Flight Booking Database

The project brief was to reverse engineer a commercial flight booking website (in this instance easyjet.com) in order to model and implement a database system.

Due to time constraints and limited knowledge of databases and sql, the project focused solely on the flight booking process. Other items that are available via the EasyJet website such as insurance, car hire and holiday booking are outside the scope of this task.

The group and I began the project by scrutinising the EasyJet.com website in order to start the process of entity discovery. This allowed us to create an initial Entity Relationship diagram together before continuing individually for the rest of the project.

Functionality of the Site

The easyJet.com website allows a visitor to:

- Sign up for a user account
- Book and purchase a single flight or many flights from a pre-defined set of origin airports to another set of pre-defined destination airports.
- Manage, change and cancel their booking.

Design Challenges

- A user making the booking isn't necessarily a passenger.
- A user may have multiple bookings.
- A passenger will have a choice of bags, seating, food etc options which all have cost implications.
- One or many passengers can travel on a single booking.
- Bookings can be paid for in multiple currencies.

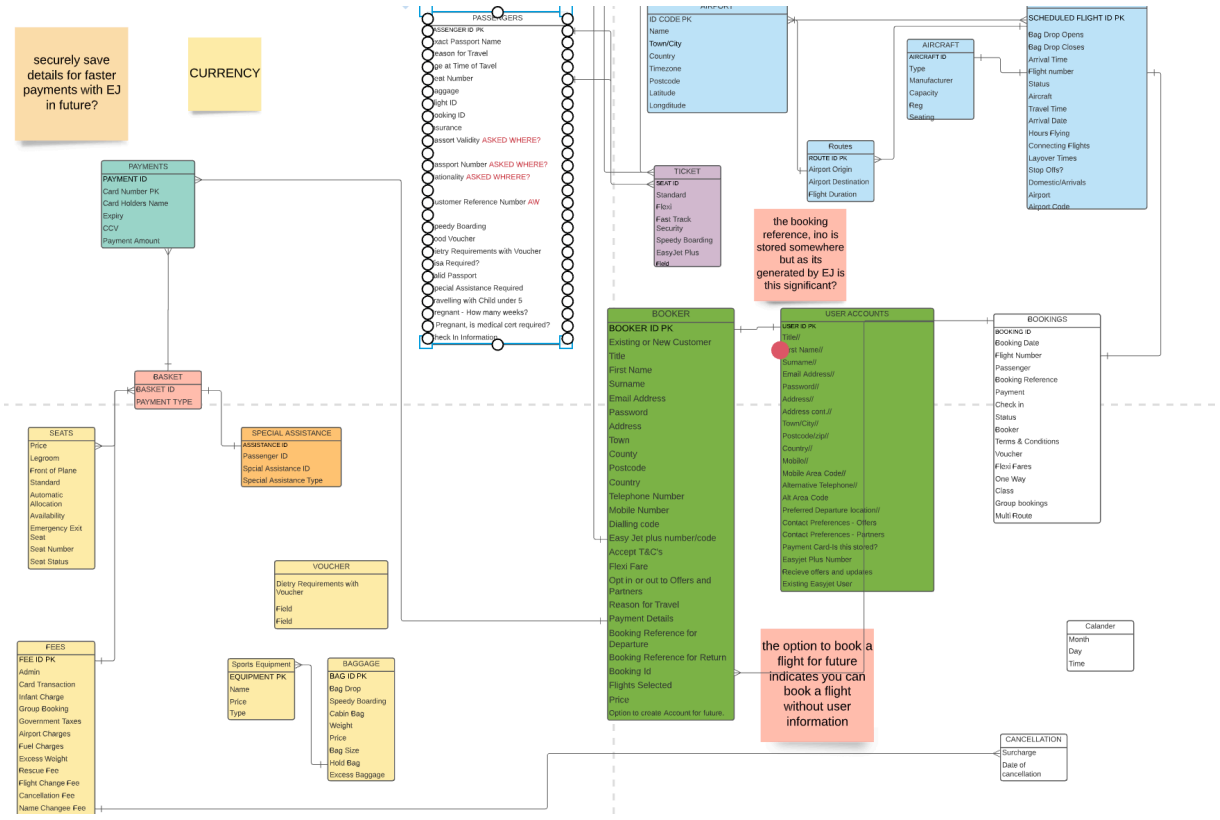


Figure 1 Original ERD compiled from Group Design

Figure 1 shows the original entities and attributes compiled by the group at the beginning of the project.

This model is cluttered with surplus fields and there is little flow to see how the stages of the booking are connected.

Figure 2 shows the connections of the main elements in the current model, which shows how I have allowed for a single user to have one or more bookings each of which can contain one or more passengers, seats and bagging choices.

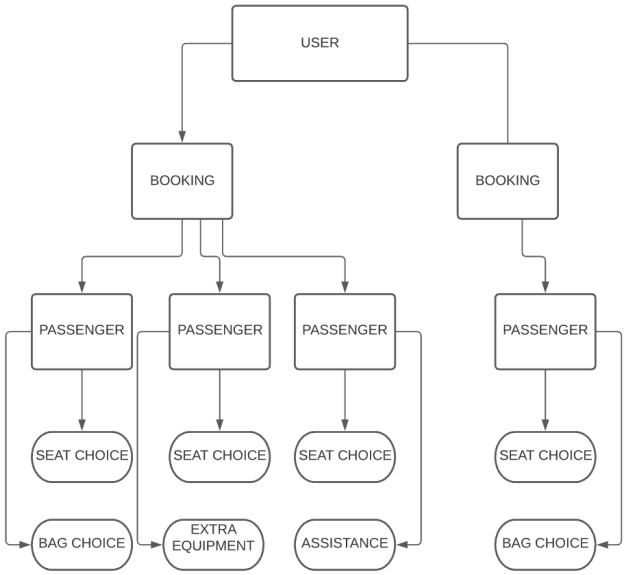


Figure 2

***** *I have designed the relationships to only allow one passenger per seat. The one exception to this is infants who share their parents seat. I have considered this an exception as they aren't awarded the same luggage allowance as a regular passenger.*

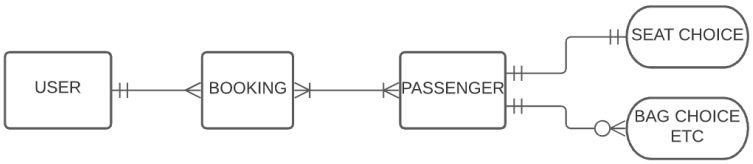


Figure 3

Figure 3 depicts the entity relationship process between the user, the booking, the passenger and their choices.

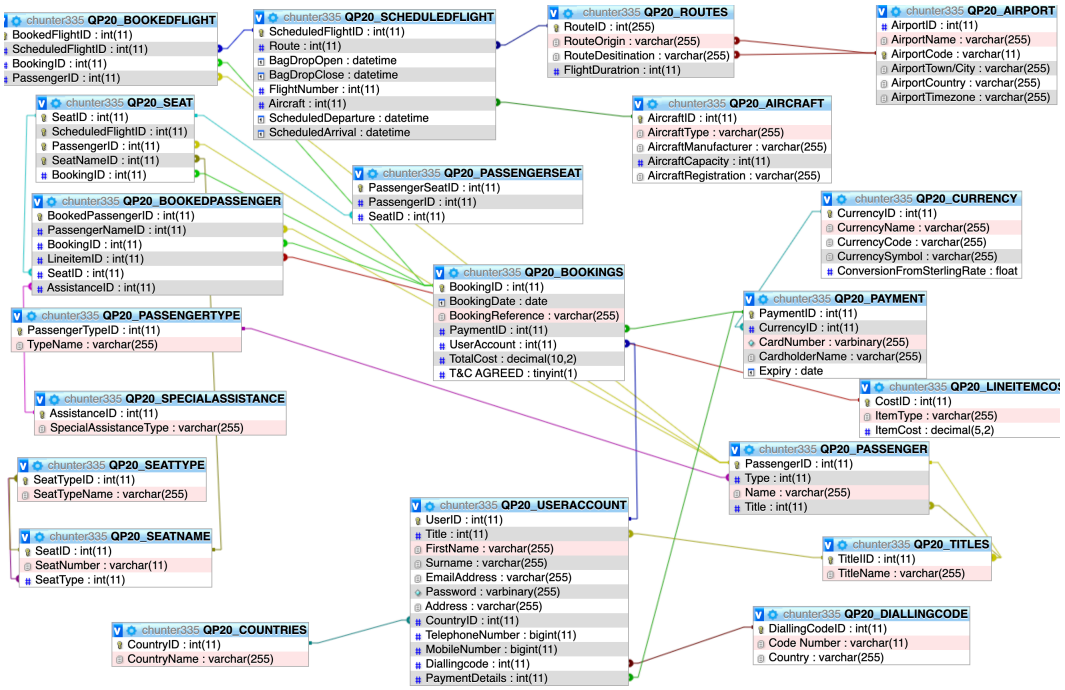


Fig 4 The finalised Diagram

The finalised design in comparison to the original group effort is significantly slimmed down. There are a lot less fields in each table, and many are iD'S from another table acting as a foreign key to link everything together.

User and Passenger

The ***UserAccount*** table holds the personal, contact and payment information for the user; In the real world these are saved and a profile is created. This allows a single user to create and bookings that can be accessed again or amended as they see fit.

The ***Titles, Countries*** and ***DiallingCode*** fields were normalised out in separate tables to minimise human error when inputting data from the front end but linked via a foreign key to the ***UserAccount*** table.

The ***UserAccount*** table is also constrained to the ***PaymentDetails*** table via a foreign key. This is not a mandatory storage field, so some users may choose not to save payment information.
Password field is stored as a varbinary as it is encrypted using AES_ENCRYPT and stores the byte strings rather than the raw values themselves. This allows for safer storage and the details can be decrypted when they are needed.
In submitting their email address and password choice, the user is creating their username and password key used to store their details for the future.

Options

	BookingID	BookingDate	BookingReference	PaymentID	UserAccount	TotalCost	T&C AGREED
Edit Copy Delete	5	2020-09-21		8	5	0	0
Edit Copy Delete	6	2020-07-12		8	5	0	0

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	UserID	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	Title	int(11)			No	None			Change Drop More
3	FirstName	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
4	Surname	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
5	EmailAddress	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
6	Password	varbinary(255)			No	None			Change Drop More
7	Address	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
8	CountryID	int(11)			No	None			Change Drop More
9	TelephoneNumber	bigint(11)			No	None			Change Drop More
10	MobileNumber	bigint(11)			No	None			Change Drop More
11	Diallingcode	int(11)			No	None			Change Drop More
12	PaymentDetails	int(11)			Yes	NULL			Change Drop More

Figure 5 UserAccount table

The central table in my database is the **Bookings** table. This models and represents the real world concept of a booking which holds all of the pertinent information for your itinerary. This table is linked to the **UserAccount** table via a foreign key constraint and also to the **BookedFlight** and **BookedPassenger** tables in similar fashion.

To allow for a one-to-many relationship between bookings and both flights and passengers, I created separate tables **BookedFlight** and ***BookedPassenger*.**

The **BookedFlight** table is then linked to the **ScheduledFlight** table which in turn is linked to the **Airport, Aircraft** and **Routes** table while the **BookedPassenger** table links to the **Seat, Voucher, Special Assistance** and **LineItemCost** tables. *Figure 7* demonstrates this more clearly in the form of an entity relationship diagram.

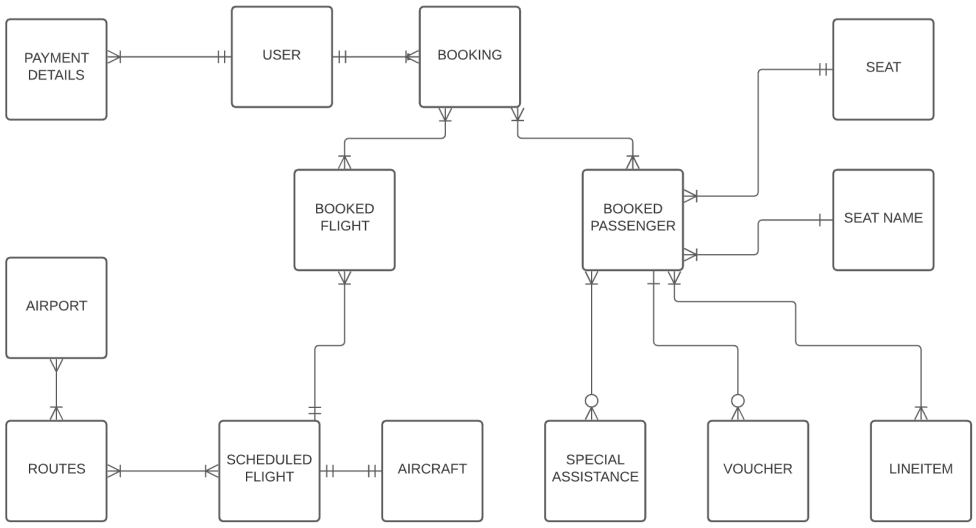


Figure 7 Booked Flight and Passenger ERD

When using the website, once a user has selected the cities they wish to travel between, they are shown all available flights within a chosen date range. This can be retrieved from the database by cross referencing the **Routes** table with the ***ScheduledDeparture*** and ***ScheduledArrival*** fields in the **ScheduledFlight** table. These fields use the 'datetime' data type and for the purposes of this project the timezone is assumed to be Universal Time Co-ordinated (UTC), regardless of the chosen origin and destination.

The same route may appear several times but each flight is designated unique status through the date time data type associated through the ***ScheduledDeparture*** and ***ScheduledArrival*** *fields.*

+ Options

ScheduledFlightID	Route	BagDropOpen	BagDropClose	FlightNumber	Aircraft	ScheduledDeparture	ScheduledArrival	RouteID	RouteOrigin	RouteDesitination
5	1	2020-11-25 10:00:00	2020-11-25 10:30:00	27036511	5	2020-11-25 11:00:00	2020-11-25 13:00:00	1	BFS	BCN
6	1	2020-11-25 14:00:00	2020-11-25 14:30:00	27036515	2	2020-11-25 15:00:00	2020-11-25 17:00:00	1	BFS	BCN

Fig 8 The SQL query result of a search between specific dates and certain cities.

On each ***RouteID*** in the **Routes** table, there is a connection to at least one or more entries in the **ScheduledFlight**t table. This is then linked to the **Aircraft** table through a foreign key constraint. The **Aircraft** table then contains the ***AircraftCapacity*** field, which can be compared to the total number of seats which do not have a null *PassengerID (meaning they have been booked)* to determine if there are any seats available on that specific scheduled flight.

+ Options

ScheduledFlightID	Route	BagDropOpen	BagDropClose	FlightNumber	Aircraft	ScheduledDeparture	ScheduledArrival	AircraftID
6	1	2020-11-25 14:00:00	2020-11-25 14:30:00	27036515	2	2020-11-25 15:00:00	2020-11-25 17:00:00	2
7	11	2020-12-23 16:00:00	2020-12-23 18:00:00	434270183	4	2020-12-23 18:30:00	2020-11-23 20:00:00	4
8	5	2021-01-16 16:00:00	2021-01-16 18:00:00	67827018	1	2021-01-16 18:00:00	2021-01-16 18:00:00	1
9	1	2021-01-17 19:14:27	2021-01-17 19:14:27	654324	4	2021-01-17 19:14:27	2020-11-25 21:14:27	4
10	4	2021-01-27 20:08:16	2021-01-27 20:08:16	765345	3	2021-01-27 20:08:16	2020-11-25 22:08:16	3
11	13	2021-02-23 15:46:09	2021-01-23 15:46:09	978765	5	2021-02-23 15:00:00	2021-02-23 18:00:00	5
12	3	2021-02-09 15:46:09	2021-02-09 15:46:09	432567	2	2021-02-09 15:46:09	2021-01-09 15:46:09	2

Fig 9 The result of an SQL query returning the flights of the passengerID'S which are not null AND less than the capacity of the aircraft.

The **BookedPassenge**r table has a relational link to the **PassengerType** table. The options in the **PassengerType** table are '*adult'*, '*child'* and '*infant'*. This is to ensure the ticket price is correctly allocated to each individual - this would be handled by the website. In the real world, adults travelling with an infant child under 2 years old will carry the child on their lap and so the infant traveller isn't assigned a seat ID. This presented a challenge when comparing the ***AircraftCapacity*** to the number of ***BookedPassenger*** *ID's* booked onto a specific flight, but If the ***PassengerType*** *field* is joined, we can search for ***BookedPassengerID*** of type - '*Adult*' to determine what seats are taken or available.

For every option a passenger has that may incur a cost I have created a **LineItemCost** table.

CostID	ItemType	ItemCost
1	Flight- Adult	60.00
2	Automatic Allocation Seat	9.00
3	Emergency Exit Seat	25.49
4	Standard Seat	6.00
5	Front of Plane Seat	25.49
6	Hands Free Baggage	7.00
7	Cabin bag	0.00
8	Hold bag <15kg	35.49
9	Sports Equipment Small	37.00
10	Excess Baggage per KG	12.00
11	Name Change	25.00
12	Flight Change	25.00
13	Sports Equipment Large	47.00
14	Hands Free Bundle	16.00
15	Infant charge	15.00
16	Child Flight	40.00
17	Hold bag <23 kg	42.49
18	Food voucher	7.50
19	Speedy Boarding	19.99

FIG 9 The LineItemCost table

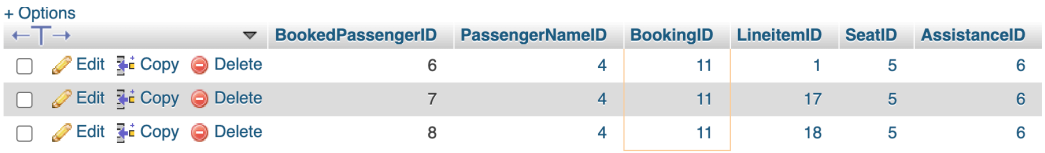
Each traveller will have a seat choice and in most instances a bag choice so this design will allow for a one-to-many relationship between the passenger and their choices.

These options are added to the **BookedPassenger** table via a constraint through the ***CostID,*** connected to the ***PassengerID****.*

The **LineItemCost** table will also cover outsized baggage and changes made to the booking at a later date so these can be stored and easily updated by adding or removing the ***LineItemID*** to the relevant booking.

In making a booking, a user may require physical assistance when they arrive at the airport. I have normalised this into a separate table called **SpecialAssistance**. We can see in *FIG 10* the ***AssistanceID*** field is also linked to the ***BookingID*** field. This corresponds to the

SpecialAssistance table, and each request is given an ID. For example, ID value 6 is connected toto ‘no help required’ rather than have a null value, which helps ensure the user has read about and is aware of the options available.



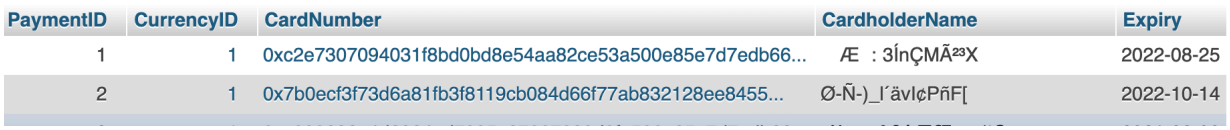
+ Options

	BookedPassengerID	PassengerNameID	BookingID	LineitemID	SeatID	AssistanceID
Edit Copy Delete	6	4	11	1	5	6
Edit Copy Delete	7	4	11	17	5	6
Edit Copy Delete	8	4	11	18	5	6

FIG 10 The result of an SQL query to show the everything connected to a specific booking ID

Payments

The ***cardNumber***, ***name*** and ***expiry*** fields in this table have been encrypted using the **AES_ENCRPYT()** function using separate secret keys for the separate entries. To comply with the Payment Card Industry Standards, the CCV has not been stored. This is not a designated storage field, so some users may choose not to save payment information.



PaymentID	CurrencyID	CardNumber	CardholderName	Expiry
1	1	0xc2e7307094031f8bd0bd8e54aa82ce53a500e85e7d7edb66...	Æ : 3ÍnÇMÃ²³X	2022-08-25
2	1	0x7b0ecf3f73d6a81fb3f8119cb084d66f77ab832128ee8455...	Ø-Ñ-)_Í´ävI¢PñF[	2022-10-14

Fig 11 The separate AES_encryptions working in the Payment Details table

The **PaymentDetails** table is linked to the **Currency** table, again by a foreign key constraint on the ***PaymentID*** field, using a **SUM()** query with a JOIN on the **BookedPassenger** table. This design allows for totalling up the cost of all the separate line items on the booking and select the chosen currency to pay in.

Having the line items as a decimal but not attached to a specific currency allows for a cleaner total cost, but it does have its limitations if the value of the chosen currency is drastically different from the origin price.

There is a field to choose the currency conversion rate and this could be completed using a sum query to bring back the **@LineitemTotalCost** variable.

Changing the Booking

A unique booking can be accessed from the ***BookingID*** field in the **Bookings** table, but also from the ***BookingReference*** field which would generate a unique number to give to the user. Using this key the user can update or change the passenger name, seat choice, bag choice or flight (availability permitting). This would be executed by updating corresponding line items attached to the booking and paying the potential fare difference.

Cancelling the Booking

The foreign key constraints from the **Booking, BookedPassenger ,BookedFlight** and **Seat** table are all set to 'cascade on delete' this means that when you delete information from one, it will delete all connected information in the linked tables.

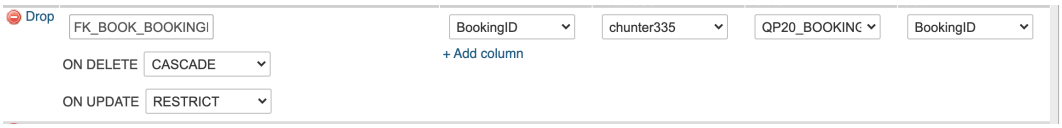


Fig 12 The On Delete selection

Improvements

In further iterations, the flight costing element of the design could be greatly improved upon. I have kept it very simplistic in the current design to show the model working, however in the real world, flight pricing would be based on a data reliant management system, dynamically changing the costs based on preempted human behaviour patterns. - eg :the more seats booked the higher the cost would be or certain time periods would be anticipated to be more popular on a Tuesday compared to a Friday so again a different cost to the standard base fare.

The payment info is also very basic in its function, and whilst it is encrypted with separate keys, this isn't particularly secure and should really be outsourced for a more complex encryption solution.

I don't feel it was very elegant in having the **SpecialAssistance** linked to the **BookedPassenger** table. The use of the **BookedPassenger** table was to allow many passengers to travel on one booking, or for one passenger to have multiple line items associated with their booking, so if they had multiple bags, a food voucher, and a sports item you had to declare the assistance every time, where as if it were connected to the booking, it would be declared once and not contain redundant data.

I would like to develop further the seat selection. Currently my design is limited to the availability of a seat on the whole plane, whereas I would like to be able to see the specific seats that are free, as this would give the user an instant price back for the seat type - which may affect their decision in purchasing their ticket.

Conclusion

The design meets the requirements of the brief given the time constraints. One user may book multiple bookings containing one or more passengers each with a separate choice for bag seat and Special Assistance.

The user can update and amend the booking at a later date, and pay using various currencies.

It has met the requirements, though is somewhat limited in capacity to what extent it can do.

I would like to pursue further the extra options such as the ‘Flexi seat’ with automatic updates in certain fields or the ‘Easyjet Plus’ members club and benefits to see how that impacts the database.

Appendix

DB query 1:

```
SELECT
        *
    FROM
        `QP20_SCHEDULEDFLIGHT`
    INNER JOIN `QP20_AIRCRAFT`
    ON QP20_SCHEDULEDFLIGHT.Aircraft = QP20_AIRCRAFT.AircraftID
    WHERE (
        SELECT COUNT(*) FROM QP20_SEAT
        WHERE QP20_SEAT.PassengerID IS NOT NULL
        AND QP20_SEAT.ScheduledFlightID = QP20_SCHEDULEDFLIGHT.ScheduledFlightID
        )
        <
        QP20_AIRCRAFT.AircraftCapacity


    AND QP20_SCHEDULEDFLIGHT.ScheduledFlightID in (


                    SELECT QP20_SCHEDULEDFLIGHT.ScheduledFlightId FROM
`QP20_SCHEDULEDFLIGHT`
                    inner JOIN QP20_ROUTES
                    on QP20_SCHEDULEDFLIGHT.Route = QP20_ROUTES.RouteID
        WHERE QP20_ROUTES.RouteOrigin = 'BFS' AND QP20_ROUTES.RouteDesitination =
'BCN'
    )


    AND QP20_SCHEDULEDFLIGHT.ScheduledDeparture Between '2020-11-25' AND
'2020-11-27';
```

DB Query: 2

```
SELECT * FROM QP20_SEAT
```

```
WHERE QP20_SEAT.ScheduledFlightID = 6

AND QP20_SEAT.PassengerID IS NULL
```

DB QUERY 3:

```
 INSERT

    INTO

      QP20_USERACCOUNT (`Address`, `CountryID`, `Diallingcode`, `EmailAddress`, `FirstName`, `MobileNumber`, `Password`, `PaymentDetails`, `Surname`, `TelephoneNumber`, `Title`, `UserID`)


       VALUES('13 YELLOW BRICK ROAD', 3, 1, 'OZ@yahoo.com', 'Dorothy', '0827367376747', AES_ENCRYPT('myPass','mySecretKey'), null, 'OZ', '73776576467484', 2, 0)
```

DB Query: 4

```
INSERT INTO QP20_PAYMENT (`PaymentID`, `CurrencyID`, `CardNumber`, `CardholderName`, `Expiry`)

VALUES (0, 1, AES_ENCRYPT('556467546','mySecretPassword'), AES_ENCRYPT('MS DOROTHY OZ','mySecretName'), '2022-05-25')
```

DB QUERY 4.1

```
 INSERT INTO QP20_BOOKINGS (`BookingID`,`BookingDate`,`BookingReference`,`PaymentID`,`UserAccount`,`TotalCost`,`T&C AGREED`)

VALUES (0, '20/12/01', 'WIZZ400', '11','17',NULL, '1')
```

DB Query:5

```
INSERT INTO QP20_PASSENGER(`PassengerID`,`Type`,`Name`,`Title`)
```

VALUES (0, 2, 'Dorothy OZ', 1)

DB Query:6

UPDATE QP20_SEAT

SET QP20_SEAT.PassengerID = 30

WHERE QP20_SEAT.SeatID = 26

AND QP20_SEAT.ScheduledFlightID = 6

- DB Query :7 to update BookedPassenger:

START TRANSACTION;

INSERT INTO QP20_BOOKEDPASSENGER
(`BookedPassengerID`,`PassengerNameID`,`BookingID`,`LineitemID`,`SeatID`,`AssistanceID`)

VALUES (0, 28, 14, 1, 25, 6),

(0, 28, 14, 5, 20, 6),

(0, 28, 14, 7, 20, 6),

(0, 28, 14, 17, 20, 6);

COMMIT;

-DB Query 8 to update BookedFlight

INSERT INTO
QP20_BOOKEDFLIGHT(`BookedFlightID`,`ScheduledFlightID`,`BookingID`,`PassengerID`)

VALUES(0, 6, 14, 28)

DB QUERY : 9 to retrieve an itinerary of information from the booking - in script.

SELECT BookingReference, TotalCost, QP20_BOOKINGS.BookingID, QP20_PASSENGER.Name,
QP20_LINEITEMCOST.ItemType, QP20_LINEITEMCOST.ItemCost,

```
QP20_SEATNAME.SeatNumber, QP20_SCHEDULEDFLIGHT.FlightNumber,
QP20_ROUTES.RouteOrigin, QP20_ROUTES.RouteDesitination

FROM QP20_BOOKINGS

INNER JOIN QP20_BOOKEDPASSENGER

ON QP20_BOOKINGS.BookingID = QP20_BOOKEDPASSENGER.BookingID

INNER JOIN QP20_PASSENGER

ON QP20_BOOKEDPASSENGER.PassengerNameID = QP20_PASSENGER.PassengerID

INNER JOIN QP20_LINEITEMCOST

ON QP20_BOOKEDPASSENGER.LineitemID = QP20_LINEITEMCOST.CostID

INNER JOIN QP20_BOOKEDFLIGHT

ON QP20_BOOKINGS.BookingID = QP20_BOOKEDFLIGHT.BookingID

INNER JOIN QP20_SEATNAME

ON QP20_BOOKEDPASSENGER.SeatID = QP20_SEATNAME.SeatID

INNER JOIN QP20_SCHEDULEDFLIGHT

ON QP20_BOOKEDFLIGHT.ScheduledFlightID = QP20_SCHEDULEDFLIGHT.ScheduledFlightID

INNER JOIN QP20_ROUTES

ON QP20_SCHEDULEDFLIGHT.Route = QP20_ROUTES.RouteID

WHERE QP20_BOOKINGS.BookingID = 13
```

DB QUERY 10 :Change a users booked flight from 12 to 13 (Ryan Marrow) and add a line item to charge for it

```
START TRANSACTION;

/* Update passengers scheduled flight for the booking */

UPDATE QP20_BOOKEDFLIGHT

SET QP20_BOOKEDFLIGHT.ScheduledFlightID = 13

WHERE QP20_BOOKEDFLIGHT.PassengerID = 8

AND QP20_BOOKEDFLIGHT.BookingID = 13;

/* Delete the charge for the previous seat */

DELETE FROM QP20_BOOKEDPASSENGER

WHERE QP20_BOOKEDPASSENGER.BookingID = 13

AND QP20_BOOKEDPASSENGER.LineitemID = 4

AND QP20_BOOKEDPASSENGER.PassengerNameID = 8;

/* Assign passenger to new seat */

UPDATE QP20_SEAT

SET QP20_SEAT.PassengerID = 8

WHERE QP20_SEAT.SeatID = 23;

/* Add line items to booking to charge for flight change and new seat */

INSERT INTO QP20_BOOKEDPASSENGER(`BookedPassengerId`, `PassengerNameID`,
`BookingID`, `LineItemID`, `SeatID`, `AssistanceID`)

VALUES (0, 8, 13, 12, 23, 6),

(0, 8, 13, 5, 23, 6);

COMMIT;
```

DB QUERY 11: To delete a booking

```
delete from QP20_BOOKINGS where QP20_BOOKINGS.UserAccount = 5
```