for the flight at *FlightID* '10'. I then set another variable, using the count feature, to total the number of passengers currently recorded against *FlightID* '10'. I used an update query to fill the *SeatsRemaining* column in the *Flight* table, to the total of the aircraft capacity variable, minus the number of passengers variable. I have included the resulting data below.

FlightID	FlightNumber	Route	DepartureDate	DepartureTime	ArrivalDate	ArrivalTime	FlightDuration	AircraftType	Flight Cost	SeatsRemaining
10	ROMED6251	16	2020-12-28	12:00:00	2020-12-28	14:00:00	3	2	80.00	542

Figure 25 – Results from Query 12

Conclusions and Potential Improvements

Given the limited time and resources, there is much in the database that could be improved going forward. For example, the security of the system could be greatly improved. As the sensitive data entered was fictitious, I used the same encryption key for every entry out of convenience. If the database contained real sensitive information, this encryption key would need to be more complicated and individually assigned to each piece of data. Ideally the encryption would need to take place on another program outside of the database for maximum security.

Another area that could be improved would be the lack of a programming language. In some of the queries, I was able to use variables to filter data from some parts of the database into others. It would not be feasible in a real-world situation to constantly carry out these queries to change data and would be better to be automated through programming.

A programming language could also be used to auto-generate the flight numbers and booking reference numbers so they would not need to be entered by the user. This would also ensure uniqueness.

Despite these drawbacks amongst others, my database largely meets the specification for a flight booking system replicating EasyJet.com. I have built it with such flexibility that, a customer can have many bookings, the bookings can contain many flights and many passengers, a particular route can contain many flights, and airports can have many flights departing and arriving at them. The use of normalisation and foreign key constraints has been greatly affective in insuring this. The use of foreign key constraints also reduces data repetition and potential for human error.

Appendix

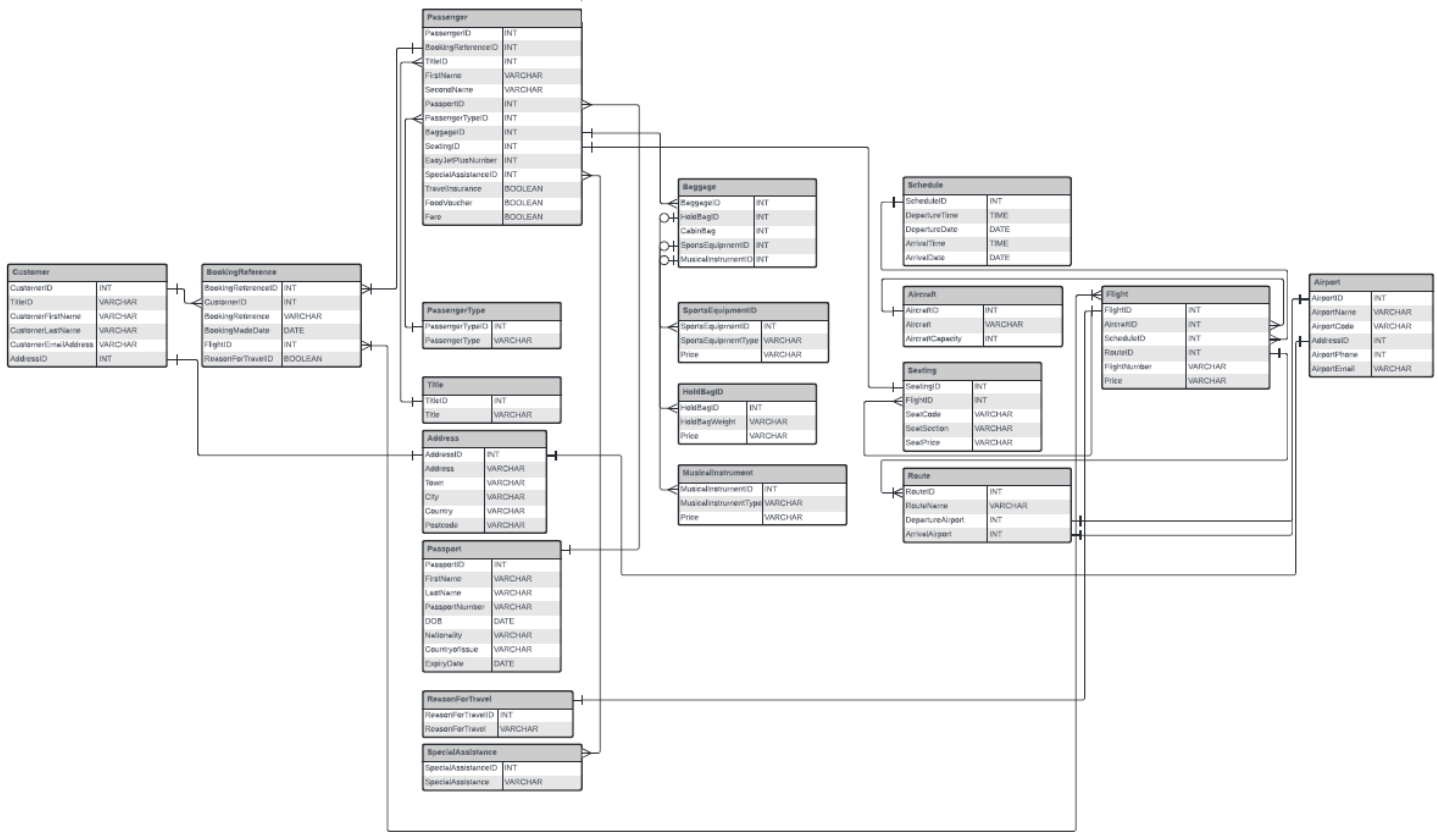


Figure 3 – Final group ERD

CustomerUsername	CustomerPassword	AES_DECRYPT (CustomerUsername, 'mySecretKey')	AES_DECRYPT (CustomerPassword, 'mySecretKey')
0x43b5cc1e0be39d8b6aac8dea69568f01	0x4a0f09f6bcb6b582bad3bf524903586d	watsy93	celtic93
0x9378974cf9e61cbc861e94be84b14130	0x4162d0c2218dabfb48be3df5176128bf	drpaul	lovescience
0x4406370f5d928442a679ecccbb0d0a290	0x2491b0d34327b59ce4bb1bb212a51f74	pmbj	number10
0x263d8d27e2bba9c19bcb140cc82e1ba0	0xd81fc163f82fe029e35c66e11fad33d	ShivL	surfergirl92
0xf1e867689ea9a432edf0ee7452a9036b	0x63c812c126643d72189cfd96ab8995f8	AntoK	underthebridge
0xfda4b303071915cdb0ce1d1de99cfb28	0xfcaf482fa73648741ea42cf1c7669643	Prince	Kiss
0xa11c531170afaa8633e44da9692f48a7	0x9050e45986c58ac9e6099fdd02d586d6	TameKev	slowrush
0x1c8851261702250d00973e4a5c166a42	0xe40b8a6e5dcd6d80f9bbe23dbc42cd9ab766daf9d486817...	DaQueen	Saxe-Coburg-Gotha
0xad14a066b1e73d80d3bc8651340a26c9	0x7b5a16a86a8fee57b5149fe104ef163e	APirlo	Midfielder
0x79bd82b712b1a7668cc269a11e010d38	0x73c9d132a72549977fb46f103c5e90c9	Maradona10	handofgod
0xfeb6b78079d824d794b9113ef8df6a26	0xd007b412be59ca489344577917677eb3	MitchelwithoneL	haggis

Figure 8 – Decrypted data from the Login table

CustomerFirstName	CustomerSurname	CustomerID	BookingRefNumber	BookingMadeDate	▲ 1
Andreas	Pirlo	10	EJZ1001	2020-11-25 21:41:00	
Rian	Watson	2	EJZ1003	2020-11-25 21:41:58	
Anthony	Kiedas	6	EJZ1005	2020-11-25 21:48:46	
Elizabeth	Windsor	9	EJZ1007	2020-11-25 21:49:51	
Kevin	Parker	8	EJZ1009	2020-11-25 21:50:49	
Kevin	Parker	8	EJZ1010	2020-11-25 21:50:49	
TheArtistFormallyKnowAs	Prince	7	EJZ1011	2020-11-25 21:51:56	
Siobhan	Lindores	5	EJZ1013	2020-11-25 21:54:38	
Boris	Johnson	4	EJZ1015	2020-11-25 21:56:04	
Boris	Johnson	4	EJZ1016	2020-11-25 21:56:04	
Paul	Wilson	3	EJZ1017	2020-11-25 21:57:58	
Paul	Wilson	3	EJZ1018	2020-11-25 21:57:58	

Figure 12 – Results from Query 7

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	PriceBreakdownID	int(11)			No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/> 2	BookingID	int(11)			No	None			Change Drop More
<input type="checkbox"/> 3	PriceBreakdownName	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 4	PriceBreakdownDesc	text	latin1_swedish_ci		No				Change Drop More
<input type="checkbox"/> 5	PriceBreakdownCost	decimal(12,2)			No	None			Change Drop More

Figure 13 – PriceBreakdown table

PriceBreakdownID	BookingID	PriceBreakdownName	PriceBreakdownDesc	PriceBreakdownCost
1	3	Passenger Andreas Flight ROMED6251	Adult flight from Rome to Edinburgh charged at sta...	80.00
2	3	Passenger Susan Flight ROMED6251	Adult flight from Rome to Edinburgh charged at sta...	80.00
3	3	Passenger Andreas seat A1 chosen flight ROMED6251	Extra leg room	20.00
4	3	Passenger Susan seat A2 chosen flight ROMED6251	Upfront	10.00
5	3	Passenger Andreas hold bag chosen flight ROMED6251	15kg hold bag	17.99
6	3	Passenger Susan hold bag chosen flight ROMED6251	15kg hold bag	17.99
7	3	Passenger Andreas special baggage chosen flight RO...	Hold golf bag	70.00
8	3	Passenger Andreas food voucher chosen flight ROMED...	£20 food voucher to be spent on flight	20.00
9	3	Passenger Andreas food voucher chosen flight ROMED...	£20 food voucher to be spent on flight	20.00
10	3	Passenger Andreas Flight EDROM6594	Adult flight from Edinburgh to Rome charged at sta...	70.00
11	3	Passenger Susan Flight EDROM6594	Adult flight from Edinburgh to Rome charged at sta...	70.00
12	3	Passenger Andreas seat A1 chosen flight EDROM6594	Extra leg room	20.00
13	3	Passenger Susan seat A2 chosen flight EDROM6594	Upfront	10.00
14	3	Passenger Andreas hold bag chosen flight EDROM6594	15kg hold bag	17.99
15	3	Passenger Susan hold bag chosen flight EDROM6594	15kg hold bag	17.99
16	3	Passenger Andreas special baggage chosen flight ED...	Hold golf bag	70.00
17	3	Passenger Andreas food voucher chosen flight EDROM...	£20 food voucher to be spent on flight	20.00
18	3	Passenger Susan food voucher chosen flight EDROM65...	£20 food voucher to be spent on flight	20.00

Figure 14 – Results from Query 8

<div><div><div><div><div></div><div></div></div><div></div><div></div></div><div></div></div></div>				CurrencyExchangeID	CurrencyName	ExchangeRate
<div><div><div></div></div></div>	<div><div><div></div></div><div>Edit</div></div>	<div><div><div></div></div><div>Copy</div></div>	<div><div><div></div></div><div>Delete</div></div>	1	GBP	1.00
<div><div><div></div></div></div>	<div><div><div></div></div><div>Edit</div></div>	<div><div><div></div></div><div>Copy</div></div>	<div><div><div></div></div><div>Delete</div></div>	2	EUR	1.12
<div><div><div></div></div></div>	<div><div><div></div></div><div>Edit</div></div>	<div><div><div></div></div><div>Copy</div></div>	<div><div><div></div></div><div>Delete</div></div>	3	USD	1.34
<div><div><div></div></div></div>	<div><div><div></div></div><div>Edit</div></div>	<div><div><div></div></div><div>Copy</div></div>	<div><div><div></div></div><div>Delete</div></div>	4	AUD	1.82
<div><div><div></div></div></div>	<div><div><div></div></div><div>Edit</div></div>	<div><div><div></div></div><div>Copy</div></div>	<div><div><div></div></div><div>Delete</div></div>	5	Bitcoin	0.07

Figure 16 – The CurrencyExchange table




















































#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	PassengerID 	int(11)			No	None		AUTO_INCREMENT	 Change  Drop  More
2	PassengerBookingID 	int(11)			No	None			 Change  Drop  More
3	PassengerFlightID 	int(11)			No	None			 Change  Drop  More
4	PassengerTitle 	int(11)			No	None			 Change  Drop  More
5	PassengerFirstName	varchar(255)	latin1_swedish_ci		No	None			 Change  Drop  More
6	PassengerSurname	varchar(255)	latin1_swedish_ci		No	None			 Change  Drop  More
7	PassengerEmail	varchar(255)	latin1_swedish_ci		No	None			 Change  Drop  More
8	PassengerPhoneNo	varchar(255)	latin1_swedish_ci		No	None			 Change  Drop  More
9	PassportID	int(11)			No	None			 Change  Drop  More
10	PassengerType 	int(11)			No	None			 Change  Drop  More
11	PassengerSeatNumber 	int(11)			Yes	NULL			 Change  Drop  More
12	TravellInsurance 	int(11)			No	None			 Change  Drop  More
13	FoodVoucher 	int(11)			Yes	NULL			 Change  Drop  More
14	PassReasonForTravel 	int(11)			No	None			 Change  Drop  More

Figure 18 – The Passenger table












#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	SeatID 	int(11)			No	None		AUTO_INCREMENT	 Change  Drop  More
2	SeatNumber	varchar(255)	latin1_swedish_ci		No	None			 Change  Drop  More
3	SeatType 	int(11)			No	None			 Change  Drop  More

Figure 20 – Seat table structure











Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
BaggageID 	int(11)			No	None		AUTO_INCREMENT	 Change  Drop  More
BaggageType	varchar(255)	latin1_swedish_ci		No	None			 Change  Drop  More
BaggageCost	double			No	None			 Change  Drop  More

Figure 21 – Baggage table structure

SpecialAssistanceID	SpecialAssistanceType
1	WheelchairUser
2	Blind
3	Deaf
4	Epileptic

Figure 23 – Special Assistance data

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	FlightID	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	FlightNumber	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
3	Route	int(11)			No	None			Change Drop More
4	DepartureDate	date			No	None			Change Drop More
5	DepartureTime	time			No	None			Change Drop More
6	ArrivalDate	date			No	None			Change Drop More
7	ArrivalTime	time			No	None			Change Drop More
8	FlightDuration	int(11)			No	None			Change Drop More
9	AircraftType	int(11)			No	None			Change Drop More
10	Flight Cost	decimal(12,2)			No	None			Change Drop More
11	SeatsRemaining	int(11)			No	None			Change Drop More

Figure 24 – Flight table structure

Report Queries

Query 1 - SELECT `CustomerFirstName`, `CustomerSurname`, `CustomerEmail` FROM `Customer`
WHERE `CustomerCity` = 3 AND `OffersAndUpdates` = 1

Query 2 - SELECT `CustomerUsername`, `CustomerPassword` FROM `LogIn` WHERE 1

Query 3 - SELECT `CardholderName`, `CardNumber`, `CardSecurityCode` FROM `PaymentCard`
WHERE 1

Query 4 - SELECT

`CustomerUsername`, `CustomerPassword`,
AES_DECRYPT (CustomerUsername, 'mySecretKey'),
AES_DECRYPT (CustomerPassword, 'mySecretKey')
FROM LogIn WHERE 1

Query 5 - SELECT

`CardholderName`, `CardNumber`, `CardSecurityCode`,
AES_DECRYPT (CardholderName, 'mySecretKey'),
AES_DECRYPT (CardNumber, 'mySecretKey'),
AES_DECRYPT (CardSecurityCode, 'mySecretKey')
FROM PaymentCard WHERE 1

Query 6 –

SELECT Customer.CustomerFirstName, Customer.CustomerSurname, Booking.BookingID,
Booking.BookingRefNumber, Passenger.PassengerFirstName, Passenger.PassengerSurname,
Flight.FlightNumber FROM Customer JOIN Booking ON Customer.CustomerID = Booking.CustomerID
JOIN Passenger ON Booking.BookingID = Passenger.PassengerBookingID JOIN FlightBooking ON
Booking.BookingID = FlightBooking.BookingID JOIN Flight ON FlightBooking.FlightID = Flight.FlightID
WHERE Customer.CustomerID =4

ORDER BY Booking.BookingID

Query 7 - SELECT Customer.CustomerFirstName, Customer.CustomerSurname, Booking.CustomerID,
Booking.BookingRefNumber, Booking.BookingMadeDate FROM Booking JOIN Customer ON
Booking.CustomerID = Customer.CustomerID WHERE 1

ORDER BY Booking.BookingMadeDate

Query 8 - SELECT * FROM `PriceBreakdown` WHERE `BookingID` = 3

Query 9 - SET @pirloTotal = (SELECT SUM(PriceBreakdownCost)FROM PriceBreakdown WHERE BookingID = 3);

UPDATE Booking SET TotalBookingPrice=@pirloTotal WHERE BookingID = 3;

Query 10 - SET @eurExchange = (SELECT ExchangeRate FROM CurrencyExchange WHERE CurrencyExchangeID = 2);

UPDATE Booking SET ExchangeRate = 2 WHERE BookingID =3;

UPDATE Booking SET `TotalInChosenCurrency` = @eurExchange * `TotalBookingPrice` WHERE BookingID = 3

Query 11 - SELECT Passenger.PassengerFirstName, Passenger.PassengerSurname, PassengerBaggage.PassengerBaggageID, PassengerBaggage.PassengerID, PassengerBaggage.BaggageID FROM PassengerBaggage JOIN Passenger ON PassengerBaggage.PassengerID = Passenger.PassengerID WHERE Passenger.PassengerID = 7

Query 12 - SET @flight10Cap=(SELECT `SeatCapacity` FROM `Aircraft` WHERE `AircraftID` = 2);

SET @flight10Pass=(SELECT COUNT(PassengerID) FROM `Passenger` WHERE PassengerFlightID = 10);

UPDATE Flight SET SeatsRemaining= @flight10Cap - @flight10Pass WHERE FlightID = 10

Queries Used in video

Display Customer

SELECT * FROM `Customer` WHERE `CustomerID` = 15

Encrypted Login

INSERT INTO Login (CustomerID, CustomerUsername, CustomerPassword)

VALUES (

'15',

AES_ENCRYPT('cmartin89', 'mySecretKey'),

AES_ENCRYPT('databases yo', 'mySecretKey')

);

Decrypted Login

```
SELECT
`CustomerUsername`,`CustomerPassword`,
AES_DECRYPT (CustomerUsername, 'mySecretKey'),
AES_DECRYPT (CustomerPassword, 'mySecretKey')
FROM Login WHERE `CustomerID` = 15
```

Make Booking

```
INSERT INTO `Booking` (`BookingID`, `BookingRefNumber`, `BookingMadeDate`, `CustomerID`,
`TotalBookingPrice`, `ExchangeRate`, `TotalInChosenCurrency`) VALUES (NULL, 'EJZ1119',
current_timestamp(), '15', NULL, NULL, NULL);
```

Attach Flights to Booking

```
INSERT INTO `FlightBooking` (`FlightBooking`, `BookingID`, `FlightID`) VALUES (NULL, '26', '19'),
(NULL, '26', '20');
```

Creating Passengers

```
INSERT INTO `Passenger` (`PassengerID`, `PassengerBookingID`, `PassengerFlightID`, `PassengerTitle`,
`PassengerFirstName`, `PassengerSurname`, `PassengerEmail`, `PassengerPhoneNo`, `PassportID`,
`PassengerType`, `PassengerSeatNumber`, `TravelInsurance`, `FoodVoucher`, `PassReasonForTravel`)
VALUES (NULL, '26', '19', '2', 'Curtis', 'Martin', 'cmartin89@qub.ac.uk', '07712345678', '65984', '1',
'2', '2', '2', '2'), (NULL, '26', '19', '1', 'Siobhan', 'Lindores', 'sioblindores@gmail.com', '0754989596',
'659874', '1', '3', '2', '2', '2'), (NULL, '26', '20', '2', 'Curtis', 'Martin', 'cmartin89@qub.ac.uk',
'07712345678', '65984', '1', '7', '2', NULL, '2'), (NULL, '26', '20', '1', 'Siobhan', 'Lindores',
'sioblindores@gmail.com', '0754989596', '659874', '1', '8', '2', NULL, '2');
```

Adding Passenger Baggage

```
INSERT INTO `PassengerBaggage` (`PassengerBaggageID`, `PassengerID`, `BaggageID`) VALUES (NULL,
'43', '2'), (NULL, '44', '3'), (NULL, '45', '2'), (NULL, '46', '3');
```

Special Baggage – Surfboard

```
INSERT INTO `PassengerSpecialBaggage` (`PassengerSpecialBagID`, `PassengerID`,
`SpecialBaggageID`) VALUES (NULL, '44', '2'), (NULL, '46', '2');
```

Adding Price Breakdown

```
INSERT INTO `PriceBreakdown` (`PriceBreakdownID`, `BookingID`, `PriceBreakdownName`,  
`PriceBreakdownDesc`, `PriceBreakdownCost`) VALUES (NULL, '26', 'Curtis, Siobhan BELSYD3254', '2x  
Adult Standard Flight Price Belfast to Sydney ', '1400.00'), (NULL, '26', 'Chosen Seats A1, B1  
BELSYD3254', '2x Extra leg room seats ', '40.00'), (NULL, '24', 'Food vouchers BELSYD3254', '2x £10  
food vouchers ', '20.00'), (NULL, '26', 'Curtis baggage BELSYD3254', '1x 15KG Hold Bag', '17.99'),  
(NULL, '26', 'Siobhan baggage BELSYD3254', '1x 23kg Hold Bag', '21.49'), (NULL, '26', 'Special  
Baggage', '1x Surfboard', '100'), (NULL, '26', 'Curtis, Siobhan SYDBEL9827', '2x Adult Standard Flight  
Price Sydney to Belfast', '1400.00'), (NULL, '26', 'Chosen Seats A2, B2 SYDBEL9827', '2x Upfront  
Seats', '20.00'), (NULL, '26', 'Curtis Baggage SYDBEL9827', '1x 15kg Hold Bag', '17.99'), (NULL, '26',  
'Siobhan Baggage SYDBEL9827', '1x 23kg Hold Bag', '21.49'), (NULL, '26', 'Special Baggage  
SYDBEL9827', '1x Surfboard ', '100.00');
```

View Price Breakdown

```
SELECT * FROM `PriceBreakdown` WHERE `BookingID`=26
```

Add Total Breakdown Cost

```
SET @curtisTotal = (SELECT SUM(PriceBreakdownCost)FROM PriceBreakdown WHERE BookingID =  
26);
```

```
UPDATE Booking SET TotalBookingPrice=@curtisTotal WHERE BookingID = 26;
```

Show Total Breakdown Cost

```
SELECT * FROM `Booking` WHERE `BookingID`=26
```