Introduction

The goal of this project is to create a relational database that supports an online flight booking system based on the EasyJet model. The design structure has been developed through reverse engineering the flight booking service on easyjet.com. The design of the database has been implemented using MySQL and has been populated with sample data to demonstrate how the database functions, and the relationships between entities. The core elements of flight booking that the database has been designed to support, have been identified through the entity discovery exercise carried out at the initial stage of the project.

The real life flight booking system for EasyJet is certainly a significantly large system overall. Therefore, in line with the project specifications I am focusing on replicating aspects of the EasyJet model that are most significant to flight booking. Any aspect of EasyJet's online flight booking service that involves external offerings such as insurance, and car hiring from third party organisations is not relevant to this project and is therefore out of scope.

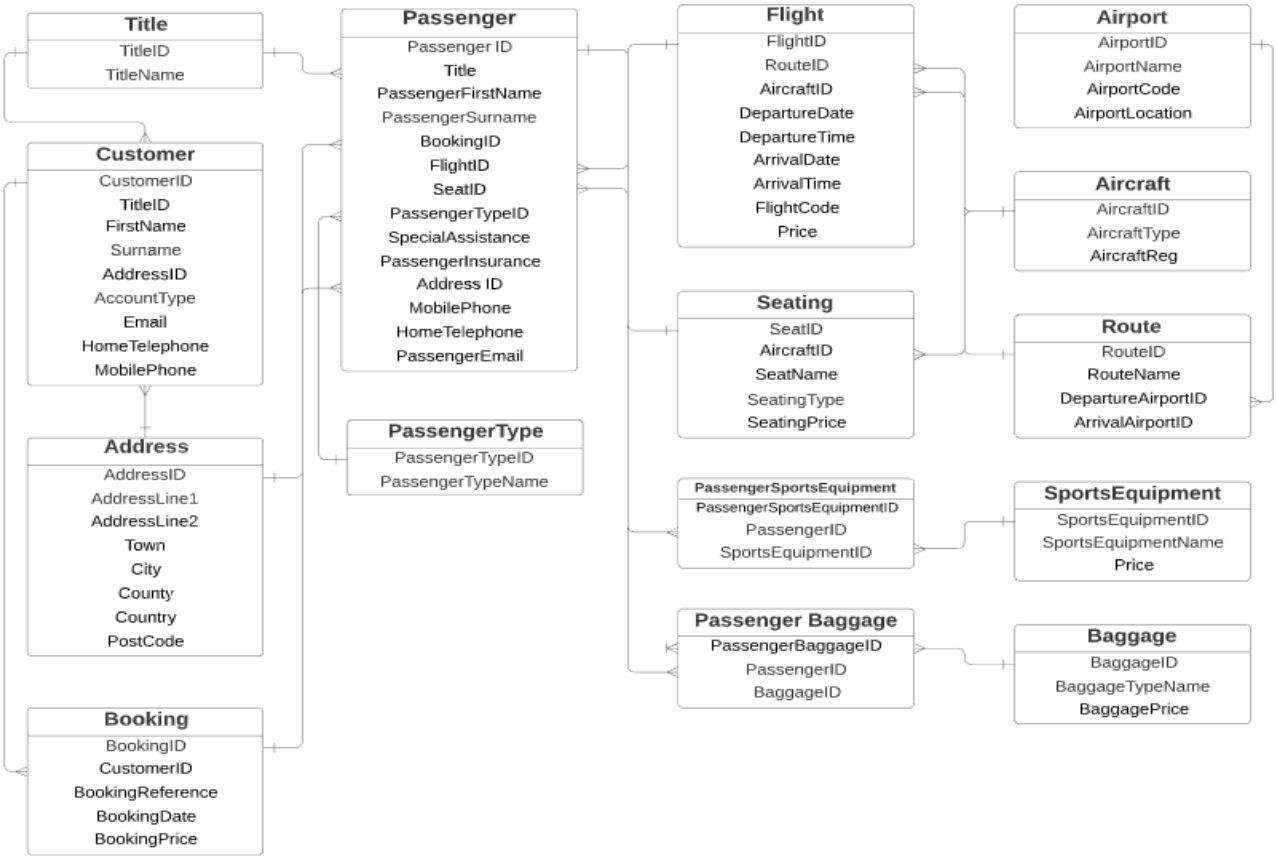
Entity discovery

A flight booking system requires airports and aircrafts, while routes exist between airports and flights are instances of these routes. Customers can purchase tickets to flights using their preferred payment method, book in their luggage and make their seat selections for these flights. These are the core elements to a flight booking system.

A database that supports a flight booking system must store; customer details, booking and passenger details, along with passport and payment information. Flights and the routes they are operating on, seat selection and passenger luggage, along with the aircraft and airports being used for journeys must also be stored. The real life relationships between these entities need to be reflected in the design structure of the database.

Through using EasyJet's online flight booking service in the entity discovery stage of the project, I constructed an ER diagram that describes relationships between entities in a flight booking system. As groups we compared ER diagrams and then independently constructed a further ER diagram to use as a base point for the design of the database. The Initial ER diagram is limited in its design but describes key entities in a flight booking system and the relationships between them.

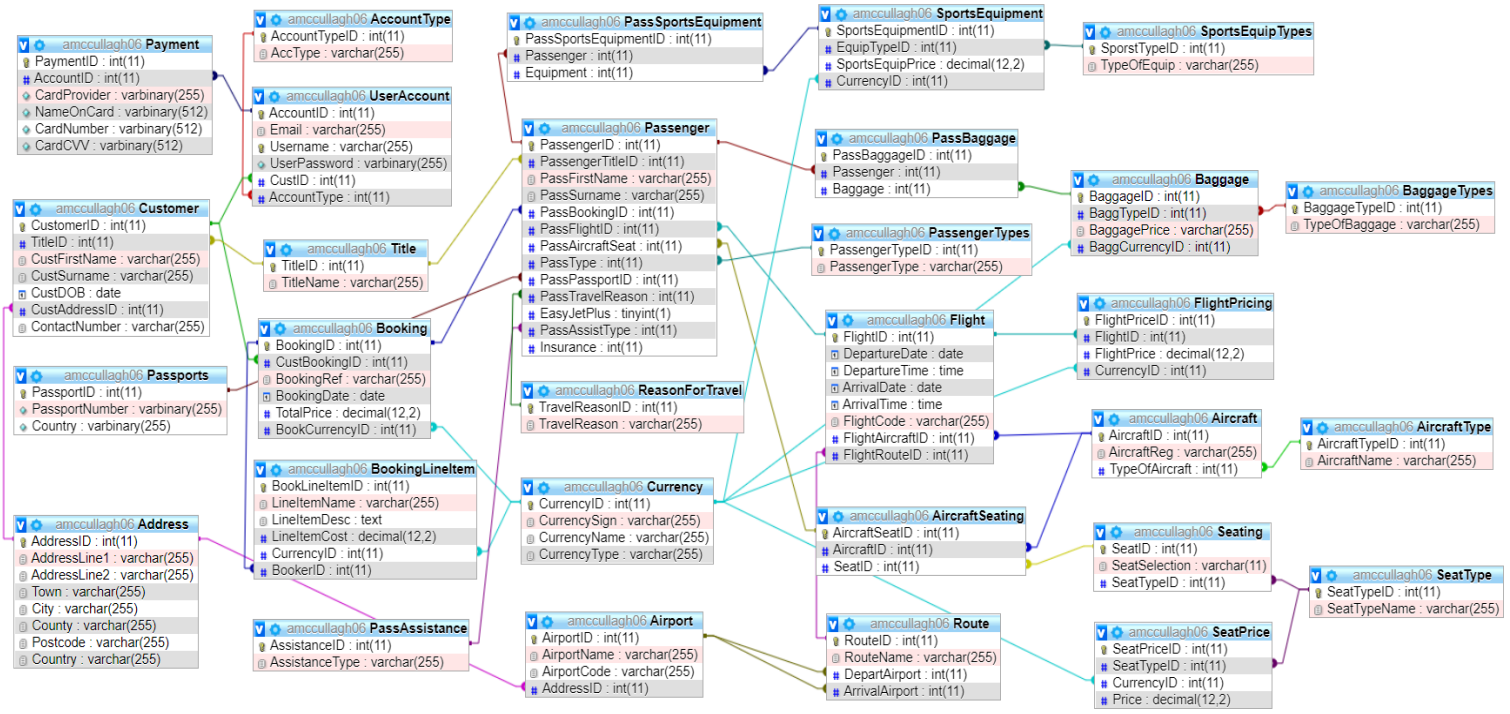
Figure 1 shows the ER diagram constructed from the initial entity discovery exercise.



Final Design

During the construction of the database, I used the initial ER diagram as a starting point. I made amendments and expansions to the initial design plans in order to design a database that best meets the requirements of a real life flight booking system.

Figure 2 shows the ER diagram that displays the final design of the database.



Primary Keys

Primary Keys are essential in the construction of a relational database as they enable the unique identification of data. A unique identifier in a database is used to discriminate a particular row of data from all the other rows of data in the same table, and throughout the entire database. In the relational database model, a table cannot contain duplicate rows, as this would create ambiguities in data retrievals.

I implemented a Primary Key for all tables in the database, with the data type set to “Integer”, whilst also using the “A_I” (Auto Increment) function. The AI function enables the assignment of a particular attribute within the table to the role of Primary Key, using an auto incremented value for each row. Therefore, every table in the database is in charge of its own uniqueness, and any values added over time will be unique.

Foreign Key Constraints

In order to create relationships between entities in the database I used Foreign Key constraints to link data in different tables. Foreign Key constraints enable an attribute in one table to refer to a Primary Key in another table, and are therefore crucial when establishing relationships between entities as they act as a cross reference. The data entered into an attribute that has a Foreign Key constraint comes directly from the Primary Key of the table its represents. Through establishing connections between tables we can use ‘JOIN Queries’ to fetch data.

Throughout this report, I will discuss some of the Primary Keys and describe the Foreign Key constraints between tables in the database. I will discuss the design decisions I have made and justify the reasons for these decisions based on the core elements of flight booking, conclusions from entity discovery and assumptions I have made about flight booking systems. The design overview will include a discussion of the airline flight-based elements, followed by a discussion of the customer-based elements. A description of the design structure which connects the core elements of flight booking will be provided. I have used SELECT queries and JOIN queries to demonstrate the relationships between tables in the database. I will describe how the database functions to support an airline company and solve problems.

This section of the report details how the database stores: aircrafts, airports, routes, flights, and seating. I will describe how these core elements of a flight booking system are connected by the design structure of the database, providing an overview of the implementation of each table and a justification for the design decisions.

Aircrafts

The Primary Key in the **Aircraft** table is *AircraftID*, which is linked to the **Flight** table as a Foreign Key constraint. A one-to-many relationship is established to reflect the fact that one aircraft can be used for many flights. The *AircraftReg* attribute stores the aircraft registration which has been set to 'unique', as this will ensure it can only appear once in the table. The *TypeOfAircraft* attribute in the **Aircraft** table is a Foreign Key constraint linking to the Primary Key in the **AircraftType** table. Aircraft types have been normalised as there a multiple aircrafts of the same type on EasyJet's fleet, for example, there are several Airbus A320-200's. A one-to-many relationship between the **AircraftType** and **Aircraft** tables is established**.**

AircraftID	AircraftReg	AircraftName
1	G-EZAB	Airbus A319-100
2	G-EZAC	Airbus A319-100
3	G-EZAF	Airbus A319-100
4	G-EZAG	Airbus A320-200
5	G-EZGX	Airbus A320-200
6	G-EZGY	Airbus A320-200
7	G-EZOA	Airbus A320neo
8	G-EZOF	Airbus A320neo
9	G-UZHD	Airbus A320neo
10	G-UZHE	Boeing 737-200
11	G-UZMC	Boeing 737-200
12	G-UZHX	Boeing 757-200

*Figure 3 shows the results from a query that searches for aircrafts, using the relationship between the Aircraft and AircraftTypes tables. (**Query 1**)*

Airports

The Primary Key in the **Airport** table is *AirportID*, which is linked to the **Route** table as a Foreign Key constraint. The *AirportName*, *AirportCode* and *AddressID*, store all relevant details to describe an airport in a flight booking system. The *AddressID* constraint is linked to the Primary Key in the **Address** table, where addresses have been normalised. The normalisation of addresses helps keep data consistent and accurate, as all address entries will follow the same format.

Routes

The Primary Key in the **Route** table is *RouteID*, which is linked to the **Flight** table as a Foreign Key constraint. It is necessary to create a one-to-many relationship between the **Route** table and the **Flight** table, as many flights can travel the same route. The **Route** table contains a description of the route under the *RouteName* attribute. The Foreign Key constraints in the **Route** table are *DepartAirport* and *ArrivalAirport* that link to the Primary Key in the **Airport** table. The one-to-many relationship between the **Airport** table and **Route** allows routes to be linked to their departure and arrival airports, while the same airports are used for many routes.

Flights

The Primary Key in the **Flight** table is *FlightID*, which is linked to the **Passenger** and **FlightPricing** tables through Foreign Key constraints. The **Flight** table effectively stores important details on all flights. The *DepartureDate*, and *ArrivalDate* attributes are setup with the data type of 'date', while the *DepartureTime* and *ArrivalTime* attributes are setup with the data type of 'time'. The *FlightCode* attributes data type is 'varchar' to support numbers and characters, this has also been set to 'unique' to avoid duplicate entries. The Foreign key constraints in the Flight table are, *FlightAircraftID* which links to the **Aircraft** table, and *FlightRoute*, which links to the **Route** table. A flight can now be linked to the route that it travels and the airports from which it takes off and arrives at. (**Query 2**) shows how a flight that hasn't yet been subject to bookings can be removed from the database in the event that it is cancelled.

Through using the date stored in the *DepartureD*ate attribute, the amount of days until departure for a specific flight can be worked out using the NOW( ) function in SQL.

FlightID	DepartureDate	NOW()	Days to Depart
11	2020-12-14	2020-11-25 17:26:37	19

*Figure 4 shows the results of a query that searches for days until departure for FlightID 11. (**Query 3.1**)*

The **FlightPricing** table allows for a storage of flight prices in multiple currencies. The Foreign Keys in this table link to the **Flight** table and the **Currency** table. An airline company can make adjustments on the price of a flight as the departure date gets closer; (***Query 3.2***) shows how the *FlightPrice* attribute in the **FlightPricing** table is updated for FlightID 11.

The structure of the database and the Foreign Key relationships established between tables will enable the airline to track flights, the routes they are travelling, the departure and arrival airports, aircrafts used and additional details for a specific date if required.

FlightID	FlightCode	DepartureDate	DepartureTime	ArrivalTime	Route	Departure Airport	Arrival Airport	AircraftName	AircraftReg
1	EZY1000	2021-01-15	17:30:00	19:00:00	Belfast to Amsterdam	Belfast International Airport	Amsterdam Schiphol Airport	Airbus A319-100	G-EZAB
16	EZY5432	2021-01-15	08:00:00	10:00:00	Alicante to Naples	Alicante Airport	Naples International Airport	Airbus A319-100	G-EZAB
17	EZY6543	2021-01-15	10:00:00	11:45:00	Amsterdam to Belfast	Amsterdam Schiphol Airport	Belfast International Airport	Airbus A319-100	G-EZAC
35	EZY2345	2021-01-15	10:00:00	12:00:00	Belfast to Amsterdam	Belfast International Airport	Amsterdam Schiphol Airport	Airbus A319-100	G-EZAC
18	EZY6421	2021-01-15	12:15:00	13:45:00	Belfast to Amsterdam	Belfast International Airport	Amsterdam Schiphol Airport	Airbus A319-100	G-EZAF
19	EZY8521	2021-01-15	13:35:00	15:15:00	Belfast to Berlin	Belfast International Airport	Berlin Schönefeld Airport	Airbus A320-200	G-EZAG
20	EZY2954	2021-01-15	16:15:00	19:25:00	Belfast to London Gatwick	Belfast International Airport	London Gatwick Airport	Airbus A320-200	G-EZGX
21	EZY7312	2021-01-15	14:25:00	16:25:00	Berlin to Belfast	Berlin Schönefeld Airport	Belfast International Airport	Airbus A320-200	G-EZGY
22	EZY1074	2021-01-15	18:15:00	20:00:00	Berlin to London Gatwick	Berlin Schönefeld Airport	London Gatwick Airport	Boeing 737-200	G-UZHE
32	EZY1420	2021-01-15	19:00:00	21:00:00	Manchester to Faro	Manchester Airport	Faro Airport	Boeing 737-200	G-UZMC
33	EZY9163	2021-01-15	16:45:00	19:25:00	Alicante to Naples	Alicante Airport	Naples International Airport	Boeing 737-200	G-UZMC
23	EZY1063	2021-01-15	15:15:00	16:35:00	Liverpool to Naples	John Lennon Airport Liverpool	Naples International Airport	Boeing 757-200	G-UZHX

*Figure 5 shows how flights with the same departure date are tracked, using the relationships between flights, routes, airports and aircrafts. (**Query 4**)*

Aircraft Seating

The **AircraftSeating** table holds all available seats for different aircrafts. The Foreign Keys are the *AircraftID attribute* that links to the Primary Key in the **Aircraft** table, and the *SeatID* attribute that links to the Primary Key in the **Seating** table.

The **Seating** table stores details that describe the seat, with the *SeatSelection* attribute storing the letter and row number of the seat, e.g. A1. The *SeatTypeID* attribute links to the **SeatType** table where the seat types have been normalised, e.g. extra legroom, up front, and standard.

The **SeatPrice** table has been created to store the price of seats in different currencies. The Foreign Key constraints in this table are the *SeatTypeID* attribute, that links to the Primary key in the **SeatType** table, and the *CurrencyID* attribute that links to the Primary Key in the **Currency** table.

SeatPriceID	Seat Type	Price	Currency
1	Extra Legroom	20.00	GBP
5	Up Front	18.50	GBP
9	Standard	5.50	GBP
2	Extra Legroom	22.00	EUR
6	Up Front	20.00	EUR
10	Standard	6.10	EUR
3	Extra Legroom	26.50	USD
7	Up Front	26.50	USD
11	Standard	7.30	USD
4	Extra Legroom	100.00	PLN
8	Up Front	92.50	PLN
12	Standard	27.50	PLN

*Figure 6 shows the results from a query that searches for seat types and their prices in different currencies. (**Query 5**)*

The **AircraftSeating** table allows us to identify what seats are on an aircraft that is being used for a flight, prior to passenger seat selection.

AircraftName	SeatSelection	SeatTypeName	FlightID
Airbus A319-100	A1	Extra Legroom	10
Airbus A319-100	B1	Extra Legroom	10
Airbus A319-100	C1	Extra Legroom	10
Airbus A319-100	D1	Extra Legroom	10
Airbus A319-100	E1	Extra Legroom	10
Airbus A319-100	F1	Extra Legroom	10
Airbus A319-100	A12	Extra Legroom	10
Airbus A319-100	A2	Up Front	10
Airbus A319-100	B2	Up Front	10
Airbus A319-100	C2	Up Front	10
Airbus A319-100	D2	Up Front	10
Airbus A319-100	E2	Up Front	10
Airbus A319-100	F2	Up Front	10
Airbus A319-100	A3	Up Front	10
Airbus A319-100	B3	Up Front	10

Figure 7 *shows some of the result set for a query that searches for aircraft seats for FlightID 10. (**Query 6.1**)*

As the **AircraftSeating** table and the **Flight** table are both connected to the **Passenger** table through Foreign Keys constraints, we can now run a query to identify which seats have been assigned to passengers for a specific flight.

FlightID	Seats Taken	PassengerID
10	C3	13
10	D18	14
10	F16	15
10	A12	16
10	B16	17
10	A2	18
10	B3	19
10	C16	20
10	E18	21
10	D1	22

Figures 8 shows the results of a query that searches for seats taken on FlightID 10. (**Query 6.2**)

The results from (*Query 6.2*) can be checked against the results from (*Query 6.1*), and the seat map for FlightID 10 can be updated on the airlines website to remove the seats that are no longer available for selection.

FlightID	Aircraft Seats
10	25

*Figure 9 shows the results from a query that counts the amount of seats on FlightID 10. (**Query 7.1**)*

PassFlightID	Seats Taken
10	10

*Figure 10 shows the results from a query that counts seats taken on FlightID 10. (**Query 7.2**)*

If a passenger wishes to change their seat selection, they can change to another available seat up until to two hours before check in. (**Query 7.3**) shows how seat selection for a passenger can be updated on the database.

Another way seating could have been implemented would have been be to add the *FlightID* to **Seating** table, however I decided against this approach as it would be inefficient due to new seat entries having to be added to the **Seating** table for every occurrence of a flight.

The next section of the report details how the database stores: customers, user accounts, payment details, passengers and their luggage. I will describe how these core elements of flight booking are connected by the structural design of their representing tables and provide a justification for the design decisions made. In this section of the report we can also see how these entities link to: flights, airports, aircrafts, routes and seat selection.

Customers

The Primary Key in the **Customer** table is *CustomerID*, which is linked to the **Booking** table and **UserAccount** table through Foreign Key constraints. Customers are described by the *CustFirstName*, *CustSurname, CustDOB, and ContactNumber* attributes. The Foreign Key constraints that are the *TitleID* attribute, which links to the **Title** table and *CustAddressID* attribute which links to the **Address** table also describe the customer. The normalisation of titles and addresses ensures consistency in data, makes updates less time consuming and improves data accuracy. The data type for *CustDOB* is set to 'date' and is stored instead of age to avoid repeated updates to customer details in the future.

First Name	Surname	DOB	AGE
John	Smyth	1987-01-01	33
Mary	O'Neill	1990-02-03	30
Paul	Muller	1975-02-15	45
Rachel	McElroy	1992-06-22	28
Sarah	Armstrong	1988-08-15	32
Andrew	Roberts	1996-11-23	24
Bradley	Phillips	1990-11-23	30

*Figure 11 shows the data set from a query that lists the age of customers. (**Query 8.1**)*

Should a customer need to make amendments to their customer account, for example, if there is a typo in the name or their Date of Birth has been entered incorrectly, a query can be used to update their record once the correct information has been provided by the customer. (**Query 8.2**). Address updates can be made on behalf of customers by using a transaction query that adds a new address to the **Address** table and updates the customers *AddressID* in the **Customer** table. (**Query 8.3**)

A one-to-many relationship between the **Customer** table, and the **Booking** table is established to support the fact that one customer can make many bookings. The Primary Key in the **Customer** table links to the *CustBookingID* attribute in the **Booking** table.

First Name	Surname	CustomerID	BookingID	BookingDate	Booking Code
John	Smyth	1	1	2020-07-01	BKN1000
John	Smyth	1	6	2021-01-31	BKN1455
John	Smyth	1	9	2020-05-26	BKN4588

*Figure 12 shows results from a query to search for multiple bookings from the same customer. (**Query 9**)*

UserAccounts

A Foreign Key constraint has been setup in the **UserAccount** Table, linking the *CustID* attribute to the Primary Key in the **Custome**r table, establishing a one-to-many relationship between the **Customer** and **UserAccount** tables. It is necessary to establish this relationship as a customer can have more than one user account with different email addresses. The *Email* attribute in the **UserAccount** table has the data type of 'varchar' to support numbers, letters and special characters, and is set to 'unique', so that one email address can only be assigned to one account.

AccountID	Type of Account	CustomerID	UserName	Email	UserPassword
1	easyJet Standard	3	pmuller11	pmuller@msn.com	0x46d3c9ea539eabff2ae17b35a28b5577
2	easyJet Standard	3	petermull123	paulmull@outlook.com	0x5b4ee60c66ec49f5068694761dbe18b6
4	easyJet Plus	3	pmuller4	pmuller2@gmail.com	0x0c8c8a854c73746b94ae902fd2514b8f

*Figure 13 shows results from a query to search for user accounts setup by CustomerID 3. (**Query 10**)*

The account types have been normalised to avoid repetitive data entry and improve the accuracy and consistency of data. The **AccountType** table lists the account types available. The Primary Key in the **AccountType** table is linked to the **UserAccount** table through a Foreign Key constraint.

AccountID	TitleName	Account Type	First Name	Surname	Address	City	Country
1	Mr	easyJet Standard	Paul	Muller	15 Grand Avenue	Munich	Germany
2	Mr	easyJet Standard	Paul	Muller	15 Grand Avenue	Munich	Germany
8	Mr	easyJet Standard	Andrew	Roberts	Mossley Hill	Liverpool	United Kingdom
10	Mr	easyJet Standard	Bradley	Phillips	54 WalkvWay Grove	Glasgow	Scotland
4	Mr	easyJet Plus	Paul	Muller	15 Grand Avenue	Munich	Germany
7	Mr	easyJet Plus	Andrew	Roberts	Mossley Hill	Liverpool	United Kingdom
9	Mr	easyJet Plus	John	Smyth	55 Westway Park	Dublin	Ireland
3	Mrs	easyJet Plus	Mary	O'Neill	5 Camden Street	Belfast	Northern Ireland
6	Mrs	easyJet Plus	Sarah	Armstrong	22 Calle Pescadoro	Madrid	Spain
5	Miss	easyJet Standard	Rachel	McElroy	105 Glendore Way	Manchester	United Kingdom

*Figure 14 shows the data set from a query that searches for user accounts belonging to customers, combined with customer details. (**Query 11**)*

Encryption

To protect sensitive data that is stored on the database from misuse or malicious intent, passport numbers, account login passwords and card payment details have been encrypted.

The **UserAccount** table stores user's login passwords under the *UserPssword* attribute. Password are encrypted when being added to the database using AES Encryption and an encryption key that is unique to each password. I have chosen to assign each password its own encryption key as this is more secure than having a common encryption key for all passwords. The Data type for encrypted password is set to varbinary. (***Query 12***) shows how a customer's login details are added to the **UserAccount** table using AES encrypt.

The **Passports** table holds encrypted passport details that are added using AES Encrypt. The *PassportNumber* attribute is also set to 'unique' to avoid duplicate entries. Passports can be linked to passengers through the Foreign Key constraint in the **Passenger** table that links to the Primary Key in the **Passports** table. Passports can now be stored securely and decrypted if necessary.

PassportID	PassportNumber	Country	AES_DECRYPT(PassportNumber, 'mySecretKey1')	AES_DECRYPT(Country, 'mySecretKey1')
1	0x56b7e8f29421082f1e554654a0b964aac5a30c66d10ef697...	0xd7f3f665362909ce95ab02e789385177	1234374542793547345789348 9	Ireland

*Figure 15 shows the decryption of PassportID 1. (**Query 13**)*

A Foreign Key constraint has been setup in the **Payment** table linking payment details to the Primary Key in the **UserAccount** table. An EasyJet customer can make bookings through their user account without having to use the same payment method every time. Therefore, the database requires a one-to-many relationship between the **UserAccount** and **Payment** tables as one user account can have many payment methods. The *NameOnCard*, *CardProvider, CardNumber* and *CardCVV* attributes in the **Payment** table store payment details that and are encrypted when being added to the database using AES Encryption. The data type for these attributes is 'varbinary'.

PaymentID	AccountID	CardProvider	NameOnCard	CardNumber	CardCVV
7	4	0x549ac7590b412f52113bb5a5b85c7764	0xf06c21b1df4681558bbbb99935437faf	0x6ee94caf399574ee6eef5b1bb8de7883d4a46dcdc337e8e7...	0x5993afff3f6322fa5ab1291419bac9a7

*Figure 16 shows encrypted details for PaymentID 4. (**Query 14**) shows how the encrypted details are added to the database.*

AES_DECRYPT(NameOnCard, 'mySecretKey15')	AES_DECRYPT(CardProvider, 'mySecretKey15')	AES_DECRYPT(CardNumber, 'mySecretKey15')	AES_DECRYPT(CardCVV, 'mySecretKey15')
Mary O Neill	Santander	3456 5687 1298 3417	428

*Figure 17 shows results from a query to decrypt PaymentID 4. (**Query 15**)*

Bookings

The Primary Key in the **Booking** table is *BookingID,* which is linked to the **Passenger** table and the **BookingLineItem** table through Foreign Key constraints. A Foreign Key constraint has been setup in the **Passenger** table**,** which links to the Primary Key in the **Booking** table, creating a one-to-many relationship between **Booking** and **Passenger.** This relationship is necessary, as one booking can be made for one or more passengers in a real life booking system. For example, a customer can book multiple return flights for a group of people, creating many passengers.

PassengerID	FlightID	BookingID	FirstName	Surname
4	4	3	Paul	Muller
5	4	3	Ciara	Muller
6	4	3	Peter	Muller
7	5	3	Paul	Muller
8	5	3	Ciara	Muller
9	5	3	Peter	Muller

*Figure 18 shows the results of a query that searches for multiple passengers that were booked onto different flights through BookingID 3. (**Query 16**). FlightID 4 is the departing flight, and FlightID 5 is the return flight.*

The Foreign Key constraints in the **Booking** table are *CustBookingID,* which links to the Primary Key in the **Customer** table, and *BookCurrencyID, which* links to the Primary Key in the **Currency** table. A customer can make multiple bookings whilst having the option to pay with different currencies, and this is supported by the structure of the database. The *TotalPrice* attribute in the **Booking** table can now be connected to the currency used. The *BookingRef* attribute has been set to 'unique' to prevent duplicate entries of the same booking reference.

BookingID	BookingDate	TotalPrice	CurrencyID	Currency	CurrencyType
1	2020-07-01	169.50	1	£	British Pounds
4	2020-09-30	75.00	1	£	British Pounds
7	2020-11-16	850.00	1	£	British Pounds
2	2020-07-06	267.60	2	€	Euros
3	2020-09-02	793.70	2	€	Euros
5	2020-10-02	108.45	2	€	Euros
8	2020-11-25	95.00	3	$	US Dollars

*Figure 19 shows results from a query to search for bookings made in different currencies in 2020. (**Query 17**).*

An airline can use the **Booking** table which stores the *TotalPrice* and *BookingDate*, to calculate revenue from bookings in different currencies over a specific time period, if required for accounting purposes.

Booking Revenue (£) 2020	Booking Revenue (€) 2020
1194.50	1169.75

*Figure 20 shows the results from queries to select the sum of Bookings for different currencies in 2020 based on sample data. (**Query 18**)*

BookingLineItems

A Foreign Key has been setup in the **BookingLineItem** table linking to the Primary Key in the **Booking** table, to create a one-to-many relationship between **Booking** and **BookingLineItem**. This relationship is necessary as one booking can have many line items, for example, flight tickets, food vouchers, baggage, sports equipment and seat selections.

BookingID	Item	Description	Cost	Currency
3	Flight Tickets	2 Adult tickets - Return 1 Child...	390.00	€
3	Passenger Baggage	32kg hold bag x 2 Return Flight EZY1050 Flight E...	134.00	€
3	Food and Drink Voucher	3 Food and drinks vouchers x2 Return Flight EZY10...	50.00	€
3	Sports Equipment	Golf bag x2 Return Bicylce x2 Return ...	183.10	€
3	Seats	3x Standard Seats 2 way return	36.60	€

*Figure 21 shows the results when searching for line items purchased by BookingID 3. (**Query 19**)*

The **BookingLineItem** table allows for a storage of prices while providing itemisation in terms of products. The items in this table can then totalled up and added to the *TotalPrice* attribute in the **Booking** table. (***Query 20***). This feature of the database ensures that the *TotalPrice* in the **Booking** table is accurate and up to date.

Passengers

The Primary Key in the **Passenger** table is *PassengerID*, which is linked to the **PassBaggage** table, and **PassSportsEquipment** table, through Foreign Key constraints.

Passengers are linked to their flights through the *PassFlightID* attribute, which is a Foreign Key constraint that links to the **Flight** table, and passengers are linked to their seats through the *PassAircraftSeat* attribute, which is a Foreign key constraint that links to the **AircraftSeating** table.

FlightID	Route	DepartureDate	DepartureTime	BookingID	PassengerID	FirstName	Surname	SeatSelection
10	Liverpool to Naples	2021-02-04	10:00:00	7	13	Andrew	Roberts	C3
10	Liverpool to Naples	2021-02-04	10:00:00	7	14	John	Stevens	D18
10	Liverpool to Naples	2021-02-04	10:00:00	7	15	Sheamus	O'Reilly	F16
10	Liverpool to Naples	2021-02-04	10:00:00	7	16	Louise	Roberts	A12
10	Liverpool to Naples	2021-02-04	10:00:00	7	17	Louise	Sweeney	B16
10	Liverpool to Naples	2021-02-04	10:00:00	7	18	Ciara	Foster	A2
10	Liverpool to Naples	2021-02-04	10:00:00	7	19	Tom	Spencer	B3
10	Liverpool to Naples	2021-02-04	10:00:00	7	20	Sarah	Cathcart	C16
10	Liverpool to Naples	2021-02-04	10:00:00	7	21	Tom	Forde	E18
10	Liverpool to Naples	2021-02-04	10:00:00	7	22	John	White	D1

*Figure 22 shows the results from a query that searches for passengers and their seat selection from BookingID 7, on Flight ID 10. (**Query 21.1**)*

There is three types of passenger as defined by EasyJet as, Adult, Child and Infant. The passenger types have been normalised through the creation of a **PassengerTypes** table to keep the data consistent, and prevent errors from repetitive data entries. The *PassengerType* attribute in the **Passenger** table links both tables, forming a one-to-many relationship between **PassengerTypes** and **Passenger**.

The passenger's reason for travel has also been normalised for efficiency and data consistency purposes. A **ReasonForTravel** table has been setup with the Primary Key being linked to the *PassTravelReason* attribute in the **Passenger** table.

PassFlightID	PassengerID	PassTypeID	PassengerType	TravelReasonID	TravelReason
6	23	1	Adult	2	Leisure
6	24	1	Adult	2	Leisure
6	25	1	Adult	2	Leisure

*Figure 23 shows the results from a query to search for Passengers currently booked on FlightID 6, who are adults and chose 'Leisure' as their reason for travel. (**Query 21.2**)*

The PassAssistType attribute in the **Passenger** table accounts for occurrences where a passenger requires assistance when travelling with EasyJet. A Foreign Key constraint has been setup connecting the PassAssistType attribute to the Primary Key in **PassAssistance** table that stores passenger assistance options as listed on the EasyJet website. It is crucial for the database to incorporate this feature so that any passenger requirements can be identified.

FlightID	PassengerID	AssistanceID	Assistance Type
4	5	6	nut allergy
4	6	6	nut allergy

*Figure 24 shows the results from a query to search for passengers on FlightID 4 with nut allergies. (**Query 22**)*

A passenger on board an EasyJet flight who is not the customer who booked the flight, may have EasyJet Plus membership, in which case the booker can enter EasyJet plus details on their behalf when booking the flight. The passenger may also have EasyJet travel insurance. To account for these possibilities, the *EasyJetPlus and Insurance* attributes in the **Passenger** table are setup with data types set to 'Boolean'. These attributes require a 1 (True) or 0 (False), to indicate if the passenger is an EasyJet plus member and if they have travel insurance, respectively.

PassengerID	First Name	Surname	PassFlightID	Insurance?	EasyJetPlus?
13	Andrew	Roberts	10	0	1
14	John	Stevens	10	0	1
15	Sheamus	O'Reilly	10	0	0
16	Louise	Roberts	10	0	1
17	Louise	Sweeney	10	0	0
18	Ciara	Foster	10	0	1

*Figure 25 shows a result set from a query to search for passengers on FlightID 10 who do not have travel insurance. (**Query 23**)*

The *PassPassportID* attribute in the **Passenger** table is a Foreign Key constraint that links to the *Primary Key* in the **Passports** table, where the passport numbers for passengers are encrypted when added to the database. The Passport numbers can be decrypted when necessary, for example when a passenger is checking in at an airport and they show their passport, the passport number that is stored may need decrypted to compare passport numbers. (**Query 13**)

Baggage

Many Passengers on board a flight can have many types of baggage, and it is therefore essential that the database supports a many-to-many relationship between the **Passenger** table and the **Baggage** table. The **Baggage** table stores different baggage types and their prices in various currencies. I have normalised the baggage types into a **BaggageType** table, which describes baggage by its weight. The *BaggTypeID* attribute in the Baggage table is a foreign key constraint that link to the Primary Key in the **BaggageTypes** table. The normalisation of baggage types is useful if amendments need made to baggage types on offer, and prevents having to update large amounts data entries while also keeping data consistent and accurate.

In order to create the many-to-many relationship between the **Passenger** and **Baggage** tables, a junction table called **PassBaggage** has been created. The **PassBaggage** table supports two, one-to-many relationships, as one passenger can appear many times in the **PassBaggae** table and one item of baggage can appear many times in the **PassBaggage** table. We can now efficiently link passengers to baggage items.

FlightID	Baggage	Currency	Price	BookingID	PassengerID	First Name	Surname
36	15kg hold bag	£	20.00	16	32	John	Jones
36	29kg hold bag	£	47.00	16	33	Bruce	Wayne
36	29kg hold bag	£	47.00	16	33	Bruce	Wayne
36	15kg hold bag	£	20.00	16	34	Ciara	Wayne
36	15kg hold bag	£	20.00	16	34	Ciara	Wayne

*Figure 26 shows the results of a query that searches for baggage belonging to Passengers from BookingID 16 on FlightID 36. (**Query 24**)*

Sports Equipment

The **PassSportsEquipment** table is a junction table that creates a many-to-many relationship between the **Passenger** and **SportsEquipment** tables**.** It is necessary to form this many-to-many relationship to support the fact that many passengers can have many types of sports equipment on board a flight. The Foreign Key constraints in the **PassSportsEquipment** table link the *Passenger* attribute to Primary Key in the **Passenger** table, and the *Equipment* attribute to Primary Key in the **SportsEquipment** table that stores equipment types in multiple currencies.

There are eight types of sports equipment options to choose from when selecting luggage on easyjet.com. The types of sports equipment have been normalised in the **SportsEquipTypes** table and linked to the **SportsEquipment** table through a Foreign Key constraint. This normalisation allows for the type of sports equipment to be repeatedly listed in the **SportsEquipment** table to account for prices in different currencies without having to type in each item of equipment, while improving data accuracy and consistency in the database.

FlightID	SportsEquipment	Currency	SportsEquipPrice	BookingID	PassengerID	First Name	Surname
10	Golf bag	£	37.00	7	13	Andrew	Roberts
10	Golf bag	£	37.00	7	13	Andrew	Roberts
10	Hang glider	£	45.00	7	14	John	Stevens
10	Bicycle	£	45.00	7	14	John	Stevens
10	Snowboard	£	37.00	7	15	Sheamus	O'Reilly
10	Canoe	£	45.00	7	15	Sheamus	O'Reilly
10	Other small sports equipment	£	37.00	7	16	Louise	Roberts
10	Windsurfing board	£	45.00	7	16	Louise	Roberts

*Figure 27 shows results of a query that searches for sports equipment belonging to Passengers from BookingID 7, on FlightID 10. (**Query 25**)*

In a real life flight booking system it is expected that changes will be made to bookings on occasions, for example, one of the passengers that has been booked onto a flight are no longer going ahead with their journey. In this situation, the Passenger, PassBagagae, PassSportsEquipment, BookingLineItem, and Booking tables can be updated using a transaction query. (**Query 26**). The customer who made the booking would then be issued a refund for the money paid on behalf of the passenger using the payment details stored in the Payment table.

Potential Improvements;

The database stores payment details, passport numbers and account passwords, which are encrypted using AES Encryption. The drawbacks to using AES Encryptions are that it uses a simple algebraic structure, every block is encrypted the same way, and it is considered hard to implement with software. If someone got access to the encryption key for a users payment details with the intent to commit fraud, a customer would lose money and the financial implications would be disastrous for the airline company. Therefore, the encryption of this type of sensitive information would be more secure and efficient if it was handled by an established third party who specialises in encryption of user details on a wide scale. Alternatively , the airline company ycould hire an encryption expert that uses a programming language to encrypt security sensitive information.

Flight codes for flights are manually entered onto the flight table, ex. 'EZY 2468'. The flight code is often used on airport screens to identify flights and is also printed on passenger receipts. Therefore having such an important piece of information manually entered onto the system is inefficient as errors can likely happen with manual data entry on such a wide scale. The alternative is to create a new table to store flight codes with the auto incremented primary key set to 'FlightCode', and link this primary key to the Fight table. Another option would be to change the Primary Key in the Flight table to 'FlightCode'. Aircraft Registration details are also manually entered onto the database in the Aircraft table which increases the chance of errors and inconsistencies in data. A better option would be to normalise aircraft registrations, linking the primary key in a newly created AircraftRegistration table to the Aircraft table.

In its current state, the database allows for distinguishing between passengers with standard accounts and passengers with EasyJet Plus accounts but doesn't implement any of the features that come with EasyJet Plus membership. Customers with EasyJet plus membership can select any seat on a plane without extra cost, along with other added benefits. A better implementation of EasyJet plus membership would involve the creation of a table to store EasyJet plus members and their EasyJet Plus account numbers. The pricing system could then be updated in the database to account for EasyJet Plus members.

In conclusion, considering the potential improvements, the database meets the requirement to support a flight booking system by providing storage of the core elements to flight booking while establishing relationships between entities through the structure of their representing tables. The database accounts for a storage of aircrafts, airports, flight and routes, along with customers, bookings, seat allocation and passenger luggage types. The structure of the database enables useful result sets to be achieved from SQL queries that combine attributes of entities. The design of the database allows for many passengers to be booked through one customer booking, while enabling many bookings to be made by one customer. Passengers can be linked to their flights, baggage and sports equipment, along with seat selection and payment methods, while the total price of bookings can be accurately fetched. The structure of the database allows records to be updated efficiently when required, while the normalisation of the database improves consistency and data integrity , improving efficiency for future use.

APPENDIX

QUERIES;

1.

```
SELECT `AircraftID`, `AircraftReg`, AircraftName
FROM `Aircraft` INNER JOIN AircraftType
ON Aircraft.TypeOfAircraft = AircraftType.AircraftTypeID
```

2

```
DELETE FROM Flight WHERE FlightID = 37;
```

3.1

```
SELECT `FlightID`, `DepartureDate`,
NOW(), DATEDIFF(DepartureDate, NOW()) AS 'Days to Depart'
FROM `Flight`
WHERE DepartureDate >= NOW() && FlightID = 11;
```

3.2

```
UPDATE `FlightPricing` SET `FlightPrice` = `FlightPrice`* 1.4 WHERE `FlightID` = 11;
```

4.

```
SELECT FlightID, FlightCode, DepartureDate, DepartureTime, ArrivalTime,
RouteName AS Route, DepartAirport.AirportName AS "Departure Airport",
ArrivalAirport.AirportName AS "Arrival Airport", AircraftName, AircraftReg
FROM `Route`
INNER JOIN Airport DepartAirport
ON Route.DepartAirport = DepartAirport.AirportID
INNER JOIN Airport ArrivalAirport
ON Route.ArrivalAirport = ArrivalAirport.AirportID
INNER JOIN Flight ON Flight.FlightRouteID = Route.RouteID
INNER JOIN Aircraft ON Aircraft.AircraftID = Flight.FlightAircraftID
INNER JOIN AircraftType ON AircraftType.AircraftTypeID = Aircraft.TypeOfAircraft
WHERE date(DepartureDate) = "2021-01-15";
```

5.

```
SELECT `SeatPriceID`, SeatTypeName AS 'Seat Type',Price,  CurrencyName AS Currency
FROM `SeatPrice`
INNER JOIN Currency
ON Currency.CurrencyID = SeatPrice.CurrencyID
INNER JOIN SeatType On SeatType.SeatTypeID = SeatPrice.SeatTypeID
```

6.1

```
SELECT AircraftName,SeatSelection,SeatTypeName, FlightID FROM AircraftSeating LEFT JOIN Flight
ON Flight.FlightAircraftID = AircraftSeating.AircraftID
INNER JOIN Aircraft ON Aircraft.AircraftID = AircraftSeating.AircraftID
INNER JOIN AircraftType ON AircraftType.AircraftTypeID = Aircraft.TypeOfAircraft
INNER JOIN Seating ON Seating.SeatID = AircraftSeating.SeatID
INNER JOIN SeatType ON SeatType.SeatTypeID = Seating.SeatTypeID
WHERE FlightID = 10;
```

6.2

```
SELECT FlightID, SeatSelection AS 'Seats Taken',
PassengerID FROM `AircraftSeating` LEFT JOIN Passenger
on Passenger.PassAircraftSeat = AircraftSeating.AircraftSeatID
INNER JOIN Seating ON Seating.SeatID = AircraftSeating.SeatID
INNER JOIN Flight on Flight.FlightID = Passenger.PassFlightID
WHERE FlightID =10;
```

7.1

```
SELECT  FlightID, COUNT(*) AS 'Aircraft Seats'  FROM AircraftSeating
INNER JOIN Flight
ON Flight.FlightAircraftID = AircraftSeating.AircraftID
WHERE FlightID = 10;
```

7.2

```
SELECT PassFlightID ,COUNT(*) AS 'Seats Taken' FROM `AircraftSeating`
LEFT JOIN Passenger
on Passenger.PassAircraftSeat = AircraftSeating.AircraftSeatID
WHERE PassFlightID =10;
```

7.3

```
UPDATE Passenger
SET PassAircraftSeat = '25'
WHERE PassengerID = 5;
```

8.1

```
SELECT `CustFirstName` AS 'First Name' ,
`CustSurname` AS Surname, `CustDOB` AS DOB,
timestampdiff(year, `CustDOB`,now()) AS AGE
FROM Customer;
```

8.2

```
START TRANSACTION ;

UPDATE Customer
SET CustDOB = '1990-02-04'
WHERE `CustomerID` = 4;
UPDATE Customer
SET CustSurname = 'Roberts'
WHERE `CustomerID` = 4;

COMMIT;
```

8.3

```
START TRANSACTION ;

INSERT INTO `Address` (`AddressID`, `AddressLine1`, `AddressLine2`, `Town`, `City`, `County`, `Postcode`, `Country`)
VALUES (NULL, '5 Forest Green Park', '', '', 'Dublin', '', 'D12 754DB', 'Ireland');
SET @last_id_in_addressTable = LAST_INSERT_ID();
UPDATE Customer SET CustAddressID = @last_id_in_addressTable
WHERE `CustomerID` = 7;

COMMIT;
```

9.

```
SELECT `CustFirstName` AS 'First Name', `CustSurname` AS Surname,
`CustomerID`, BookingID, BookingDate, BookingRef AS 'Booking Code'
FROM `Customer` INNER JOIN Booking
On Customer.CustomerID = Booking.CustBookingID
Where CustomerID = 1;
```

10.

```
SELECT AccountID, AccType AS 'Type of Account',
CustomerID,UserName,Email, UserPassword
FROM `Customer`
INNER JOIN UserAccount ON UserAccount.CustID = Customer.CustomerID
INNER JOIN AccountType ON AccountType.AccountTypeID = UserAccount.AccountType
WHERE CustomerID = 3;
```

11.

```
SELECT AccountID, TitleName, AccType AS 'Account Type',
`CustFirstName` AS 'First Name',`CustSurname` AS Surname,
AddressLine1 AS Address, City,Country FROM `Customer`
INNER JOIN Address ON Address.AddressID = Customer.CustAddressID
LEFT JOIN UserAccount ON UserAccount.CustID = Customer.CustomerID
INNER JOIN AccountType On AccountType.AccountTypeID = UserAccount.AccountType
INNER JOIN Title ON Title.TitleID = Customer.TitleID
```

12.

```
INSERT INTO UserAccount (Email, Username, UserPassword, CustID, AccountType)
VALUES (
'bwayne11@msn.com',
'bwayne12345',
AES_ENCRYPT('Password100','mySecretKey20'),
'14', '1'
);
```

13.

```
SELECT `PassportID`, `PassportNumber`, `Country`,
AES_DECRYPT(PassportNumber, 'mySecretKey1') ,
AES_DECRYPT(Country, 'mySecretKey1')
FROM Passports WHERE `PassportID` = 1;
```

14.

```
INSERT INTO Payment (AccountID, CardProvider, NameOnCard, CardNumber, CardCVV)
VALUES (
'4',
AES_ENCRYPT('Santander','mySecretKey15'),
AES_ENCRYPT('Mary O Neill','mySecretKey15'),
AES_ENCRYPT('3456 5687 1298 3417','mySecretKey15'),
AES_ENCRYPT('428','mySecretKey15')
);
```

15.

```
SELECT  `NameOnCard`, `CardNumber`, `CardCVV`,
AES_DECRYPT(NameOnCard, 'mySecretKey15') ,
AES_DECRYPT(CardProvider, 'mySecretKey15') ,
AES_DECRYPT(CardNumber, 'mySecretKey15'),
AES_DECRYPT(CardCVV, 'mySecretKey15')
From Payment WHERE `PaymentID` = 7;
```

16.

```
SELECT PassengerID, PassFlightID AS FlightID, BookingID,
PassFirstName AS FirstName, PassSurname as Surname
FROM `Booking` INNER JOIN Passenger
ON Booking.BookingID = Passenger.PassBookingID
Where BookingID = 3;
```

17.

```
SELECT `BookingID`, BookingDate ,`TotalPrice`,CurrencyID,
CurrencySign AS Currency, CurrencyType FROM `Booking`
INNER JOIN Currency ON Currency.CurrencyID = Booking.BookCurrencyID
INNER JOIN Customer On Customer.CustomerID = Booking.CustBookingID
WHERE year(BookingDate) = 2020
```

18.

```
SELECT SUM(`TotalPrice`) AS 'Booking Revenue (£) 2020'
FROM Booking
WHERE year(`BookingDate`) = 2020 && `BookCurrencyID` = 1 ;

SELECT SUM(`TotalPrice`) AS 'Booking Revenue (€) 2020'
FROM Booking
WHERE year(`BookingDate`) = 2020 && `BookCurrencyID` = 2 ;
```

19.

```
SELECT `BookingID`, LineItemName AS Item, LineItemDesc AS Description,
LineItemCost AS Cost, CurrencySign AS Currency FROM `Booking`
INNER JOIN BookingLineItem
ON Booking.BookingID= BookingLineItem.BookerID
INNER JOIN Currency
ON BookingLineItem.CurrencyID = Currency.CurrencyID
WHERE BookingID = 3;
```

20.

```
SELECT * FROM `BookingLineItem` WHERE 1;
SET @sumTotal = (SELECT SUM(LineItemCost) FROM `BookingLineItem` WHERE`BookerID`=2);
UPDATE `Booking` SET `TotalPrice` = @sumTotal WHERE `Booking`.`BookingID` = 2;
```

21.1

```
SELECT BookingID, `PassengerID`, PassFirstName AS 'FirstName', PassSurname AS Surname,
SeatSelection, FlightID, DepartureDate, DepartureTime, RouteName  FROM `Passenger` INNER JOIN
Booking ON Booking.BookingID = Passenger.PassBookingID
INNER JOIN AircraftSeating ON AircraftSeating.AircraftSeatID = Passenger.PassAircraftSeat
INNER JOIN Seating ON Seating.SeatID = AircraftSeating.SeatID
INNER JOIN Flight ON Flight.FlightID = Passenger.PassFlightID
INNER JOIN Route ON Route.RouteID = Flight.FlightRouteID
WHERE BookingID = 7 && FlightID = 10;
```

21.2

```
SELECT `PassFlightID`, `PassengerID`,  `PassType` AS 'PassTypeID',PassengerType,
PassTravelReason AS 'TravelReasonID', TravelReason FROM `Passenger`
INNER JOIN PassengerTypes
On Passenger.PassType = PassengerTypes.PassengerTypeID
INNER JOIN ReasonForTravel
ON ReasonForTravel.TravelReasonID = Passenger.PassTravelReason
WHERE PassType = 1 && `PassFlightID`= 6 && TravelReason = 'Leisure';
```

22.

```
SELECT `PassFlightID` AS FlightID,`PassengerID`, `PassAssistType`
AS AssistanceID, AssistanceType
FROM `Passenger` INNER JOIN PassAssistance
ON Passenger.PassAssistType = PassAssistance.AssistanceID
WHERE PassFlightID = 4 && `PassAssistType` = 6;
```

23.

```
SELECT `PassengerID`, `PassFirstName` AS 'First Name',
`PassSurname` AS 'Surname', `PassFlightID`, `Insurance`AS 'Insurance?',
`EasyJetPlus`AS 'EasyJetPlus?' FROM `Passenger`
WHERE `PassFlightID` =10 && Insurance = 0;
```

24.

```
SELECT FlightID, TypeOfBaggage AS Baggage , CurrencySign AS Currency, BaggagePrice AS Price ,
BookingID, PassengerID, PassFirstName AS 'First Name', PassSurname AS Surname
FROM `Passenger` INNER JOIN PassBaggage
ON PassBaggage.Passenger = Passenger.PassengerID
INNER JOIN Baggage ON Baggage.BaggageID = PassBaggage.Baggage
INNER JOIN BaggageTypes ON Baggage.BaggTypeID = BaggageTypes.BaggageTypeID
INNER JOIN Currency ON Currency.CurrencyID = Baggage.BaggCurrencyID
INNER JOIN Booking ON Booking.BookingID = Passenger.PassBookingID
INNER JOIN Flight ON Flight.FlightID = Passenger.PassFlightID
WHERE BookingID = 16 && FlightID = 36;
```

25.

```
SELECT FlightID,TypeOfEquip AS SportsEquipment, CurrencySign AS Currency,
SportsEquipPrice, BookingID, PassengerID, PassFirstName AS 'First Name', PassSurname AS Surname
FROM `Passenger` INNER JOIN PassSportsEquipment
ON PassSportsEquipment.Passenger = Passenger.PassengerID
INNER JOIN SportsEquipment ON SportsEquipment.SportsEquipmentID = PassSportsEquipment.Equipment
INNER JOIN SportsEquipTypes ON SportsEquipTypes.SporstTypeID = SportsEquipment.EquipTypeID
INNER JOIN Currency ON Currency.CurrencyID = SportsEquipment.CurrencyID
INNER JOIN Booking ON Booking.BookingID = Passenger.PassBookingID
INNER JOIN Flight on Flight.FlightID = Passenger.PassFlightID
WHERE BookingID = 7 && FlightID =10;
```

26.

```
START TRANSACTION ;

DELETE FROM Passenger WHERE `PassengerID` = 25;
DELETE FROM PassBaggage WHERE `Passenger` = 25;
DELETE FROM PassSportsEquipment WHERE `Passenger` = 25;
DELETE FROM BookingLineItem WHERE BookerID = 10 && LineItemDesc = 'Adult Ticket x1 - one way' ;
DELETE FROM BookingLineItem WHERE LineItemName =  'Seat' && LineItemDesc = 'Seat D7 Flight EZY1245' ;
SELECT * FROM `BookingLineItem` WHERE 1;
SET @sumTotal = (SELECT SUM(LineItemCost) FROM `BookingLineItem` WHERE`BookerID`=10);
UPDATE `Booking` SET `TotalPrice` = @sumTotal WHERE `Booking`.`BookingID` = 10;

COMMIT;
```