Database Design Project: *jet2holidays.com*

Background and Context

The goal of this project was to design and implement a functional database system for the 'jet2holidays.com,' package holiday booking system. In doing so, the aim was to develop a system which would mirror the booking process that a real-world user would encounter when booking a package holiday using the website.

It was important to take into account the scope of the project when designing the database system. Instead of simply adding everything seen on the user interface upon accessing the website, part of the challenge was to stay within the scope of the project. The difficulty lay in providing an in-depth, usable database but avoiding the addition of various entities which were provided by third parties, for example car rental, or were not directly related to booking a package holiday, for example suitcase weight allowance onboard a flight.

Another key aspect to creating a practical, operative database was to decide on some set assumptions that would be made in relation to the inner-workings of the database. Due to the inherently short-term, fixed-timescale nature of a project like this, these decisions were made to streamline the process of database construction and to avoid tackling some in-depth concepts which may not have directly related to the creation of a working database. Of course, it is important to recognise that by adhering to these assumptions some real-world transferability may have been lost, however potential adjustments and areas for improvement will be covered at a later stage within this report.

Other essential details explored throughout the project included the use of a relational database model. This was an idea conceived by Ted Codd, and contrasts to a hierarchical data model. While a hierarchical model is represented by a highly structured, tiered approach to storing data, Codd's idea was that data within a data set has the possibility of being more nuanced, divulging that relationships must be present between different pieces of data. This

relational model was more likened to a real world entity, such as a university, where the idea of a clearly defined and distinguishable hierarchy within personnel is not a realistic one.
Chen then developed the idea of ER Notation on top of the foundation that was the relational data model. Chen (1976) stated that
"The entity-relationship model adopts a more natural view that the real world consists of entities and relationships. It incorporates some of the important semantic information about the real world."
This semantic information refers to real entities and their information. Throughout the Entity Relation (ER) Diagram in *Figure 3,* we see examples of these instances. For example, 'passenger,' which references somebody who is a part of a booking, has attributes such as 'first_name,' 'surname,' and 'date_of_birth.' Moreover, a 'passenger,' is likely to be present on a booking within the 'jet2holidays.com,' database, meaning that there must intrinsically be a relationship between the 'passenger,' and their 'booking.'

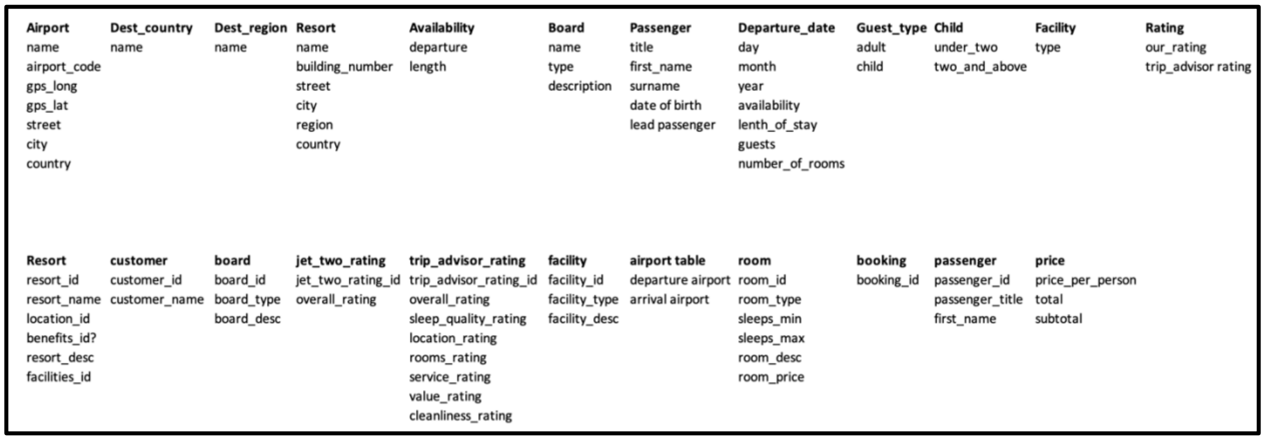
Further pre-requisites to designing a working database system such as initial entity discovery, entity relations, cardinality constraints and normalisation decisions will be discussed further within this piece. Included will be an overview of how, as a group, the initial diagrams were created, why certain design decisions and assumptions were undertaken, any justifications I have for various design and Normalisation decisions that I made independently, and why I chose to use particular data types within the database.

Initial Entity Discovery

When we first came together as a group, the initial target that we set for ourselves was to use the 'jet2holidays.com' website to create a spreadsheet of different entities and their various attributes. We used the website to form *Figure 1,* which represents our first attempts at analysing the website and outlining entities and their various fields. Upon observing the website, we saw that certain entities had multiple attributes that we would have to later consider branching off into their own table when undergoing Normalisation. For example, if we compare the 'room' column in *Figure 1*, with the 'room' table and 'room_type' table within my final ER Diagram in *Figure 3*, ‘room_type’ can be seen as an entity within its own table. This allowed various attributes to be mapped to each room type, as well as each physical room within a resort to have an assigned type through use of a foreign key. This one-to-many relationship was a key component when linking a lot of our initial entities to one another.

As with most projects, the first draft is unlikely to be identical to the final product, as was the case with this one. When creating the diagram in *Figure 1,* some entries did not align with the scope of the project. This meant that some of this entities were discarded. Keeping in mind that one of the challenges was to avoid the inclusion of features which were offered by a third party. The ‘trip_advisor_rating’ was originally included within our plans. Although upon further reflection, we decided this was provided by a third party, and so it removed from our designs.

We also had placed ‘departure_airport’ and ‘arrival_airport’ within the ‘airport’ table. However, when revisiting the reality of storing this information in our database we realised that every airport had the potential to be either an arrival or departure airport. To overcome this, we removed both attributes from the airport table and instead stored them both in a new ‘route’ table, designed for the purpose of storing flights from departure airport destinations to arrival airport destinations. The importance of this is that it allowed a foreign key to be set up between ‘airport_id’ and both ‘departure_airport_id’ and ‘arrival_airport_id’. This helped to minimise redundancy, a key part of Normalisation which was a key influence throughout the database designing process.



Airport	**Dest_country**	**Dest_region**	**Resort**	**Availability**	**Board**	**Passenger**	**Departure_date**	**Guest_type**	**Child**	**Facility**	**Rating**
name	name	name	name	departure	name	title	day	adult	under_two	type	our_rating
airport_code			building_number	length	type	first_name	month	child	two_and_above		trip_advisor rating
gps_long			street		description	surname	year				
gps_lat			city			date of birth	availability				
street			region			lead passenger	lenth_of_stay				
city			country				guests				
country							number_of_rooms				

Resort	**customer**	**board**	**jet_two_rating**	**trip_advisor_rating**	**facility**	**airport table**	**room**	**booking**	**passenger**	**price**
resort_id	customer_id	board_id	jet_two_rating_id	trip_advisor_rating_id	facility_id	departure airport	room_id	booking_id	passenger_id	price_per_person
resort_name	customer_name	board_type	overall_rating	overall_rating	facility_type	arrival airport	room_type		passenger_title	total
location_id		board_desc		sleep_quality_rating	facility_desc		sleeps_min		first_name	subtotal
benefits_id?				location_rating			sleeps_max			
resort_desc				rooms_rating			room_desc			
facilities_id				service_rating			room_price			
				value_rating						
				cleanliness_rating						

Figure 1 - Initial Entity Discovery

<u>Group ERD Design</u>

The group ER diagram came next in the database design process. We wanted to create a general design as a starting point so that each member could then manipulate it to best suit their personal preference and ideas for the database design. It was throughout this stage that we began to make certain assumptions to serve our design purposes. We figured that a central part of any international package holiday booking would be revolve around flights to reach the destination. Stemming from this, we decided to assume that a flight would always be available for the start date of a holiday. Moreover, every flight would be a direct flight, with no stopovers present. We decided to do this because the scope of the project was in relation to the package booking. Therefore, we did not want to diverge from this focus. These assumptions would allow us to streamline the booking process inside the database itself.

We also chose to use one table to store all addresses. Then, using an auto-incremented primary key called 'address_id', any new addresses added to the system will be given their own unique identifier, allowing them to be accessed easily within the database. Since deciding we were going to have one address table, inside it we had to choose meaningful attribute names which could be applied to home addresses, airport addresses and resort addresses. In the end, we decided on 'building_name,' 'building_number,' 'address_line_one,' 'address_line_two,' 'city_id,' and 'postcode,' with 'postcode,' being the least ambiguous of the attributes. Regarding the use of 'postcode,' I would later choose to assign it a VARCHAR data type. This meant that countries using different types of postal codes such as the Spanish 'Código Postal,' or the Irish 'Éircode,' could also be stored, as these both consist of a differing amount of characters than the British postcode.

Of course, throughout the creation of this group ERD, we had to take into account Normalisation. We used 1st Nominal Form (1NF) and the use of primary keys for their atomic nature to ensure that each row within the database had a distinct reference. Through following 1NF we then ensured that elements were not repeated within a table. Any elements which would appear to do so were split into two (or more) relational tables. For example, if we had ‘address’ within the ‘passenger’ table as opposed to in one of its own, the address of passengers who lived in the same household would have to be entered in numerous times within the database (passenger will be used interchangeably with customer in this project). Then, if they were to change residence then the knock-on effect would result in having to accurately change address information in multiples rows within the database. By developing a one-to-many relationship between ‘address’ and ‘passenger’ and using a foreign key to show their relationship, we ensure that data would only have to change in the ‘address’ table to affect the multiple passengers that have changed residency.

2nd Nominal Form (2NF) and 3rd Nominal Form (3NF) were also at the forefront of design decisions. As can be seen in *Figure 2,* we separated columns that were not intrinsically related to the primary key into different tables, for example, initially the ‘login’ table was inside the ‘lead_passenger’ table, but was then separated out. Other examples were ‘country’, ‘city’, and ‘title’. Whilst designing the diagram, we also ensured that any attributes which were not the primary key were not related. Decisions like these allowed us to normalise our database, minimise redundancy and make it easier for developers to understand and make changes to the database.

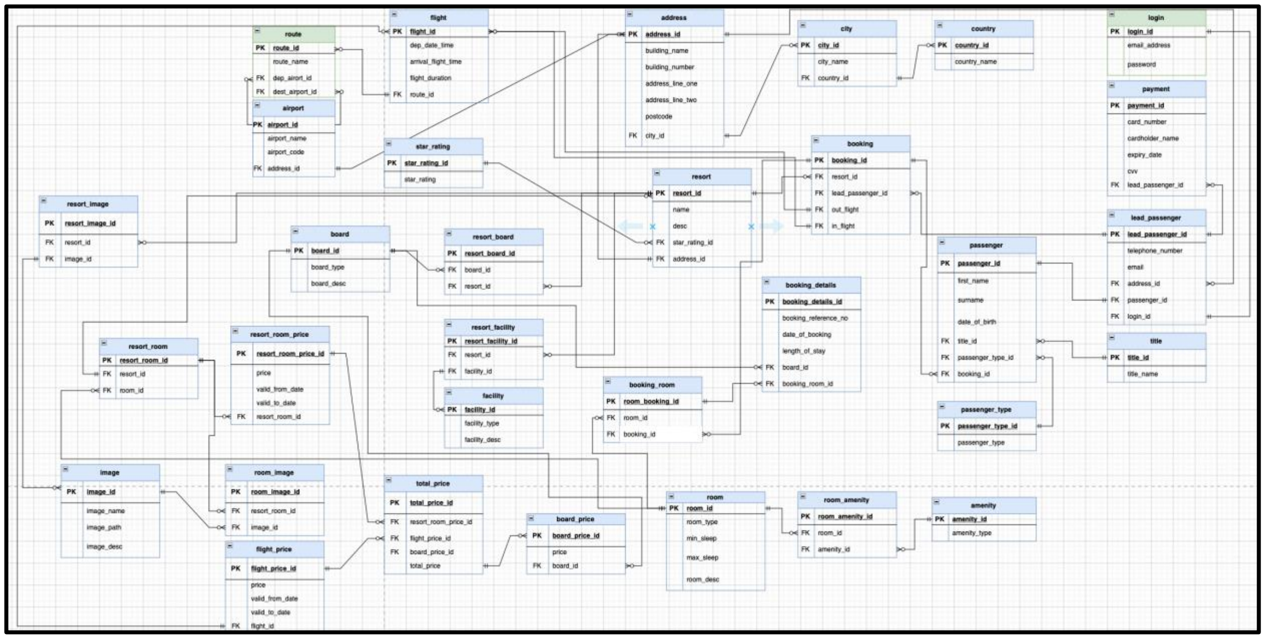


Figure 2 - Group ER Diagram

<u>**Individual Design and Database Creation**</u>

Once I was happy with my ER diagram, I began to create my database. I began by creating each table, adding an auto-incrementing primary key and assigning data types to each of the attributes. Before adding data into each table, I began to enforce referential integrity by setting up foreign key relationships. I represented this in my individual ERD by using 'FK' to the left of attributes which were foreign keys and then using lines which represent the type of relationship between two entities to link each foreign key to the primary key that it stemmed from.

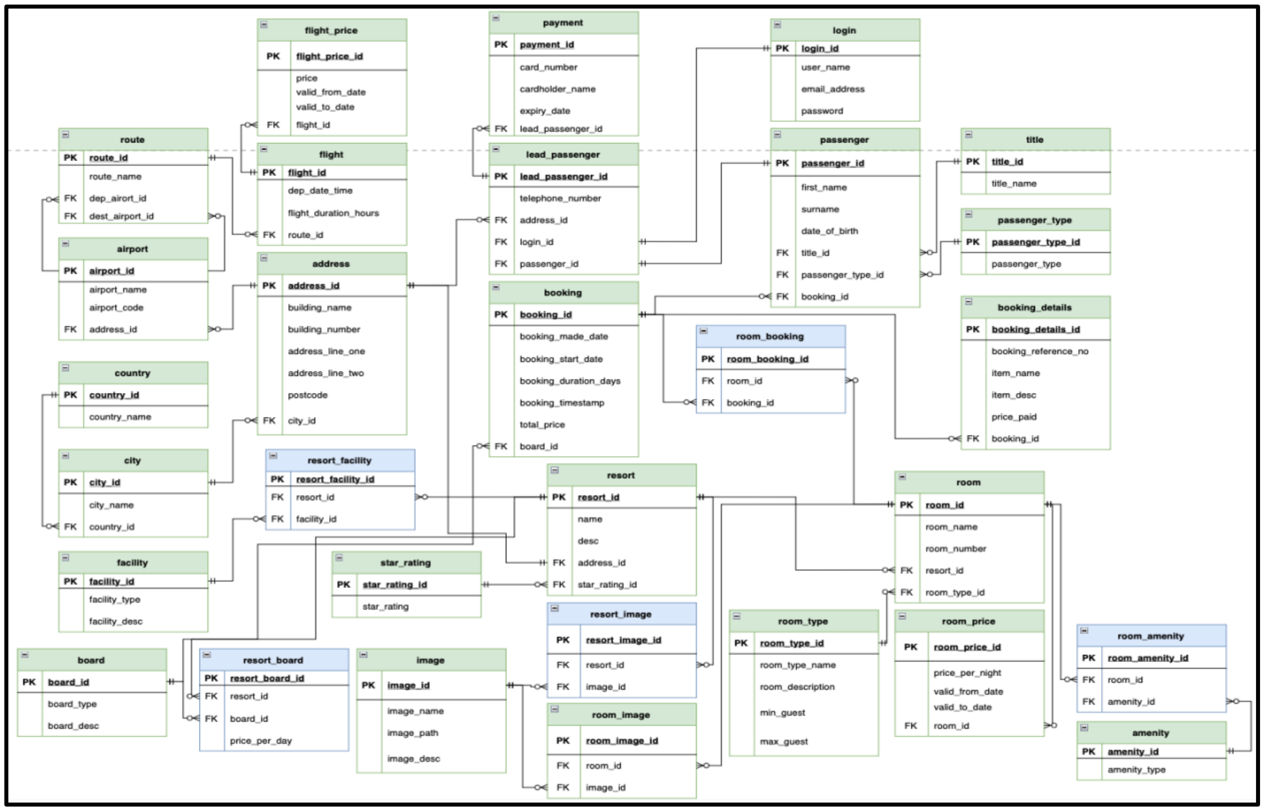


Figure 3 - Individual ER Diagram

Whilst creating the group ER diagram, a point of contention was the storage of derived attributes. Whilst it can be useful to see the age of a passenger, for example, it also means that extra data is being used to store this number. In contrast, use of knowledge already present in the database to derive this number would achieve the same result but use less storage space. By storing the passenger's date of birth and by knowing the current date and time we could derive their age. Therefore, it made sense to avoid storing derivable attributes in most cases.

However, I did include a ‘total_price’ field within my booking table. I think that the importance of being able to see a clear total when paying for a holiday outweighed the downfall of said information taking up storage space in the database.

To represent the many-to-many relationships within the database, I had to decide between using candidate keys to create the relationships, or surrogate keys. I chose to use surrogate keys throughout all of my ER diagrams and within my database. Due to the nature of a candidate key consisting of two primary keys, I thought that avoiding the inherently messy nature of viewing each primary key to understand a relationship between two entities was the easiest course of action to improve the presentability of the database both for myself, and for potential future developers who may need to view the internals of the database.

To create a (relatively) dynamic pricing model, I accounted for four different pricing periods: December through February, March through May, June through August and September through November, with the idea in mind that real-world pricing changes in relation to perceived seasonal demand. Even though jet2holidays.com allow bookings to be done through multiples of 7 days, I chose to use a ‘price_per_night’ system. This was an important decision as it allows users to book holidays over two pricing periods instead of being restricted to one particular holiday season.

To ensure functionality, I also chose to insert the ‘booking_id’ into the ‘passenger’ table, creating a one-to-many relationship. This was an interesting decision to make as it means that the if a passenger wants to go on more than one holiday with *jet2holidays.com* they will have to go through the entire sign-up process again – all information like address, e-mail and telephone number, to name a few, would have to be entered in again. I decided to take this route based on the assumption that when you vacation with *jet2holidays.com* they do not care if you have been a customer with them before. This would mean that recurring customers would not have an easy way of receiving benefits for booking with the same company repeatedly. I did this as I do not think that the possibility of reusing a customer’s information for future bookings is essential to the scope of the project. I think that this could be an area for future consideration if I were to implement improvements within the database design. A possible many-to-many relationship between ‘booking’ and ‘passenger’ using two one-to-many relationships connecting to a ‘booking_passenger’ table could be a possible solution to this issue.

I also chose to create a ‘booking-room’ table within my plans. This table represents a many-to-many relationship between bookings and passengers. To accomplish this, I needed to set up a one to many relationship from ‘booking’ to ‘booking_room’ and another from ‘room’ to ‘booking_room’. The result is that if a passenger would like to have more than one room on their booking and have separate check-out dates on their booking, that they now would have the option to do so.

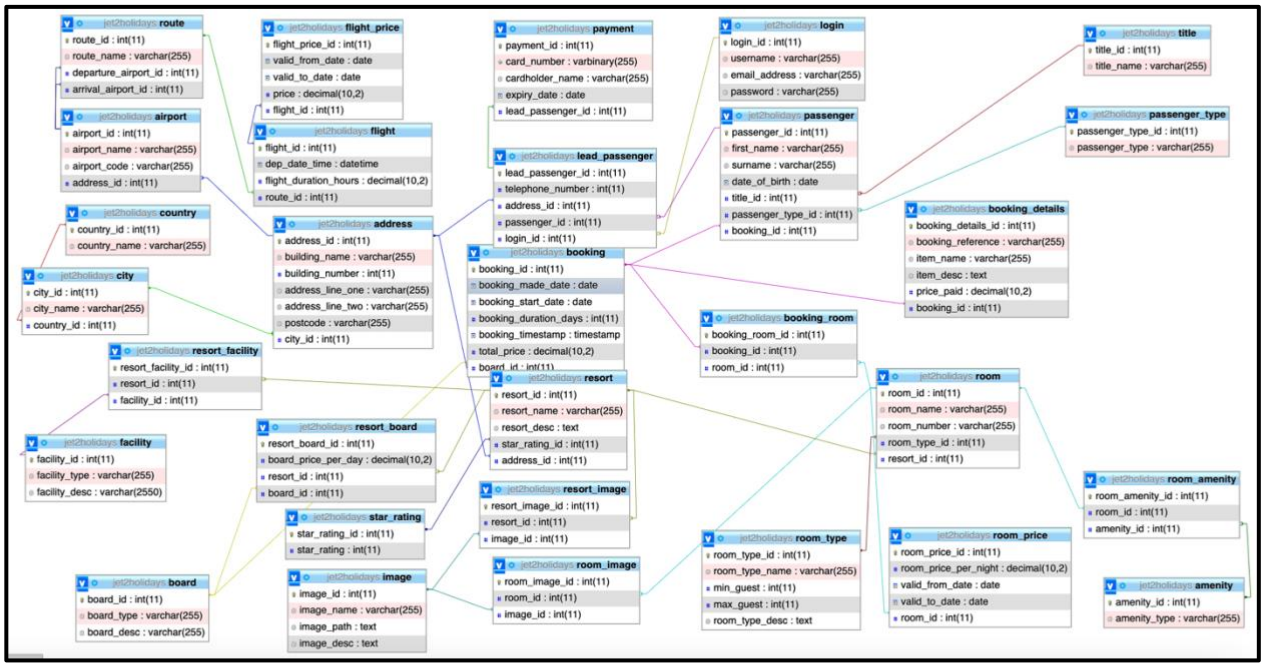


Figure 4 - Database Designer View

<u>Protection and Storage of Sensitive Data</u>

To preface this section, the techniques used to encrypt, decrypt and store data within this database will only be used within this project and the database video demonstration. In a real-world production system external code would be used for securing card payments or passwords and all storage of payment details and Personally Identifiable Information (PII) would be compliant with PCI. For the purpose of observing basic encryption and decryption methods, I included the 'payment' and 'login' tables.

By placing the 'login_id' into the 'lead_passenger' table, I was able to link each lead passenger with a login so that they could access their booking details after creating their booking. To encrypt the login password I used a salt and hash technique. I began the encryption by generating a six random character salt and combining it with the password. I stored the salt so in the database so that it could be accessed again (allowing a future password checking service through decryption). I then hashed them using the SHA1 algorithm. This meant the password could not be easily cracked by an infiltrator of the database. To simulate a password check, I could then retrieve the hash which was stored in clear text, get the hash of the salted password at sign up, concatenate the salt and login password attempt, then hash them. If this matched the stored hashed password, then the login attempt will allow the user access to their account.

On the other hand, to encrypt the payment information I used the Advanced Encryption Standard (AES) algorithm. I wanted to encrypt and store the card long number ('card_number' in *Figure 4*) so I had to change its datatype to VARBINARY. This meant that the encrypted output would be created by changing the card long number into a binary string through the use of a secret key. The AES_DECRYPT could then also be used to access this sensitive information by decrypting the secret key so that the original card long number can be viewed. As aforementioned, it would be much safer to use a third party to deal with all payments and sensitive information such as passwords. In terms of both data protection and GDPR, it would this sensitive information could not be stored in the main database. As PII has to be destroyed after seven years if the owners of the data have no reasonable use for the information, it would not make sense to store it from both a data deletion and a safety perspective.

Improvements and Adjustments

Whilst designing the database there were areas which may need adjusted if they were to fit a real-world, live database system. I would also propose that, for the purpose of increasing efficiency, it could be useful to de-normalise some aspects of the database. This would simplify querying and increase database performance, but may add more complexity to the backend service. For example, as it stands, when the user is entering their information on the front end we would have to call to the backend to fetch all the available titles from the 'title' table in the database to give them the options for title. This would add unnecessary complexity and could have a negative effect on performance and speed. As a potential solution to this, I could store a row called 'title' in the 'passenger' table. Then, by displaying a drop-down menu when a user is filling out their details on the website, the different titles (Mr., Ms., and Mrs.) could be hardcoded into the backend service.

I also think that it could be beneficial to create a system with even more specific pricing periods. During the winter months, relatively low prices were accounted for, however times like Christmas could be busier than the other winter months, and therefore it would make sense to improve the pricing model used within the database. One of the assumptions we made at the start was that 'lead_passenger' was assigned to the booker, although I believe that it would be good to separate the two, allowing for one passenger to book the package and then assign another as the lead passenger. As mentioned previously, I would also have liked to incorporate a system through which returning passengers would already have some of their information and past bookings stored within the system.

The Final Product

Overall, the designed database functions accurately for the purpose of allowing a user to book onto a package holiday. The goal of this project was to create an operational and relatively accurate representation of the *jet2holidays.com* package holiday booking system and I feel that the database that I created does so brilliantly. Granted, adjustments could be made to further reflect the website and to ensure a higher level of safety and security for customers, as well as to allow different features, however I believe that I have also outlined basic ideas that could be implemented in a wider timeframe. Paired with the video demonstration which will accompany this report, I believe the performance of this database will be sufficiently evidenced.

Reference List

Chen, Peter (March 1976). “The Entity-Relationship Model – Toward a Unified View of Data”. ACM Transactions on Database Systems

Appendix

```
1.SELECT *
FROM resort
INNER JOIN room
ON resort.resort_id=room.resort_id
INNER JOIN room_type
ON room.room_type_id=room_type.room_type_id
WHERE room_type.room_type_name = 'Studio';
```

```
2. SELECT *
FROM resort
INNER JOIN resort_facility
ON resort.resort_id=resort_facility.resort_id
INNER JOIN facility
ON resort_facility.facility_id=facility.facility_id
WHERE facility.facility_id = '1';
```

```
3. SELECT *
FROM resort_board
INNER JOIN board
ON resort_board.board_id = board.board_id
WHERE board.board_id = '4';
```

```
4. SELECT * FROM room WHERE resort_id = '5';
SELECT resort_name, room_number, booking_room_id, room.room_id
```

```
FROM resort
INNER JOIN room
ON resort.resort_id=room.resort_id
INNER JOIN booking_room
ON room.room_id = booking_room.room_id;
```

```
5. START TRANSACTION;
INSERT INTO booking (booking_made_date, booking_start_date, booking_duration_days,
total_price, board_id)
   VALUES (2022-11-26, 2023-07-01, 7, 0, 4);
   SET @last_id_in_booking = LAST_INSERT_ID();
   INSERT INTO booking_room (booking_id, room_id)
   VALUES (@last_id_in_booking, 16);
   INSERT INTO passenger (first_name, surname, date_of_birth, title_id, passenger_type_id,
booking_id)
   VALUES ('Peter', 'Parker', 1994-12-12, 1, 1, @last_id_in_booking);
   SET @last_lead_passenger = LAST_INSERT_ID();
   INSERT INTO passenger (first_name, surname, date_of_birth, title_id, passenger_type_id,
booking_id)
   VALUES ('Gwen', 'Stacey', 1995-01-21, 3, 1, @last_id_in_booking);
   INSERT INTO passenger (first_name, surname, date_of_birth, title_id, passenger_type_id,
booking_id)
   VALUES ('Peni', 'Parker', 2020-02-18, 3, 2, @last_id_in_booking);
          INSERT INTO address (building_name, building_number, address_line_one,
address_line_two, postcode, city_id)
   VALUES ('Parker Residence', 20, 'Ingram Street', 'Holylands', 'BT7 1QJ', 1);
          SET @last_id_in_address = LAST_INSERT_ID();
COMMIT;
```

```
6. SET @username = 'pparker';
SET @email = 'pparker@icloud.com';
SET @plainPassword = 'parkerspassword';
SELECT @salt := SUBSTRING(SHA1(RAND()), 1, 6);
SELECT @saltedHash := SHA1(CONCAT(@salt, @plainPassword)) AS salted_hash_value;
SELECT @storedSaltedHash := CONCAT(@salt,@saltedHash) AS password_to_be_stored;
INSERT INTO login (login_id, username, email_address, password)
                        VALUES (NULL, @username, @email, @storedSaltedHash);

7. INSERT INTO lead_passenger (telephone_number, address_id, passenger_id, login_id)
            VALUES ('02817682728', 16, 5, 8);



8. SET @loginPassword = 'parkerspassword';
SET @loginUsername = 'pparker';
SELECT @saltInUse := SUBSTRING(password, 1, 6) FROM login WHERE username =
@loginUsername;
SELECT @storedSaltedHashInUse := SUBSTRING(password, 7, 40) FROM login WHERE username
= @loginUsername;
SELECT @saltedHash := SHA1(CONCAT(@saltInUse, @loginPassword)) AS
salted_hash_value_login;

9. SELECT @duration := booking_duration_days FROM booking;
SELECT @roomPrice := @duration*room_price_per_night FROM room_price WHERE
valid_from_date >= '2023-06-01' AND valid_to_date <= '2023-08-31' AND room_id = '16';
SELECT @boardPrice := (@duration - 1)*board_price_per_day FROM resort_board WHERE
resort_board_id = '1';
SELECT @returnFlightPrice := price*2 FROM flight_price WHERE flight_id = '1';
SELECT @totalPrice := @roomPrice + @returnFlightPrice + @boardPrice;
```

```
UPDATE 'booking' SET 'total_price' = @totalPrice WHERE booking.booking_id = '3';
INSERT INTO booking_details (booking_details_id, booking_reference, item_name, item_desc,
price_paid, booking_id)
VALUES (NULL, 'KDJSI297', 'Studio room', 'Studio room in Globales Gardenia Hotel &
Aquamijas', @roomPrice, 3);
INSERT INTO booking_details (booking_details_id, booking_reference, item_name, item_desc,
price_paid, booking_id)
VALUES (NULL, 'KDJSI298', 'Board - bed and breakfast', 'Bed and breakfast for 7 nights',
@boardPrice, 3);
INSERT INTO booking_details (booking_details_id, booking_reference, item_name, item_desc,
price_paid, booking_id)
VALUES (NULL, 'KDJSI299', 'Return flight from Belfast International Airport to Malaga Airport',
'Fly from Belfast to Malaga on 1 July. Fly from Malaga to Belfast on 8 July.', @returnFlightPrice,
3);

10. SET @cardholdername = 'Mr Peter Parker';
SET @cardnumber = '1234 5678 1234 5678';
SET @cardenddate= '2025-10-08';
SET @secretPasssword = 'greenGoblin';
SET @cardnumber = AES_ENCRYPT(@cardnumber,@secretPasssword);
INSERT INTO payment (payment_id, card_number, cardholder_name, expiry_date,
lead_passenger_id)
VALUES (NULL, @cardnumber, @cardholdername, @cardenddate, 4);
SET @secretPasssword = 'greenGoblin';
11. SELECT payment_id, cardholder_name, AES_DECRYPT(card_number, @secretPasssword),
expiry_date, lead_passenger_id
FROM payment WHERE payment_ID = 4;
```