EASYJET DATABASE REPORT

Matthew Johnston

Student Number: 40247362
mjohnston77@qub.ac.uk

School of Electronics, Electrical Engineering & Computer Science

Queen's University Belfast

A signed and completed cover sheet must accompany the submission of the Individual Software Development dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Declaration of Academic Integrity

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.
6. Software and files are submitted on a memory stick.
7. Journal has been submitted.

I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.

I am aware of the disciplinary consequences of failing to abide and follow the School and Queen's University Regulations on Plagiarism.

Name: (BLOCK CAPITALS) **MATTHEW JOHNSTON**

Student Number: **40247362**

Student's signature **Matthew Johnston** *Date of submission* **23.11.18**

Introduction

EasyJet is a British low-cost passenger airline. It operates domestic and international flights in more than 30 countries worldwide

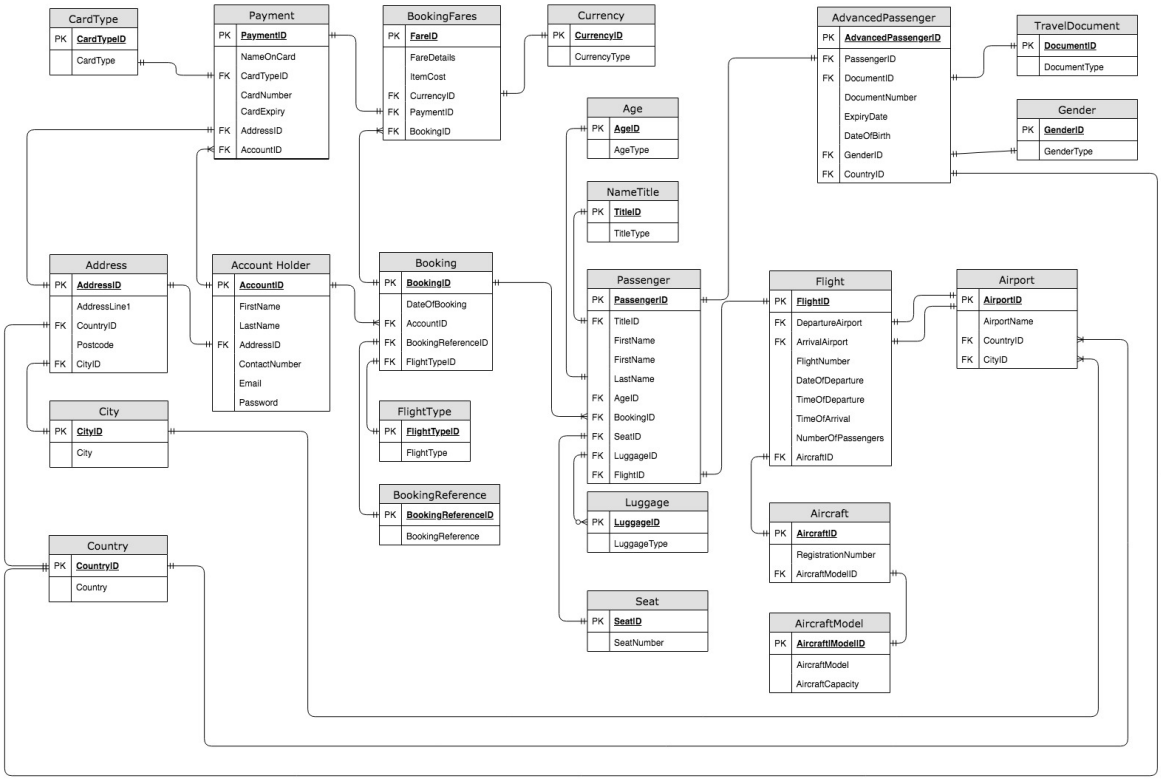
In this report I have detailed a database based on the booking system of the easyjet.com website. This starts with working back from what information EasyJet require to book a flight and what I also think would be contained in the database after that. From this I will figure out the different entities, attributes and the relationships between them and build a database focused on the process of booking a flight. There were other areas such as hotels and car rentals, but they are out of scope for this project.

I began what I will call the 'investigative stage' of this process as part of a group. We each focused on different areas of the website to spread out the work load. This was a learning process as none of us had much of an idea of what we were really doing but as we saw what each person was bringing back we were able to see similarities and that put us on the right track.

We made a quick ER diagram of what we thought different entities would be and some of the attributes associated with it (Figure 1). After this we took a few sections each to try and thoroughly assess what Attributes would go with each Entity. We began to compile these in tables in a word document, this enabled us to start to look at foreign keys and try to bring our tables together (Figure2). This was the last work we completed as a group.

In the group ER Diagram the bits that a user would actually go though (account, passenger, booking etc) are most of what ended up being in the final database, this is because we could see that information on the EasyJet website. The area to do with the actual flight took more work to get it to a functional standard. It is also clear in this diagram that we didn't understand foreign keys or normalization yet. This can be seen in the 'Card Type' and 'Booking' tables, where instead of just an incrementing single row we had it where it would have been a list full of Yes and No's next to a specific type. There are cases throughout the group diagram where tables are repeated. The 'Add on Fare' is what would just become the entire cost table, whereas the cost tables were dropped as again they are just a list of what would be Yes and No's. This is where I began looking at what tables could go and what normalisation I could do.

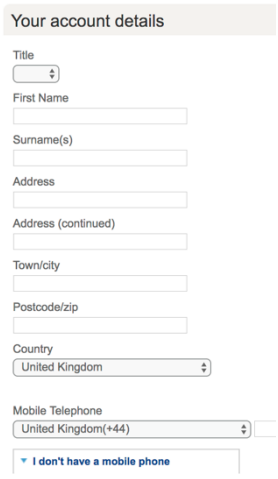
ER Diagram



The above image is what my final ER Diagram looks like. It includes the Attributes of each Entity as well as the relationships and the relationship types.

The main areas are Account Holder, Booking, Passenger and Flight. The rest of the tables branch out using foreign keys.

The account holder area consists of the main account table then an address table that is linked to it using a foreign key. Within the address table there are also foreign keys to tables for Country and City. Below shows how these tables reflect on the sign up process of the EasyJet site.



#	Name
1	AccountID
2	FirstName
3	LastName
4	AddressID
5	ContactNumber
6	Email
7	Password

#	Name
1	AddressID
2	AddressLine1
3	CityID
4	Postcode
5	CountryID

#	Name
1	CityID
2	City

#	Name
1	CountryID
2	Country

The next main area of focus is the Booking table, within it there is a FlightType and a BookingReference. The Booking table connects to the AccountHolder using a foreign key.

#	Name
1	BookingID
2	DateOfBooking
3	AccountID
4	BookingReferenceID
5	FlightTypeID

Name
BookingReferenceID
BookingReference

#	Name
1	FlightTypeID
2	FlightType

From the Booking table the BookingFares connects to it using a foreign key. The BookingFares then connects to the Payment table which then connects to a CardType table. The Payment table also connects back to the AccountHolder table using a foreign key.

#	Name
1	FareID
2	FareDetails
3	ItemCost
4	CurrencyID
5	BookingID
6	PaymentID

#	Name
1	CurrencyID
2	CurrencyType

#	Name
1	PaymentID
2	NameOnCard
3	CardTypeID
4	CardNumber
5	CardExpiry
6	AddressID
7	AccountID

#	Name
1	CardTypeID
2	CardTypeName

Payment card

Visa debit
Matthew Johnston
3224 - 06/18 **EXPIRED**

The next main area is the Passenger table. It is made up mostly of foreign keys. There are areas of the Passenger table I have chosen not to include such as special assistance and EasyJet plus. I decided these were too large and I put them out of scope. The advanced passenger table is connected to the Passenger table. This table would be used for items like passports for international flights. This was the first area where I wasn't sure what would definitively be in the table as I wasn't able to input this data on the website, but I feel like I have a good representation of what it could be.

#	Name
1	PassengerID
2	TitleID
3	FirstName
4	LastName
5	AgeID
6	BookingID
7	SeatID
8	LuggageID
9	FlightID

#	Name
1	AgeID
2	AgeType

#	Name
1	LuggageID
2	LuggageType

#	Name
1	SeatID
2	SeatNumber

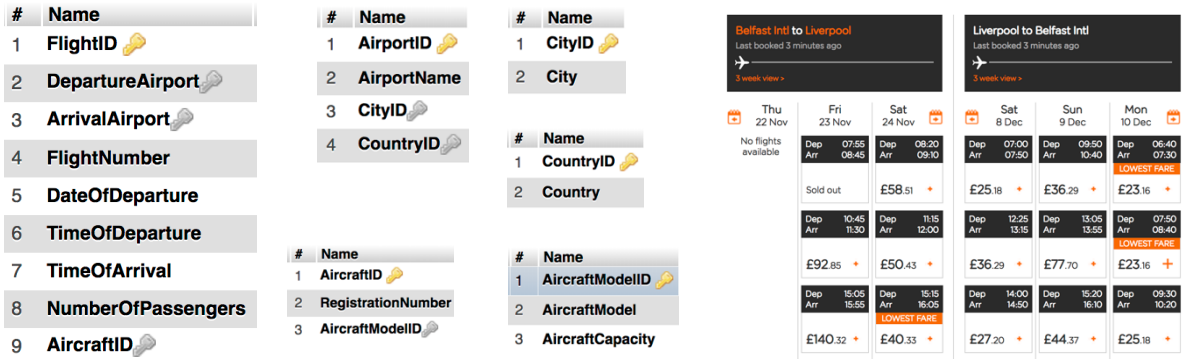
#	Name
1	TitleID
2	TitleType

#	Name
1	AdvancedPassengerID
2	PassengerID
3	DocumentID
4	DocumentNumber
5	ExpiryDate
6	DateOfBirth
7	GenderID
8	CountryID

#	Name
1	GenderID
2	GenderType

#	Name
1	DocumentID
2	DocumentType

The final main area is the Flight table. This table contains the information that you would select when you are selecting where you are flying from and to, then the rest of the data I assume is assigned to it by EasyJet. The flight table contains an AircraftID which comes from a table that also contains an AircraftModelID. The DepartureAirport and ArrivalAirport come from a table called Airport that contains each airport name then they are also connected to the Country and City table. The flight area is where I can't really look into the EasyJet website and I'm going from information from previous bookings and images of boarding passes online.



Primary Keys

The primary key is the fundamental area which a relational database is formed, without it the tables would not link to or reference each other. Primary keys are important to uniquely identify tables, which comes into play where data could be repeated such as names or ages. Each table contains an ID corresponding to the table that it is in. An example is the AccountHolder contains an AccountID, which can in turn identify the information related to the account holder. These keys are set to an incrementing unique value throughout and can then be linked to other tables by referencing them as a Foreign Key. They always have a unique value and are never set to null.

Foreign Keys

A Foreign key is used to cross-reference a Primary key in another table where you want to link the two together. These use the Unique identifier of the Primary key so data can be easily linked together while only referencing one thing. Foreign keys have a cardinality between the tables such as one to one, one to many and many to many. An example of one to one is how a booking can only have one payment, as seen in the connection between my BookingFares and Payment tables which are connected by the PaymentID. An example of a one to many relationship would be how one Account Holder can make many bookings, again this can be seen between my AccountHolder and Booking table which are connected by the AccountID in the booking table.

Data Types

Int

I have used the data type int with the value set as 11 for all Primary and Foreign keys within the table. This is good for this because it can only be set as a number and ensures a letter cannot accidently be added. I could have used uniqueidentifier if I needed to make sure my keys were unique across my entire schema, but for the size of my database I don't think this is necessary. I was going to use int in areas that only contain numerical value such as card number but decided against this as some people type it with spaces or dash and this would come back with an error when the data entered was correct.

Varchar

I have used varchar set to 255 is the datatype in all other fields in my database. It is useful for this as it is big enough to enter any data that I need to enter. I used it where I was adding time and date over the time and date datatypes as I wanted to use the styling format that suited me. In the BookingFare table I used varchar on the cost as with varchar I can do a SUM query to add up all the fares set to a bookingID. Varchar is useful over the text datatype because you can choose the size of the text and limit the memory that can be used.

If I had added a description section into the BookingFare table to elaborate on the item booking I would have used a larger datatype such as Blob.

Design

I will begin to highlight the key changes I made in my database from what we had come up with in the group diagram.

The first area I removed was the Schedule table and combined it with the flight table. I made this decision because the two tables were the same and it was just a repeat of data.

I removed the Terminal table as I feel this would not be stored in a database but would be generated on the day at the airport. Some flights may have terminals that they usually use but if there was any maintenance being done that day and another terminal used that would conflict with the terminal stored in the database for that flight.

I removed the Check-in Table because again that would be an action at the airport or online, but it would not be stored in the database. This in an instance of an action for has or hasn't checked in and not actual information.

I have combined the Add on Fares, Flight cost, Luggage cost and Seat cost into one table under BookingFares as a line item table where each row lists the item and price and the

BookingID, the final price can be calculated using the SUM query. I don't know how I would have added it from four different tables.

In the Password field within the AccountHolder table and the CardNumber and ExpiryDate in the Payment table I have encrypted the information using the MD5 feature as this information should not be readable. This is an example of encryption to show that it exists. As MD5 has been cracked in a real-world scenario I would use AES as it is industry standard.

Normalisation

I began the normalisation process in the AccountHolder table where I took the Address table that had previously been attached to payment and linked to AccountHolder. I then further normalised the Address table by adding the City and Country tables. This stops instances of cities and countries being repeated within a table.

I normalised the Booking table firstly by adding a FlightTypeID that linked to a FlightType table that contains the option for One Way or Return Flight. Before this One Way and Return were listed as Attributes and would have been a yes or a no and did not make sense in the database. I also added a BookingReferenceID that linked to a BookingReference table that contained the booking reference where a list of generated booking references are contained.

The last example of normalisation that I will give is in my Flight table. I have assigned AirportID to the Departure and Arrival airports which has been further normalised with a CityID and a CountryID linking to the City and Country tables. In the Flight table there is also an AircraftID linking to the aircraft table that has been further normalised to link to the AircraftModel table using the AircraftModelID.

Reflection

Within this database if I were to do it again there are things I would do differently. Firstly, I would include Special Requirements in the Passenger table as it gives a good example of a many to many relationships as many passengers within a booking can have many special requirements such as wheel chair access and a nut allergy. I would adjust how the foreign keys are in some tables as I think some may be backwards as I'm having to enter data backwards, such as AddressID in AccountHolder. I missed the Title table out of passenger that is required in the EasyJet booking process. I think my luggage area needs work because as it stands a passenger can only have one bag in the hold. I would do this as a line item table, similar to how I've done my BookingFare table. In the AccountHolder table I wouldn't have linked City to it. If it were to be a drop down on the EasyJet website there would have to be a list of every Town and City that EasyJet fly from, I would just have this as an input field.

Business Intelligence Report

For an airline to get the longest use out of each of their aircrafts they must ensure flights are being distributed evenly between the fleet. If one airplane is being used more than all the others it will reach the end of its lifespan before the rest.

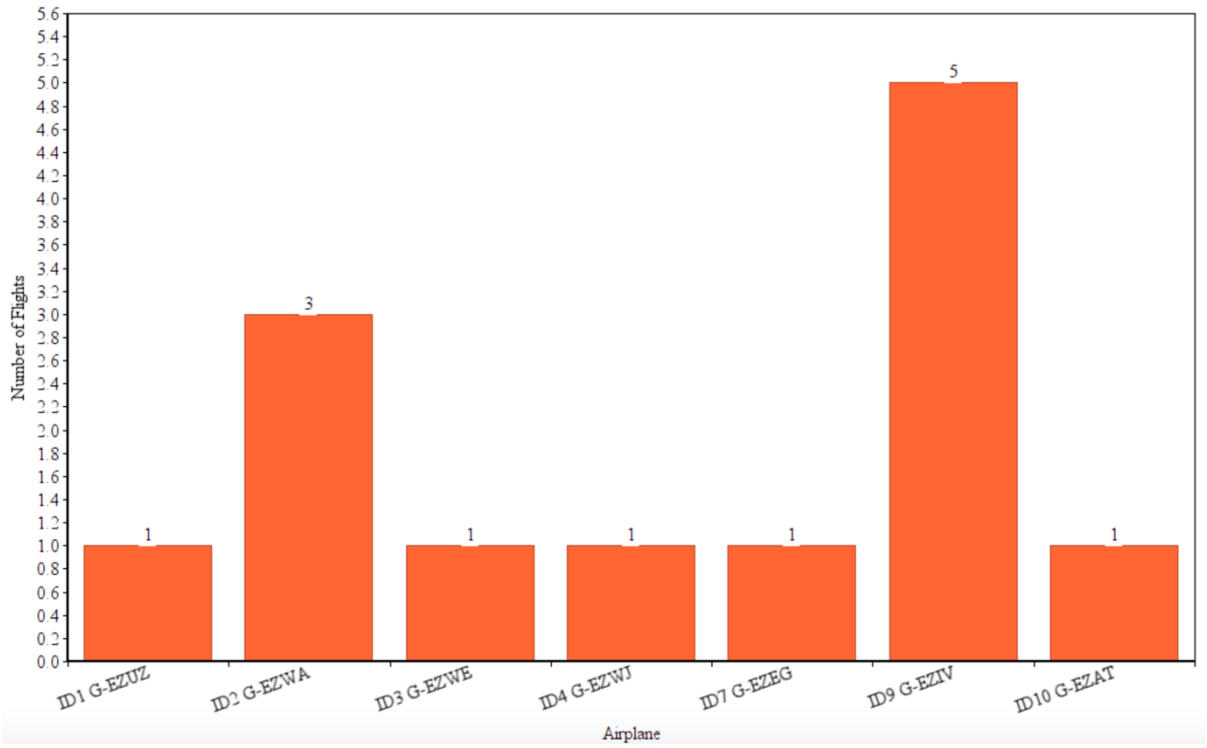
To check flight distribution within my database I have used the following sql query:

```
SELECT AircraftID, COUNT(*) FROM EJ_Flight GROUP BY AircraftID ORDER BY
`COUNT(*)` DESC
```

It returns a result that looks like this:

AircraftID	COUNT(*) 1
9	5
2	3
1	1
3	1
4	1
7	1
10	1

From this result set you can see the AircraftID which is unique to each aircraft and the number of flights it has been on in the database. This information is represented in a bar chart below.



This information could also be used to schedule deep cleaning or maintenance.

Appendix.

Figure 1.

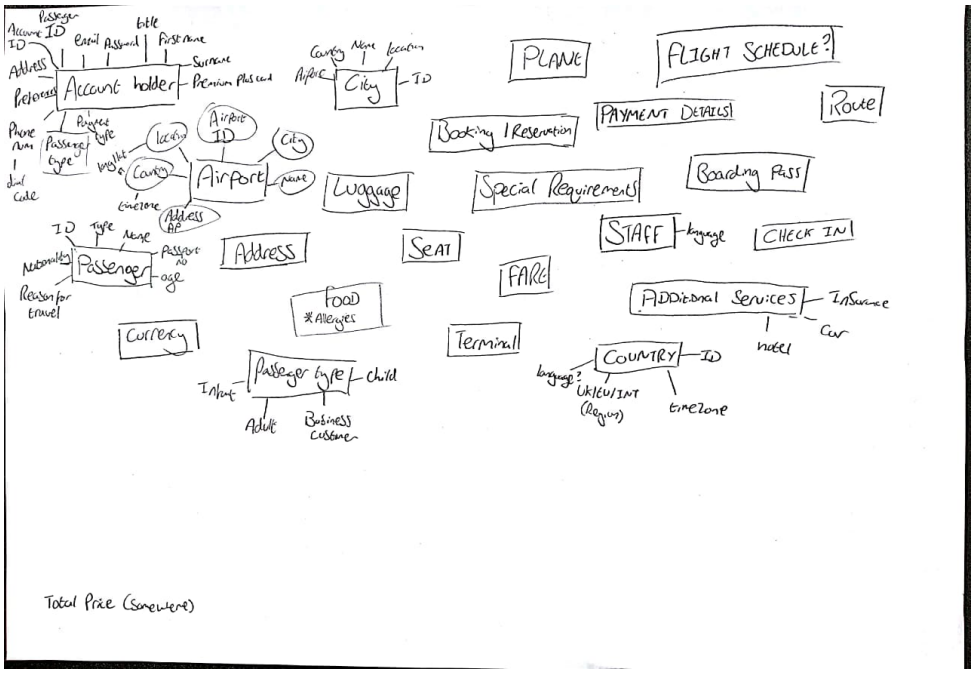
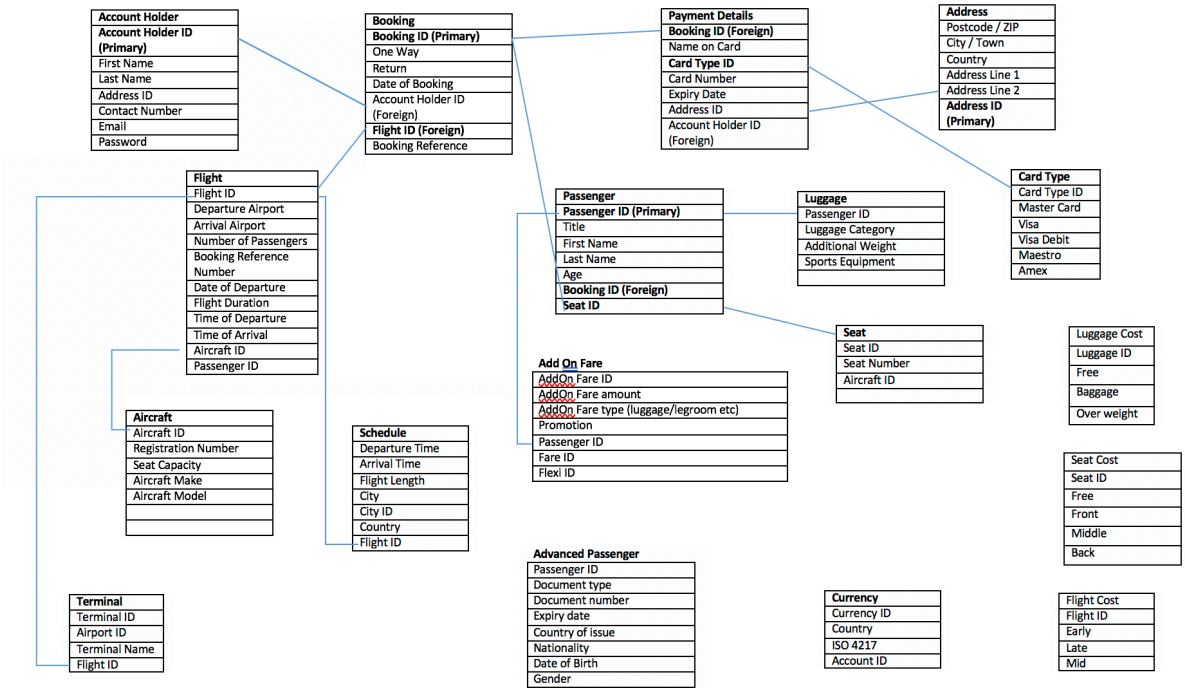


Figure 2.



// BI Query Most reoccurring aircrafts

```
SELECT
   AircraftID,
   COUNT(*)
FROM
   EJ_Flight
GROUP BY
   AircraftID
ORDER BY
   `COUNT(*)`
DESC
```

//Join flight and Aircraft table on AircraftID using inner join

```
SELECT
   *
FROM
   EJ_Flight
INNER JOIN EJ_Aircraft ON EJ_Flight.AircraftID = EJ_Aircraft.AircraftID;
```

// Find all passengers on a flight under one booking with date of booking.

```
SELECT
   *
FROM
   EJ_Passenger
INNER JOIN EJ_Booking ON EJ_Passenger.BookingID = EJ_Booking.BookingID
WHERE
   EJ_Passenger.BookingID = '6';
```

// to calculate the total cost of a booking

```
SELECT
   SUM(ItemCost)
FROM
   `EJ_BookingFares`
WHERE
   BookingID = 6;
```

// Find where a single item cost was over £100

```
SELECT
   FareDetails,
   ItemCost
FROM
   `EJ_BookingFares`
WHERE
   ItemCost > 100
ORDER BY
         ItemCost DESC
```

//Insert Query for add new passenger

```
INSERT INTO `EJ_Passenger`(
    `PassengerID`,
    `TitleID`,
    `FirstName`,
    `LastName`,
    `AgeID`,
    `BookingID`,
    `SeatID`,
    `LuggageID`,
    `FlightID`
)
VALUES(
    55,
    1,
    'Ryan',
    'Richardson',
    1,
    2,
    5,
    2,
    8
)
```

// insert new table

```
CREATE TABLE EJ_SpecialAssistance(
    AssistanceID INT(11) AUTO_INCREMENT,
    AssistanceType VARCHAR(255),
    PRIMARY KEY(AssistanceID)
)
```

// fill table

```
INSERT INTO `EJ_SpecialAssistance` (`AssistanceID`, `AssistanceType`) VALUES (NULL, 'Peanut Allergy'),
(NULL, 'Penicillin Allergy'), (NULL, 'Blind'), (NULL, 'Deaf'), (NULL, 'Learning Disability'), (NULL, 'Guide Dog
Assisted'), (NULL, 'Amputee');
```

// DROP TABLE

```
DROP TABLE EJ_SpecialAssistance
```