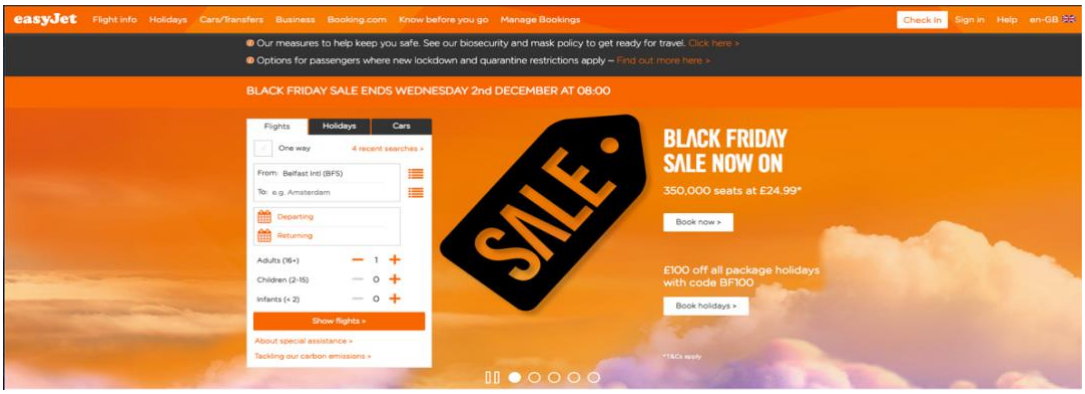
EasyJet Database Design Report



Introduction

Easyjet.com is an airline booking system website in association with EasyJet Airlines. The website provides a platform for customers to book a variety of destination flights and package holidays. In addition, it provides customers with products and other services from third party suppliers that are often required when booking a holiday or travelling. Additional products include hotel bookings, holiday / travel insurance, car rental and airport transfers.

The remit of this assignment was to design and implement a working database of easyjet.com. Due to the size and complexity of easyjet.com we have opted to mirror the operation of only a few select aspects of the real airline booking system. The design of the database has been constructed through MySQL and populated with sample data to demonstrate the databases features.

In planning the assignment In our project group, in the early stages we made some of the following design assumptions: (I have also included some additional early design assumptions I have made outside of the group)

- The model will assume that the website focuses on the sale of flights only.
- We have excluded the sales of additional add ons e.g. hotel bookings, holiday / travel insurance, car rental and airport transfers.
- It is essential to be have an EasyJet account (User) to book a flight.
- EasyJet Terms and conditions are automatically accepted by the user when booking a flight.
- All flights will be direct flights. (My own assumption)
- Users will see all prices in pound sterling – conversions will be completed at booking total stage to relevant currency if applicable. (My own assumption)

To begin our design implementation we firstly had to work on the entity relationships that existed within the EasyJet website. This is defined as creating an entity-relationship model. The model is the design of a database that that can be implemented as a working database after all the discoveries are made. We began by reverse engineering the flight booking process of the website to discover the structure of the database and the possible entities and attributes that may be relevant to the assignment . To display this structure visually, we represented these entitles and attributes using an Entity-relationship diagram (ER Diagram).

Entity Relationship Diagram Development

Figure.1 is our Initial group brainstorm design of entity relationships from the flight booking process of easyjet.com.

Figure.2 represents the final group ER Diagram design complete with relational lines to represent cardinality between entities.

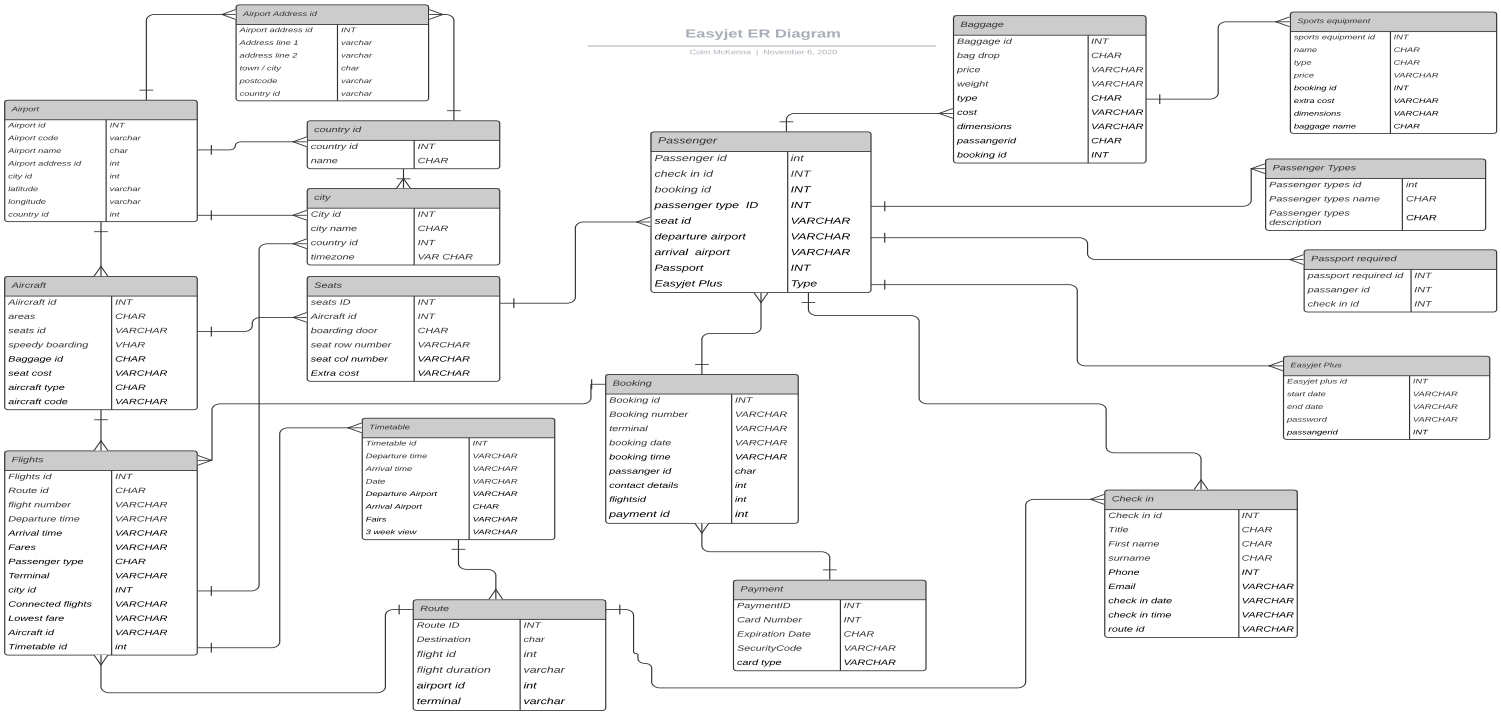
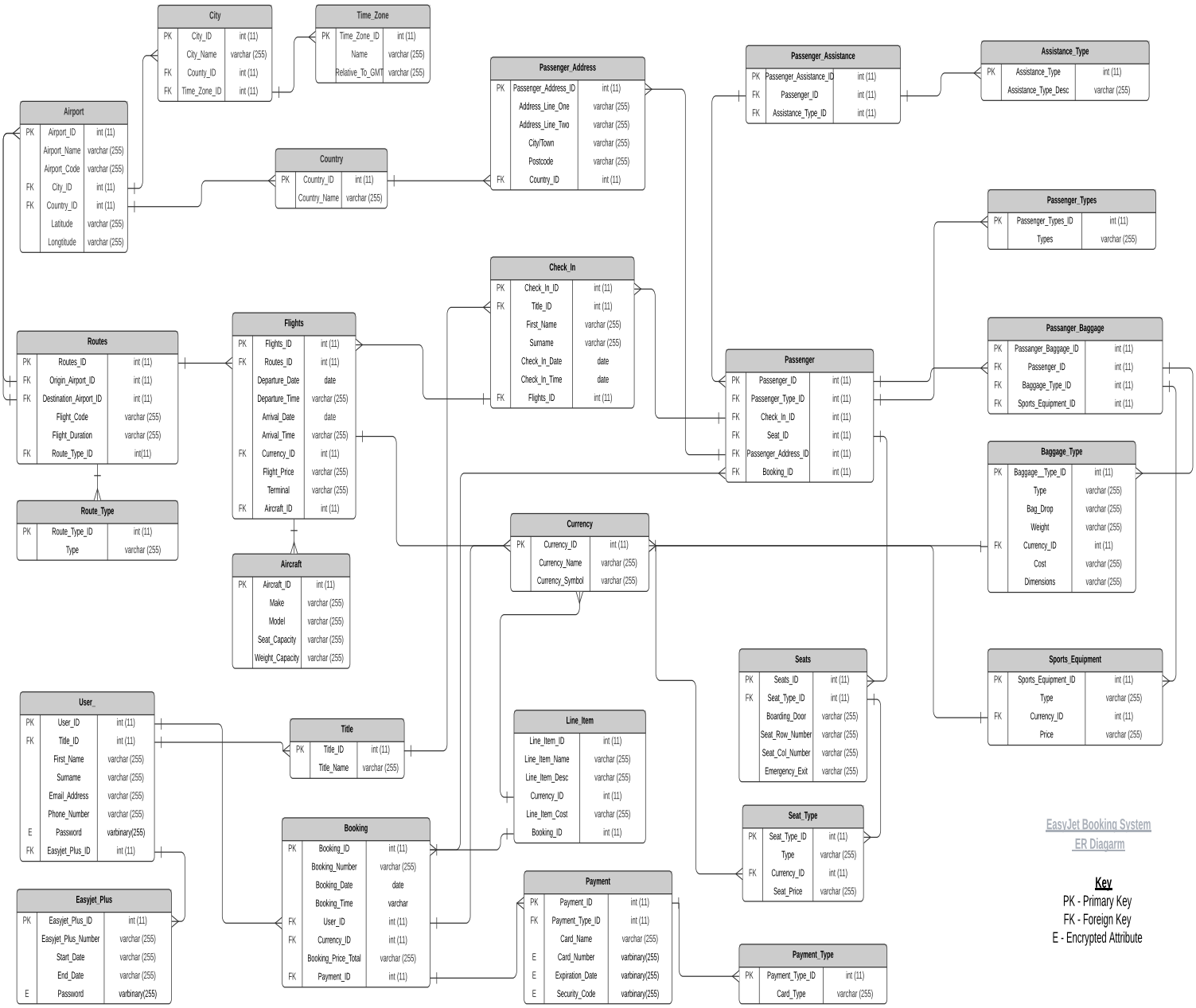


Figure.3 displays the final representation of the ER Diagram after the completion of the database.



- Note Figure.3 is complete with a key to represent the primary and foreign key relationships within the database and shows the attributes that have been encrypted.
- Data types and lengths are also represented in the ER Diagram.

The development of the database design is apparent through the continual improvement in the overall design over the course of the assignment, from the original brainstormed Excel tables, through to the final group ER diagram and then the final completed ER diagram. It is evident that the design evolved as the tables that included repeated data in the initial design were normalised out. Tables were removed from the initial design e.g. restricted items as we narrowed our focus solely on the flight booking process and then other relevant tables were inserted in their place to model the real world relationships as closely as possible.

The finalized ER Diagram can be broken down in to three abstract layers that become interlinked as the flight booking process is completed by the user.

- Layer 1 - Represents the link between airports and their geographical location (Countries, cities, time zones).
- Layer 2 - Represents the link between routes, flights and the aircraft involved.
- Layer 3 - Represents the link between the user and the relationships between the user/booking, passengers and passenger add ons. User payment relationships are also included in this layer.

Looking more in depth at a selection of the tables in the final completed design we can investigate how the tables were constructed, the relationships between tables and the data types used within the tables.

User Table

The User table is key to our overall database design. The user is essential in the flight booking process and is part of the basis in which our design is focused. The role of the user was one of the earliest design assumptions made and is the starting point for the flight booking process. Without being a user (having an EasyJet account) you are unable to book a flight in our model.

In the User table, User_ID is assigned as the primary key. A primary key is a unique identifier for a table in a database. When setting up this primary key I made use of the Auto Increment feature of phpMyAdmin. Auto-Increment allows a unique number to be automatically generated when a new record is inserted into a table and adds a value of 1 to the new insert from the previous insert . This entity is of type int 11 meaning any data recorded will be a whole number with maximum 11 digits.

Title_ID and Easyjet_Plus_ID are also set as type int 11 however they are assigned as foreign keys in the table. A foreign key is a key used to link two tables together. The foreign key refers to the primary key of another table.

First_Name, Surname, Email_Address and Phone_Number are set as type Varchar 255. This mean they can be populated with a variety of letters and numbers with maximum 255 characters.

The Password entity in this table is of data type Varbinary 255. Varbinary is similar to varchar but stores binary byte strings rather than character strings with a maximum 255. What also differentiates the Password entity from the rest of the attributes in the table is that the data held is encrypted. The data has been encrypted using the AES_Encrypt function . This will help reduce the possibility of unwanted users accessing sensitive or confidential information and preventing a data breach in the database.

Booking Table

In the Booking table, Booking_ID is assigned as the primary key. This is advantageous to us as now every booking is unique and it is easily identifiable to us when searching what passengers and what line items are linked to an individual booking. This entity is of type int 11 meaning any data recorded will be a whole number with maximum 11 digits.

User_ID , Currency_ID and Payment_ID are set as type int 11 and are assigned as foreign keys in the table. Foreign keys in this table are useful as they provide us with the ability to link the overall booking with the User table and the Payment methods.

Booking_Number, booking_Time and Booking_Price_Total are set as type Varchar 255. This mean they can be populated with a variety of letters and numbers with maximum 255 characters like in the previous table. The Booking_Number is individual to each booking in our model but this number has been manually inputted in the database design end and has not been auto incremented by phpMyAdmin.

Booking_Date is a unique data type in this table as it is set as type Date. The data is inserted in the form YYYY/MM/DD and is set at 11 characters. The functionality of the date data type is useful when calculating dates. In our database we can use this to check the length of time between a set date and the date a flight is departing.

Routes Table

In the Routes table, Routes_ID is assigned as the primary key. As with all primary keys in our database it as set as data type int 11.

Route_Type_ID, Origin_airport_ID and Destination_Airport_ID are assigned as foreign key constraints in this table with data type int 11. What makes this unlike any other table is that two foreign keys are linked to one primary key, in this instance both foreign keys (Origin_airport_ID and Destination_Airport_ID) are linked to the Airport table's Airport_ID primary key. This iterates the value and importance of naming conventions within the database as creating unique names in this situation is essential for referencing the foreign keys at a future date and removes any chance of confusion.

Flight_Code and Flight_Duration are set as type Varchar 255. Similarly to The Booking_Number in our previous example, Flight_Code incorporates a number that is individually set to each flight in our model and this number has been manually inputted in the database design end and has not been auto incremented by phpMyAdmin.

Database Design Features

Overall database design features

The features below present some of the overall key design elements that I considered to be essential to the overall database quality.

- The database will reflect the real-world structure of the flight booking system.
- All tables and relationships should be easily identifiable and accessible to the database user.
- The database avoids data redundancy.
- The database Supports data integrity.

I will now go further in to the important design decisions, highlighting the key design decisions made throughout the creation of the database.

Flight Booking Process

These are the key design features of the booking section of the database:

- Users are the starting point in the booking process.
- Users are the primary link between the overall booking and the passenger.
- One user booking can be linked to multiple passengers.
- Many bookings can be linked to one individual flight.
- Booking total price can be grouped using the unique booking ID.

As flights are commonly booked as a group booking e.g. a family holiday booking , it was essential that users had the capability of booking multiple passengers to a single flight and/or booking multiple passengers to multiple flights. The database was then responsible for the amalgamation of these as one singular booking that is identifiable through a unique booking ID. The total cost of the booking is also linked through the unique booking ID. In doing so, any costs that relate to passengers such as flight ticket price, seat picks and baggage cost can be totalled and be collectively stored under the one booking. The need for the unique booking ID is also essential in the possible situation of users being incorrectly charged for an item or flight as it becomes easier to spot and rectify the mistake using this unique key. Users payment details are stored in this section of our database and the decision to use AES_Encrypt encryption was made to ensure the integrity and security of the sensitive data.

Flight Availability

Flight availability is another key design decision of the database. The seating capacity of a flight is established through the Flights table and the Aircraft table. An individual aircraft is linked to a flight and so we can see from this early connection the seating capacity availability for users on the flight. Visually the amount of seats available can be seen by the user when picking a seat in the booking process so it is crucial our database is able to replicate simultaneously the data the website and availability that the user can see. The database stores the information to calculate the remaining seat availability on the flight, simultaneously reducing the available seats as the flight is booked by users. This is essential to eliminate the chance of a flight becoming overbooked and key to the logistics of the flight as we can now have the information to connect the individual seat with the passenger and so in the event of an emergency or a passenger being late they are easily identifiable through this link.

Another essential flight availability design is the capability of the database to store data relating to multiple flights departing from the same location with the same end destination. This mirrors the real life booking system as it is common to have multiple flights flying daily between the same destinations. In the database these are identifiable through unique flight codes. The flight prices and dates can also be obtained from the database and so in turn leads to greater user engagement and flexibility in the booking process as they have a choice when choosing flights to meet their needs and requirements.

Geographical Location

Geographical Location is an integral part of the database design that we had to factor in to make key decisions. The database design had to have the ability to represent a number of different factors relating to geographical location. As EasyJet flies to multiple countries from various locations worldwide we had to be able to store and retrieve data surrounding this. The database stores a selection of information relating to various countries, cities and time zones. The database has the capability of representing the airports, routes and flights that fly from multiple countries and cities. It must be able to show all the available airports in a country and a city, as it is feasible that a city may have multiple airports. It is also possible to show a correlation between passengers and their closest airport through the passenger address table. Airport data referencing an individual airports latitudes and longitudes is present in the database. This is key to air travel as these are used to examine the fastest, shortest, and most efficient routes between two airports.

Database Normalisation

Database normalisation is the process of organising a relational database. The purpose of Normalisation is to reduce data redundancy and to improve data integrity. For the assignment we considered the first 3 rules of Database normalisation known as First, Second and Third Normal Form (abbreviated as 1NF, 2NF, and 3NF).

First, Second and Third Normalisation Defined;

First Normal Form 1NF: The information is stored in a relational table with each column containing atomic values. There are no repeating groups of columns.

Second Normal Form 2NF: The table is in first normal form and all the columns depend on the table's primary key. There are no partial dependencies on a concatenated key.

Third Normal Form 3NF: The table is in second normal form and all of its columns are not transitively dependent on the primary key

I will now explain some of the important normalisation decisions taken in the database design.

Airport

Normalising out the Airport table was one of the first normalisation decisions made in the process. This was because a singular airport has many flights arriving and departing each day. As a result data would have been constantly repeated in our database and would have broken the First Normal Form rule. Normalising the data helped with eliminating the redundancy of data. Airport_ID became the primary key in the Airport table and was used as a foreign key constraint in the Routes table.

Title

When recording user names and check in names in the relevant tables there would have been a large amount of repeated data as many users would have used the same prefix title again breaking the First Normal Form rule. As a result the prefix title was normalised out to a new table and Title ID was used as the Primary key of the new table. TitleID also became a foreign key in the User table and Check_In table.

Baggage Type

Adding passenger baggage is a key element of the flight booking process. EasyJet provide a select amount of baggage options for each passenger. As all passengers will require at least one of the options it was essential to create a baggage type table and make this normalisation decision. With the potential of having upwards of 180 passengers on a flight, creating the Baggage_Type_Id made this population of this data linked to the passenger much more efficient and massively helped with data redundancy in our database. Using the Second Normal Form rule made Baggage_Types_ID easily identifiable through the primary key in its use as a foreign key in multiple tables. We were also able to store the price of the item in the table.

Currency

There are various elements throughout our database that required the use of a currency prefix. As a result the decision was made to normalise this data. This means that both the First and Second Normal Form rules were met. The currency table was constructed and used in relation to other tables. In the final design the currency primary key was the table referenced by the most other tables as a foreign key constraint.

Database Design Improvements

There are various ways in which improvements to the database design could be made.

We can firstly look at the overall scope of the design we decided on and include some of the early design decision assumptions we chose to remove for the purpose of the assignment. The reintroduction of add ons e.g. hotel bookings, holiday / travel insurance, car rental and airport transfers would more accurately represent the real life booking system as a whole and would make our database more complete and complex.

The database lacks some sense of flexibility in terms of dealing with costings and currency.
With EasyJet being used worldwide, it would be advantageous that users in different countries were shown costings in their native currency throughout the booking process and not just at the end like in our database design. Currency conversion could be coded in using java and be linked to current

exchange rates that update in real time. It would also be beneficial to have the ability to store the price history in the database. In future I would add this element as it gives the database user more insight to price and booking patterns and means they are be able to investigate more data relating to this aspect e.g. if they were investigating the price of an item on a set date.

Although we have attempted to maintain data integrity through encryption I would improve the security of our database. AES Encryption was used but for the purpose of our database we have used the same secret key multiple times. In the real world this would be unacceptable as our database would be easily susceptible to hackers and data breaches. To further increase security I would use an alternative encryption method or use a third party database security company that specialises in data encryption and the overall security features of the database.

The previous paragraph leads us in to our final database improvement, making sure our database is GDPR (General Data Protection Regulation) compliant. GDPR establishes rules on how companies, governments and other entities can process the personal data of citizens who are EU citizens or residents. GDPR aims to strengthen and unify data protection laws for all individuals across the European Union. Taking these new laws in to consideration we can enact them to create greater database integrity. We can firstly create and enforce roles and permissions, making sure a user approves terms and conditions of the storing of their data. We can limit the amount of users who have access to the database and create alerts that notify us of any attempted breaches of data. Finally, we can map the flow of data to have the ability to review effective ways of data processing and be able to identify any unintended uses.

EasyJet and Big Data

Big data is a term that describes the large volume of data that a business receives on a daily basis. it is possible for EasyJet to analyse big data from purchase activity to see demand patterns, known as business intelligence. If it sees the demand for a certain route increasing, they can adjust prices accordingly. From this information that is stored in the database, EasyJet can also identify which customer segments are price sensitive, and determine a segment's price range for a given route. [1]

EasyJet also invested in an artificially intelligent algorithm that determines seat pricing automatically, depending on demand. The system can also analyse historical data to predict demand patterns up to a year in advance. [2] This use of big data impacts future decision-making about new routes, schedules, and all this comes from data that is stored in the database over time.

Conclusion

In conclusion, the database design successfully meets the requirements of the flight booking system under the constraints and assumptions we proposed early in the design process. The implemented system allows for multiple bookings to be registered in the system and checked against flight availability. With the some of the mentioned improvements implemented, the design could be a close representation of the database used by EasyJet. The use of data normalisation helps to maintain data integrity and redundancy and overall provides the basis for business intelligence and analytics through big data.

Appendix

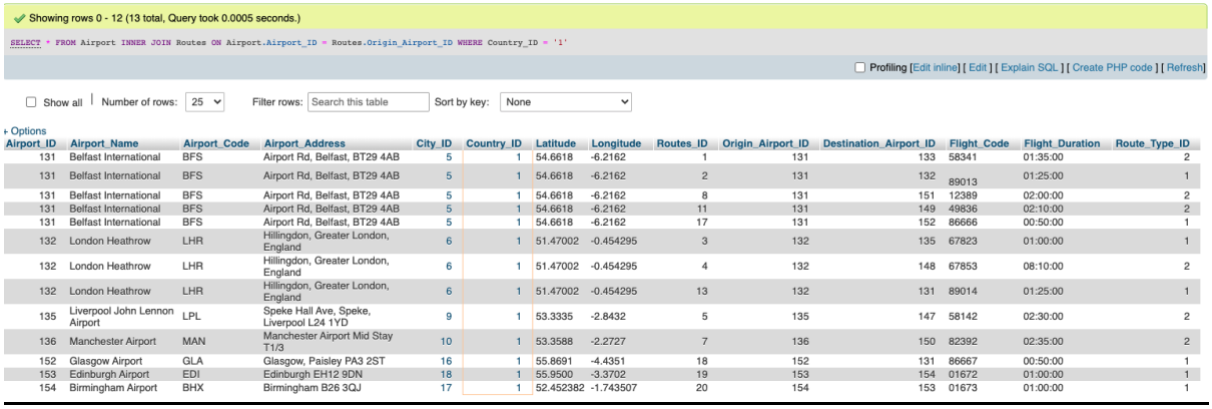
[1] https://blog.datumize.com/5-relevant-examples-of-a-big-data-case-study-from-the-airline-industry

[2] https://blog.datumize.com/5-relevant-examples-of-a-big-data-case-study-from-the-airline-industry

Video link - https://youtu.be/fMJYUb81wQY

Queries

1) Show all routes from uk



Showing rows 0 - 12 (13 total, Query took 0.0005 seconds.)

```
SELECT * FROM Airport INNER JOIN Routes ON Airport.Airport_ID = Routes.Origin_Airport_ID WHERE Country_ID = '1'
```

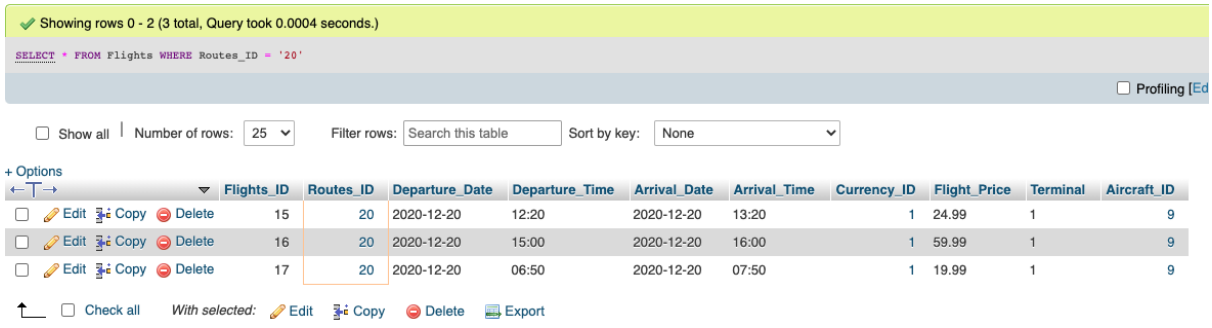
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

+ Options

Airport_ID	Airport_Name	Airport_Code	Airport_Address	City_ID	Country_ID	Latitude	Longitude	Routes_ID	Origin_Airport_ID	Destination_Airport_ID	Flight_Code	Flight_Duration	Route_Type_ID
131	Belfast International	BFS	Airport Rd, Belfast, BT29 4AB	5	1	54.6618	-6.2162	1	131	133	58341	01:35:00	2
131	Belfast International	BFS	Airport Rd, Belfast, BT29 4AB	5	1	54.6618	-6.2162	2	131	132	89013	01:25:00	1
131	Belfast International	BFS	Airport Rd, Belfast, BT29 4AB	5	1	54.6618	-6.2162	8	131	151	12389	02:00:00	2
131	Belfast International	BFS	Airport Rd, Belfast, BT29 4AB	5	1	54.6618	-6.2162	11	131	149	49836	02:10:00	2
131	Belfast International	BFS	Airport Rd, Belfast, BT29 4AB	5	1	54.6618	-6.2162	17	131	152	86666	00:50:00	1
132	London Heathrow	LHR	Hillingdon, Greater London, England	6	1	51.47002	-0.454295	3	132	135	67823	01:00:00	1
132	London Heathrow	LHR	Hillingdon, Greater London, England	6	1	51.47002	-0.454295	4	132	148	67853	08:10:00	2
132	London Heathrow	LHR	Hillingdon, Greater London, England	6	1	51.47002	-0.454295	13	132	131	89014	01:25:00	1
135	Liverpool John Lennon Airport	LPL	Speke Hall Ave, Speke, Liverpool L24 1YD	9	1	53.3335	-2.8432	5	135	147	58142	02:30:00	2
136	Manchester Airport	MAN	Manchester Airport Mid Stay T1/3	10	1	53.3588	-2.2727	7	136	150	82392	02:35:00	2
152	Glasgow Airport	GLA	Glasgow, Paisley PA3 2ST	16	1	55.8691	-4.4351	18	152	131	86667	00:50:00	1
153	Edinburgh Airport	EDI	Edinburgh EH12 9DN	18	1	55.9500	-3.3702	19	153	154	01672	01:00:00	1
154	Birmingham Airport	BHX	Birmingham B26 3QJ	17	1	52.452382	-1.743507	20	154	153	01673	01:00:00	1

2) Show multiple flights from same route from same location with same destination airport on same date.



Showing rows 0 - 2 (3 total, Query took 0.0004 seconds.)

```
SELECT * FROM Flights WHERE Routes_ID = '20'
```

Profiling [Ed

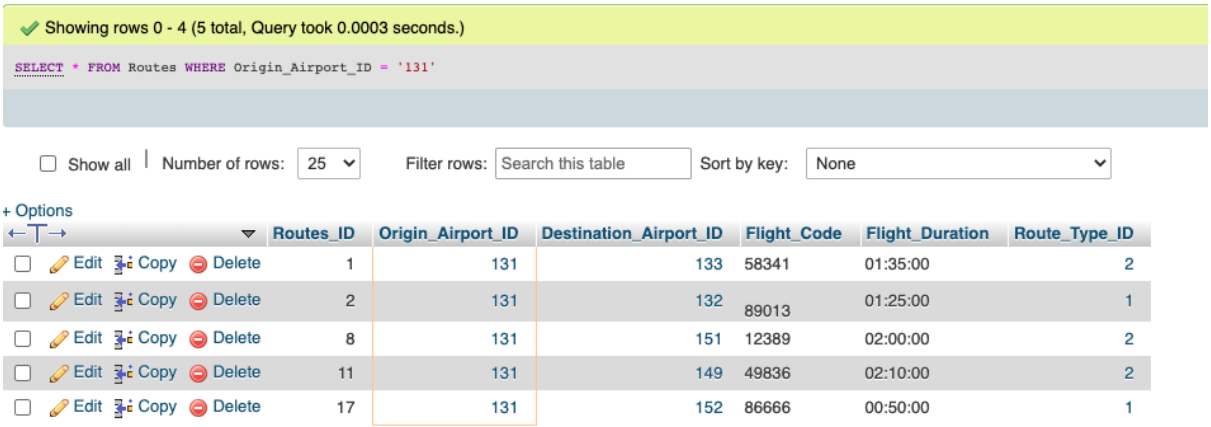
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

+ Options

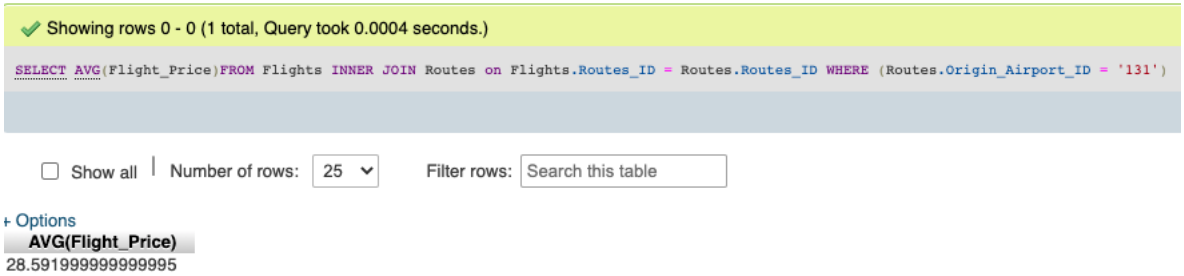
	Flights_ID	Routes_ID	Departure_Date	Departure_Time	Arrival_Date	Arrival_Time	Currency_ID	Flight_Price	Terminal	Aircraft_ID
Edit Copy Delete	15	20	2020-12-20	12:20	2020-12-20	13:20	1	24.99	1	9
Edit Copy Delete	16	20	2020-12-20	15:00	2020-12-20	16:00	1	59.99	1	9
Edit Copy Delete	17	20	2020-12-20	06:50	2020-12-20	07:50	1	19.99	1	9

Check all With selected: Edit Copy Delete Export

3) show flights from Belfast to various destinations



4) Show capabilities of the database - show Average price of flights from Belfast
Round number - SELECT ROUND(28.591999999999995,2)



---

5) Transaction to insert data in passenger and check in tables

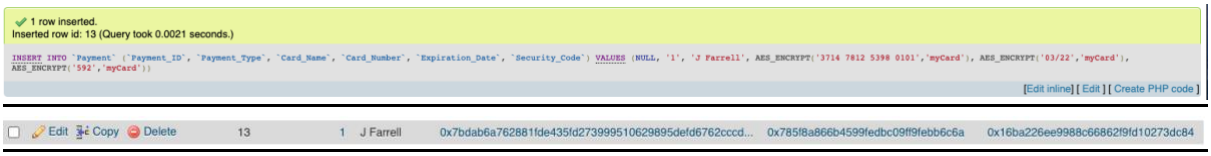
START TRANSACTION;

INSERT INTO `Passenger` (`Passenger_ID`, `Passenger_Type_ID`, `Check_In_ID`, `Seat_ID`, `Passenger_Address_ID`, `Booking_ID`) VALUES
(NULL, 1, 64, 27, 1, 1);
SET @last_id_in_Passenger_ID = LAST_INSERT_ID();

INSERT INTO `Check_In` (`Check_In_ID`, `Title_ID`, `First_Name`, `Surname`, `Check_In_Date`, `Check_In_Time`, `Flights_ID`) VALUES
(NULL, 1, 'James', 'McManus', '2020-12-20', 17:30, 2);
SET @last_id_in_Check_In_ID = LAST_INSERT_ID();

COMMIT;

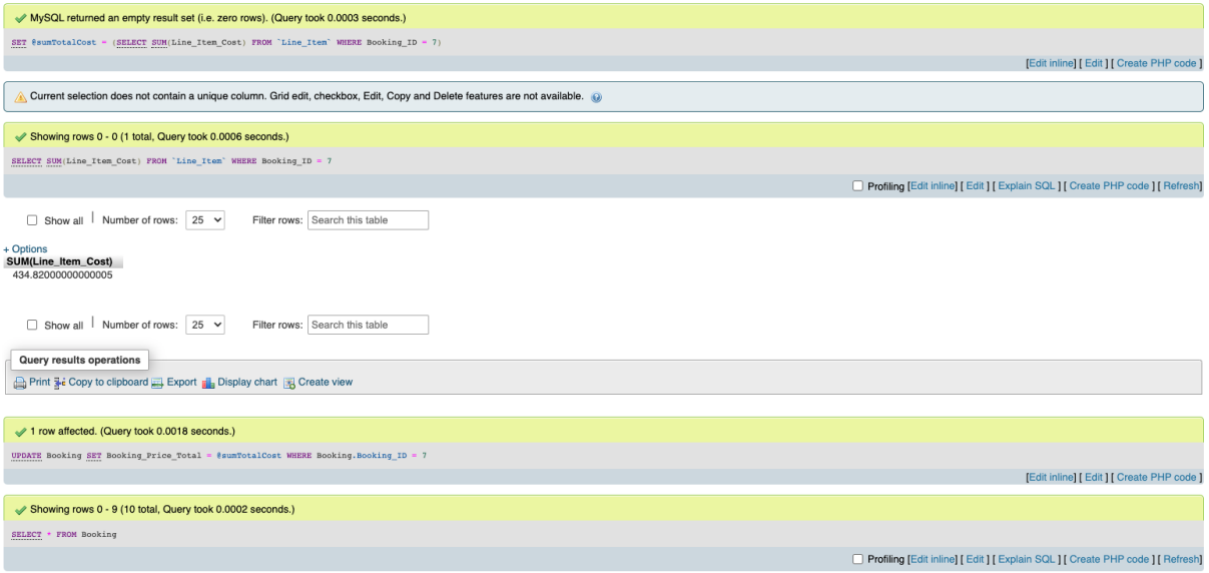
6) Encrypt card details for user



7) Get cost of line items – set in user booking

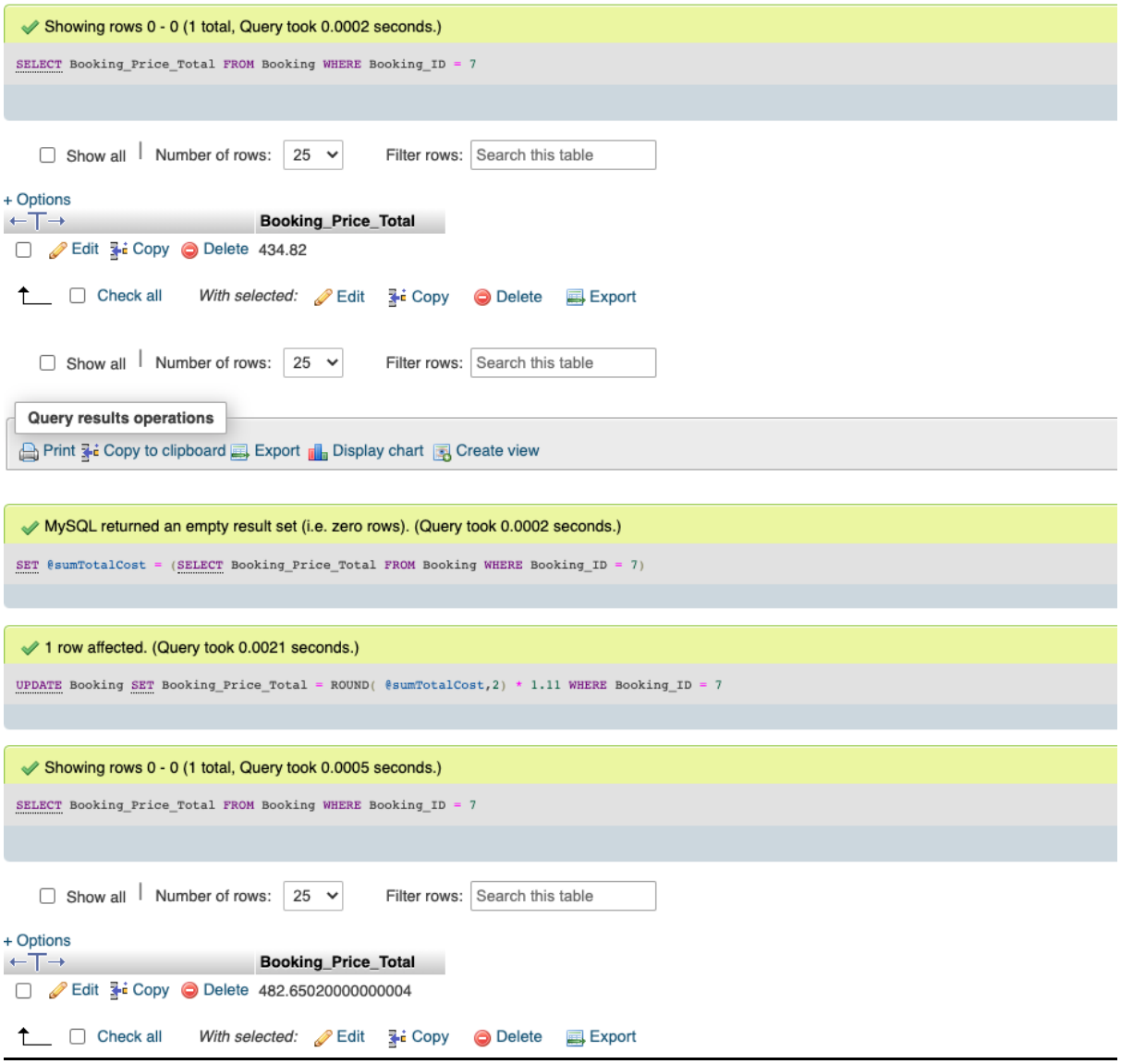
SET @sumTotalCost = (SELECT SUM(Line_Item_Cost) FROM `Line_Item` WHERE Booking_ID = 7);
SELECT SUM(Line_Item_Cost) FROM `Line_Item` WHERE Booking_ID = 7;
UPDATE Booking SET Booking_Price_Total = @sumTotalCost WHERE Booking.Booking_ID = 7;
SELECT * FROM Booking;

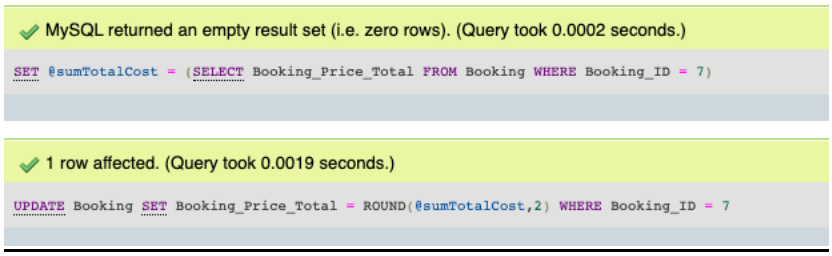
(Round to 2 decimal)
SELECT Booking_Price_Total FROM Booking WHERE Booking_ID = 7;
SET @sumTotalCost = (SELECT Booking_Price_Total FROM Booking WHERE Booking_ID = 7);
UPDATE Booking SET Booking_Price_Total = ROUND(@sumTotalCost,2) WHERE Booking_ID = 7;



9) Convert to relevant currency - Euro

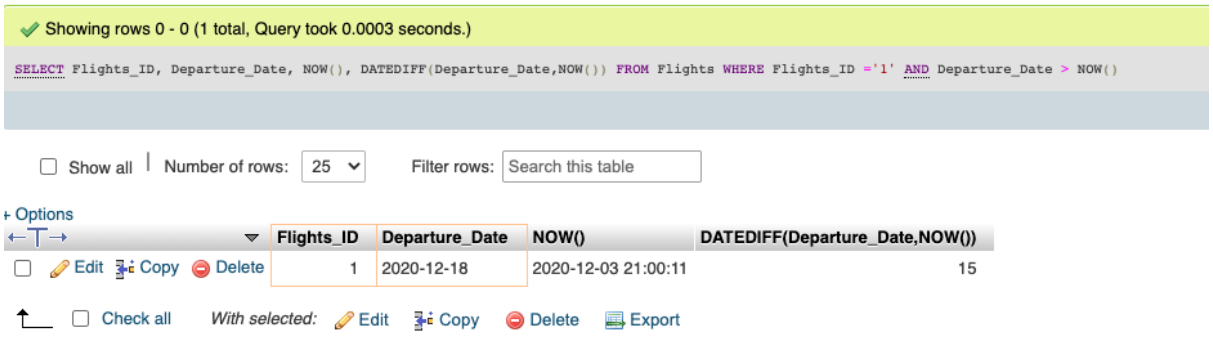
```
SELECT Booking_Price_Total FROM Booking WHERE Booking_ID = 7;
SET @sumTotalCost = (SELECT Booking_Price_Total FROM Booking WHERE
Booking_ID = 7);
UPDATE Booking SET Booking_Price_Total = ROUND( @sumTotalCost,2) * 1.11
WHERE Booking_ID = 7;

SELECT Booking_Price_Total FROM Booking WHERE Booking_ID = 7;
SET @sumTotalCost = (SELECT Booking_Price_Total FROM Booking WHERE
Booking_ID = 7);
UPDATE Booking SET Booking_Price_Total = ROUND(@sumTotalCost,2) WHERE
Booking_ID = 7;
```





Queries that didn't make the 10 minute video cut

10) Check when flight is leaving



11) Get seat capacity on aircraft after booking – Subtract amount of seats and checked in passengers

SET @totalOnAircraft = SELECT COUNT(Check_In.Check_In_ID from Check_In WHERE (Flights_ID = '4')
SET @totalbooked = SELECT Seat_Capacity FROM Aircraft INNER JOIN Flights ON Aircraft.Aircraft_ID = Flights.Aircraft_ID WHERE Flights_ID = 4
SET @totalAvailable = @totalbooked = @totalOnAircraft
SELECT @totalAvailable