

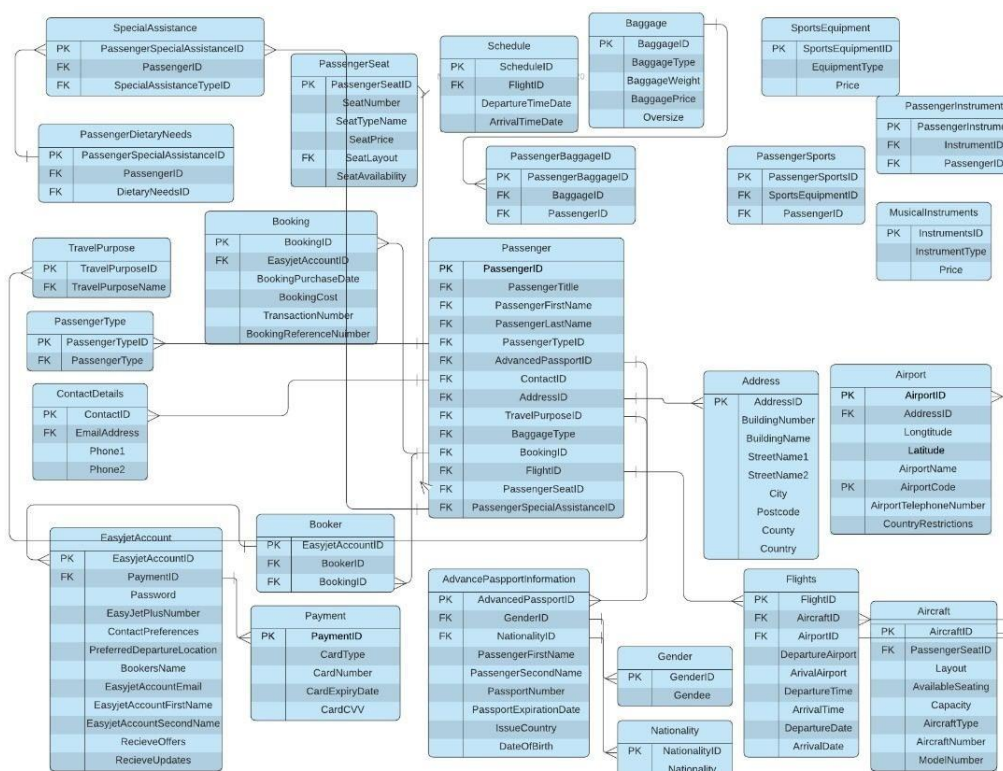
Aims and Scope

The overall aim of this project was to design and implement a relational database management system in phpMyAdmin formulated on selected aspects of a real commercial airline's booking system, EasyJet.com. The key functionality of the database is to emulate the account creation, point-of-sale, and flight booking management systems found at EasyJet.com. Primarily, the database should facilitate the accurate handling of data and the secure storage of sensitive information. Additionally, the relational design should make full use of normalisation techniques (1NF, 2NF, 3NF) and referential integrity constraints (PK, FK) to ensure atomicity and limit redundancy. In practice, the database facilitates the creation of an EasyJet account, allows the user to easily book uniquely identifiable seats on an aircraft travelling on a particular route, and populates the booking with one or more passengers and their relevant baggage, assistance requirements, etc. Certain aspects of the booking process such as car hire and accommodation were determined to be out of scope for the project due to their reliance on third-party data structures and/or APIs.

Overview of Database Design (ERD) & Design Assumptions

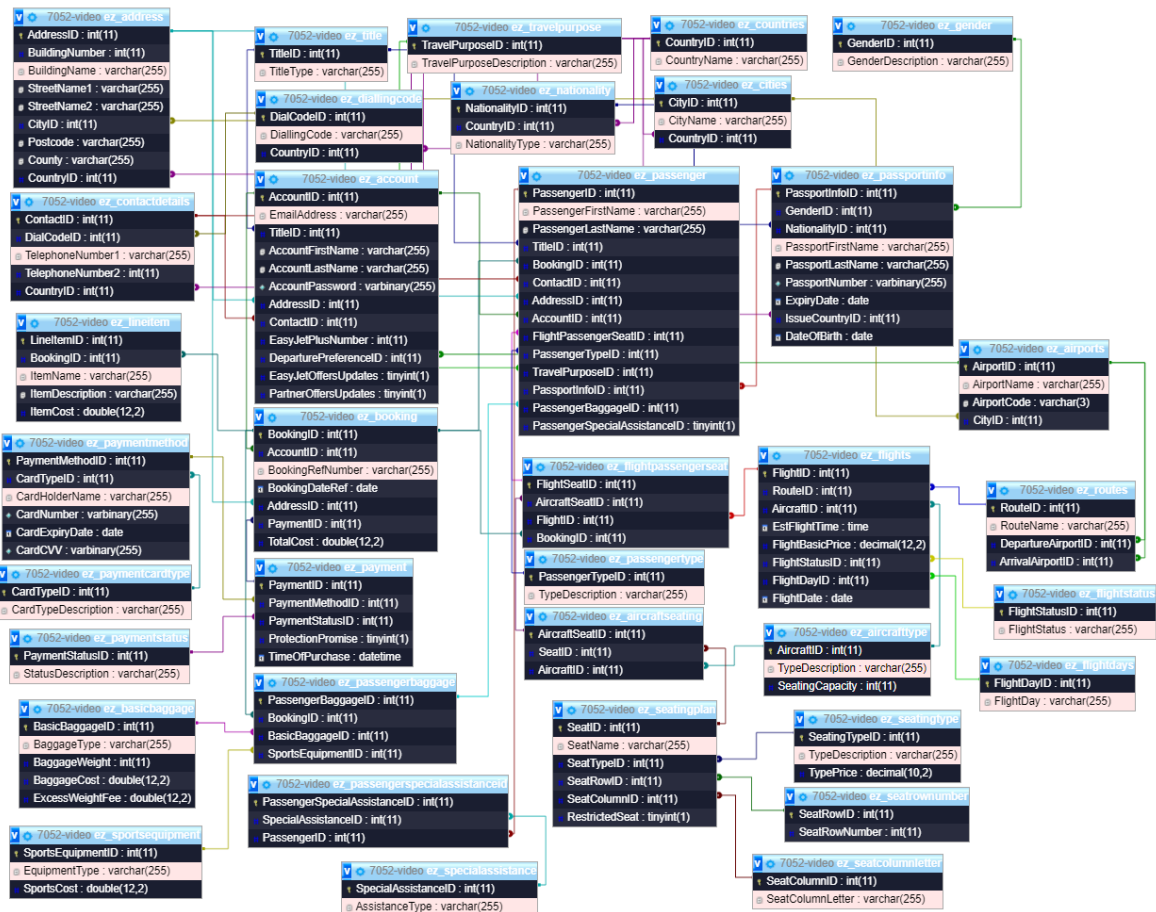
By thoroughly reviewing the flight booking system found at EasyJet.com the initial group design team was able to identify key entities, attributes and relationships that form the basis of EasyJet's flight booking system. Entities are linked to one another via relationships such as many-to-one, one-to-one, many-to-many and so forth. At this stage, the 'Booking' and 'EasyJetAccount' were identified as essential for holding unique user log-in data for repeat use. Additionally, early designs of junction tables such as 'PassengerBaggage' and 'PassengerSeat' provided a starting point for expressing many-to-many relationships within the database – for example, a passenger can have multiple baggage items/types and this can be uniquely expressed by querying the junction table with a 'WHERE' clause specifying the passenger's ID. The Figure 1 demonstrates the relationship types between tables:

Figure 1.



The final group ER diagram (Fig. 1) was used as the basis for developing a more comprehensive individual ER diagram (Fig. 2) for the purpose of independently designing and implementing a functional relational database. Additional junction tables such as 'ez_FlightPassengerSeat' have been included to facilitate a means of uniquely identifying every aircraft seat on every flight for a relevant passenger.

Figure 2.



For the purposes of this project, and from thorough review of EasyJet's flight booking system UI, it is assumed that no booking can be complete until an EasyJet user account is registered, and payment details are confirmed. With this in mind, referential integrity constraints have been implemented to prevent passengers being created without bookings, etc. By default, the database runs in an auto-commit mode. This means that unless statements occur inside transactions (flagged by the START TRANSACTION command) they are treated as atomic. The SQL ROLLBACK command is an effective method of emulating the input of user data without finalising a flight booking. Statements are also rolled back automatically if they contain logic or syntax errors – for this reason referential integrity has become an essential component of the functioning database. Furthermore, it is broadly assumed that in compliance with GDPR right-to-erasure, the relational design would also function to remove user data on request. As this project is the individual designer's first

database design, the approach to implement this has been a simple 'ON DELETE CASCADE' on relevant foreign key constraints.

Primary Keys

An essential component of the individual flight booking DBMS' relational design are primary keys. The primary key field forms the basis for uniquely identifying records in any table. To do this, the primary key must be completely unique to the row and must not be null. Throughout the individual database all primary keys are given the data type INT, specifically of length (11) within the scope of this project. The reason for this is so that each primary key can automatically increment and update its value when a new set of records is inserted into the table. To simplify things further while working with many tables, the postfix 'ID' is used with the 'tablename' in all relevant primary keys, for example, the primary key for the 'ez_Passenger' table is 'PassengerID'. The primary key in all cases is set to auto increment every time a new set of records is inserted. By ensuring every primary key is set to 'auto increment' the row is assigned an atomic value every time.

Foreign Keys

Arguably just as indispensable as primary keys, foreign key constraints are consistently and reliably used throughout this project's relational database to ensure both referential integrity and to also ensure that records are not entered in isolation. Many important tables such as passenger, booker, account receive repeated user input. To overcome the likelihood of poor data quality and inconsistency, select fields such as title have been normalised out into their own table. The 'ez_Titles' table is referenced by foreign key constraints in both the account and passenger tables, allowing users to select their title from a precomposed list of available inputs. This method of using foreign key constraints minimises the risk of user errors such as typos or trolling that would compromise data integrity.

Additionally, optional clauses in the foreign key declaration have been considered in the relational design of the flight booking DBMS. For example, the foreign key constraint that links passengers to their parent 'BookingID' is set with the referential clause ON DELETE CASCADE (Fig. 4). This referential action performs two functions; firstly, it ensures referential integrity between passengers and bookings (as a passenger cannot exist without a 'parent' booking) and secondly, it is assumed by the designer that the real-world flight booking system must adhere to general data protection regulation when handling user data.

Data Types

Throughout the database, the most frequently used data type/length is INT (11). This is due largely to the many referential integrity constraints present in the relational design that rely on a numeric value to work effectively. By using this data type in primary key and foreign key constraints the auto-increment attribute can function effectively and ensure atomicity. VARCHAR (255) is also used widely through the database to store information such as first names, last names, email addresses, telephone numbers and dialling codes. By using this data type, important user data can easily be output to screen as cleartext. Airport

codes are the only VARCHAR field with a variation in length, being (3) rather than the (255) used throughout the rest of the database. One noticeable limitation of this datatype decision is the reasonable likelihood of typos when querying the airports table or the possibility that airport codes may be extended beyond a 3-character length in the future.

DATE is featured in fields such as 'BookingDateRef' or 'FlightDate' in the 'ez_Booking' and 'ez_Flights' tables respectively. TIME is used in the fields such as 'EstFlightTime' in the ez_Flights table. The standardised formatting of these datatypes ensures that the database can be queried easily and effectively. A small range of fields with the 'TINYINT' (1) data type can be seen, it is assumed by the designer that these are likely used as Boolean flags for EasyJet's frontend team/analytics.

The datatype DOUBLE is used for storing monetary values such as the basic price of a flight or a hold-baggage fee. For the essential purpose of encrypting sensitive user data the VARBINARY (255) type/length was used. This data type facilitates the AES_ENCRYPT() method built-in to phpMyAdmin that allows the database to perform a simple encryption with the aim of securing user input data. Within the scope of this project, fields that EasyJet account password, card number and CVV as specified in the 'PaymentMethod' table.

Lastly, where appropriate variables have been used to store and pass reference data between tables. The decision was made to use variables where possible rather than set up derived attributes which would potentially cause problems with data integrity (such as total costs which may fluctuate with changes in price or the addition of new records). Variables such as 'LastVarID' are used with the LAST_INSERT_ID() function to return the value of the last auto-incremented key. When querying multiple tables this becomes highly valuable for ease of reference when passing data into fields that rely on foreign key relationships. In addition, variables such as '@TotalOnAircraft' and '@TotalBooked' are used to sum the remaining number of seats on a given flight, which is set to a '@TotalStillAvailable' variable that returns the numeric value. The use of variables to store reference data mitigates the need to use multi-level subqueries/joins when interrogating the database. Variables can also be used in conjunction with the relational design to make records more readable and/or accessible, such as when handling dialling code records in the 'ez_DiallingCode' and telephone numbers in the 'ez_ContactDetails' table – by using variables when handling these records the data can easily be concatenated into a string literal and returned as cleartext for the user.

Figure 3.

	PaymentMethodID	CardTypeID	CardHolderName	CardNumber	CardExpiryDate	CardCVV
■ Edit Copy Delete	1	2	MR JOHN F SMITH	0x1423c6f853ced0d8eb29d5b3dfe92410a059910eac45bc6a...	2025-01-01	0x0062b8c8f24a23a8121fdeb7307caf3
■ Edit Copy Delete	2	4	MRS SALLY P SMITH	0xce504d9908e85a2ae22329fd97cbb747f5a07ed6c90ee405...	2023-04-01	0x0fa03f8337ccbe3876bc66e75ce13258
■ Edit Copy Delete	3	3	MRS LAUREN E STEELE	0x5b960703210a175ab63aeb706e5af74beafcf5bc2935baef...	2024-08-01	0x5eeda3195a56ab2493cc7530b0258169
■ Edit Copy Delete	14	5	MR CHRIS P LAWRENCE	0x56e65fc862897200300671122c5f2de52eaf35fd2f064f30...	2025-10-13	0xc183462a6df3865e200c3b5de5ff01a
■ Edit Copy Delete	15	4	MRS DIANE H LAURENT	0xec7f040cbd180a92afe1a481fe6ab0ec78168e2102f948ec...	2024-08-01	0xb27675ca5443c4d88e6d371949f14b54
■ Edit Copy Delete	18	4	MR KEVIN 'SILENT BOB' SMITH	0xec7f040cbd180a92afe1a481fe6ab0ec78168e2102f948ec...	2024-08-01	0xb27675ca5443c4d88e6d371949f14b54

Figure 4.

Drop

FK_PassengerBooking

ON DELETE

CASCADE

ON UPDATE

RESTRICT

Discussion and Justification of Important Design Decisions

Infant Seating

While reviewing the EasyJet booking system it became apparent that bookers cannot purchase exclusive seats for children under the age of 2. In place of this, EasyJet allows passengers with children that fall into the 'Infant (<2)' category to let their child sit in their lap. With respect to this, the 'PassengerType' table (Fig. 5) was created to uniquely identify the age group that any given passenger falls into. The table reflects EasyJet's real-world age categories of 'Adult', 'Child' and 'Infant' and allows for future age categories to be created with an atomic value using an auto-incremented primary key. The 'FlightPassengerSeat' junction table (Fig. 4) allows the database to uniquely identify every passengers seat on every flight and its corresponding aircraft. In terms of infant seating, the atomic value returned from this table as 'FlightPassengerSeatID' can be used to link an adult and child passenger with the same BookingID to a single seat. However, to ensure the data integrity of the number of available seats on any given flight, seats on relevant flights should be totalled as well as the number of distinct 'FlightPassengerSeatID's linked to a given BookingID.

Line Item Table

The introduction of a line item table to the project database serves as an invoice for customers upon completion of their booking. Within the line item table is an 'ItemName', 'ItemDescription' and 'ItemCost' field linked to a unique 'BookingID' that is used in conjunction with the 'LineItemID' primary key to form a composite key. The table is used to mitigate data integrity failures when item costs change suddenly. If an item is linked directly to the booking via a foreign key constraint, the total price of a booking agreed by the customer at point-of-sale is liable to change. Through the relational design of the database, summing the total cost of a booking can be efficiently and reliably done using a SELECT query such as App. 4. However, booking line-items and their costs must be manually inserted into the line item table. This decision was made to mitigate the likelihood of key-dependent records causing booking costs to change unexpectedly when flight or seat costs change in the future. A limitation of this is that line item records must be repetitively entered, increasing the likelihood of poor data quality through typos, etc. A possible solution to this that would be the integration of the flight booking DBMS with a third-party invoicing software as outlined in the potential improvements section below.

Multiple Baggage Items

At present, EasyJet's booking system allows only one cabin-bag included with the cost of the flight ticket. Additional cabin-bags and hold luggage incur an extra cost as reflected in the 'BasicBaggageID' and 'SportsEquipmentID' tables. As each passenger in a booking may potentially have more than one item of baggage with them per flight, it was important to reflect this in the relational design. With this in mind, the passenger table has been designed to hold a foreign key for the atomic value passed from a junction table titled 'PassengerBaggageID' – which similarly holds foreign key values for 'BookingID', 'BasicBaggageID' and 'SportsEquipmentID' respectively. In this way, a booker can purchase

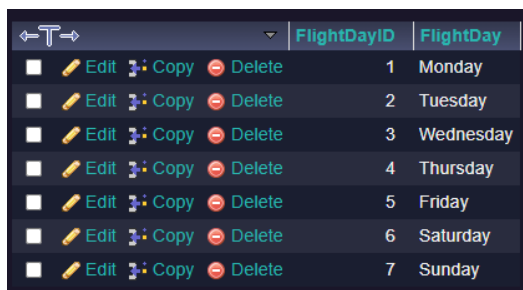
numerous additional baggage types for one or more passengers on each flight.

Encryption of Sensitive Information

In contrast to the original group design, the final individual database makes important consideration of encryption for sensitive user information such as passwords, credit card details and passport numbers. The database makes use of phpMyAdmin's built-in AES encryption method that is used to encrypt data held within the database and offer rudimentary data protection for bookers and passengers. A secure password facility is essential for allowing EasyJet account holders access to their account and latest booking information. Similarly, bookers' payment information such as their card number and card CVV is encrypted to minimise the risk of user data becoming compromised. However, it's important to note that with a larger, commercial database management system it is highly likely that a third-party encryption software would be used in-line with PCI DSS compliance.

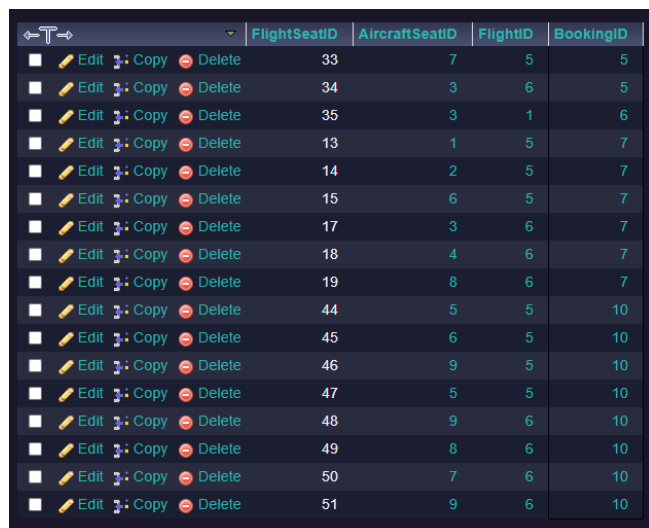
Flight Days Table

Unlike the initial group ERD, the final database design incorporates a flight days table (Fig. 3). This mirrors the real-world functionality of EasyJet.com's flight booking system and offers bookers even more control over the planning of their flights. This relational design choice also means that the flights table can be queried using a WHERE clause to return a list of flights operating on a given day. A 'FlightDaysID' acts as the designated primary key and is referenced in the flights table via foreign key constraint.



	FlightDayID	FlightDay
<input type="checkbox"/> Edit Copy Delete	1	Monday
<input type="checkbox"/> Edit Copy Delete	2	Tuesday
<input type="checkbox"/> Edit Copy Delete	3	Wednesday
<input type="checkbox"/> Edit Copy Delete	4	Thursday
<input type="checkbox"/> Edit Copy Delete	5	Friday
<input type="checkbox"/> Edit Copy Delete	6	Saturday
<input type="checkbox"/> Edit Copy Delete	7	Sunday

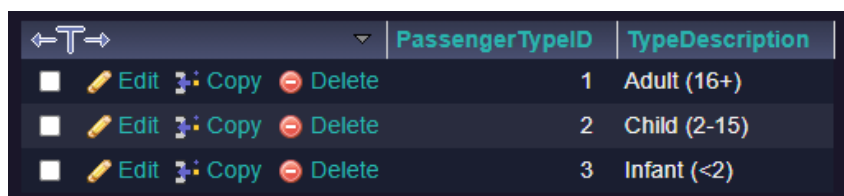
Figure 5.



	FlightSeatID	AircraftSeatID	FlightID	BookingID
<input type="checkbox"/> Edit Copy Delete	33	7	5	5
<input type="checkbox"/> Edit Copy Delete	34	3	6	5
<input type="checkbox"/> Edit Copy Delete	35	3	1	6
<input type="checkbox"/> Edit Copy Delete	13	1	5	7
<input type="checkbox"/> Edit Copy Delete	14	2	5	7
<input type="checkbox"/> Edit Copy Delete	15	6	5	7
<input type="checkbox"/> Edit Copy Delete	17	3	6	7
<input type="checkbox"/> Edit Copy Delete	18	4	6	7
<input type="checkbox"/> Edit Copy Delete	19	8	6	7
<input type="checkbox"/> Edit Copy Delete	44	5	5	10
<input type="checkbox"/> Edit Copy Delete	45	6	5	10
<input type="checkbox"/> Edit Copy Delete	46	9	5	10
<input type="checkbox"/> Edit Copy Delete	47	5	5	10
<input type="checkbox"/> Edit Copy Delete	48	9	6	10
<input type="checkbox"/> Edit Copy Delete	49	8	6	10
<input type="checkbox"/> Edit Copy Delete	50	7	6	10
<input type="checkbox"/> Edit Copy Delete	51	9	6	10

Figure 6.

Figure 7.



	PassengerTypeID	TypeDescription
<input type="checkbox"/> Edit Copy Delete	1	Adult (16+)
<input type="checkbox"/> Edit Copy Delete	2	Child (2-15)
<input type="checkbox"/> Edit Copy Delete	3	Infant (<2)

Discussion and Justification of Important Normalisation Decisions

Overview of Normalisation Decisions

To minimise redundancy and preserve data integrity the relational design aimed to achieve normalisation by adhering to first, second and third normal forms. Specifically, key normalisation decisions were made when considering the relational design of the account, passenger and 'FlightPassengerSeat' entities. Overall, the relational design aimed to avoid repetition of data as well as the presence of non-keyed dependencies between attributes.

Normalisation Decisions in ez_Account

The account table receives a large amount of user-inputted data. Without normalisation, fields concerning titles, telephone numbers, dialling codes and contact details including address, city and country would likely be entered by users inconsistently. To minimise the risk of errors in user data input, and to remove the need for repeated data input (in violation of 1NF), the fields outlined above were normalised out into their own tables and linked via foreign key constraints. Additionally, the account table illustrates the achievement of second normal form in the project DBMS' relational design. All non-keyed attributes – such as the user's unique login details, 'EmailAddress' and 'Password' – are dependent on the primary key. This allows the data to be interrogated simply and efficiently, while retaining overall atomicity.

Normalisation Decisions in ez_Passenger

Similarly to the account table, the passenger table is liable to receive a large amount of data from the relevant booker. To further reduce the risk of errors, fields such as title, passenger type (age-group) and travel purpose have been normalised out into their own unique helper tables and linked to the passenger table via foreign key constraints. These normalisation decisions when considering the relational design of the database ensure that the first, second and third normal forms are adhered too effectively and reduce the risk of redundant data as well as repetitive input while retaining a uniquely-identifiable 'ID' atomic value for each passenger.

Normalisation Decisions across Seating Entities

Without effective normalisation when considering the relational design of the database, the seating tables in particular were liable to get completely out of hand. For the individual DBMS, it was decided that the best approach was to set up a master 'FlightPassengerSeat' entity that provides each passenger with an atomic value referencing their unique seat on every aircraft, as well as the relevant flight and their booking number. By using junction tables, the booker can select an exact seat on any aircraft that flies their chosen route. In this way, all attributes within each seating table are dependent on the primary key; multiple seating row numbers and column letters across the same aircraft will always receive a unique identification number, eliminating redundant data and thus retaining atomicity.

Conclusion and Discussion of Potential Improvements

EasyJet offers a wide range of international flights to customers – with this in mind, rather than the estimated flight-time field that is currently used within the database, a time-zone entity or attribute could potentially be implemented and used to calculate distinct arrival and departure times for all flights. Moreover, with regards to the summing of booking line-item costs, additional attributes such as tax and currency types could potentially be implemented to better-allow the database to integrate with third-party invoicing software. At present, these details require manual entry into the line item table. Although this mitigates the likelihood of a key-dependency causing booking costs to change unexpectedly, it also means that data must be repetitively and potentially-redundantly entered into the line item table.

Attributes such as card payment status have been implemented with the aim of providing a basis for Boolean flags that the front-end team can utilise when constructing the end-user web API. At present, fields such as card payment status and flight-status are not being fully utilised by the database. With more development time (and potentially a greater working-knowledge of PHP) these fields could be used more effectively to automate web APIs, potentially via SQL triggers. Comparatively, fields such as 'EasyJetPlusNumber' and Boolean fields such as 'EasyJetOffersUpdates' and 'PartnerOffersUpdates' are arguably not being used to their full effectiveness within the relational design. In EasyJet's real-world flight booking system, EasyJet plus membership means that a booker can select any seat for a passenger as well as bring additional cabin-baggage on board without incurring any additional cost. A potential better use of this field would be to normalise it out into a 'EasyJetPlusMembership' table that allows a booker to be uniquely identified as having a distinct membership as well as membership type. The specific membership type could then be referenced for each passenger by linking a unique 'EasyJetMembership' + 'BookingID' composite key via foreign key constraint to every distinct passenger on a booking. Although these fields are non-key dependent and adhere to the principles of good normalisation, they are not used functionally in the database in the same way that EasyJet's real-world flight booking system likely uses them. Similar to the potential uses of 'status' attributes as outlined above, these fields could potentially be used to automate web APIs and provide a solid basis for user analytics as part of EasyJet's overall business strategy.

In regard to referential integrity constraints, certain foreign keys in the relational design are set with the optional clause ON DELETE CASCADE to ensure both referential integrity between bookings and passengers, and also to adhere to general data protection regulation. However, with a real-world commercial flight booking system it is likely that hundreds of thousands of booking records will be inserted and deleted over time. The use of the CASCADE referential action can therefore be seen to increase overall memory usage and potentially be detrimental to the performance of the database at large-scale. A possible solution to this is to use an automated delete via a trigger rather than a cascading deletion clause.

Appendix of SQL Queries

#Query for 'Kevin Smith' to create his account:

```
1 START TRANSACTION
2 ;
3 INSERT INTO ez_address(
4     AddressID,
5     BuildingNumber,
6     BuildingName,
7     StreetName1,
8     StreetName2,
9     CityID,
10    Postcode,
11    County,
12    CountryID
13 )
14 VALUES(
15     NULL,
16     20,
17     NULL,
18     'Big Country House',
19     NULL,
20     22,
21     '61128',
22     NULL,
23     2
24 );
25 SET
26     @VarLastID = LAST_INSERT_ID();
27 INSERT INTO ez_contactdetails(
28     ContactID,
29     DialCodeID,
30     TelephoneNumber1,
31     TelephoneNumber2,
32     CountryID
33 )
34 VALUES(
35     @VarLastID,
```

```
VALUES(
    @VarLastID,
    4,
    '6443 59920',
    NULL,
    2
);
INSERT INTO ez_account(
    AccountID,
    EmailAddress,
    TitleID,
    AccountFirstName,
    AccountLastName,
    AccountPassword,
    AddressID,
    ContactID,
    EasyJetPlusNumber,
    DeparturePreferenceID,
    EasyJetOffersUpdates,
    PartnerOffersUpdates
)
VALUES(
    NULL,
    'kevinsmitht@yahoo.fr',
    1,
    'Kevin',
    'Smith',
    AES_ENCRYPT(
        'mysupersecretpassword99',
        'mysecretkey'
    ),
    @VarLastID,
    @VarLastID,
    '5668001',
    27,
    1,
    1
);
COMMIT
;
```

#Query to select flight departure airport, arrival airport and flight dates/times:

```
1 SELECT
2     *
3 FROM
4     ez_airports
5 INNER JOIN ez_cities ON ez_cities.CityID = ez_airports.CityID
6 INNER JOIN ez_routes ON ez_airports.AirportID = ez_routes.DepartureAirportID
7 WHERE
8     ez_cities.CityName = 'Paris';
9 SELECT
10    *
11 FROM
12    ez_airports
13 INNER JOIN ez_cities ON ez_cities.CityID = ez_airports.CityID
14 INNER JOIN ez_routes ON ez_airports.AirportID = ez_routes.ArrivalAirportID
15 WHERE
16    ez_cities.CityName = 'London';
17 SELECT
18    *
19 FROM
20    ez_flights
21 WHERE
22    RouteID = 8 OR RouteID = 7;
23
```

#Query to view all seats on a distinct flight, and select the seat '1B':

```
1 SELECT
2   *
3 FROM
4   ez_flights
5 INNER JOIN ez_aircraftseating ON ez_flights.AircraftID = ez_aircraftseating.AircraftID
6 INNER JOIN ez_seatingplan ON ez_aircraftseating.SeatID = ez_seatingplan.SeatID
7 WHERE
8   ez_flights.FlightID = 5 OR ez_flights.FlightID = 6;
9 SELECT
10  *
11 FROM
12   ez_flights
13 INNER JOIN ez_aircraftseating ON ez_flights.AircraftID = ez_aircraftseating.AircraftID
14 INNER JOIN ez_seatingplan ON ez_aircraftseating.SeatID = ez_seatingplan.SeatID
15 INNER JOIN ez_seatcolumnletter ON ez_seatingplan.SeatColumnID = ez_seatcolumnletter.SeatColumnID
16 WHERE
17   (
18     ez_flights.FlightID = 5 OR ez_flights.FlightID = 6
19   ) AND(ez_seatingplan.SeatRowID = 1) AND(SeatColumnLetter = 'B');
```

#Query to create a Kevin's booking, bookings aren't finalised without payment info so we
#work from the attributes 'inside-out', restricted by relevant FK constraints that mitigate
#redundant/void bookings:

```
1 START TRANSACTION
2   ;
3 INSERT INTO ez_paymentmethod(
4   PaymentMethodID,
5   CardTypeID,
6   CardHolderName,
7   CardNumber,
8   CardExpiryDate,
9   CardCW
10 )
11 VALUES(
12   NULL,
13   4,
14   'MR KEVIN 'SILENT BOB' SMITH',
15   AES_ENCRYPT(
16     '1334 6783 0200 7912',
17     'mysecretkey'
18   ),
19   '2024-08-01',
20   AES_ENCRYPT('313', 'mysecretkey')
21 );
22 SET
23   @VarLastID = LAST_INSERT_ID();
24 INSERT INTO ez_payment(
25   PaymentID,
26   PaymentMethodID,
27   PaymentStatusID,
28   ProtectionPromise,
29   TimeOfPurchase
30 )
31 VALUES(
```

```
31 VALUES(
32   NULL,
33   @VarLastID,
34   2,
35   1,
36   '2020-12-03 11:54:43'
37 );
38 SET
39   @VarLastID = LAST_INSERT_ID();
40 INSERT INTO ez_booking(
41   BookingID,
42   BookingRefNumber,
43   BookingDateRef,
44   AddressID,
45   AccountID,
46   PaymentID,
47   TotalCost
48 )
49 VALUES(
50   NULL,
51   'EZ999392',
52   '2020-12-03',
53   8,
54   8,
55   @VarLastID,
56   0
57 );
58 COMMIT
59   ;
```

#Query showing Kevin booking distinct seats for himself and three family members, his
#infant child shares his 'AircraftSeatID' (1) but is assigned their own unique 'FlightSeatID'
#that ensures atomicity:

```
1 INSERT INTO `ez_flightpassengerseat`(  
2     `FlightSeatID`,  
3     `AircraftSeatID`,  
4     `FlightID`,  
5     `BookingID`  
6 )  
7 VALUES(NULL, '1', '5', '7'),(NULL, '2', '5', '7'),(NULL, '3', '5', '7'),(NULL, '1', '5', '7');  
8 INSERT INTO `ez_flightpassengerseat`(  
9     `FlightSeatID`,  
10    `AircraftSeatID`,  
11    `FlightID`,  
12    `BookingID`  
13 )  
14 VALUES(NULL, '1', '6', '7'),(NULL, '2', '6', '7'),(NULL, '3', '6', '7'),(NULL, '1', '6', '7');
```

#Query to confirm selected seats:

```
1 SELECT  
2     *  
3 FROM  
4     ez_flightpassengerseat  
5 WHERE  
6     BookingID = 10;
```

#Query to insert baggage records and confirm baggage:

```
1 INSERT INTO `ez_passengerbaggage`(  
2     `PassengerBaggageID`,  
3     `BookingID`,  
4     `BasicBaggageID`,  
5     `PassengerSportsID`  
6 )  
7 VALUES(NULL, '10', '4', '10');  
8 SELECT  
9     *  
10 FROM  
11     ez_passengerbaggage  
12 WHERE  
13     BookingID = 10;
```

#Query to confirm log-in details:

```
1 SELECT
2     *
3 FROM
4     ez_account
5 WHERE
6     ez_account.EmailAddress = 'kevinsmith@yahoo.fr';
7 SELECT
8     AccountID,
9     AES_DECRYPT(AccountPassword, 'mysecretkey')
10 FROM
11     ez_account
12 WHERE
13     AccountID = 10;
```

#Query to insert passenger information, special assistance requirements, passport info for #each family member (4). Please note this query is unformatted due to its large size for #ease of reference:

```
1 START TRANSACTION;
2 INSERT INTO ez_passportinfo (PassportInfoID, GenderID, NationalityID, PassportFirstName, PassportLastName, PassportNumber, ExpiryDate, IssueCountryID, DateOfBirth)
3 VALUES (NULL, 1, 'Kevin', 'Smith', AES_ENCRYPT('PS5022011', 'mysecretkey'), 2022-06-11, 2, 1979-09-01);
4 SET @VarLastID = LAST_INSERT_ID();
5 INSERT INTO ez_passenger (PassengerID, PassengerFirstName, PassengerLastName, TitleID, BookingID, ContactID, AddressID, AccountID, FlightPassengerSeatID, PassengerTypeID, TravelPurposeID, PassportInfoID, PassengerBaggageID,
6 PassengerSpecialAssistanceID)
7 VALUES (NULL, 'Kevin', 'Smith', 1, 10, 11, 11, 10, 44, 1, 2, @VarLastID, 7, NULL);
8 INSERT INTO ez_passportinfo (PassportInfoID, GenderID, NationalityID, PassportFirstName, PassportLastName, PassportNumber, ExpiryDate, IssueCountryID, DateOfBirth)
9 VALUES (NULL, 1, 2, 'Chloe', 'Smith', AES_ENCRYPT('PS0870399', 'mysecretkey'), 2024-09-05, 2, 1975-05-09);
10 SET @VarLastID = LAST_INSERT_ID();
11 INSERT INTO ez_passenger (PassengerID, PassengerFirstName, PassengerLastName, TitleID, BookingID, ContactID, AddressID, AccountID, FlightPassengerSeatID, PassengerTypeID, TravelPurposeID, PassportInfoID, PassengerBaggageID,
12 PassengerSpecialAssistanceID)
13 VALUES (NULL, 'Chloe', 'Smith', 2, 10, 11, 11, 10, 45, 1, 2, @VarLastID, 8, 0);
14 INSERT INTO ez_passportinfo (PassportInfoID, GenderID, NationalityID, PassportFirstName, PassportLastName, PassportNumber, ExpiryDate, IssueCountryID, DateOfBirth)
15 VALUES (NULL, 2, 1, 'Charlotte', 'Smith', AES_ENCRYPT('PS5601029', 'mysecretkey'), 2027-02-01, 2, 2000-11-13);
16 SET @VarLastID = LAST_INSERT_ID();
17 INSERT INTO ez_passenger (PassengerID, PassengerFirstName, PassengerLastName, TitleID, BookingID, ContactID, AddressID, AccountID, FlightPassengerSeatID, PassengerTypeID, TravelPurposeID, PassportInfoID, PassengerBaggageID,
18 PassengerSpecialAssistanceID)
19 VALUES (NULL, 'Charlotte', 'Smith', 2, 10, 11, 11, 10, 46, 1, 2, @VarLastID, 9, 0);
20 INSERT INTO ez_passportinfo (PassportInfoID, GenderID, NationalityID, PassportFirstName, PassportLastName, PassportNumber, ExpiryDate, IssueCountryID, DateOfBirth)
21 VALUES (NULL, 2, 1, 'Baby', 'Smith', AES_ENCRYPT('PS7051119', 'mysecretkey'), 2031-01-01, 2, 2019-10-25);
22 SET @VarLastID = LAST_INSERT_ID();
23 INSERT INTO ez_passenger (PassengerID, PassengerFirstName, PassengerLastName, TitleID, BookingID, ContactID, AddressID, AccountID, FlightPassengerSeatID, PassengerTypeID, TravelPurposeID, PassportInfoID, PassengerBaggageID,
24 PassengerSpecialAssistanceID)
25 VALUES (NULL, 'Baby', 'Smith', 2, 10, 11, 11, 10, 44, 3, 2, @VarLastID, 10, 0);
26 COMMIT;
```

#Query to confirm Kevin's payment information before finalising his purchase:

```
1 SELECT
2     *
3 FROM
4     ez_booking
5 INNER JOIN ez_payment ON ez_booking.PaymentID = ez_payment.PaymentID
6 INNER JOIN ez_paymentmethod ON ez_payment.PaymentMethodID = ez_paymentmethod.PaymentMethodID
7 WHERE
8     BookingID = 10;
9 SELECT
10    *,
11    AES_DECRYPT(CardNumber, 'mysecretkey'),
12    AES_DECRYPT(CardCW, 'mysecretkey')
13 FROM
14     ez_payment
15 INNER JOIN ez_paymentmethod ON ez_payment.PaymentMethodID = ez_paymentmethod.PaymentMethodID
16 WHERE
17     PaymentID = 10;
```

#Query to confirm the total cost of Kevin's booking:

```
1 SELECT DISTINCT
2     ez_flightpassengerseat.BookingID,
3     FlightSeatID,
4     TypePrice,
5     FlightBasicPrice,
6     (
7     SELECT DISTINCT
8         (
9             SUM(FlightBasicPrice + TypePrice)
10        )
11    FROM
12        ez_flightpassengerseat
13    INNER JOIN ez_aircraftseating ON ez_flightpassengerseat.AircraftSeatID = ez_aircraftseating.AircraftSeatID
14    INNER JOIN ez_seatingplan ON ez_aircraftseating.SeatID = ez_seatingplan.SeatID
15    INNER JOIN ez_seatingtype ON ez_seatingplan.SeatTypeID = ez_seatingtype.SeatingTypeID
16    JOIN ez_flights ON ez_flightpassengerseat.FlightID = ez_flights.FlightID
17    WHERE
18        ez_flightpassengerseat.BookingID = 10
19 ) AS TotalFlightSeatCost
20 FROM
21     ez_flightpassengerseat
22 INNER JOIN ez_aircraftseating ON ez_flightpassengerseat.AircraftSeatID = ez_aircraftseating.AircraftSeatID
23 INNER JOIN ez_seatingplan ON ez_aircraftseating.SeatID = ez_seatingplan.SeatID
24 INNER JOIN ez_seatingtype ON ez_seatingplan.SeatTypeID = ez_seatingtype.SeatingTypeID
25 JOIN ez_flights ON ez_flightpassengerseat.FlightID = ez_flights.FlightID
26 WHERE
27     ez_flightpassengerseat.BookingID = 10;
28 SELECT DISTINCT
29     PassengerBaggageID,
30     BookingID,
31     BaggageCost,
32     SportsCost,
33     (
34     SELECT
35         (SUM(BaggageCost + SportsCost))
36     FROM
37         ez_passengerbaggage
38     INNER JOIN ez_basicbaggage ON ez_passengerbaggage.BasicBaggageID = ez_basicbaggage.BasicBaggageID
39     INNER JOIN ez_sportsequipment ON ez_passengerbaggage.SportsEquipmentID = ez_sportsequipment.SportsEquipmentID
40     WHERE
41         ez_passengerbaggage.BookingID = 10
42 ) AS TotalCost
43 FROM
44     ez_passengerbaggage
45 INNER JOIN ez_basicbaggage ON ez_passengerbaggage.BasicBaggageID = ez_basicbaggage.BasicBaggageID
46 INNER JOIN ez_sportsequipment ON ez_passengerbaggage.SportsEquipmentID = ez_sportsequipment.SportsEquipmentID
47 WHERE
48     ez_passengerbaggage.BookingID = 10;
```

#Query to return the number of seats still available on a given flight:

```
1 SET
2     @TotalBooked =(
3     SELECT
4         COUNT(FlightPassengerSeatID)
5     FROM
6         ez_passenger
7     JOIN ez_flightpassengerseat ON ez_passenger.FlightPassengerSeatID = ez_flightpassengerseat.FlightSeatID
8     WHERE
9         FlightID = 5
10 );
11 SET
12     @TotalOnAircraft =(
13     SELECT
14         COUNT(SeatID)
15     FROM
16         ez_aircraftseating
17     WHERE
18         AircraftID = 2
19 );
20 SET
21     @TotalStillAvailable = @TotalOnAircraft - @TotalBooked;
22 SELECT
23     @TotalStillAvailable;
```