Daniel McAuley - 40125497

# Database Design Report – CSC7052

## Introduction

This project aims to reverse engineer the database of a popular online flight booking system, EasyJet.com.

By examining the front end of the website, as a group the process of entity discovery could begin. Once the main entities and their attributes where identified, a first sketch of an Entity Relationship Diagram (ERD) could be drawn up as a group (*Fig 1.1*). After several iterations of the ERD, the tables where normalised do reduce redundant data and relationships between entities where discussed (*Fig 1.2*).

Next, PHPMyAdmin was used to independently create the database. Once all the tables had been set up with the appropriate foreign keys constraints, sample data was inputted and SQL queries where ran to test these relationships (*Fig 1.3*).

For this project, it was decided that the focus would only be within the scope of a flight booking system. Insurance, hotel bookings, car rental and other additional services where not included.

This report will give an insight into the core entities and relationships within the database, design decisions, an explanation of the foreign key constraints, normalisation justification and improvements that could be made. SQL query results are provided on a regular basis as evidence to show the database working as intended. For reference, normalisation, which is referred to throughout the report, is the process of organising the fields and tables in a database to minimise data repetition (redundancy).

## Initial Design and Challenges

An Entity Relationship (ER) model is a way that allows the reader to visualise how a real-world database system would work. The relationships can be seen from the lines between each of the main entities. The main relationships in an ERD are one to one, one to many and many to many, all of which appear at some point throughout the database design.

Some of the core requirements regarding a flight booking system are to allow:

- A Booker to book one or more passengers a seat on a flight
- One passenger to carry one or more bags on to their flight
- Every seat on a flight to be assigned up to one passenger but allowing passengers to book seats on more than one flight (i.e. the departing and returning flight).
- Many flights to take the same one route, all of which have multiple airports/ destinations.
- Each booking to have many different costs related to the one booking, allowing a total price to be charged at the end of the booking process.

A simple sketch was designed as a group to begin to allow the visualisation of the booking process and to help identify some of the main entities, where the rectangles present the entities and the diamonds represent the relationships.

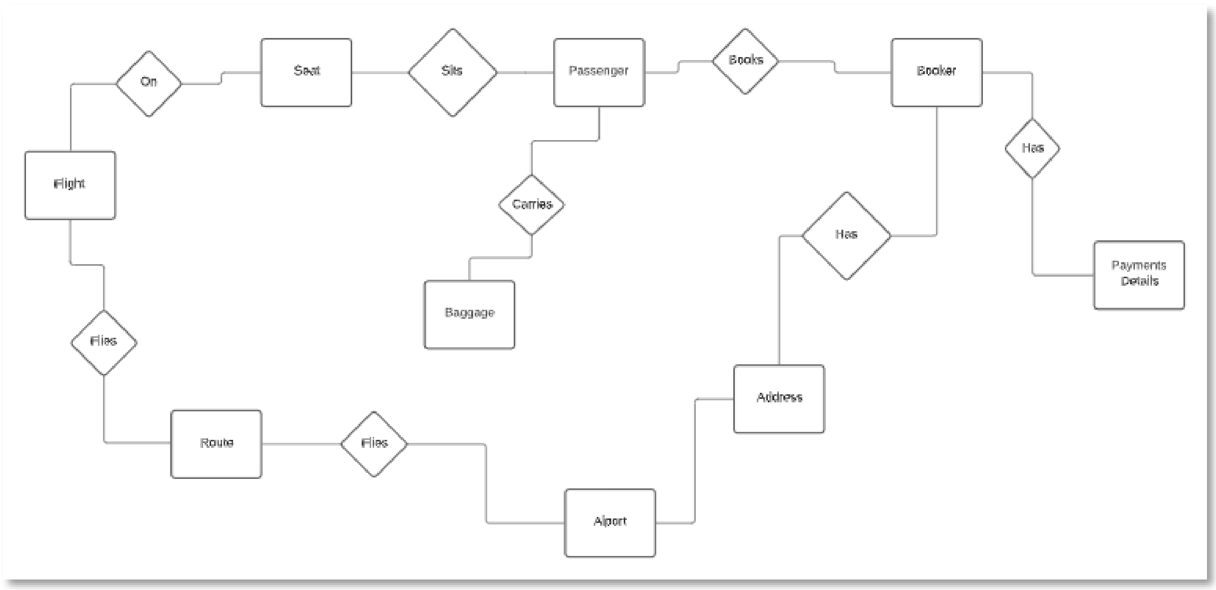


***Fig 1.1:*** *First group design of the Entity Relationship Diagram*

After multiple iterations, the group design was fleshed out and a second version of the diagram was agreed upon as seen in *Fig 1.2*. Through the discovery process, the attributes that where linked to each entity, could be placed into the appropriate tables, expanding the diagram, and giving an idea of how the database may look.

At each stage, the core concept of getting a passenger onto a seat on a flight remained at the forefront. Some normalisation was carried out at this stage, like the **passenger type**, **passenger title**, **booker details** and the **address**. This decision was made as this data would appear many times when adding information into the database.

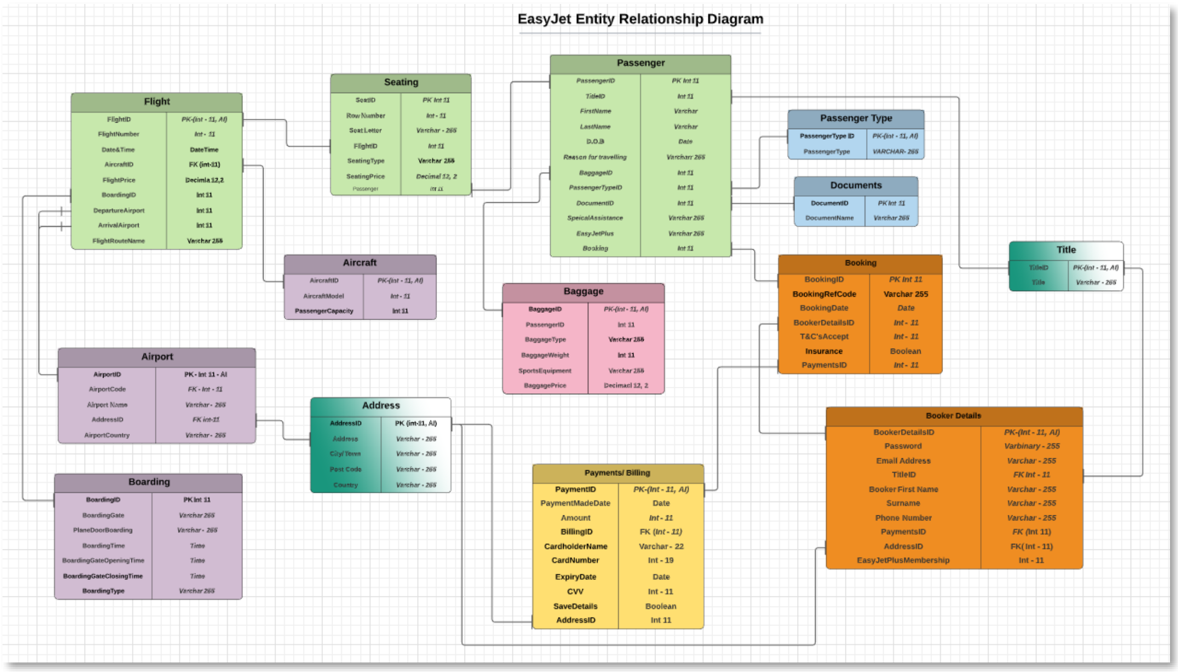


***Fig 1.2:*** *Final group iteration of the ERD*

After several weeks of design with the group, induvial changes were made and some of the aspects of the database were altered or refined, giving a final version of the Entity Relationship Diagram, emulating the EasyJet.com website. This is the version that was built in PHPMyAdmin and will be the one referred to in this report, unless stated otherwise.

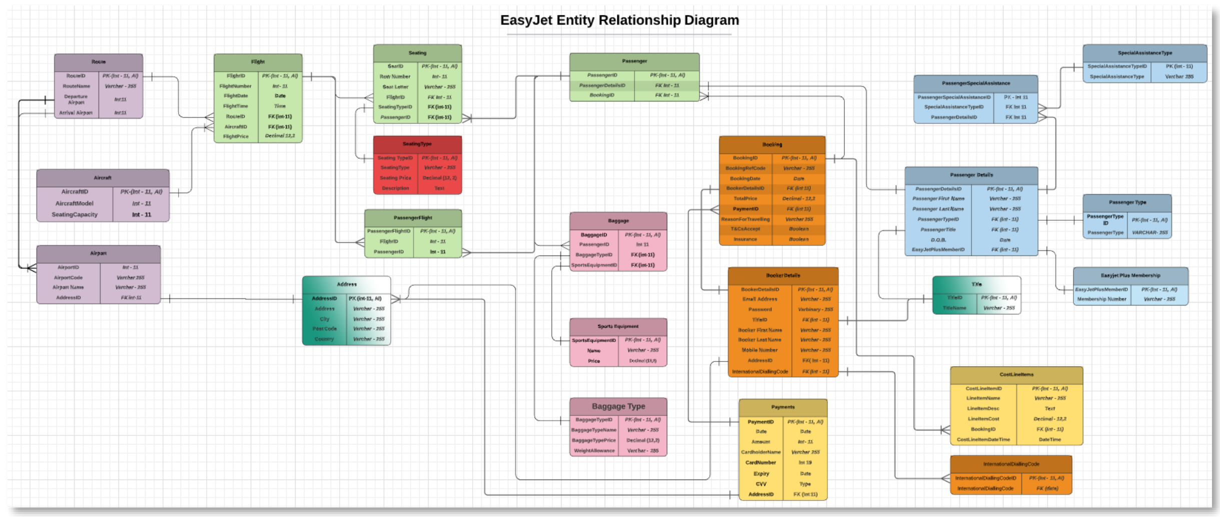


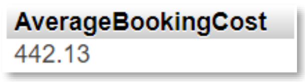
***Fig 1.3:** Final individual version of the ERD built in PHPMyAdmin*

## Booking

One of the main entities of the flight booking system is the **booking** table. It is linked to several other important tables like **passenger**, the **line item** table which allows for all the costs associated with a particular booking to be seen in a list format, the **payments** entity and finally the **booker details** entity.

The aim of the booker table is to store the information about the *booking reference code*, the *booking date* and the *total price* along with additional details like *reason for travelling*, *terms and conditions* and if there is *insurance* with the booking, stored as data types varchar, date, decimal, and Boolean in PHPMyAdmin. As *T&C's* and *insurance* are either yes or no answers, a Boolean was decided to be the most appropriate data type here where 1 represents yes and 0 represents no.

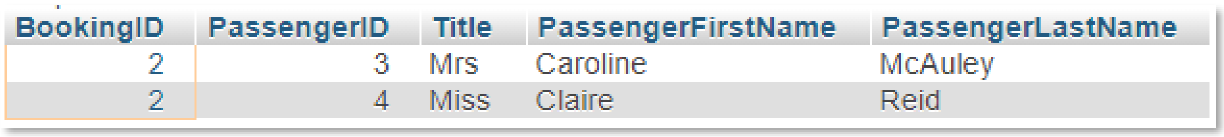
The AVG() function within SQL could be used to retrieve the average overall cost, which may be useful for a business to know as they could then compare spending on a monthly / yearly basis.



| AverageBookingCost |
|---|
| 442.13 |

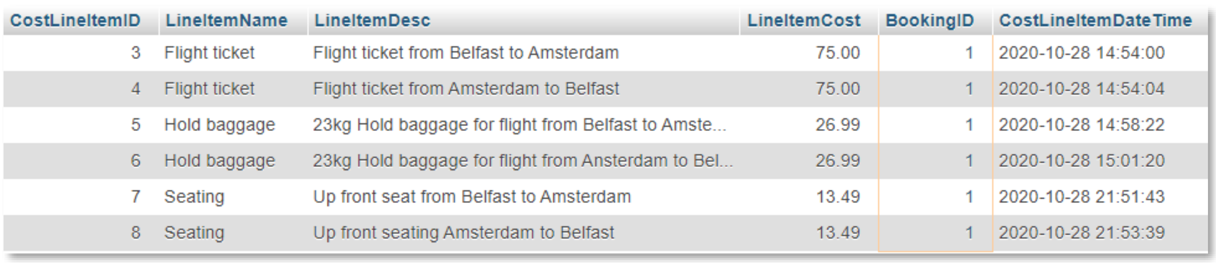
***Fig 2.1:** Average overall booking costs*

The primary key acts as a unique identifier for each booking, in line with the third rule of first normal form. The data type, integer of length 11 was used with any subsequent foreign keys in other tables also of this type as to allow links or foreign key constraints to me made in PHPMyAdmin. This data type layout is common in all the primary keys for the database. The primary key of the **booking** entity can be seen in both the **passenger** and **costLineItem** table, representing a one to many relationship. As a result, one booking can have many passengers and one booking can also have many costs, as represented below.



| BookingID | PassengerID | Title | PassengerFirstName | PassengerLastName |
|---|---|---|---|---|
| 2 | 3 | Mrs | Caroline | McAuley |
| 2 | 4 | Miss | Claire | Reid |

***Fig 2.2:** One booking can have many passengers*

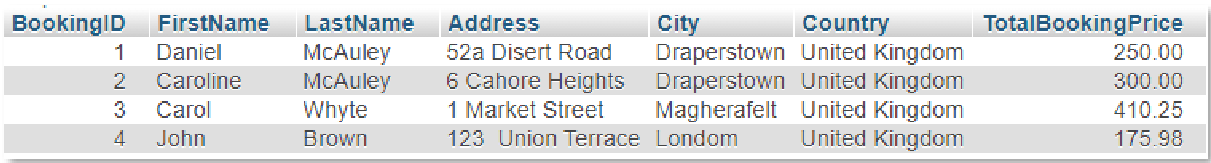


| CostLineItemID | LineItemName | LineItemDesc | LineItemCost | BookingID | CostLineItemDateTime |
|---|---|---|---|---|---|
| 3 | Flight ticket | Flight ticket from Belfast to Amsterdam | 75.00 | 1 | 2020-10-28 14:54:00 |
| 4 | Flight ticket | Flight ticket from Amsterdam to Belfast | 75.00 | 1 | 2020-10-28 14:54:04 |
| 5 | Hold baggage | 23kg Hold baggage for flight from Belfast to Amste... | 26.99 | 1 | 2020-10-28 14:58:22 |
| 6 | Hold baggage | 23kg Hold baggage for flight from Ansterdam to Bel... | 26.99 | 1 | 2020-10-28 15:01:20 |
| 7 | Seating | Up front seat from Belfast to Amsterdam | 13.49 | 1 | 2020-10-28 21:51:43 |
| 8 | Seating | Up front seating Amsterdam to Belfast | 13.49 | 1 | 2020-10-28 21:53:39 |

***Fig 2.3:** One booking can have many costs associated with it*

The foreign keys, *BookerDetailsID* and *PaymentID* allow for a one to one relationship. Meaning each booking is associated with the details of one person and each booking is connected to one set of payment details.

For example, the *name*, *address*, *phone number* and total *price* of one booking is a result of SQL queries running multiple “Inner Join” statements, which can link several tables through their foreign keys.

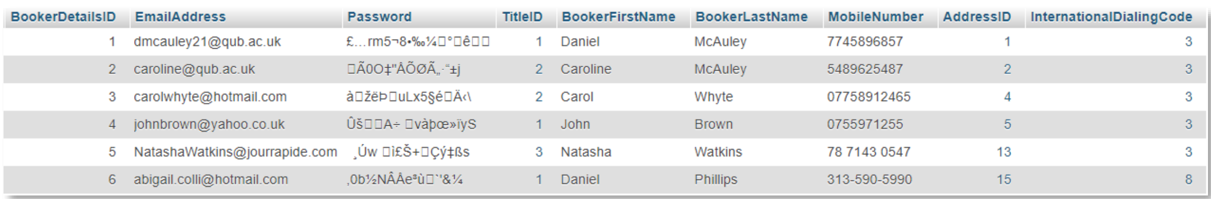


| BookingID | FirstName | LastName | Address | City | Country | TotalBookingPrice |
|---|---|---|---|---|---|---|
| 1 | Daniel | McAuley | 52a Disert Road | Draperstown | United Kingdom | 250.00 |
| 2 | Caroline | McAuley | 6 Cahore Heights | Draperstown | United Kingdom | 300.00 |
| 3 | Carol | Whyte | 1 Market Street | Magherafelt | United Kingdom | 410.25 |
| 4 | John | Brown | 123  Union Terrace | Londom | United Kingdom | 175.98 |

***Fig 2.4:** Each booking is linked to the details of one booker and has one overall booking price*

The **booker details** have been normalised to reduce data redundancy. Following second normal form, data which is not directly related to the primary key should be split into its own table.

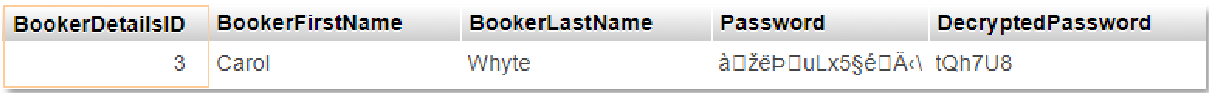
As the **booker details** table contains sensitive information about the user’s login details, an encryption function has been applied to the *password*. Encryption is a way of encoding information into “cipertext” which is unreadable to anyone who has access to the database. A method with PHPMyAdmin, called AES_ENCRYPT() is used to achieve this using a secret key.



| BookerDetailsID | EmailAddress | Password | TitleID | BookerFirstName | BookerLastName | MobileNumber | AddressID | InternationalDialingCode |
|---|---|---|---|---|---|---|---|---|
| 1 | dmcauley21@qub.ac.uk | £…rm5¬8•‰¼□°□ê□□ | 1 | Daniel | McAuley | 7745896857 | 1 | 3 |
| 2 | caroline@qub.ac.uk | □Ã0O‡"ÂÕØÃ„·“±j | 2 | Caroline | McAuley | 5489625487 | 2 | 3 |
| 3 | carolwhyte@hotmail.com | à□žëÞ□uLx5§é□Ä‹\ | 2 | Carol | Whyte | 07758912465 | 4 | 3 |
| 4 | johnbrown@yahoo.co.uk | Ûš□□A÷ □vàþœ»ïyS | 1 | John | Brown | 0755971255 | 5 | 3 |
| 5 | NatashaWatkins@jourrapide.com | ¸Úw □ì£Š+□Çý‡ßs | 3 | Natasha | Watkins | 78 7143 0547 | 13 | 3 |
| 6 | abigail.colli@hotmail.com | ‚0b½NÂÅeªù□`'&¼ | 1 | Daniel | Phillips | 313-590-5990 | 15 | 8 |

***Fig 2.5:** Booker details table with encrypted password information*

To decrypt the password, if required, the function AES_DECRYPT() is used with the secret key.



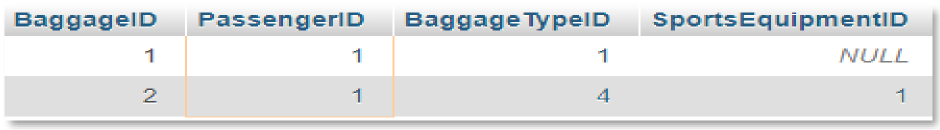
| BookerDetailsID | BookerFirstName | BookerLastName | Password | DecryptedPassword |
|---|---|---|---|---|
| 3 | Carol | Whyte | à□žëÞ□uLx5§é□Ä‹\ | tQh7U8 |

***Fig 2.6:** Booker details of decrypted password using the secret key*

## Passenger

The two attributes within the **passenger** table, outside of the primary key are foreign keys. This is because the **passenger** entity is a central piece for a flight booking system. Some of the links to the passenger include the **passenger details**, the **booking**, the **seating** details, and the actual **flights** the passenger is linked to.

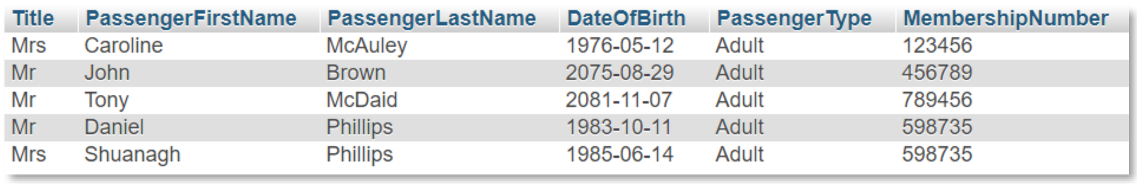
The aim of the passenger table is to hold all the information about the passenger of a particular fight with the primary key acting as a unique identifier for that table. This primary key can be seen in the **baggage** table, demonstrating that many bags can be linked to the one passenger. *PassengerID* is also seen in the **PassengerFlight** and **Seating** entities, which will be discussed retrospectively in their own individual sections.



| BaggageID | PassengerID | BaggageTypeID | SportsEquipmentID |
|---|---|---|---|
| 1 | 1 | 1 | NULL |
| 2 | 1 | 4 | 1 |

***Fig 3.1:** One passenger can have multiple bags*

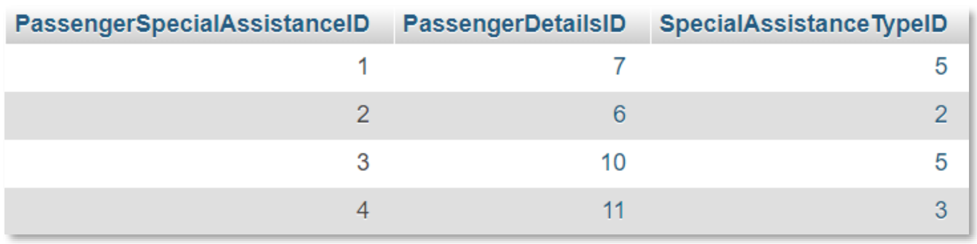
Originally within the group design, the **passenger** table was one of the larger tables in the database, but a design decision was made to normalise out much of this, in line with the recomendations of first, second and third normal form which aim to divide larger tables into smaller tables, isolating data and making it easier to insert, update or delete from the database in the future. For example, if details are needed from different tables about a passenger's *title*, *name*, *date of birth*, *passenger type* and *EasyJet Plus number*, multiple joins can be run within an SQL query to achieve this.



| Title | PassengerFirstName | PassengerLastName | DateOfBirth | PassengerType | MembershipNumber |
|---|---|---|---|---|---|
| Mrs | Caroline | McAuley | 1976-05-12 | Adult | 123456 |
| Mr | John | Brown | 2075-08-29 | Adult | 456789 |
| Mr | Tony | McDaid | 2081-11-07 | Adult | 789456 |
| Mr | Daniel | Phillips | 1983-10-11 | Adult | 598735 |
| Mrs | Shuanagh | Phillips | 1985-06-14 | Adult | 598735 |

***Fig 3.2:** Passenger information taking from the Title, PassengerDetails, PassengerType & EasyJetPlusMembership tables*

A many to many relationship was set up between **passenger details** and **special assistance** as one passenger can have many types of special assistance and one type of special assistance can be assigned to many passengers. This was achieved through a smaller joining table containing both primary keys, *PassengerDetailsID* and *SpecialAssistanceID*.

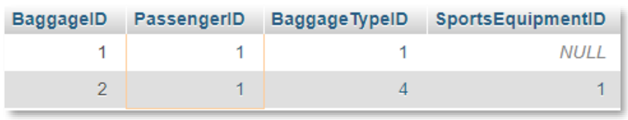


| PassengerSpecialAssistanceID | PassengerDetailsID | SpecialAssistanceTypeID |
|---|---|---|
| 1 | 7 | 5 |
| 2 | 6 | 2 |
| 3 | 10 | 5 |
| 4 | 11 | 3 |

***Fig 3.3:** The passenger special assistance joining table facilitating a many to many relationship*

## Baggage

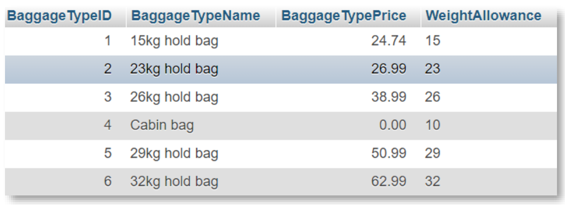
In terms of baggage, it was assumed that one passenger would be able to carry more than one bag.



| BaggageID | PassengerID | BaggageTypeID | SportsEquipmentID |
|---|---|---|---|
| 1 | 1 | 1 | NULL |
| 2 | 1 | 4 | 1 |

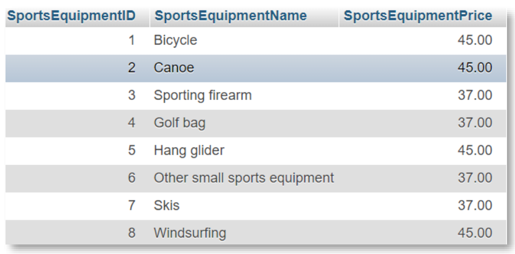
***Fig 4.1:** One passenger can have more than one bag*

On the EasyJet website, it also seemed that different **baggage types** and **sports equipment** where treated as separate tables. This is reflected in the design of the database of this project and are normalised out from the **baggage** table. Each *bag* can be linked to one *baggage type* and one piece of *sports equipment*, meaning queries can be done to show the different types of bags and sports equipment linked to each **passenger**, as can be seen in the previous *Fig 4.1*, where the sports equipment and baggage type ID represent different kinds of items.



| BaggageTypeID | BaggageTypeName | BaggageTypePrice | WeightAllowance |
|---|---|---|---|
| 1 | 15kg hold bag | 24.74 | 15 |
| 2 | 23kg hold bag | 26.99 | 23 |
| 3 | 26kg hold bag | 38.99 | 26 |
| 4 | Cabin bag | 0.00 | 10 |
| 5 | 29kg hold bag | 50.99 | 29 |
| 6 | 32kg hold bag | 62.99 | 32 |

***Fig 4.2:** Baggage type table*

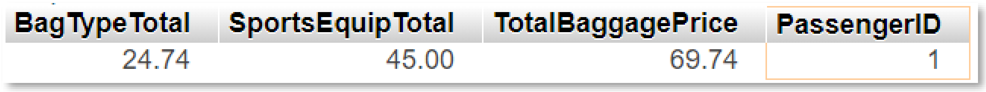


| SportsEquipmentID | SportsEquipmentName | SportsEquipmentPrice |
|---|---|---|
| 1 | Bicycle | 45.00 |
| 2 | Canoe | 45.00 |
| 3 | Sporting firearm | 37.00 |
| 4 | Golf bag | 37.00 |
| 5 | Hang glider | 45.00 |
| 6 | Other small sports equipment | 37.00 |
| 7 | Skis | 37.00 |
| 8 | Windsurfing | 45.00 |

***Fig 4.3:** Sports equipment table*

The **baggage type** table and **sports equipment** table hold information like the *name* of the item, the *weight allowance* related to that piece of baggage and the *prices* of each baggage type/ sports equipment. Price was stored as a decimal with a length of up to 12 numbers and a precision of 2 decimal places, representing pence from 1-99.

Even though prices of different baggage types and pieces of sports equipment are held in their own tables, SQL fortunately has a SUM() function which allows the total baggage price of one passenger to be queried.

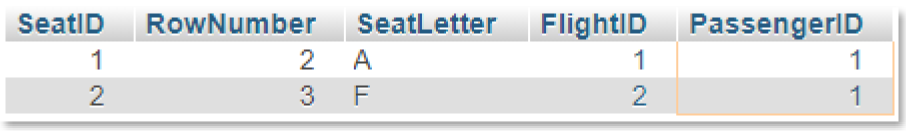


| BagTypeTotal | SportsEquipTotal | TotalBaggagePrice | PassengerID |
|---|---|---|---|
| 24.74 | 45.00 | 69.74 | 1 |

***Fig 4.4:** the total of each baggage type, sports equipment, and combined baggage total for a particular passenger*

## Seating

In a real-life scenario when a booking is made, a passenger would be assigned two seats, one for the departing flight and one for the return flight. The primary key of the **passenger** table, *PassengerID* can be seen in the **seating** table as a foreign key, allowing seats on different flights to be assigned to one passenger.

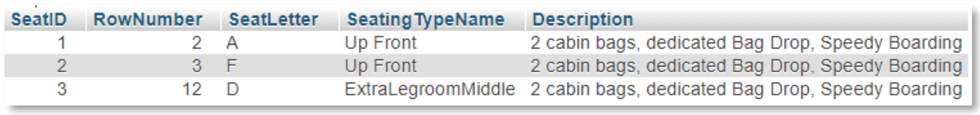


| SeatID | RowNumber | SeatLetter | FlightID | PassengerID |
|---|---|---|---|---|
| 1 | 2 | A | 1 | 1 |
| 2 | 3 | F | 2 | 1 |

***Fig 5.1:** One passenger can be having seats on the departing and return flight*

In the group iteration of the ERD, one passenger could only have one seat but in hindsight, this was changed to represent a more realistic model regarding departing and returning flights.

The remaining two foreign keys in the table include *FlightID*, allowing many seats to be assigned to the one flight, along with *SeatingTypeID*, representing one seat to have one seating type. This is shown in the SQL query results below.

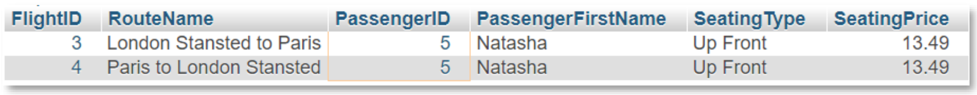


| SeatID | RowNumber | SeatLetter | SeatingTypeName | Description |
|---|---|---|---|---|
| 1 | 2 | A | Up Front | 2 cabin bags, dedicated Bag Drop, Speedy Boarding |
| 2 | 3 | F | Up Front | 2 cabin bags, dedicated Bag Drop, Speedy Boarding |
| 3 | 12 | D | ExtraLegroomMiddle | 2 cabin bags, dedicated Bag Drop, Speedy Boarding |

***Fig 5.2:** One seat can have one seating type name and description*

Outside of the foreign keys, the seating table holds information about the seat *row number* and the *seat letter*, attributes directly related to any seat on an aircraft.

Like the **passenger** and **booking** table, seating was normalised into another table called **SeatingType**, to emulate the EasyJet website, and contains information about the *prices* of the different seats and a *description* of the seating types. This information appeared to be stored in its own table during the entity relationship discovery process. This means that queriers can be run which show the *prices* and *types of seats* for each *passenger* or for a group of passengers.

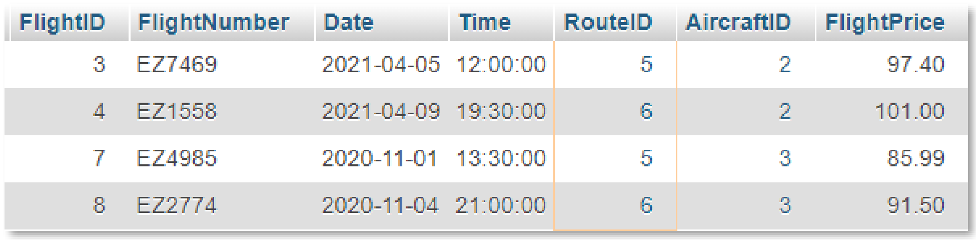


| FlightID | RouteName | PassengerID | PassengerFirstName | SeatingType | SeatingPrice |
|---|---|---|---|---|---|
| 3 | London Stansted to Paris | 5 | Natasha | Up Front | 13.49 |
| 4 | Paris to London Stansted | 5 | Natasha | Up Front | 13.49 |

***Fig 5.3:** Seating types and prices of one passenger along with their name and route details.*

## Flight

Next in the database is the **flight** entity. This table holds all the relevant information regarding different flights that passengers can board. The main design assumption when it came to the **flight** entity is that a flight must have a destination. This is represented through a foreign key, linked to a **route** table which will be further discussed in the next section, but is meant so that many flights can take the same route.

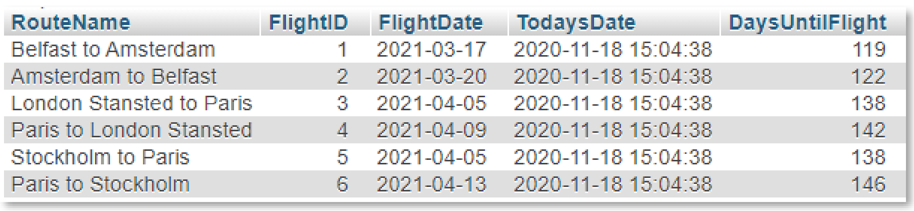


| FlightID | FlightNumber | Date | Time | RouteID | AircraftID | FlightPrice |
|---|---|---|---|---|---|---|
| 3 | EZ7469 | 2021-04-05 | 12:00:00 | 5 | 2 | 97.40 |
| 4 | EZ1558 | 2021-04-09 | 19:30:00 | 6 | 2 | 101.00 |
| 7 | EZ4985 | 2020-11-01 | 13:30:00 | 5 | 3 | 85.99 |
| 8 | EZ2774 | 2020-11-04 | 21:00:00 | 6 | 3 | 91.50 |

***Fig 6.1:** Multiple flights can take the same route*

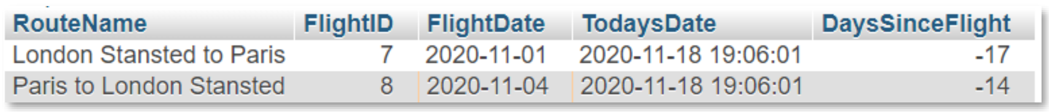
The other foreign key constraint in this table is the link to an **aircraft**. With each flight needing to have a physical aircraft to fly, the **aircraft** table can hold information about the *aircraft model* and the *seating capacity*, information which may be used by a back-end or front-end programming language to calculate how many seats are left on a flight and show to the customer in live-time.

Excluding the foreign keys, the **flight** table provides information about the *flight number*, *date*, *time*, and the flight *price*. Date and time are stored as two separate attributes under "Date" and "Time" data types in PHPMyAdmin and allows for queries to be run about flights in the future or flights that already may have happened.



| RouteName | FlightID | FlightDate | TodaysDate | DaysUntilFlight |
|---|---|---|---|---|
| Belfast to Amsterdam | 1 | 2021-03-17 | 2020-11-18 15:04:38 | 119 |
| Amsterdam to Belfast | 2 | 2021-03-20 | 2020-11-18 15:04:38 | 122 |
| London Stansted to Paris | 3 | 2021-04-05 | 2020-11-18 15:04:38 | 138 |
| Paris to London Stansted | 4 | 2021-04-09 | 2020-11-18 15:04:38 | 142 |
| Stockholm to Paris | 5 | 2021-04-05 | 2020-11-18 15:04:38 | 138 |
| Paris to Stockholm | 6 | 2021-04-13 | 2020-11-18 15:04:38 | 146 |

***Fig 6.2:** Flights in the future of the date the SQL query was run, and the number of days left until the flight*

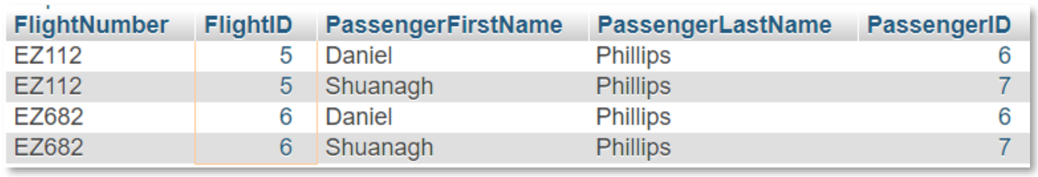


| RouteName | FlightID | FlightDate | TodaysDate | DaysSinceFlight |
|---|---|---|---|---|
| London Stansted to Paris | 7 | 2020-11-01 | 2020-11-18 19:06:01 | -17 |
| Paris to London Stansted | 8 | 2020-11-04 | 2020-11-18 19:06:01 | -14 |

***Fig 6.3:*** *Flights that already have happened since the date the SQL query was run*

A design decision was made to keep the *price* directly in the **flight** table to act as a "live" price which may be programmed to update depending on how close the flight is to its take off date. This may be done on the front end and be reflected in live time on the webpage.

An important thing to note about this table is that there is a many to many relationship between it and **passenger**. This decision was made retrospectively of the group's final version of the ERD. The reasoning for this many to many relationship is that one *flight* can have many *passengers*, and one *passenger* will usually have more than one *flight*, a departing and arrival flight. This was achieved through a small joining table called **PassengerFlight,** which has both the *flightID* and *passengerID* as foreign keys, meaning any flight could be linked to any passenger.



| FlightNumber | FlightID | PassengerFirstName | PassengerLastName | PassengerID |
|---|---|---|---|---|
| EZ112 | 5 | Daniel | Phillips | 6 |
| EZ112 | 5 | Shuanagh | Phillips | 7 |
| EZ682 | 6 | Daniel | Phillips | 6 |
| EZ682 | 6 | Shuanagh | Phillips | 7 |

***Fig 6.4:*** *Multiple flights can be linked to multiple passengers*

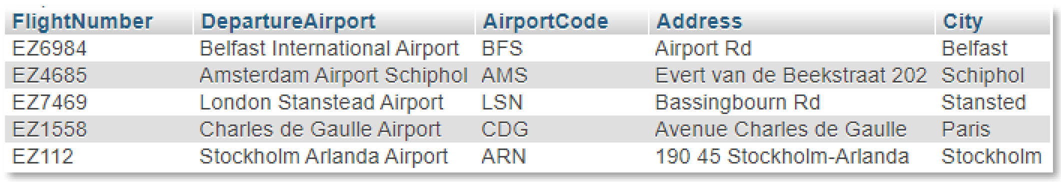
Finally, in the group design of the ERD, the **flight** table was linked to another **boarding** table. The aim of this table was to store information about the *boarding times* and *boarding gates* but as stated in the beginning of this report, this database was designed to only deal with a flight *booking* process. As a result, information outside of getting a passenger booked on a seat for flight was deemed unnecessary and so the boarding table was removed, as boarding information is only needed when the passenger is getting on to the flight.

## Route

In the group design of the ERD, the **flight** table contained all the information about the flights route, including the flight *departure airport* and the flight *arrival airport*. On further thought and again, in line with the second rule of first normal form, which explains that repeating groups of data should be removed, a separate **route** table was created.

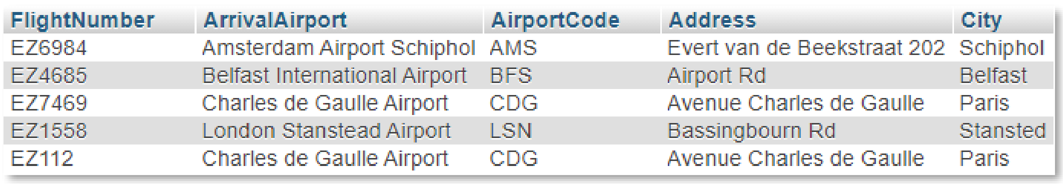
The main aim of a **route** table is to provide information about the route a flight can take, including the route *name*, the *departing airport* of the route and the *arrival airport* of the route. The primary key *RouteID* is a foreign key in the **flight** table, allowing different flights at different times to take the same route, as demonstrated in *Fig 6.1* of the **Flight** section.

The *departure airport* and the *arrival airport* are two foreign keys linked to the one primary key in the **airport** table. Both the departure airport and the arrival airport will be linked to their own separate airports. This allows the booker to see information on both **airports** and where their flights are flying from and to. For this to be demonstrated, two SQL queries must be run together showing the *departure* and *arrival* airport information for each *flight*.



| FlightNumber | DepartureAirport | AirportCode | Address | City |
|---|---|---|---|---|
| EZ6984 | Belfast International Airport | BFS | Airport Rd | Belfast |
| EZ4685 | Amsterdam Airport Schiphol | AMS | Evert van de Beekstraat 202 | Schiphol |
| EZ7469 | London Stanstead Airport | LSN | Bassingbourn Rd | Stansted |
| EZ1558 | Charles de Gaulle Airport | CDG | Avenue Charles de Gaulle | Paris |
| EZ112 | Stockholm Arlanda Airport | ARN | 190 45 Stockholm-Arlanda | Stockholm |

***Fig 7.1:** Departing airport information for first 5 flights*



| FlightNumber | ArrivalAirport | AirportCode | Address | City |
|---|---|---|---|---|
| EZ6984 | Amsterdam Airport Schiphol | AMS | Evert van de Beekstraat 202 | Schiphol |
| EZ4685 | Belfast International Airport | BFS | Airport Rd | Belfast |
| EZ7469 | Charles de Gaulle Airport | CDG | Avenue Charles de Gaulle | Paris |
| EZ1558 | London Stanstead Airport | LSN | Bassingbourn Rd | Stansted |
| EZ112 | Charles de Gaulle Airport | CDG | Avenue Charles de Gaulle | Paris |

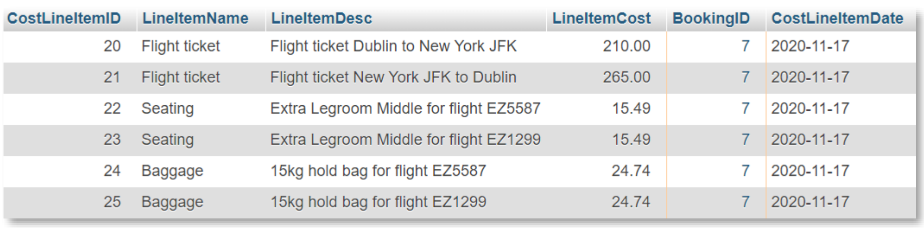
***Fig 7.2:** Arrival airport information for first 5 flights*

## Payments / Costs

Dealing with the final section of the database, the payments, and costs section, required some more thought and rethinking. Before the final group design, a **costs** table held all the prices that would be associated with booking a flight but was soon discarded as there was confusion as to how it would be linked to all the appropriate entities. Finally, it was decided that the *prices* directly linked to each entity would appear in the specified table. For example, *FlightPrice* was placed directly in the **Flight** table and would represent a "live" price, also seen in the **baggage** and **seating** tables.

This live price would be controlled by the front-end or back-end programming language and could change in live time, for example the price of a flight would change on the website as it got closer to the departing date. This decision to hold live prices was carried through to the final version of the database but with one important addition, the **CostLineItems** table.

The **CostLineItems** table acts like a transaction history or a receipt for each booking. It deals with all the associated costs of one *booking*, for example **flight**, **seating**, and **baggage** costs. The benefit of this table is that it is not linked to the live prices so the prices that appear in the table will be correct at the time of booking and will remain that way.

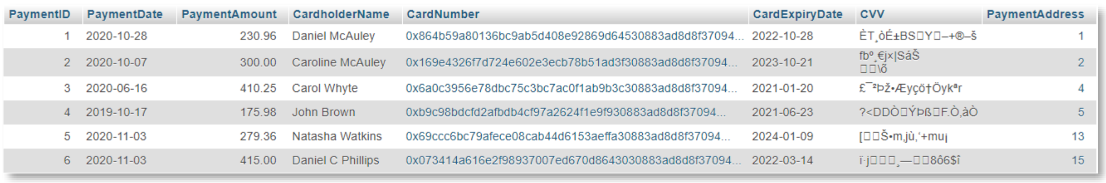


| CostLineItemID | LineItemName | LineItemDesc | LineItemCost | BookingID | CostLineItemDate |
|---|---|---|---|---|---|
| 20 | Flight ticket | Flight ticket Dublin to New York JFK | 210.00 | 7 | 2020-11-17 |
| 21 | Flight ticket | Flight ticket New York JFK to Dublin | 265.00 | 7 | 2020-11-17 |
| 22 | Seating | Extra Legroom Middle for flight EZ5587 | 15.49 | 7 | 2020-11-17 |
| 23 | Seating | Extra Legroom Middle for flight EZ1299 | 15.49 | 7 | 2020-11-17 |
| 24 | Baggage | 15kg hold bag for flight EZ5587 | 24.74 | 7 | 2020-11-17 |
| 25 | Baggage | 15kg hold bag for flight EZ1299 | 24.74 | 7 | 2020-11-17 |

***Fig 8.1:** Cost line items associated with one booking*

In the real world, card **payment** details would normally not be held within the company's database but rather outsourced to a third party, payment processor service. This would redirect the user to a check-out page hosted by the provider and means the third-party provider are responsible for handling sensitive information securely. For the sake of this project and to demonstrate a close replica of a flight booking system, **payment** information is stored but certain information hidden.

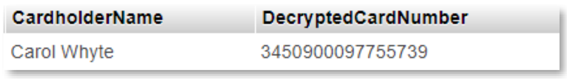
Like the *password* in the **BookerDetails** table, the AES_ENCRYPT function within PHPMyAdmin allowed user's *card number* and *CVV* details to be encrypted into an unreadable format.



| PaymentID | PaymentDate | PaymentAmount | CardholderName | CardNumber | CardExpiryDate | CVV | PaymentAddress |
|---|---|---|---|---|---|---|---|
| 1 | 2020-10-28 | 230.96 | Daniel McAuley | 0x864b59a80136bc9ab5d408e92869d64530883ad8d8f37094... | 2022-10-28 | ÈT¸òÉ±BS□Y□–+®–š | 1 |
| 2 | 2020-10-07 | 300.00 | Caroline McAuley | 0x169e4326f7d724e602e3ecb78b51ad3f30883ad8d8f37094... | 2023-10-21 | fbº¸€j×\|SáŠ □□\õ | 2 |
| 3 | 2020-06-16 | 410.25 | Carol Whyte | 0x6a0c3956e78dbc75c3bc7ac0f1ab9b3c30883ad8d8f37094... | 2021-01-20 | £¯²Þž•Æyçö†Öykªr | 4 |
| 4 | 2019-10-17 | 175.98 | John Brown | 0xb9c98bdcfd2afbdb4cf97a2624f1e9f930883ad8d8f37094... | 2021-06-23 | ?<DDÒ□ÝÞß□F.Ò¸àÒ | 5 |
| 5 | 2020-11-03 | 279.36 | Natasha Watkins | 0x69ccc6bc79afece08cab44d6153aeffa30883ad8d8f37094... | 2024-01-09 | [□□Š•m,jù,'+mu¡ | 13 |
| 6 | 2020-11-03 | 415.00 | Daniel C Phillips | 0x073414a616e2f98937007ed670d8643030883ad8d8f37094... | 2022-03-14 | ï·j□□□¸—□□8ô6$î | 15 |

***<u>Fig 8.2:</u>*** *<u>Encryption on sensitive information like CardNumber and CVV</u>*

Using the secret key, the information could also be decrypted using the AES_DECRYPT function.



| CardholderName | DecryptedCardNumber |
|---|---|
| Carol Whyte | 3450900097755739 |

***<u>Fig 8.3:</u>*** *<u>Decryption of card number using the secret key within the AES_DECRYPT function</u>*

## Potential Improvements

As with any project, there are always improvements and alterations that could be made to make the database operate at a higher standard. The first is that the database does not deal with **stop over flights**. On the EasyJet website the booker has the option to select whether they want stop-over options shown in their flight search results. While technically, a total of four **flights** could be added to deal with a stopover on the *departure* and *return* flight in the current database, the website seemed to have dealt with this as a different kind of flight, perhaps having its own attribute in the **flight** table.

The second design flaw that could be criticised, is in relation to the **baggage** table. At present *PassengerID* appears as a foreign key in the **Baggage** table, allowing for one passenger to have one or more bags associated with them but unfortunately means that there is repetition of data when many passengers have the same baggage type. Ideally, many passengers could have many different types of baggage, but to avoid repetition of data with the current design, a smaller "joining" table could be used to link **passenger** and **baggage**, like that used between **flight** and **passenger**.

Finally, as previously mentioned, a redesign of the database in a real-world situation would not have the **payments** table store the bookers sensitive card information, but for demonstration purposes this has been included in the design. At present, the function AES_ENCRYPT is used to achieve some level of protection of this sensitive data, where a *secret key* is used to encrypt and subsequently decrypt the information. In the current design, the same secret key has been used for all the pieces of data but using individual secrets keys for each one would ensure an even higher level of protection.

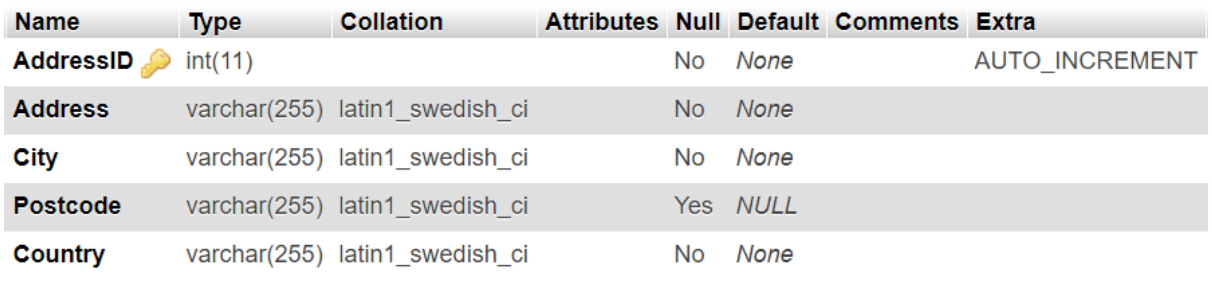
## Conclusion

To conclude, the current design of the database predominantly meets the requirements of a flight booking system, outlined in the introduction. Many passengers can be under the one booking with each passenger having multiple bags and their own set of details. Passengers can choose seats on both the departing and return flight, with many flights able to take the same route. Live prices can be adjusted in the main tables where appropriate, and a **costLineItems** table deals with a breakdown of the booking's total costs on an item by item basis. Although the level of encryption could be improved by having individual distinct secret keys, and the relationship between passenger and baggage needs some revision, the database largely accommodates data processing needs through sensible design decisions relevant of a real world booking system.

# Appendix

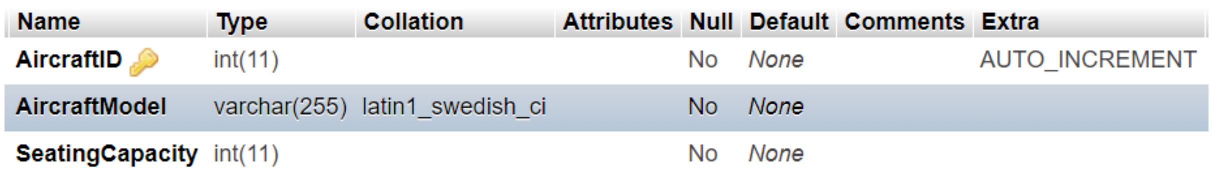
## Table Structures

### Address table



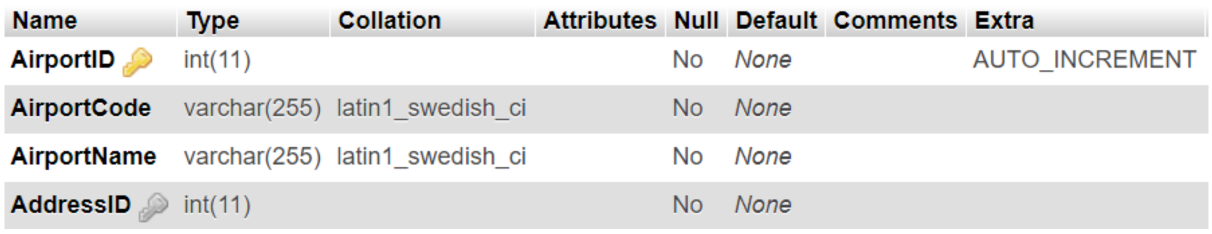
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **AddressID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **Address** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **City** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **Postcode** | varchar(255) | latin1_swedish_ci | | Yes | *NULL* | | |
| **Country** | varchar(255) | latin1_swedish_ci | | No | *None* | | |

### Aircraft



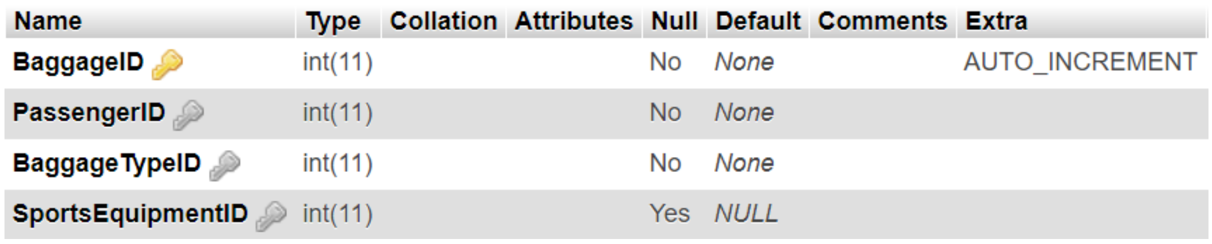
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **AircraftID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **AircraftModel** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **SeatingCapacity** | int(11) | | | No | *None* | | |

### Airport



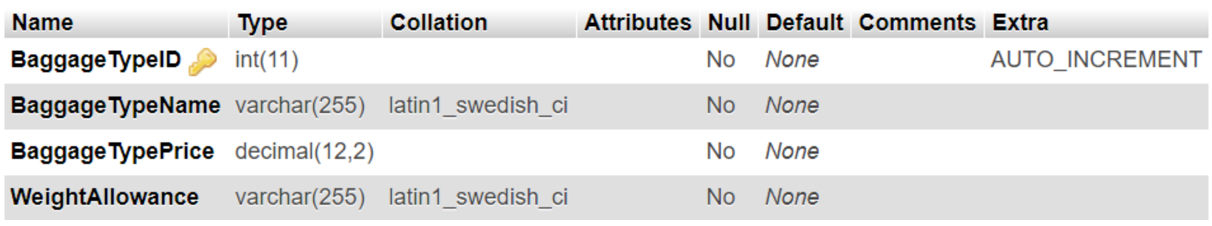
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **AirportID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **AirportCode** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **AirportName** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **AddressID** | int(11) | | | No | *None* | | |

### Baggage



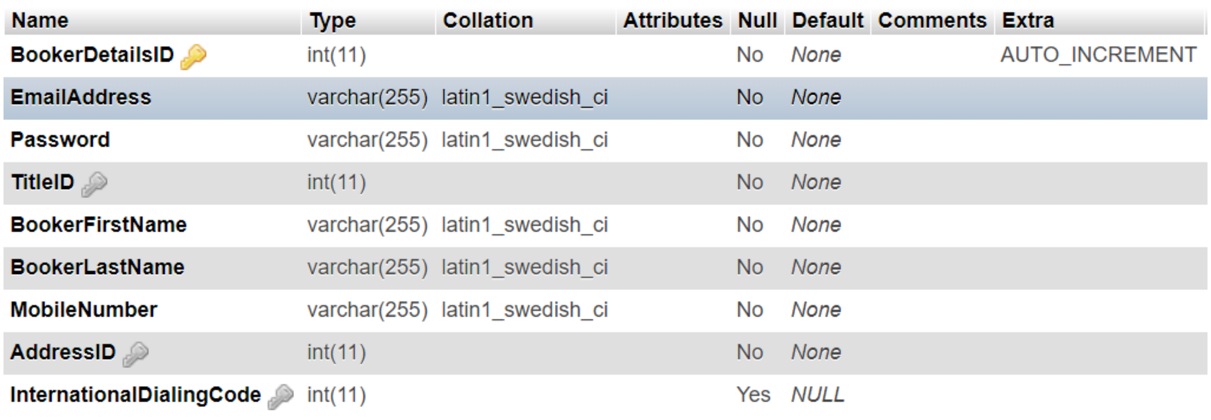
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **BaggageID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **PassengerID** | int(11) | | | No | *None* | | |
| **BaggageTypeID** | int(11) | | | No | *None* | | |
| **SportsEquipmentID** | int(11) | | | Yes | *NULL* | | |

### Baggage_type



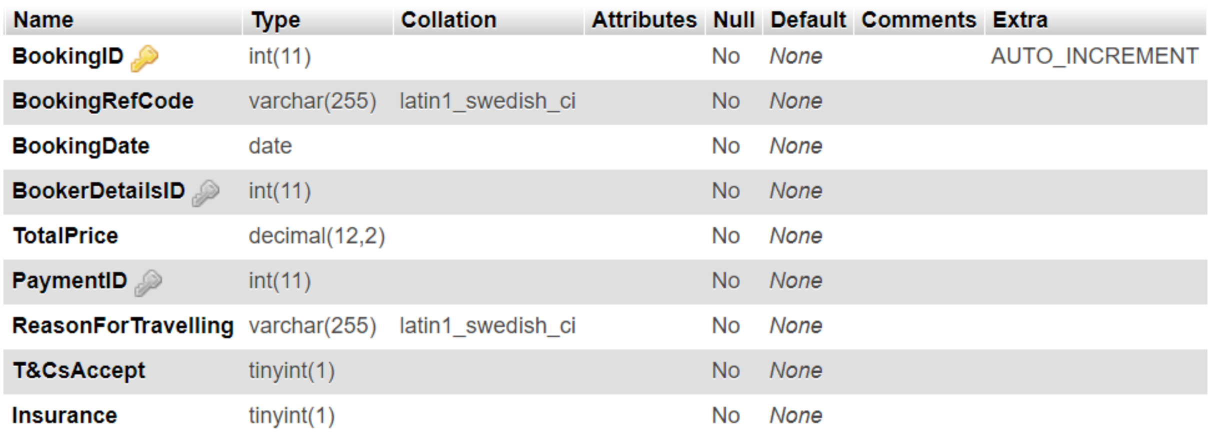
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **BaggageTypeID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **BaggageTypeName** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **BaggageTypePrice** | decimal(12,2) | | | No | *None* | | |
| **WeightAllowance** | varchar(255) | latin1_swedish_ci | | No | *None* | | |

## Booker_details



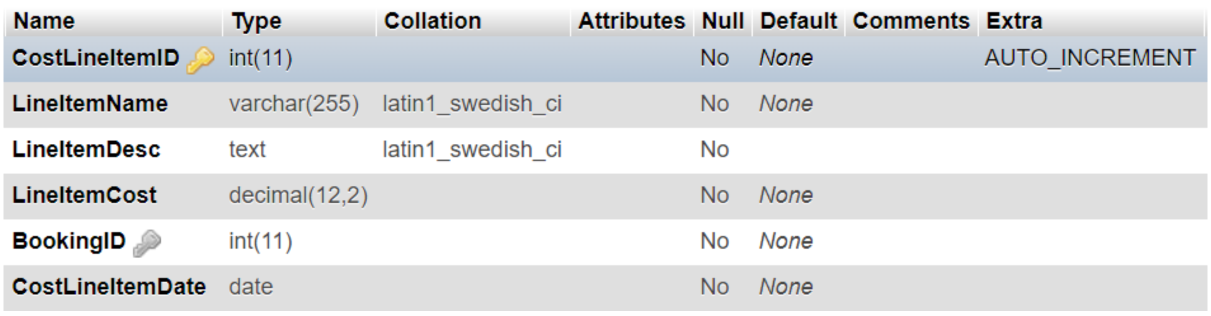
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| BookerDetailsID 🔑 | int(11) | | | No | *None* | | AUTO_INCREMENT |
| EmailAddress | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| Password | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| TitleID 🔑 | int(11) | | | No | *None* | | |
| BookerFirstName | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| BookerLastName | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| MobileNumber | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| AddressID 🔑 | int(11) | | | No | *None* | | |
| InternationalDialingCode 🔑 | int(11) | | | Yes | *NULL* | | |

## Booking



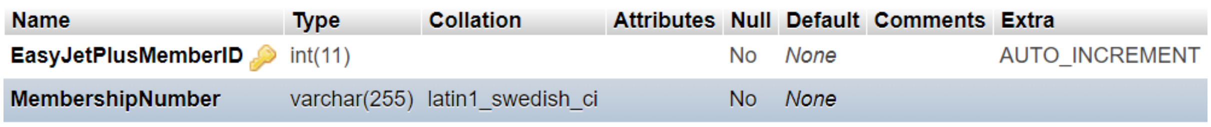
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| BookingID 🔑 | int(11) | | | No | *None* | | AUTO_INCREMENT |
| BookingRefCode | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| BookingDate | date | | | No | *None* | | |
| BookerDetailsID 🔑 | int(11) | | | No | *None* | | |
| TotalPrice | decimal(12,2) | | | No | *None* | | |
| PaymentID 🔑 | int(11) | | | No | *None* | | |
| ReasonForTravelling | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| T&CsAccept | tinyint(1) | | | No | *None* | | |
| Insurance | tinyint(1) | | | No | *None* | | |

## Cost_line_items



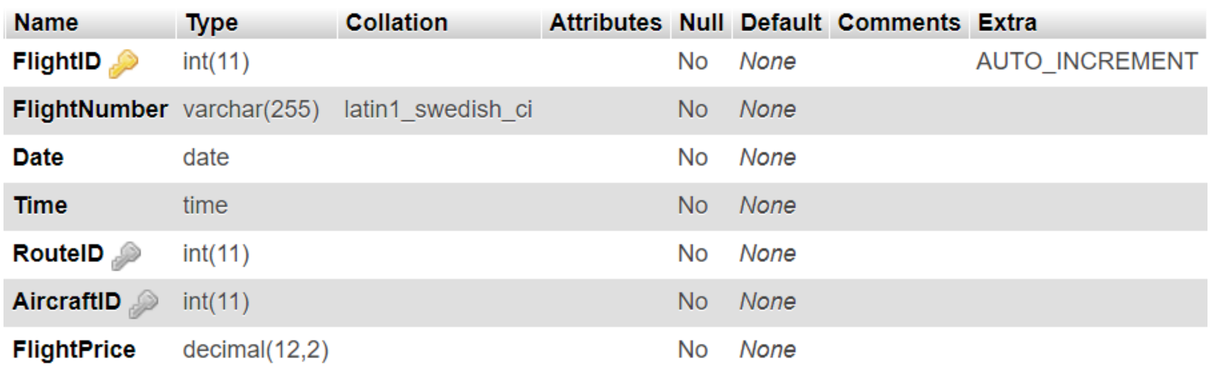
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| CostLineItemID 🔑 | int(11) | | | No | *None* | | AUTO_INCREMENT |
| LineItemName | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| LineItemDesc | text | latin1_swedish_ci | | No | | | |
| LineItemCost | decimal(12,2) | | | No | *None* | | |
| BookingID 🔑 | int(11) | | | No | *None* | | |
| CostLineItemDate | date | | | No | *None* | | |

## Easyjet_plus_membership



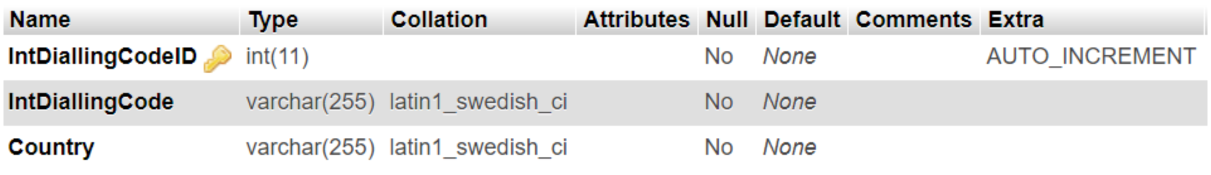
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| EasyJetPlusMemberID 🔑 | int(11) | | | No | *None* | | AUTO_INCREMENT |
| MembershipNumber | varchar(255) | latin1_swedish_ci | | No | *None* | | |

## Flight



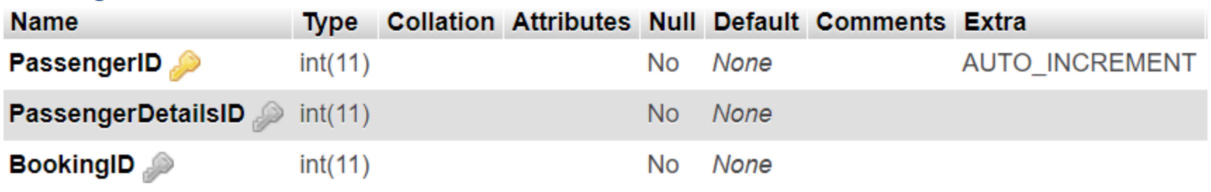
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **FlightID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **FlightNumber** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **Date** | date | | | No | *None* | | |
| **Time** | time | | | No | *None* | | |
| **RouteID** | int(11) | | | No | *None* | | |
| **AircraftID** | int(11) | | | No | *None* | | |
| **FlightPrice** | decimal(12,2) | | | No | *None* | | |

## International_dialing_codes



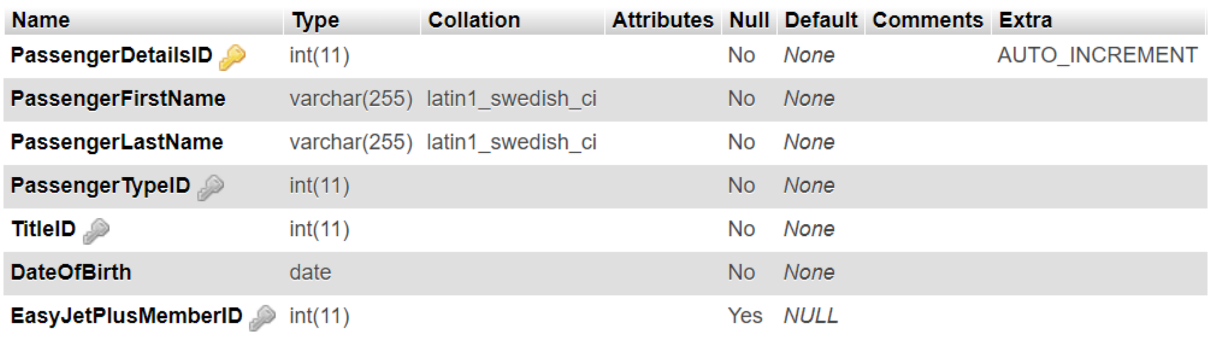
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **IntDiallingCodeID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **IntDiallingCode** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **Country** | varchar(255) | latin1_swedish_ci | | No | *None* | | |

## Passenger



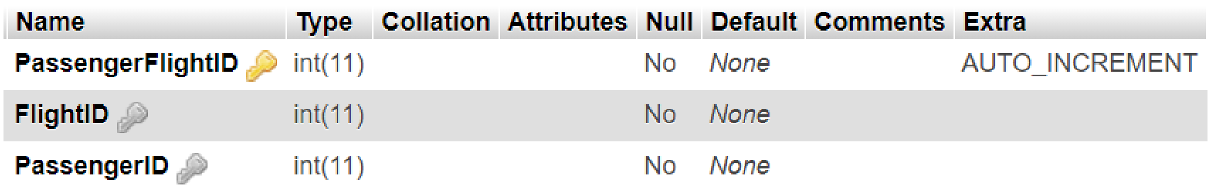
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **PassengerID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **PassengerDetailsID** | int(11) | | | No | *None* | | |
| **BookingID** | int(11) | | | No | *None* | | |

## Passenger_details



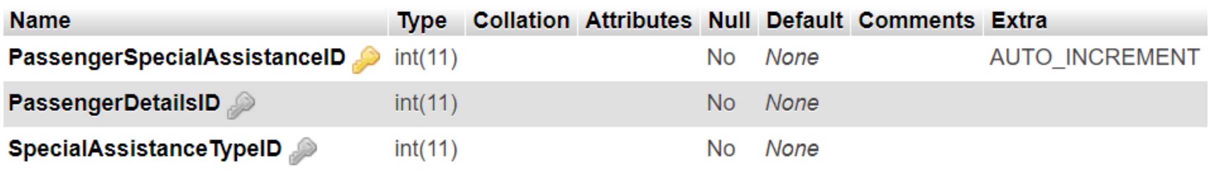
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **PassengerDetailsID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **PassengerFirstName** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **PassengerLastName** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **PassengerTypeID** | int(11) | | | No | *None* | | |
| **TitleID** | int(11) | | | No | *None* | | |
| **DateOfBirth** | date | | | No | *None* | | |
| **EasyJetPlusMemberID** | int(11) | | | Yes | *NULL* | | |

## Passenger_flight



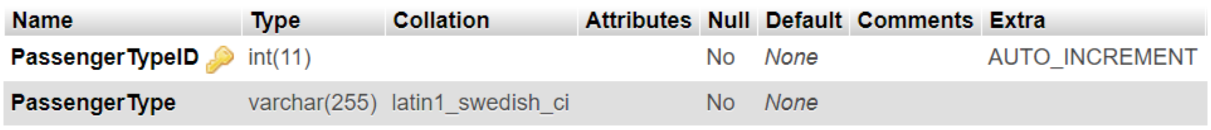
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **PassengerFlightID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **FlightID** | int(11) | | | No | *None* | | |
| **PassengerID** | int(11) | | | No | *None* | | |

## Passenger_special_assistance



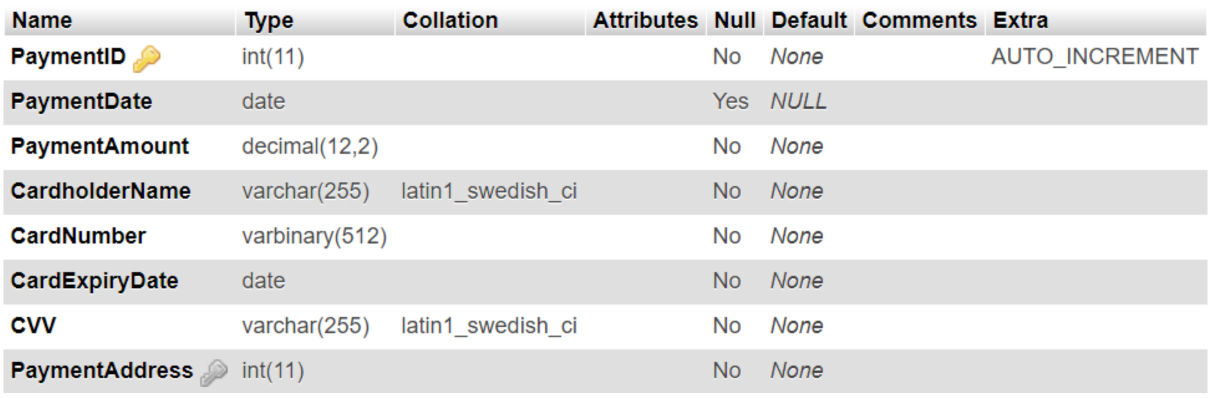
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **PassengerSpecialAssistanceID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **PassengerDetailsID** | int(11) | | | No | *None* | | |
| **SpecialAssistanceTypeID** | int(11) | | | No | *None* | | |

## Passenger_type



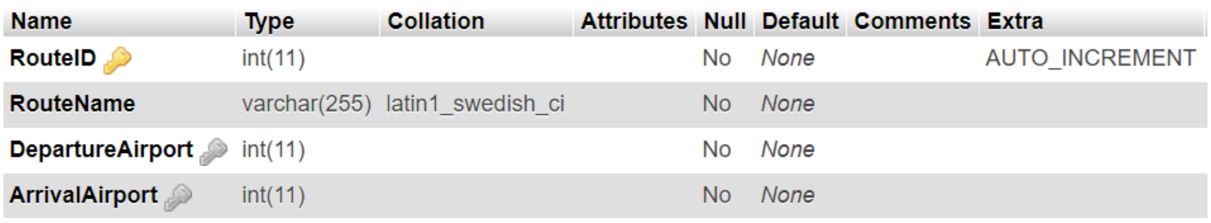
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **PassengerTypeID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **PassengerType** | varchar(255) | latin1_swedish_ci | | No | *None* | | |

## Payments



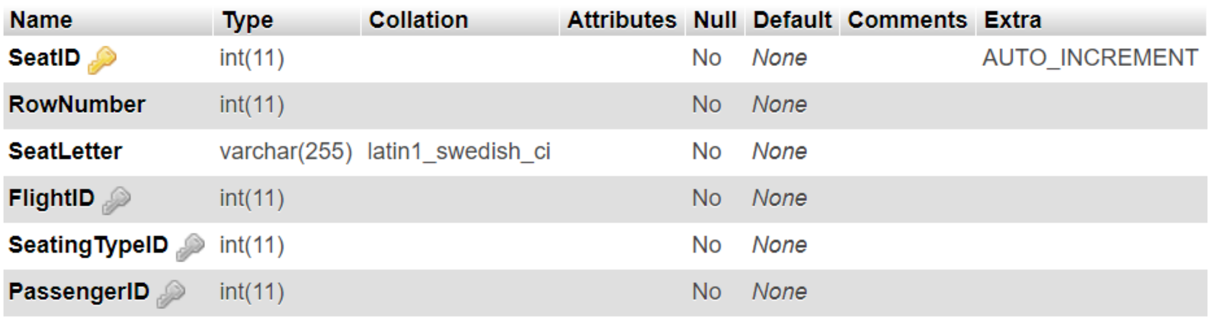
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **PaymentID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **PaymentDate** | date | | | Yes | *NULL* | | |
| **PaymentAmount** | decimal(12,2) | | | No | *None* | | |
| **CardholderName** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **CardNumber** | varbinary(512) | | | No | *None* | | |
| **CardExpiryDate** | date | | | No | *None* | | |
| **CVV** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **PaymentAddress** | int(11) | | | No | *None* | | |

## Route



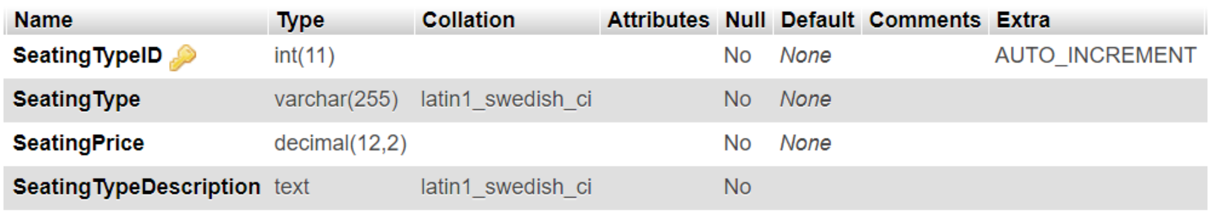
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **RouteID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **RouteName** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **DepartureAirport** | int(11) | | | No | *None* | | |
| **ArrivalAirport** | int(11) | | | No | *None* | | |

## Seating



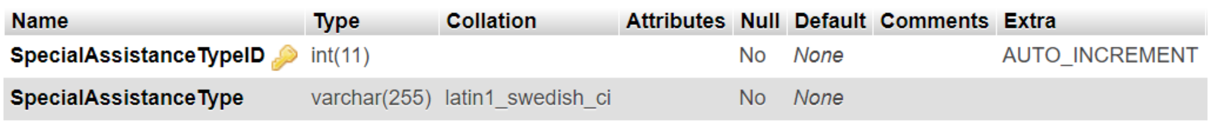
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **SeatID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **RowNumber** | int(11) | | | No | *None* | | |
| **SeatLetter** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **FlightID** | int(11) | | | No | *None* | | |
| **SeatingTypeID** | int(11) | | | No | *None* | | |
| **PassengerID** | int(11) | | | No | *None* | | |

## Seating_type



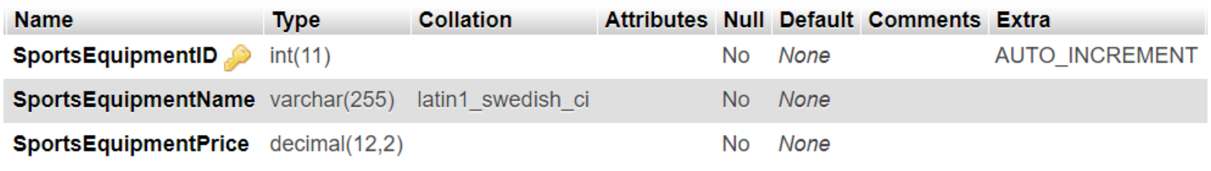
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **SeatingTypeID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **SeatingType** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **SeatingPrice** | decimal(12,2) | | | No | *None* | | |
| **SeatingTypeDescription** | text | latin1_swedish_ci | | No | | | |

## Special_assisstance_type



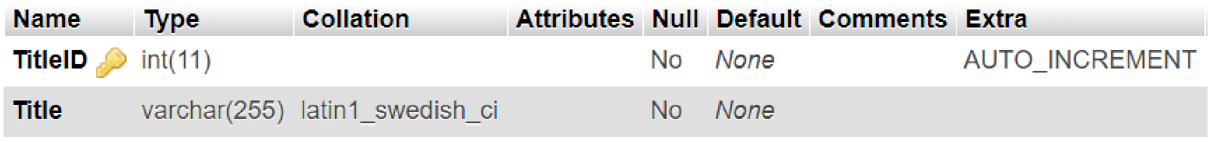
| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **SpecialAssistanceTypeID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **SpecialAssistanceType** | varchar(255) | latin1_swedish_ci | | No | *None* | | |

### Sports_equipment



| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **SportsEquipmentID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **SportsEquipmentName** | varchar(255) | latin1_swedish_ci | | No | *None* | | |
| **SportsEquipmentPrice** | decimal(12,2) | | | No | *None* | | |

### Title



| Name | Type | Collation | Attributes | Null | Default | Comments | Extra |
|---|---|---|---|---|---|---|---|
| **TitleID** | int(11) | | | No | *None* | | AUTO_INCREMENT |
| **Title** | varchar(255) | latin1_swedish_ci | | No | *None* | | |

## SQL Queries

### Fig 2.1

```
SELECT FORMAT(AVG(TotalPrice), 2) AS AverageBookingCost FROM booking;
```

### Fig 2.2

```
SELECT passenger.BookingID, PassengerID, title.Title, passenger_details.PassengerFirstName, passenger_details.PassengerLastName
FROM booking
INNER JOIN passenger ON booking.BookingID = passenger.PassengerID
INNER JOIN passenger_details ON passenger.PassengerDetailsID = passenger_details.PassengerDetailsID
INNER JOIN title ON passenger_details.TitleID = title.TitleID
WHERE passenger.BookingID = 2;
```

### Fig 2.3

```
SELECT * FROM `cost_line_items`
WHERE cost_line_items.BookingID = 1;
```

### Fig 2.4

```
SELECT booking.BookingID, booker_details.BookerFirstName AS FirstName, booker_details.BookerLastName AS LastName,
address.Address, address.City, address.Country, booking.TotalPrice AS TotalBookingPrice
FROM address INNER JOIN booker_details ON address.AddressID = booker_details.AddressID
INNER JOIN booking ON booker_details.BookerDetailsID = booking.BookerDetailsID;
```

### Fig 2.5

```
UPDATE booking_details SET Password = AES_ENCRYPT (tQh7u8, 'mySecretKey') WHERE BookerDetailsID = 3;
```

*(this was done for all the passwords in the Booker Details table)*

### Fig 2.6

```
SELECT BookerDetailsID, BookerFirstName, BookerLastName, Password, AES_DECRYPT (`Password`, 'mySecretKey') AS DecryptedPassword
from booker_details WHERE BookerDetailsID = 3;
```

*(this query is used to decrypt all the password in the table using the secret key)*

### Fig 3.1

```
SELECT * FROM `baggage` WHERE PassengerID = 1;
```

### Fig 3.2

```
SELECT title.Title, `PassengerFirstName`, `PassengerLastName`, `DateOfBirth`, passenger_type.PassengerType,
easyjet_plus_membership.MembershipNumber FROM
title INNER JOIN passenger_details ON title.TitleID = passenger_details.TitleID
INNER JOIN passenger_type ON passenger_type.PassengerTypeID = passenger_details.PassengerTypeID
INNER JOIN easyjet_plus_membership ON easyjet_plus_membership.EasyJetPlusMemberID = passenger_details.EasyJetPlusMemberID;
```

### Fig 4.1

```
SELECT * FROM `baggage` WHERE PassengerID = 1;
```

## Fig 4.2

```
SELECT
SUM(BaggageTypePrice) AS 'BagTypeTotal',
SUM(SportsEquipmentPrice) AS 'SportsEquipTotal',
(SUM(BaggageTypePrice) + SUM(SportsEquipmentPrice)) AS TotalBaggagePrice,
PassengerID
FROM baggage_type
LEFT JOIN baggage ON baggage_type.BaggageTypeID = baggage.BaggageTypeID
LEFT JOIN sports_equipment ON baggage.SportsEquipmentID = sports_equipment.SportsEquipmentID
WHERE PassengerID = 1;
```

## Fig 5.1

```
SELECT SeatID, RowNumber, SeatLetter, FlightID, seating.PassengerID FROM seating
INNER JOIN passenger on seating.PassengerID = passenger.PassengerID
WHERE seating.PassengerID = 1;
```

## Fig 5.2

```
SELECT SeatID, RowNumber, SeatLetter, seating_type.SeatingType AS SeatingTypeName, seating_type.SeatingTypeDescription AS
Description
FROM seating
INNER JOIN seating_type on seating.SeatingTypeID = seating_type.SeatingTypeID;
```

## Fig 5.3

```
SELECT seating.FlightID, route.RouteName, seating.PassengerID, PassengerFirstName, SeatingType, SeatingPrice
FROM seating INNER JOIN flight ON seating.FlightID = flight.FlightID
INNER JOIN route ON flight.RouteID = route.RouteID
INNER JOIN seating_type ON seating.SeatingTypeID = seating_type.SeatingTypeID
INNER JOIN passenger ON seating.PassengerID = passenger.PassengerID
INNER JOIN passenger_details ON passenger.PassengerDetailsID = passenger_details.PassengerDetailsID
WHERE seating.PassengerID = 5;
```

## Fig 6.1

```
SELECT * FROM flight WHERE `RouteID`= 5 OR `RouteID`=6;
```

## Fig 6.2

```
SELECT RouteName, FlightID, Date AS FlightDate, NOW() AS TodaysDate, DATEDIFF(Date, NOW()) AS DaysUntilFlight
FROM flight
INNER JOIN route ON flight.RouteID = route.RouteID
WHERE Date > NOW();
```

## Fig 6.3

```
SELECT RouteName, `FlightID`, `Date` AS FlightDate, NOW() AS TodaysDate, DATEDIFF(`Date`, NOW()) AS DaysSinceFlight
FROM flight
INNER JOIN route ON flight.RouteID = route.RouteID
WHERE Date < NOW();
```

## Fig 6.4

```
SELECT flight.FlightNumber, passenger_flight.FlightID, passenger_details.PassengerFirstName, passenger_details.PassengerLastName,
passenger_flight.PassengerID
FROM flight INNER JOIN passenger_flight ON flight.FlightID = passenger_flight.FlightID
INNER JOIN passenger ON passenger_flight.PassengerID = passenger.PassengerID
INNER JOIN passenger_details ON passenger.PassengerDetailsID = passenger_details.PassengerDetailsID
WHERE passenger_flight.FlightID = 5 OR passenger_flight.FlightID = 6;
```

## Fig 7.1

```
SELECT FlightNumber, airport.AirportName AS DepartureAirport, AirportCode, Address, City
```

```sql
FROM route
INNER JOIN airport ON route.DepartureAirport=airport.AirportID
INNER JOIN flight ON flight.RouteID=route.RouteID
INNER JOIN address ON address.AddressID=airport.AddressID LIMIT 5;
```

### Fig 7.2

```sql
SELECT FlightNumber, airport.AirportName AS ArrivalAirport, AirportCode, Address, City
FROM route
INNER JOIN airport ON route.ArrivalAirport=airport.AirportID
INNER JOIN flight ON flight.RouteID=route.RouteID
INNER JOIN address ON address.AddressID=airport.AddressID LIMIT 5;
```

### Fig 8.1

```sql
SELECT * FROM `cost_line_items` WHERE `BookingID`=7;
```

### Fig 8.2

```sql
UPDATE payments SET CardNumber = AES_ENCRYPT (3450300007755739, 'mySecretKey') WHERE PaymentID = 3;
```

*(similar queries were used for all the CardNumber's and CVV's in the Payment table)*

### Fig 8.3

```sql
SELECT CardholderName, AES_DECRYPT (`CardNumber`, 'mySecretKey') AS DecryptedCardNumber from payments WHERE PaymentsID =
3;
```

*(a similar query is used to decrypt all the card numbers in the table using the secret key)*

## Video SQL Queries

### Destinations

```sql
SELECT address.City, address.Country FROM address
INNER JOIN airport ON address.AddressID = airport.AddressID;
```

### Dates

```sql
SELECT RouteName, Date AS FlightDate, NOW() AS TodaysDateAndTime, DATEDIFF(Date, NOW()) AS DaysUntilFlight
FROM flight
INNER JOIN route ON flight.RouteID = route.RouteID
INNER JOIN airport arrival_airport ON (arrival_airport.AirportID = route.ArrivalAirport)
INNER JOIN airport departure_airport ON (departure_airport.AirportID = route.DepartureAirport)
WHERE Date > NOW() AND (arrival_airport.AirportID = 3 AND departure_airport.AirportID = 2) ORDER BY DaysUntilFlight ASC;

SELECT RouteName, Date AS FlightDate, NOW() AS TodaysDateAndTime, DATEDIFF(Date, NOW()) AS DaysUntilFlight
FROM flight
INNER JOIN route ON flight.RouteID = route.RouteID
INNER JOIN airport arrival_airport ON (arrival_airport.AirportID = route.ArrivalAirport)
INNER JOIN airport departure_airport ON (departure_airport.AirportID = route.DepartureAirport)
WHERE Date > NOW() AND (arrival_airport.AirportID = 2 AND departure_airport.AirportID = 3) ORDER BY DaysUntilFlight ASC;
```

### Passenger Types

```sql
SELECT * FROM `passenger_type`;
```

### Prices

```sql
SELECT route.RouteName, Date, Time, `FlightPrice` FROM flight
INNER JOIN route on flight.RouteID = route.RouteID
INNER JOIN airport departAirport ON route.DepartureAirport = departAirport.AirportID
INNER JOIN airport arrivalAirport ON route.ArrivalAirport = arrivalAirport.AirportID
WHERE departAirport.AirportID = 2 AND arrivalAirport.AirportID = 3;

SELECT route.RouteName, Date, Time, `FlightPrice` FROM flight
INNER JOIN route on flight.RouteID = route.RouteID
INNER JOIN airport departAirport ON route.DepartureAirport = departAirport.AirportID
INNER JOIN airport arrivalAirport ON route.ArrivalAirport = arrivalAirport.AirportID
WHERE departAirport.AirportID = 3 AND arrivalAirport.AirportID = 2;
```

### MIN() Prices

```
SELECT route.RouteName, Date, Time, MIN(FlightPrice) FROM flight
INNER JOIN route on flight.RouteID = route.RouteID
INNER JOIN airport departAirport ON route.DepartureAirport = departAirport.AirportID
INNER JOIN airport arrivalAirport ON route.ArrivalAirport = arrivalAirport.AirportID
WHERE departAirport.AirportID = 2 AND arrivalAirport.AirportID = 3;

SELECT route.RouteName, Date, Time, MIN(FlightPrice) FROM flight
INNER JOIN route on flight.RouteID = route.RouteID
INNER JOIN airport departAirport ON route.DepartureAirport = departAirport.AirportID
INNER JOIN airport arrivalAirport ON route.ArrivalAirport = arrivalAirport.AirportID
WHERE departAirport.AirportID = 3 AND arrivalAirport.AirportID = 2;
```

## Seating

### Seating capacity

```
SELECT aircraft.SeatingCapacity, flight.FlightNumber, route.RouteName
FROM aircraft
INNER JOIN flight ON aircraft.AircraftID = flight.AircraftID
INNER JOIN route ON flight.RouteID = route.RouteID
WHERE route.RouteID=1 OR route.RouteID=2;
```

### Seats booked on flights

```
SELECT flight.FlightNumber, flight.Date, route.RouteName, seating.RowNumber, seating.SeatLetter
FROM seating
INNER JOIN flight ON seating.FlightID=flight.FlightID
INNER JOIN route ON route.RouteID = flight.RouteID
WHERE route.RouteID=1 OR route.RouteID=2;
```

### Seats booked on 1 flight

```
SELECT flight.FlightNumber, flight.Date, route.RouteName, seating.RowNumber, seating.SeatLetter
FROM seating
INNER JOIN flight ON seating.FlightID=flight.FlightID
INNER JOIN route ON route.RouteID = flight.RouteID
WHERE flight.FlightID=1;
```

## Baggage

```
SELECT
SUM(BaggageTypePrice) AS 'BagTypeTotal',
SUM(SportsEquipmentPrice) AS 'SportsEquipTotal',
(SUM(BaggageTypePrice) + SUM(SportsEquipmentPrice)) AS TotalBaggagePrice,
PassengerID
FROM baggage_type
LEFT JOIN baggage ON baggage_type.BaggageTypeID = baggage.BaggageTypeID
LEFT JOIN sports_equipment ON baggage.SportsEquipmentID = sports_equipment.SportsEquipmentID
WHERE PassengerID = 1;
```

## Booker Details

### Decrypted password

```
SELECT BookerDetailsID, BookerFirstName, BookerLastName, Password, AES_DECRYPT(Password, 'mySecretKey') AS DecryptedPassword
from booker_details WHERE BookerDetailsID = 1;
```

## Costs & Payments

### Sum() of Costs

```
SELECT booking.BookingRefCode, cost_line_items.BookingID, SUM(`LineItemCost`) AS TotalBookingCost FROM cost_line_items
INNER JOIN booking ON booking.BookingID = cost_line_items.BookingID
WHERE cost_line_items.BookingID=1;
```

### Updating tables using variables

```
SET @sumTotalCost = (SELECT SUM(`LineItemCost`) FROM `cost_line_items` WHERE `BookingID` = 1);
UPDATE booking SET booking.TotalPrice = @sumTotalCost WHERE booking.BookingID = 1;
UPDATE payments SET payments.PaymentAmount = @sumTotalCost WHERE payments.PaymentID = 1;
```