**Assessment info**

* No database file needed
* Marking based entirely on report and video

**Written report info**

* Pdf form
* Submitted through canvas
* No cover page or table of contents
* Sections
  + Overview of db design – designer view/ draw io
    - Start and final – allows to explain key design decisions
    - More diagrams can go in appendix
    - Talk through noramlisations and evolution of design
  + List of design assumptions that were made
    - What is in and out of scope?
    - Thisis about package holidays
    - Aircraft baggage etc/ meals out of scope
    - Package holiday: hotel room for x days and flights
    - Assume room and flight are tied, if you cang et one then you can get the other
    - Direct flights only
    - Enumerate the most key assumptions
  + Explanation of some important primary and fk constraints
    - Surrogates?
    - Composite keys?
    - Foreign keys to prouce 1:1, 1:N, N:M etc
    - Types and why they were chosen
    - Not interested in a dictionary of everything
  + Some Important design desicisons **(paired with below – same section)**
    - Micro ones aren’t really necessary
    - Key decisions around prices, availability, pricing etc
  + Some important normalisation decisions **(paired with above)**
    - Good pragmatic decisions
    - Not so interested in specific normal forms, but the pragmatic WHY of decisions
  + Improvements
    - Design improvements (own section maybe)
    - We are only building selected aspects of the system, so there will be reasons why some tables are empty, why some things are not implemented
  + SQL Queries in appendix
    - Demo of important aspects
    - Record all ones from video in report

Text

Description automatically generated

**Notes:**

Display selectable durations

Show pricing differences for different airport/ flight dates

Balanced owed query

Add intermediate ERD to appendix

--- FEEDBACK LECTURE NOTES

Validate data type lengths above data layer is probably smarter

**Database Design Report**

**by Peter Marley (student number 13404067)**

**Overview**

This project is an attempt to reverse engineer the database layer of certain sections of the professional commercial booking system of the [Jet2Holidays](https://www.jet2holidays.com/) website, using the relational model. The Jet2Holiday website is vast, and was developed iteratively over many years and contains much functionality that would not be possible to recreate within the timescale allowed for this project. Thus, the scope of this project will be limited to the data storage requirements necessary to store direct and supporting data for 3 main areas of interest – namely passengers, flights and hotels – in such a way as to allow a developer to subsequently implement functionality visible on the website.

Initially we operated under the following design assumptions:

1. If a room is available to choose, a corresponding outbound and return flight is assumed to be available.
2. Anything involving financing or setting up recurring monthly payment of the cost of a booking was considered out of scope, but multiple part payments or a single complete payment were to be considered in scope.
3. Any data required to implement fine grained flight booking were out of scope; namely specific seating on a flight, extra baggage/ hold luggage pricing and selection, extra leg room, suitability for infant passengers, sports equipment, and flight meals.
4. Flights are direct to destination. Multiple leg flights are considered out of scope.
5. Transfers from airports to hotels were considered out of scope.
6. Car Hire was considered out of scope.
7. Any form of insurance was out of scope.
8. Scope is limited to the Jet2Holidays section of the website, excluding others – such as Agent Finder, Jet2Villas and Jet2CityBreaks, although the latter two have considerable overlap in functionality with Jet2Holidays.
9. We will not be able to implement full PCI Security Standard Council policies and procedures regarding storage of payment card information. These are detailed and prescriptive, instead we settled for implementing basic 2-way encryption of card details using AES encryption using a known Secret, adhering to the most basic principles.

A high level overview of the functionality of interest is as follows:

1. Search for a holiday package using departure airport, holiday destination, holiday duration, and number of passengers and their ages. A single passenger is nominated as the booking contact and that passenger requires extra information to be stored, namely address, email and telephone contact information.
2. Make a booking for a holiday based on selected departure airport, hotel, room type, board basis, holiday duration, passenger number, and to calculate the cost and availability of said booking. Flight and accommodation cost is dependent on date and time for the former, and date alone for the latter.
3. Create queries to demonstrate the various considerations above two use cases.

Initial entity discovery was undertaken individually. We analysed the Jet2Holidays website to try to deduce the entities and attributes underpinning data visible on in-scope sections of the website. We combined our entities into one list, and as a group pinpointed which of these bits of information were entities, and which were attributes. Furthermore, we attempted some initial pragmatic normalisation of attributes to remove repetition of data, expand composite attributes into several attributes, and decide which were more appropriate as derived/ computed attributes. Ultimately this was the beginning of the process to bring all the entities up to the standard of the 3rd normal form at a minimum.

Our initial Entity-Relationship Diagram (figure 1) describes our entity-relationship model, however it is naïve in several aspects within the specified scope of the project:

1. Naming conventions of entities and their attributes are inconsistent.
2. It does not attempt to address either time/date dependent pricing of flights and hotels, nor an availability model for our hotel rooms.
3. It does not consider how to store payment card information.
4. It does not consider how to tabulate payments made nor balanced owed on a booking.
5. Text

   Description automatically generated with medium confidence It does not adequately describe the destination to region to resort to accommodation relationship visible on the website.
6. It does not consider the cost of board basis (examples include All inclusive and Bed & Breakfast) for a hotel room booking.
7. It does not consider the ability for a user to create an account and save provisional booking information to a short list.
8. Some entities are not normalised as far as the 3rd normal form, for example, star\_rating attribute of the review entitys.
9. Some attributes are better represented as derived attributes to maintain data integrity, for example the estimated\_time attribute of the flight entity, which allows the unfavourable introduction of insertion, update and deletion dependencies.
10. The diagram does not consider data types for each attribute.

Diagram

Description automatically generated

Figure Initial Entity Relationship Diagram

Diagram

Description automatically generated

Figure Final Entity Relationship Diagram

**Important Normalisations and Evolution of Design**

We attempted to address the naivety of the original design and to improve upon its shortcomings (see appendix for intermediate diagrams).

**Important Normalisation Decisions**

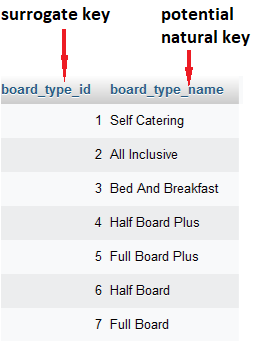
Booking contact and contact info

Geographic location information chain (location\_name, country, destination, region, resort, hotel, gps)

Bits of info shared between hotels, that have hotel specific information at the hotel level (hotel\_board\_type + board\_type// room\_type\_facility + room\_type // hotel\_facility + hf\_type + hk\_image + hf bullet)

Flight info (flight has route, airport, route\_price)

Star\_rating plus flag



**Important Primary Key Considerations**

Throughout the project I chose to use exclusively surrogate primary keys. That is, PRIMARY KEY attributes of type INTEGER used to uniquely identify each record in a table by means of AUTO INCREMENT flag. At several points I considered using natural primary keys. As in the example in *figure 3,* at first the board\_type\_name seemed like a good candidate for a natural primary key, but to ensure that this table would enjoy a long lifespan, I chose a surrogate key. I wanted to isolate the data being stored from its method of unique identification, so that it would remain trivial to edit the board\_type\_name without also violating foreign key constraints across all tables which potentially used board\_type\_name as a foreign key. This consideration was also apparent in several other entities containing only a single piece of data (for example language, email, card\_vendor, telephone\_type).

Figure Natural Key Consideration in board\_type table.

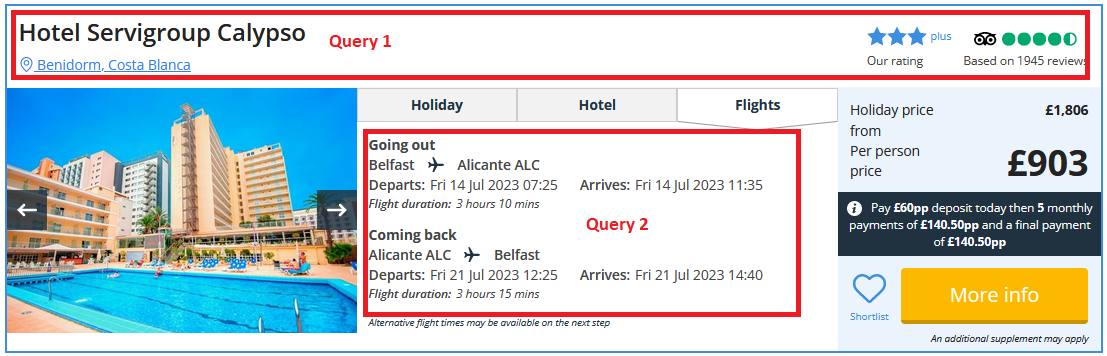
**Foreign Key Constraints**

**APPENDIX 1 : QUERIES**

Note Test data does not necessarily match website data

**Use Case - 1:**

**Generate the data visible on a hotel “card” on the website after using main search form**



**Use Case 1 – Queries 1:**

SET @HotelName = 'Spanish Hotel 1';

SELECT

  hotel.hotel\_name AS HotelName,

    region.region\_name AS RegionName,

    resort.resort\_name AS ResortName,

    star\_rating.star\_rating AS OurRating,

    star\_rating.star\_rating\_plus AS OurRatingPlusFlag,

    CAST(SUM(review\_rating.review\_rating) / COUNT(review.review\_id) AS DEC(2,1)) AS ReviewScore,

    COUNT(DISTINCT review.review\_id) AS NumberOfReviews

FROM hotel

INNER JOIN resort ON resort.resort\_id = hotel.hotel\_resort\_id

INNER JOIN region ON region.region\_id = resort.resort\_id

INNER JOIN star\_rating ON star\_rating.star\_rating\_id = hotel.hotel\_star\_rating\_id

INNER JOIN room\_type ON room\_type.room\_type\_hotel\_id = hotel.hotel\_id

INNER JOIN room\_booking ON room\_booking.room\_booking\_room\_type\_id = room\_type.room\_type\_id

INNER JOIN booking ON booking.booking\_id = room\_booking.room\_booking\_booking\_id

INNER JOIN booking\_contact ON booking\_contact.booking\_contact\_id = booking.booking\_contact\_id

INNER JOIN review ON review.reviewer\_id = booking\_contact.booking\_contact\_id

INNER JOIN review\_rating ON review\_rating.review\_rating\_id = review.review\_rating\_id

WHERE hotel.hotel\_name = @HotelName

GROUP BY hotel.hotel\_id;

**Use Case 1 – Results 1:**

Graphical user interface, application, website

Description automatically generated

**Use Case 1 – Queries 2:**

-- parameters gathered from jet2Holidays main search feature form

SET @OutboundDate = '2023-01-01';

SET @Duration = 2;

SET @HomeAirportId = 5; -- Belfast Internation Airport ID

-- parameter from website card shown above, changes for each card

SET @HotelId = 11; -- 'Spanish Hotel 1' Id

-- get route ids & compute return date

SET @AwayAirportId = (SELECT hotel.hotel\_serving\_airport\_id FROM hotel WHERE hotel.hotel\_id = @HotelId);

SET @OutboundRouteId = (SELECT route.route\_id FROM route WHERE route.departure\_airport\_id = @HomeAirportId AND route.arrival\_airport\_id = @AwayAirportId);

SET @ReturnRouteId = (SELECT route.route\_id FROM route WHERE route.departure\_airport\_id = @AwayAirportId AND route.arrival\_airport\_id = @HomeAirportId);

SET @ReturnDate = DATE\_ADD(@OutboundDate, INTERVAL @Duration DAY);

-- outbound flight info query

SELECT

  dep\_airport.airport\_name AS DepartureAirport,

    dep\_airport.airport\_iata\_code AS DepartureAirportCode,

    arr\_airport.airport\_name AS ArrivalAirport,

    arr\_airport.airport\_iata\_code AS ArrivalAirportCode,

    flight.departure\_utc\_datetime AS DepartureTime,

    flight.arrival\_utc\_datetime AS ArrivalTime,

    TIMEDIFF(flight.arrival\_utc\_datetime, flight.departure\_utc\_datetime) AS Duration

FROM flight

INNER JOIN route ON route.route\_id = flight.route\_id

INNER JOIN airport AS dep\_airport ON dep\_airport.airport\_id = route.departure\_airport\_id

INNER JOIN airport AS arr\_airport ON arr\_airport.airport\_id = route.arrival\_airport\_id

WHERE flight.route\_id = @OutboundRouteId LIMIT 1;

-- return flight info query

SELECT

  dep\_airport.airport\_name AS DepartureAirport,

    dep\_airport.airport\_iata\_code AS DepartureAirportCode,

    arr\_airport.airport\_name AS ArrivalAirport,

    arr\_airport.airport\_iata\_code AS ArrivalAirportCode,

    flight.departure\_utc\_datetime AS DepartureTime,

    flight.arrival\_utc\_datetime AS ArrivalTime,

    TIMEDIFF(flight.arrival\_utc\_datetime, flight.departure\_utc\_datetime) AS Duration

FROM flight

INNER JOIN route ON route.route\_id = flight.route\_id

INNER JOIN airport AS dep\_airport ON dep\_airport.airport\_id = route.departure\_airport\_id

INNER JOIN airport AS arr\_airport ON arr\_airport.airport\_id = route.arrival\_airport\_id

WHERE flight.route\_id = @ReturnRouteId LIMIT 1;

**Use Case 1 – Results 2:**





**Use Case - 2:**

**Demonstrate Time & Date Dependent Flight Pricing**

**Use Case 2 – Queries 1** *(ran once with each datetime variable below)***:**

SET @DateTimeToCheck = '2023-01-01 03:00:00';

-- SET @DateTimeToCheck = '2023-01-01 14:00:00';

-- SET @DateTimeToCheck = '2023-01-01 21:00:00';

-- SET @DateTimeToCheck = '2023-01-02 12:20:00';

SET @RouteId = 10; -- 'Spain to Belfast'

SELECT

  route\_price.route\_price\_gbp AS FlightPrice,

    @DateTimeToCheck AS FlightDateTime,

    route\_price.route\_price\_valid\_from\_datetime AS ValidFrom,

    route\_price.route\_price\_valid\_to\_datetime AS ValidTo FROM route\_price

INNER JOIN route ON route.route\_id = route\_price.route\_price\_route\_id

WHERE route\_price.route\_price\_route\_id = @RouteId

AND @DateTimeToCheck BETWEEN route\_price.route\_price\_valid\_from\_datetime AND route\_price.route\_price\_valid\_to\_datetime

AND route.route\_id = @RouteId;

**Use Case 2 – Results 1**

**Text

Description automatically generated with low confidence**

**Graphical user interface

Description automatically generated with medium confidence**

**Graphical user interface

Description automatically generated with medium confidence**

**A picture containing graphical user interface

Description automatically generated**

**Use Case - 3:**

**Hotel Reviews**

Graphical user interface, text, application, email

Description automatically generated

**Use Case 3 – Queries 1:**

SET @HotelName = 'Spanish Hotel 1';

SELECT

    hotel.hotel\_name AS HotelName,

    town\_city.town\_city\_name AS ReviewerTown,

    CountryName.location\_name AS ReviewerCountry,

    passenger.passenger\_first\_name AS ReviewerFirstName,

    passenger.passenger\_last\_name AS ReviewerLastName,

    DATE(review.review\_timestamp) AS ReviewDate,

    review\_rating.review\_rating AS ReviewRating,

    review.review\_content AS Review

FROM review

INNER JOIN booking\_contact ON booking\_contact.booking\_contact\_id = review.reviewer\_id

INNER JOIN passenger ON passenger.passenger\_id = booking\_contact.booking\_contact\_passenger\_id

INNER JOIN booking ON booking.booking\_contact\_id = booking\_contact.booking\_contact\_id

INNER JOIN review\_rating ON review\_rating.review\_rating\_id = review.review\_rating\_id

INNER JOIN address ON address.address\_id = booking\_contact.booking\_contact\_address\_id

INNER JOIN town\_city ON town\_city.town\_city\_id = address.town\_city\_id

INNER JOIN country ON country.country\_id = town\_city.town\_city\_country\_id

INNER JOIN location\_name AS CountryName ON CountryName.location\_name\_id = country.country\_location\_name\_id

INNER JOIN room\_booking ON room\_booking.room\_booking\_booking\_id = booking.booking\_id

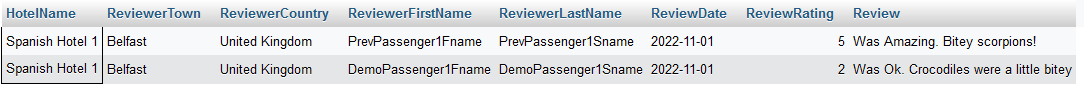
INNER JOIN room\_type ON room\_type.room\_type\_id = room\_booking.room\_booking\_room\_type\_id

INNER JOIN hotel ON hotel.hotel\_id = room\_type.room\_type\_hotel\_id

WHERE hotel.hotel\_name = @HotelName

GROUP BY review.review\_id;

**Use Case 3 – Results 1:**



**APPENDIX 2 : FINAL DRAW IO DIAGRAM**

**Put it here**