Individual Database Design Report

Introduction

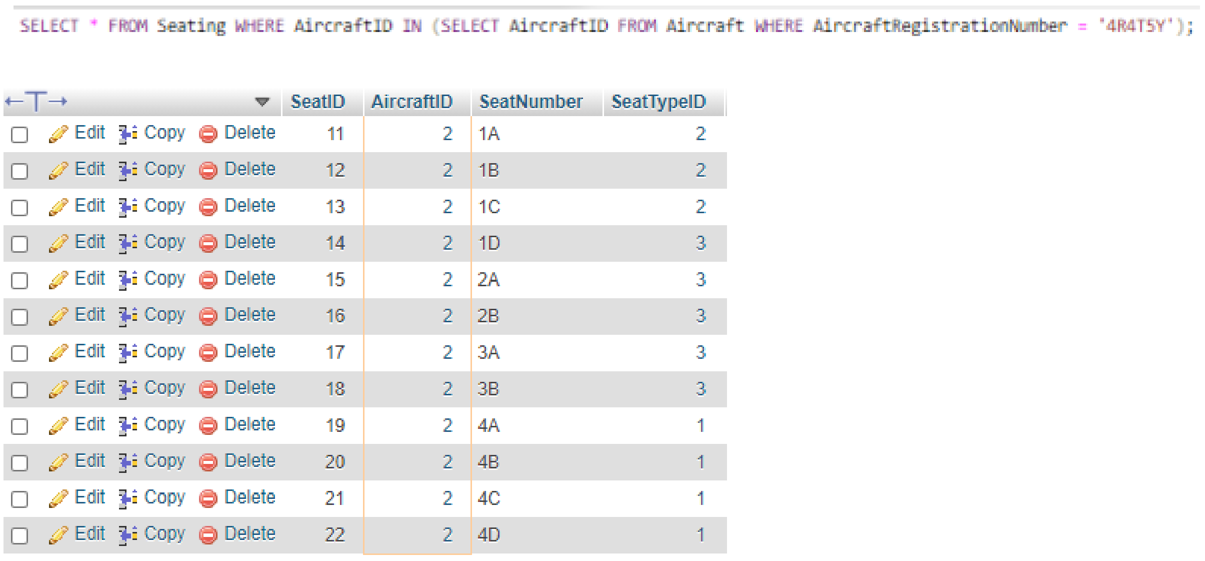
This report shall provide an analysis of the development process and major decisions behind the design and creation of my individual relational database. The model discussed is intended to support a 'real-world' flight booking system. As such, its entities, relationships and attributes are heavily inspired by EasyJet.com, as shall be detailed below. The final justifications behind the system's design and ultimate build are my own, although significant acknowledgement of my group's ardent and thoughtful efforts should not go amiss. Major areas of similarity and difference between my group's initial design and my own final model are discussed, with examples of SQL demonstration queries provided throughout, in the accompanying video submission and in the appendix to this document. In addition to discussion of the reasons behind core design decisions made for my database build, this report will a consider some plausible areas of further development, considered for the time-being to be beyond the feasible scope of this project. Some principles of relational database theory shall also be considered alongside the true-to-life practicalities which guided all design choices analysed hereafter.

Scope and Assumptions

In terms of scope, the aims of this project are confined within the flight-booking parts of the easyjet.com website. Thusly, any additional or third-party services the company provides, such as holiday insurance, car rental services and hotel bookings are superfluous to requirements, as are airport transfer services and the 'Disruption Help Hub' mentioned on the front-end of the site.

To enable feasibility and in lieu of a more formal requirements analysis process being conducted with the company, many reasonable design assumptions were made. For instance, it was deemed realistic to have one account holder be able to book as many or as few passengers, or as many or as few flights as he/she would like (so that an account can be created without needing to book a flight). It was also thought appropriate to have a flight booking entity loosely modelled around the idea of ticket or boarding pass in order to logically group certain key data required about a passenger. It was furthermore assumed that prices are liable to fluctuate frequently, as one might expect with the flight booking system of a major airline. The assumption was therefore also made that an order history or 'line-item' table to enable a saved static price of previous purchases would be a necessary and useful feature. Such an entity indeed seems to be alluded to under the name 'Basket' on the website's front-end.

Some further assumptions had to be delineated around the issue of seating on a given flight. For instance, it was assumed that infants need a passenger entity but not necessarily a corresponding seat on a plane (which can be achieved by simply excluding their PassengerTypeID from SQL queries). My final database system operates on the reasonable assumption that a finite number of seats with specific information about each seat must be stored by easyjet.com, along with aircraft details (including make, model and total seats for a particular vessel, see Figure 1).



```
SELECT * FROM Seating WHERE AircraftID IN (SELECT AircraftID FROM Aircraft WHERE AircraftRegistrationNumber = '4R4T5Y');
```

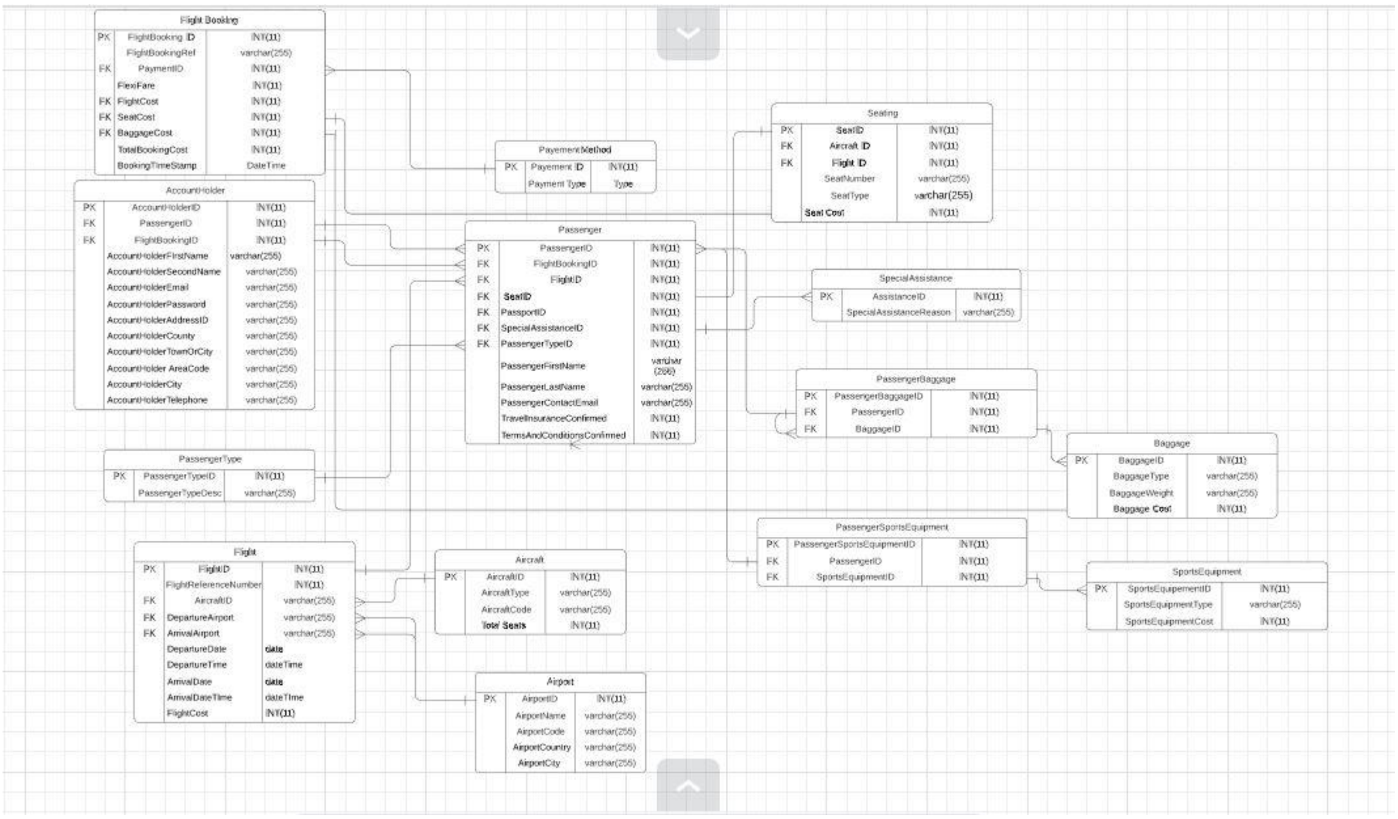
SeatID	AircraftID	SeatNumber	SeatTypeID
11	2	1A	2
12	2	1B	2
13	2	1C	2
14	2	1D	3
15	2	2A	3
16	2	2B	3
17	2	3A	3
18	2	3B	3
19	2	4A	1
20	2	4B	1
21	2	4C	1
22	2	4D	1

Figure 1: Use of a sub-query here returns which seat numbers are available on a particular aircraft with the corresponding type of each seat. This could enable the seat maps found on easyjet.com.

It was also reasonably assumed that 'many to many' cardinalities would need to exist between a given passenger and their requirements around luggage, special assistance and/or sports equipment. This assumption was based on the fact that no passengers or many passengers may require no or many instances of each of these entities to be stored.

Put simply, an account holder can have no or many flight bookings, and each flight booking can have one or many passengers. Each of those passengers would then need to be able have information persistently stored enabling them to book no luggage, sports equipment or special assistance requirements or multiple instances of each. In this way, the final database design should adhere quite effectively to a ‘real-world’ flight-booking system.

Other assumptions made primarily regarded which attributes to include within each entity table. Please see Figures 2 and 3 below for an exhaustive list of which entities, attributes and relationships were judged appropriate by both the group and myself.



Flight Booking
PK FlightBooking ID INT(11)
FlightBookingRef varchar(255)
FK PaymentID INT(11)
FlexiFare INT(11)
FK FlightCost INT(11)
FK SeatCost INT(11)
FK BaggageCost INT(11)
TotalBookingCost INT(11)
BookingTimeStamp DateTime
AccountHolder
PK AccountHolderID INT(11)
FK PassengerID INT(11)
FK FlightBookingID INT(11)
AccountHolderFirstName varchar(255)
AccountHolderSecondName varchar(255)
AccountHolderEmail varchar(255)
AccountHolderPassword varchar(255)
AccountHolderAddressID varchar(255)
AccountHolderCounty varchar(255)
AccountHolderTownOrCity varchar(255)
AccountHolder AreaCode varchar(255)
AccountHolderCity varchar(255)
AccountHolderTelephone varchar(255)
PassengerType
PK PassengerTypeID INT(11)
PassengerTypeDesc varchar(255)
PayementMethod
PK Payement ID INT(11)
Payment Type Type
Passenger
PK PassengerID INT(11)
FK FlightBookingID INT(11)
FK FlightID INT(11)
FK SeatID INT(11)
FK PassportID INT(11)
FK SpecialAssistanceID INT(11)
FK PassengerTypeID INT(11)
PassengerFirstName varchar (255)
PassengerLastName varchar(255)
PassengerContactEmail varchar(255)
TravelInsuranceConfirmed INT(11)
TermsAndConditionsConfirmed INT(11)
Seating
PK SeatID INT(11)
FK Aircraft ID INT(11)
FK Flight ID INT(11)
SeatNumber varchar(255)
SeatType varchar(255)
Seat Cost INT(11)
SpecialAssistance
PK AssistanceID INT(11)
SpecialAssistanceReason varchar(255)
PassengerBaggage
PK PassengerBaggageID INT(11)
FK PassengerID INT(11)
FK BaggageID INT(11)
Baggage
PK BaggageID INT(11)
BaggageType varchar(255)
BaggageWeight varchar(255)
Baggage Cost INT(11)
PassengerSportsEquipment
PK PassengerSportsEquipmentID INT(11)
FK PassengerID INT(11)
FK SportsEquipmentID INT(11)
SportsEquipment
PK SportsEquipementID INT(11)
SportsEquipmentType varchar(255)
SportsEquipmentCost INT(11)
Flight
PK FlightID INT(11)
FlightReferenceNumber INT(11)
FK AircraftID varchar(255)
FK DepartureAirport varchar(255)
FK ArrivalAirport varchar(255)
DepartureDate date
DepartureTime dateTime
ArrivalDate date
ArrivalDateTime dateTime
FlightCost INT(11)
Aircraft
PK AircraftID INT(11)
AircraftType varchar(255)
AircraftCode varchar(255)
Total Seats INT(11)
Airport
PK AirportID INT(11)
AirportName varchar(255)
AirportCode varchar(255)
AirportCountry varchar(255)
AirportCity varchar(255)

Figure 2: Final Group E-R Diagram.

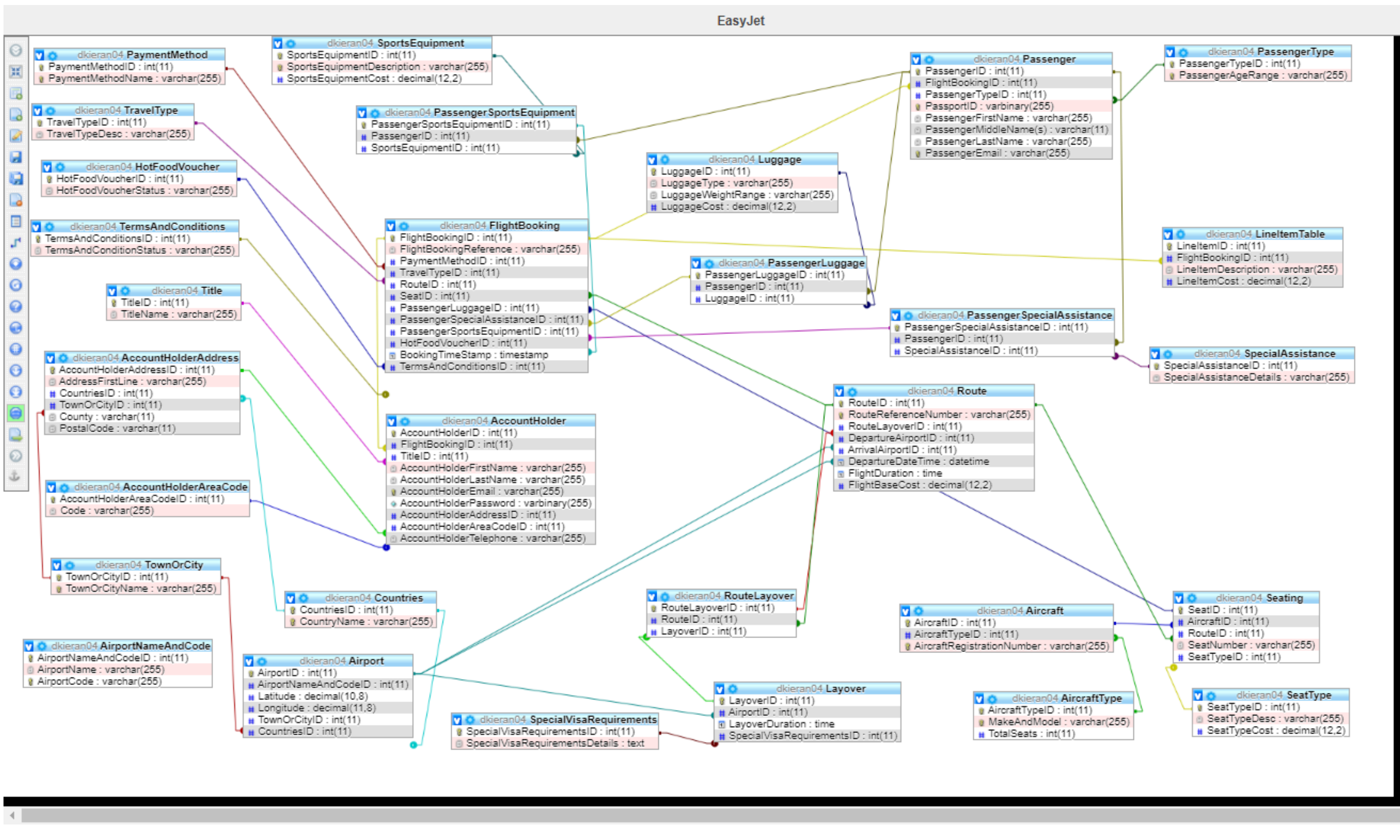
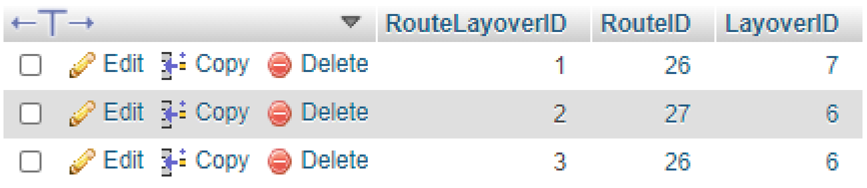


Figure 3: Final Individual Database System Model.

Significant Design Challenges and Decisions

As can be seen with Figure 2 and Figure 3, my finalised database system enacts a higher degree of normalisation than the group model. This has the effect both causing the diagram to appear more complex at first glance, and of actually making it a more easily maintained system that is less liable to suffer from inconsistencies or poor data quality stemming from unexpected user input. Before analysing specific examples, it is important to explain the theoretical justification behind my decision to diverge from my group in the implementation of a higher degree of normalisation.

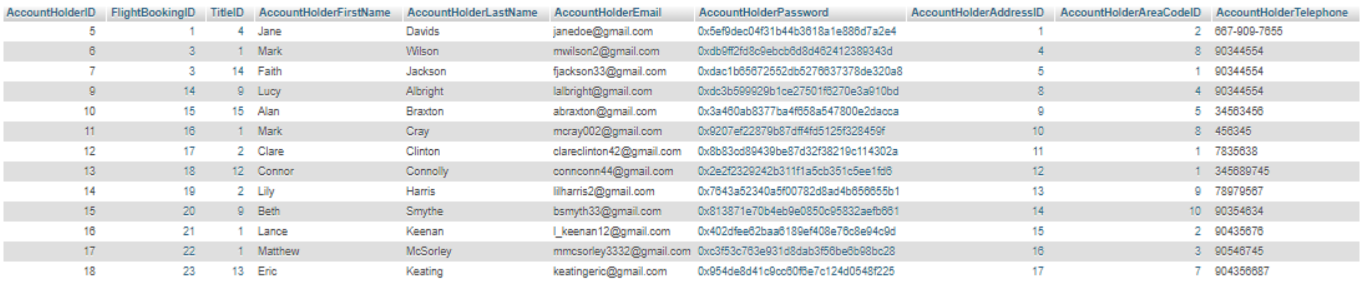
Firstly, some of the normalisation (created by usage of unique and auto-incremented foreign-keys) was simply an effective way to ensure the desired cardinality of the ‘many to many’ relationships discussed above. By adding a ‘service’ or ‘lookup’ table, two combined ‘one to many’ relationships allowed my system to store multiple special assistance/sports equipment/luggage requirements for any passenger (‘many to many’). A similar example of such a service table can be seen in Figure 4 below.



	RouteLayoverID	RouteID	LayoverID
☐ Edit Copy Delete	1	26	7
☐ Edit Copy Delete	2	27	6
☐ Edit Copy Delete	3	26	6

Figure 4 – RouteLayoverID - an example of a 'service' table enabling a 'many to many' relationship between Route and Layover. As discussed in the video submission, this is a speculative feature aimed at improving my system's future scalability, thus representing a reasonable preventative measure against the need to restructure my system's schema frequently.

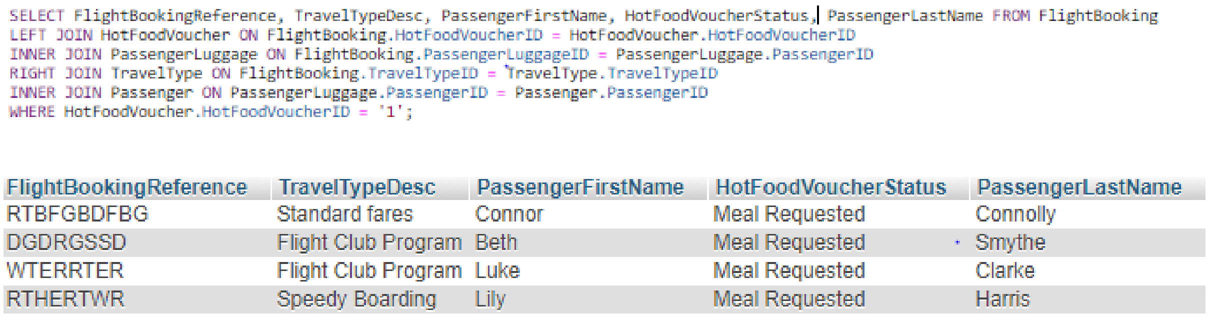
Secondly, normalisation, in so far as it involves the removal or reduction of redundancy, was an important way to ensure that my system followed as many of Dr Edgar F. Codd's rules for relational database systems as possible[1]. In particular, having more foreign-key constraints means that my system is less easily broken by adding, updating or deleting data. The lengthy consideration that was given to column headings should result in minimal need for future re-working of the schema, except where unforeseeable business circumstances emerge. Importantly, efforts were made to make the model as logical as possible so that it could be understandable and informative. This was largely achieved by taking care to silo data into relatively simple, mostly intuitive and generally small tables. An exception to this might be the AccountHolder table, one of the larger tables in my DBMS, as seen in Figure 5. I made the design decision that the attributes included in this table were fair assumptions relevant to the information asked for in the account setup section of the website.



AccountHolderID	FlightBookingID	TitleID	AccountHolderFirstName	AccountHolderLastName	AccountHolderEmail	AccountHolderPassword	AccountHolderAddressID	AccountHolderAreaCodeID	AccountHolderTelephone
5	1	4	Jane	Davids	janedoe@gmail.com	0x5ef9dec04f31b44b3618a1e886d7a2e4	1	2	667-909-7655
6	3	1	Mark	Wilson	mwilson2@gmail.com	0xdb9ff2fd8c9ebcb6d8d462412389343d	4	8	90344554
7	3	14	Faith	Jackson	fjackson33@gmail.com	0xdac1b65672552db5276637378de320a8	5	1	90344554
9	14	9	Lucy	Albright	lalbright@gmail.com	0xdc3b599929b1ce27501f6270e3a910bd	8	4	90344554
10	15	15	Alan	Braxton	abraxton@gmail.com	0x3a460ab8377ba4f658a547800e2dacca	9	5	34563456
11	16	1	Mark	Cray	mcray002@gmail.com	0x9207ef22879b87dff4fd5125f328459f	10	8	456345
12	17	2	Clare	Clinton	clareclinton42@gmail.com	0x8b83cd89439be87d32f38219c114302a	11	1	7835638
13	18	12	Connor	Connolly	connconn44@gmail.com	0x2e2f2329242b311f1a5cb351c5ee1fd6	12	1	345689745
14	19	2	Lily	Harris	lilharris2@gmail.com	0x7643a52340a5f00782d8ad4b656655b1	13	9	78979567
15	20	9	Beth	Smythe	bsmyth33@gmail.com	0x813871e70b4eb9e0850c95832aefb661	14	10	90354634
16	21	1	Lance	Keenan	l_keenan12@gmail.com	0x402dfee62baa6189ef408e76c8e94c9d	15	2	90435676
17	22	1	Matthew	McSorley	mmcsorley3332@gmail.com	0xc3f53c763e931d8dab3f56be6b98bc28	16	3	90546745
18	23	13	Eric	Keating	keatingeric@gmail.com	0x954de8d41c9cc60f6e7c124d0548f225	17	7	904356687

Figure 5 – AccountHolder Table.

Repetitive data was mostly removed to avoid violations of the second rule of First Normal Form. As mentioned in the video submission, concatenated keys were avoided in this model so violations of Second Normal Form were impossible. Care was also taken to ensure that Third Normal Form was violated as minimally as possible by ensuring that non-key attributes only consisted of facts about the unique primary(or foreign) keys in each table. This helped avoid some insert, update and deletion dependencies. However, one arguable drawback of the high degree of normalisation in my model is that some insert dependencies were inevitable, particularly in regards to the conceptually central FlightBooking table, where almost all attributes are foreign keys. One might seek to overcome such dependencies with an SQL transaction query, which would at least ensure the database does not fall into an inconsistent state. Regardless, my system design broadly fits the need of being easily queried, as exampled in Figure 6, where a combination of some inner and outer joins and use of a conditional clause shows that my system can store and return information about which passengers ordered a hot meal on a flight, and which class they are in, enabling EasyJet to make different meals for customers who have paid more or less. One might easily add joins to the Seating entity here to include a relevant seat number which could be useful in practice for flight staff.



```sql
SELECT FlightBookingReference, TravelTypeDesc, PassengerFirstName, HotFoodVoucherStatus, PassengerLastName FROM FlightBooking
LEFT JOIN HotFoodVoucher ON FlightBooking.HotFoodVoucherID = HotFoodVoucher.HotFoodVoucherID
INNER JOIN PassengerLuggage ON FlightBooking.PassengerLuggageID = PassengerLuggage.PassengerID
RIGHT JOIN TravelType ON FlightBooking.TravelTypeID = TravelType.TravelTypeID
INNER JOIN Passenger ON PassengerLuggage.PassengerID = Passenger.PassengerID
WHERE HotFoodVoucher.HotFoodVoucherID = '1';
```

FlightBookingReference	TravelTypeDesc	PassengerFirstName	HotFoodVoucherStatus	PassengerLastName
RTBFGBDFBG	Standard fares	Connor	Meal Requested	Connolly
DGDRGSSD	Flight Club Program	Beth	Meal Requested	Smythe
WTERRTER	Flight Club Program	Luke	Meal Requested	Clarke
RTHERTWR	Speedy Boarding	Lily	Meal Requested	Harris

Figure 6– Joined tables showing which passengers ordered a hot meal, and which class they are in, enabling EasyJet to make different meals for customers who have paid more or less.

Another way in which my design differs from the final group design is in the inclusion of latitude and longitude coordinate values in the Airport table. While this is obviously important information to store given the site's inclusion of a Route Map which visually pinpoints airports, storage of coordinates might also make possible more intricate pricing calculations for certain flights. A limitation here would be that, despite the appearance of the route map (see Figure 7), flight routes in actuality do not fly linearly. Rather, they take into account other aircraft and climate patterns, amongst other factors. Given the immense amount of internationally sourced, cross-corporation data it would take to adequately factor in such flight paths, this potential feature was considered far outside the scope of this database system in its current iteration.

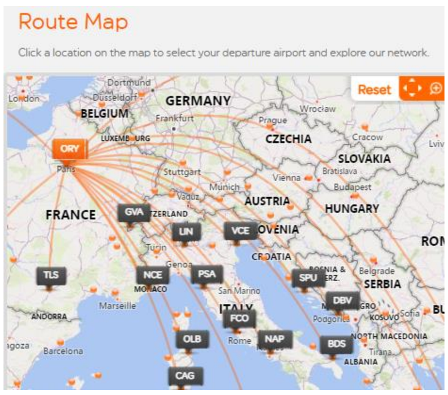
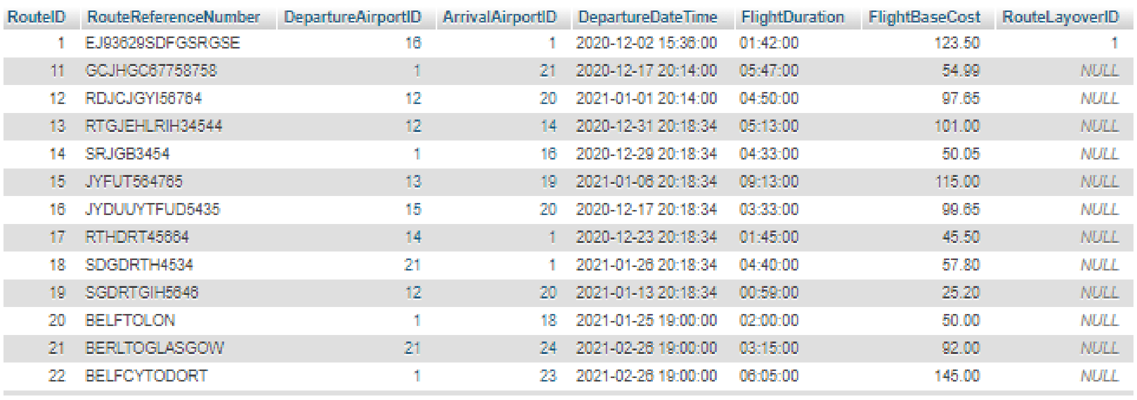


Figure 7 – Route map taken from easyjet.com

One further quite important distinction between the Group ER model and my own database is how the issue of flight times are handled. I firstly took the view that Route was a more logical name for the main flight entity, although its functional purpose remains the same in both models. In addition to this, I took the view that the more succinct table attributes (see Figure 8) used in my final design were superior to those found in our group's use of the DepartureDate, DepartureTime, ArrivalDate and ArrivalDateTime attributes. I reached this decision in consideration of the desire, as discussed above, to create generally smaller silos of information through the process of normalisation where appropriate.



RouteID	RouteReferenceNumber	DepartureAirportID	ArrivalAirportID	DepartureDateTime	FlightDuration	FlightBaseCost	RouteLayoverID
1	EJ93629SDFGSRGSE	16	1	2020-12-02 15:36:00	01:42:00	123.50	1
11	GCJHGC67758758	1	21	2020-12-17 20:14:00	05:47:00	54.99	NULL
12	RDJCJGYI56764	12	20	2021-01-01 20:14:00	04:50:00	97.65	NULL
13	RTGJEHLRIH34544	12	14	2020-12-31 20:18:34	05:13:00	101.00	NULL
14	SRJGB3454	1	16	2020-12-29 20:18:34	04:33:00	50.05	NULL
15	JYFUT564765	13	19	2021-01-06 20:18:34	09:13:00	115.00	NULL
16	JYDUUYTFUD5435	15	20	2020-12-17 20:18:34	03:33:00	99.65	NULL
17	RTHDRT45664	14	1	2020-12-23 20:18:34	01:45:00	45.50	NULL
18	SDGDRTH4534	21	1	2021-01-26 20:18:34	04:40:00	57.80	NULL
19	SGDRTGIH5646	12	20	2021-01-13 20:18:34	00:59:00	25.20	NULL
20	BELFTOLON	1	18	2021-01-25 19:00:00	02:00:00	50.00	NULL
21	BERLTOGLASGOW	21	24	2021-02-26 19:00:00	03:15:00	92.00	NULL
22	BELFCYTODORT	1	23	2021-02-26 19:00:00	06:05:00	145.00	NULL

Figure 8 – Route Table and sample data. DepartureAirportID and ArrivalAirportID are normalised by foreign-key constraint to the Airport table, enabling a defined list of available airports to be accessed for all flights/ routes.

In this case, having fewer columns did no damage to the logical structure or maintainability of the table. Thus, my system uses the column headers DepartureDateTime and FlightDuration, with the data types 'datetime' and 'time' respectively. This more succinct design allows the values held for each flight/route to be easily passed to a mid-level programming language like Java. Of course, one might chose to use MySQL's function such as DATEDIFF() to manipulate such data into an arrival time, but one might also be slightly short-sighted in doing so, given the Java class library's superior means of parsing data, casting and then converting to specific time-zones, which is naturally of the utmost importance to the over-all functionality of a 'real-world' flight booking system.

Data Types and Lengths

For the issue of payments, live prices were stored in each relevant table with the data type and length decimal(12, 2). As demonstrated in the video demonstration for this report, these live prices are then collated in the line item table via a foreign-key relationship with the FlightBookingID attribute, allowing prices to be freely manipulated without affecting existing purchases.

It was important to ensure a consistent usage of data types and lengths in order to create a straight forward and easily updatable schema, especially in light of the normalisation decisions laid out in this report. ‘Int(11)’ and ‘varchar(255)’ were most frequently used in my system, with the former being the exclusive type and length of all foreign-key linked attributes. ‘Text(2550)’ was used for the SpecialVisaRequirements section, wherein detailed information can be stored on any layover countries which might have particular travel advice in place. ‘Timestamp’ was used to enable the recording of time of purchase in the FlightBooking table and ‘time’ was judged to be appropriate for the LayoverDuration and FlightDuration attributes in the Layover and Route tables respectively. Latitudes, which range from -90 to +90 degrees, were stored with decimal(10, 8). Longitudes, which range from -180 to +180 were stored with decimal(11,8)[3]. I choose to use int(11) for some attributes which could arguably be viewed as having a simple boolean logic - TermsAndConditionsAccepted is a notable example. In opting to user an integer type I avoided some known pitfalls of the Boolean data type and also enabled a wider range of options to be stored for the user’s status, should front-end designers choose to implement this wider range of possibilities or not. This small decision increased the effective capabilities of my design model. Space usage was not a major concern due to the dynamic space allocation offered by the InnoDB storage engine.[2]

Other Features

Some attributes, such as PassengerMiddleName and PassengerEmail in the Passenger table were purposefully set to null to mimic a realistic model: obviously not all passengers would have middle names or email addresses, but a flight booking system should still be capable of storing such information if needed. Similarly, the unique modifier was enabled on many attributes to ensure atomic data was maintained where necessary. A notable example of this was FlightBookingReference which, it was assumed, would be need to be a unique and customer visible reference for each entry to the FlightBooking table. Its customer visibility in particular meant that although uniqueness was important for this attribute, it would not be wise to use an integer auto-incremented by MySQL, as was done with the FlightBookingID and all other primary key attributes in my model.

In terms of naming conventions, I opted for what I believed was a more professional naming pattern than that which was present in the earlier group model. I used pascal-case for all entities and attributes in my system, believing consistency to be of the utmost importance. Whitespace was removed between words to remove any unexpected query results and a deliberately verbose naming pattern was used for increased clarity and because auto-complete features existing on most modern relational database management systems means that there is little reason to abbreviate titles. Service tables were given a consistent naming pattern through combination of the two linked tables. For example: PassengerSpecialAssistance, PassengerLuggage, PassengerSportsEquipment.

System Limitations and Areas of Possible Further Development.

Some areas of limitation have already been mentioned in passing. This section shall elaborate on such limitations and suggest some areas of future development which could enable this database to become truly viable in a non-academic environment. Firstly, the limited scope already discussed is of course a major limitation. Features such as holiday activity bookings, car rentals, airport transfers, latest travel information and hotel bookings make up prominent portions of easyjet.com(see Figure 9), and although they are mostly administered by third party providers and not strictly part of a flight booking system, they could be included in an enhanced future version of my relational database.

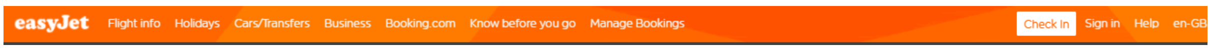
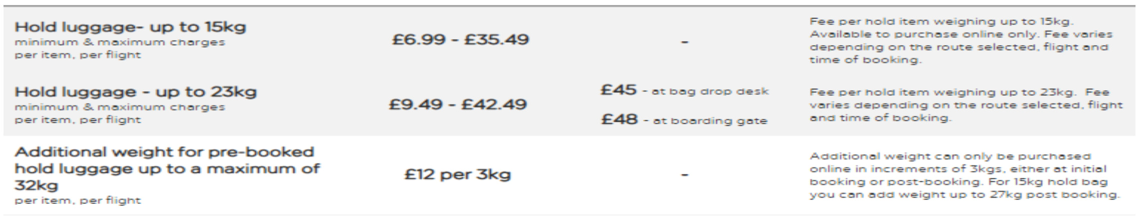


Figure 9 – Website header / banner showing site features so-far treated as being out of scope.

Of course, a front-end user-interface perhaps written in Javascript or a similar programming language would logically be the next major improvement that could be made to my current database. It is perhaps worth mentioning here that the real-life easy-jet website went through some visual changes towards the end point of my system's development. This was an unfortunate but by no means debilitating coincidence, as was the company's recent decision to charge for use of in-flight overhead compartments. This change could definitely justify its own entity or attribute in future versions of my database system.

As well the assumptions discussed above, some simplifications had to be made to certain aspects of my database for purposes of viability. For instance, luggage fees in my system are treated as a single price dependant on type and weight of the item. However, in reality the Fees and Charges section of the EasyJet website outlines a more complex pricing model based on route and time of booking (see Figure 10).



Hold luggage- up to 15kg minimum & maximum charges per item, per flight	£6.99 - £35.49	-	Fee per hold item weighing up to 15kg. Available to purchase online only. Fee varies depending on the route selected, flight and time of booking.
Hold luggage - up to 23kg minimum & maximum charges per item, per flight	£9.49 - £42.49	£45 - at bag drop desk £48 - at boarding gate	Fee per hold item weighing up to 23kg. Fee varies depending on the route selected, flight and time of booking.
Additional weight for pre-booked hold luggage up to a maximum of 32kg per item, per flight	£12 per 3kg	-	Additional weight can only be purchased online in increments of 3kgs, either at initial booking or post-booking. For 15kg hold bag you can add weight up to 27kg post booking.

Figure 10 – Example of detailed pricing information for separate items of luggage[4]

A similar simplification that had to be made in this version of my system was the decision to treat Business class as a regular attribute of the TravelType entity. Business customers in fact have their own separate section of the EasyJet website, which could simply be a marketing strategy, or perhaps indicates a separate database entirely.

Further areas of possible future development might include currencies, which were not implemented at the database level in my system to avoid having to deal with live currency fluctuations and non-aesthetic prices resulting from conversion equations, as the company would likely want to avoid this. Greater attention could also be paid to strategies around business intelligence, perhaps through the caching of customer activity to enable targeted marketing of site users. Some kind of multi-language capabilities would also be a reasonable improvement.

One debateable but important area of possible system improvement involves the storage of card details in my relational database. For this project, this was avoided due to known inadequacies with the Advanced Encryption Standard (particularly the potential for reverse engineering of secret keys). Current industry norm at the time of writing is to outsource the storage of card details to a third-party service provider such as Stripe or authorize.net[5]. Not only does this design decision aid the removal of logistical difficulties such as factoring frequently changing fees/ surcharges for certain payment methods, outsourcing the storage of card details also helps to minimise legal liabilities for any data security or GDPR breaches which might occur. AES-encryption was given to some areas of sensitive information in my system, such as passwords and passport numbers (see appendix), but to assume this level of encryption alone would suffice in a real world system lacks consideration of, amongst other things, the need for a secure server connection and appropriate database access permissions. Moreover, my decision was in line with the PCI Security Standards advice to only store card details 'if it is absolutely necessary'[7].

Conclusion

In summation, my implemented database model broadly meets the aim of enabling a full flight booking system to be constructed and maintained. Despite its limitations, my model provides a solid basis for further development and is grounded in consideration of the many practicalities which might be expected for a real-world flight booking system, as well as the theoretical study of relational database design. Should it be further developed along with a user interface, feedback from so-called 'early adopters' of such a system would undoubtedly throw up further points for consideration.

Appendix

Cited Sources

1 - (2007) Codd’s rules. In: Inside Relational Databases with Examples in Access. Springer, London. https://doi.org/10.1007/978-1-84628-687-2_24

2 - https://dev.mysql.com/doc/refman/8.0/en/innodb-file-space.html

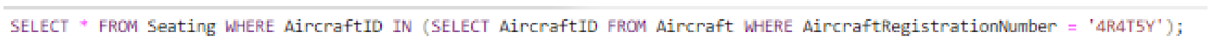
3 - https://docs.mapbox.com/help/glossary/lat-lon/#:~:text=Latitude%20and%20longitude%20are%20a,180%20to%20180%20for%20longitude

4 - https://www.easyjet.com/en/terms-and-conditions/fees

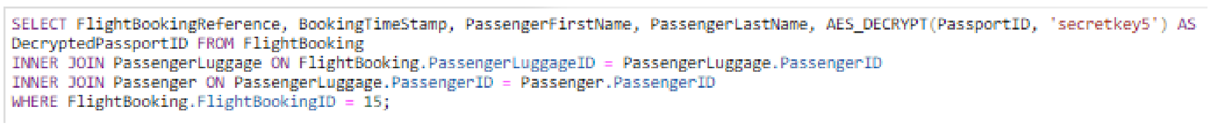
5- https://stackoverflow.com/questions/3002189/best-practices-to-store-creditcard-information-into-database

6 - https://www.pcisecuritystandards.org/pdfs/pci_fs_data_storage.pdf

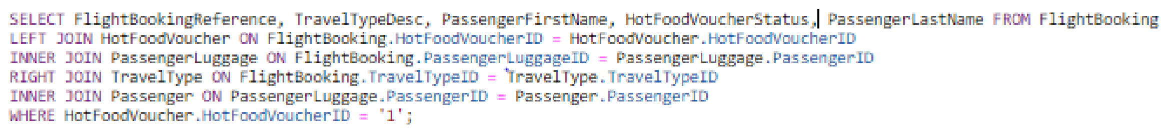
Some sample SQL Queries used in Video Demonstration and Project Development



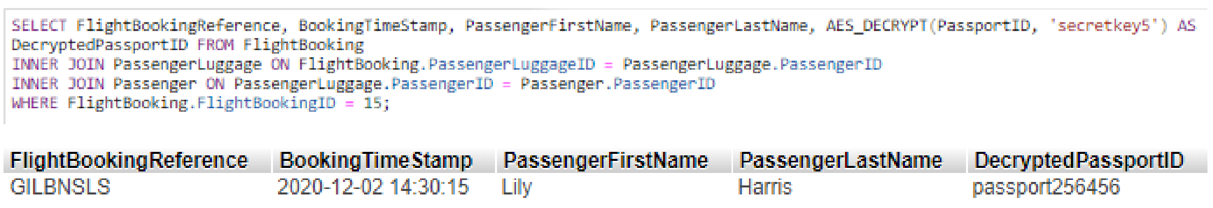
```
SELECT * FROM Seating WHERE AircraftID IN (SELECT AircraftID FROM Aircraft WHERE AircraftRegistrationNumber = '4R4T5Y');
```



```
SELECT FlightBookingReference, BookingTimeStamp, PassengerFirstName, PassengerLastName, AES_DECRYPT(PassportID, 'secretkey5') AS
DecryptedPassportID FROM FlightBooking
INNER JOIN PassengerLuggage ON FlightBooking.PassengerLuggageID = PassengerLuggage.PassengerID
INNER JOIN Passenger ON PassengerLuggage.PassengerID = Passenger.PassengerID
WHERE FlightBooking.FlightBookingID = 15;
```

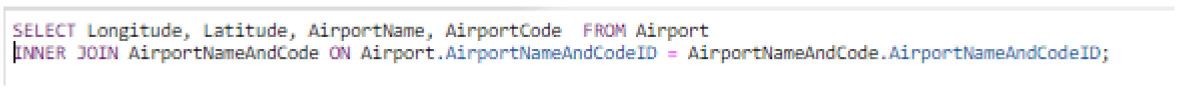


```
SELECT FlightBookingReference, TravelTypeDesc, PassengerFirstName, HotFoodVoucherStatus, PassengerLastName FROM FlightBooking
LEFT JOIN HotFoodVoucher ON FlightBooking.HotFoodVoucherID = HotFoodVoucher.HotFoodVoucherID
INNER JOIN PassengerLuggage ON FlightBooking.PassengerLuggageID = PassengerLuggage.PassengerID
RIGHT JOIN TravelType ON FlightBooking.TravelTypeID = TravelType.TravelTypeID
INNER JOIN Passenger ON PassengerLuggage.PassengerID = Passenger.PassengerID
WHERE HotFoodVoucher.HotFoodVoucherID = '1';
```

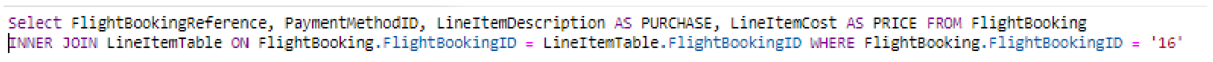


```
SELECT FlightBookingReference, BookingTimeStamp, PassengerFirstName, PassengerLastName, AES_DECRYPT(PassportID, 'secretkey5') AS
DecryptedPassportID FROM FlightBooking
INNER JOIN PassengerLuggage ON FlightBooking.PassengerLuggageID = PassengerLuggage.PassengerID
INNER JOIN Passenger ON PassengerLuggage.PassengerID = Passenger.PassengerID
WHERE FlightBooking.FlightBookingID = 15;
```

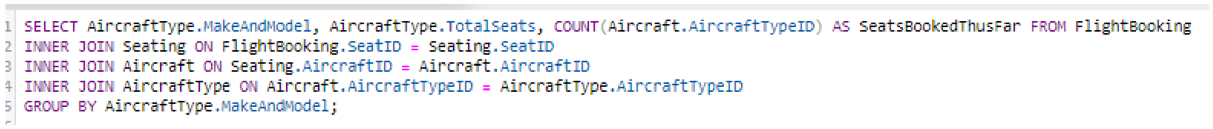
FlightBookingReference	BookingTimeStamp	PassengerFirstName	PassengerLastName	DecryptedPassportID
GILBNSLS	2020-12-02 14:30:15	Lily	Harris	passport256456



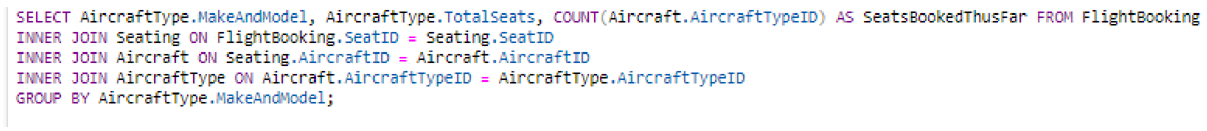
```
SELECT Longitude, Latitude, AirportName, AirportCode  FROM Airport
INNER JOIN AirportNameAndCode ON Airport.AirportNameAndCodeID = AirportNameAndCode.AirportNameAndCodeID;
```



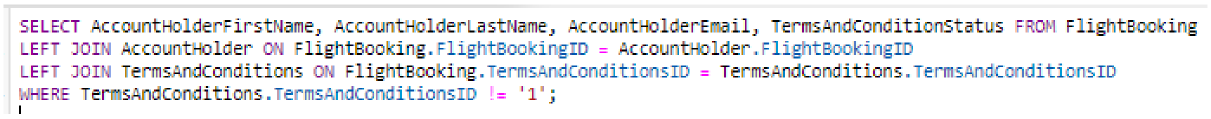
```
Select FlightBookingReference, PaymentMethodID, LineItemDescription AS PURCHASE, LineItemCost AS PRICE FROM FlightBooking
INNER JOIN LineItemTable ON FlightBooking.FlightBookingID = LineItemTable.FlightBookingID WHERE FlightBooking.FlightBookingID = '16'
```



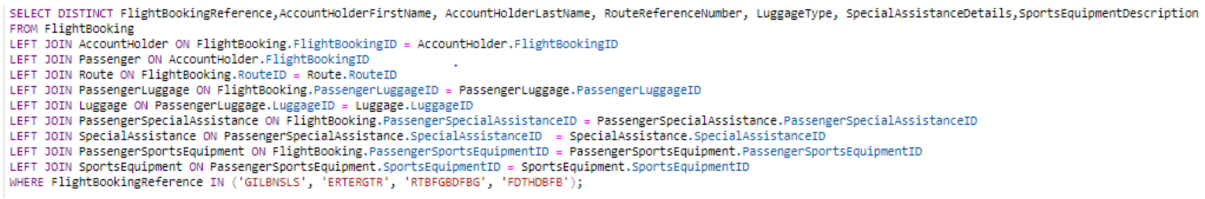
```sql
SELECT AircraftType.MakeAndModel, AircraftType.TotalSeats, COUNT(Aircraft.AircraftTypeID) AS SeatsBookedThusFar FROM FlightBooking
INNER JOIN Seating ON FlightBooking.SeatID = Seating.SeatID
INNER JOIN Aircraft ON Seating.AircraftID = Aircraft.AircraftID
INNER JOIN AircraftType ON Aircraft.AircraftTypeID = AircraftType.AircraftTypeID
GROUP BY AircraftType.MakeAndModel;
```



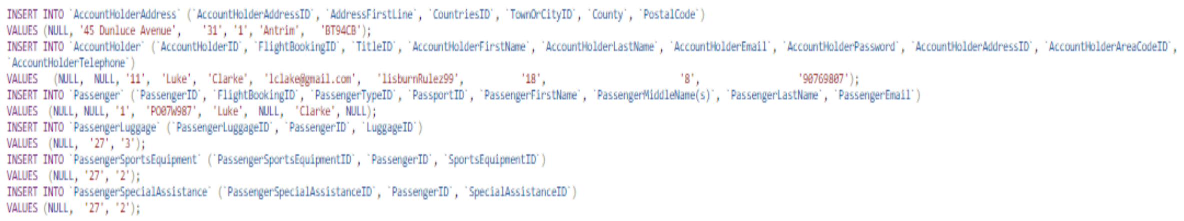
```sql
SELECT AircraftType.MakeAndModel, AircraftType.TotalSeats, COUNT(Aircraft.AircraftTypeID) AS SeatsBookedThusFar FROM FlightBooking
INNER JOIN Seating ON FlightBooking.SeatID = Seating.SeatID
INNER JOIN Aircraft ON Seating.AircraftID = Aircraft.AircraftID
INNER JOIN AircraftType ON Aircraft.AircraftTypeID = AircraftType.AircraftTypeID
GROUP BY AircraftType.MakeAndModel;
```



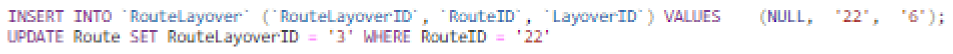
```sql
SELECT AccountHolderFirstName, AccountHolderLastName, AccountHolderEmail, TermsAndConditionStatus FROM FlightBooking
LEFT JOIN AccountHolder ON FlightBooking.FlightBookingID = AccountHolder.FlightBookingID
LEFT JOIN TermsAndConditions ON FlightBooking.TermsAndConditionsID = TermsAndConditions.TermsAndConditionsID
WHERE TermsAndConditions.TermsAndConditionsID != '1';
```



```sql
SELECT DISTINCT FlightBookingReference,AccountHolderFirstName, AccountHolderLastName, RouteReferenceNumber, LuggageType, SpecialAssistanceDetails,SportsEquipmentDescription
FROM FlightBooking
LEFT JOIN AccountHolder ON FlightBooking.FlightBookingID = AccountHolder.FlightBookingID
LEFT JOIN Passenger ON AccountHolder.FlightBookingID
LEFT JOIN Route ON FlightBooking.RouteID = Route.RouteID
LEFT JOIN PassengerLuggage ON FlightBooking.PassengerLuggageID = PassengerLuggage.PassengerLuggageID
LEFT JOIN Luggage ON PassengerLuggage.LuggageID = Luggage.LuggageID
LEFT JOIN PassengerSpecialAssistance ON FlightBooking.PassengerSpecialAssistanceID = PassengerSpecialAssistance.PassengerSpecialAssistanceID
LEFT JOIN SpecialAssistance ON PassengerSpecialAssistance.SpecialAssistanceID  = SpecialAssistance.SpecialAssistanceID
LEFT JOIN PassengerSportsEquipment ON FlightBooking.PassengerSportsEquipmentID = PassengerSportsEquipment.PassengerSportsEquipmentID
LEFT JOIN SportsEquipment ON PassengerSportsEquipment.SportsEquipmentID = SportsEquipment.SportsEquipmentID
WHERE FlightBookingReference IN ('GILBNSLS', 'ERTERGTR', 'RTBFGBDFBG', 'FDTHDBFB');
```



```sql
INSERT INTO `AccountHolderAddress` (`AccountHolderAddressID`, `AddressFirstLine`, `CountriesID`, `TownOrCityID`, `County`, `PostalCode`)
VALUES (NULL, '45 Dunluce Avenue',    '31', '1', 'Antrim',   'BT94CB');
INSERT INTO `AccountHolder` (`AccountHolderID`, `FlightBookingID`, `TitleID`, `AccountHolderFirstName`, `AccountHolderLastName`, `AccountHolderEmail`, `AccountHolderPassword`, `AccountHolderAddressID`, `AccountHolderAreaCodeID`,
`AccountHolderTelephone`)
VALUES   (NULL,  NULL, '11',  'Luke',  'Clarke',  'lclake@gmail.com',   'lisburnRulez99',         '18',                      '8',                '90769807');
INSERT INTO `Passenger` (`PassengerID`, `FlightBookingID`, `PassengerTypeID`, `PassportID`, `PassengerFirstName`, `PassengerMiddleName(s)`, `PassengerLastName`, `PassengerEmail`)
VALUES  (NULL, NULL, '1',  'PO07W987',  'Luke',  NULL,  'Clarke', NULL);
INSERT INTO `PassengerLuggage` (`PassengerLuggageID`, `PassengerID`, `LuggageID`)
VALUES  (NULL,  '27', '3');
INSERT INTO `PassengerSportsEquipment` (`PassengerSportsEquipmentID`, `PassengerID`, `SportsEquipmentID`)
VALUES  (NULL, '27', '2');
INSERT INTO `PassengerSpecialAssistance` (`PassengerSpecialAssistanceID`, `PassengerID`, `SpecialAssistanceID`)
VALUES (NULL,  '27', '2');
```



```sql
INSERT INTO `RouteLayover` (`RouteLayoverID`, `RouteID`, `LayoverID`) VALUES    (NULL,  '22',  '6');
UPDATE Route SET RouteLayoverID = '3' WHERE RouteID = '22'
```