DATABASES PROJECT REPORT

Designing a database for Hotels.com

JACK KYLE
40209514

Contents

Introduction 2

Development of the Entity Relationship Diagram through the project 2

Accommodation and rooms 4

Customer 6

Booking 8

Shared tables 10

Improvements 11

Conclusion 11

Bibliography 12

Table of figures 12

SQL queries 12

Introduction

In this project the goal was to produce a database that was modelled after the website “Hotels.com” and its function in accommodation booking. The project was decided to focus solely on accommodation bookings. This is because services also offered on Hotels.com such as flight and attraction bookings are under different companies and would be sorted as such, along with the data related to them being stored under different sections, such as the flights being under “travel.uk.hotels.com” rather than Hotels.com. This means that this would not be within the confines of the database designed in this project. This was also decided upon in order to increase the feasibility of this project.

The design of the database was based on Chen’s theory, that the database “adopts the more natural view that the real world consists of entities and relationships.” [1] This was done by considering that the data should be considered as intersecting web of entities, attributes and their relationships. This was modelled with Entity Relationship Diagrams (ERD.)

The start of the project involved each member of the group studying the website individually, creating accounts, and by doing so, finding the information the website would store, and thus the information that would need to be stored into the database for this project. This allowed for the entities of the database to be decided upon to allow for the storage of this data. This then lead on into the development of the structure of this data such as what tables each entity would go in to, which entity the primary key for each table would be and which of these should be foreign keys for other tables in order to decide how the tables related to each other, shown through the use of ERD. Once a final structure was decided upon by the group, the individual section of the project began, with the designing of the actual database in MySQL and with this building, more development was made toward the structure as once the data was entered it gave a new view of how the data would be organised in entities and tables. Once fully developed and populated, the database then had its ability for representing Hotels.com through the use of SQL queries.

Development of the Entity Relationship Diagram through the project

As already mentioned, the project began with studying the website on which the database was to be based, including making profiles in order to get further ideas of what details are stored about the customers. Using this method of creating an account and going through the website, making most of a booking to see the details taken as well as starting to set up an accommodation to see the details there allowed for a thorough examination of the type of data the database should store. This lead to the first list of entities and attributes, not yet linked up in any form of structure shown below in figure 1.

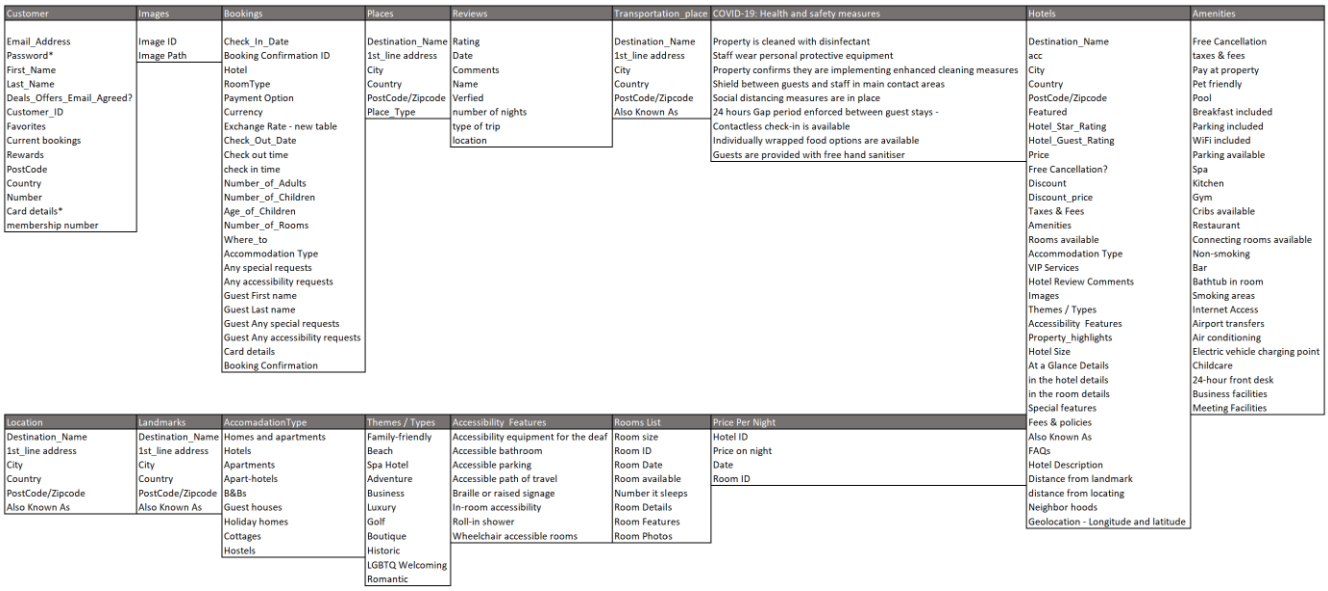

FIGURE 1: INITIAL EXCEL TABLES OF DATA TAKEN FROM HOTELS.COM

By taking down this data that was to be stored, it was then possible to start deciding what was necessary for the database and what was not. This meant that a specific function for the database was to be chosen in order to decide this. The function that was decided upon was that the database should be able to store the data of customers, accommodations, bookings that would be made by the customers for the

accommodations, reviews about these accommodations left by the customers, as well as also allowing the customers to view all data relevant to them on this database (such as their bookings made, and accommodation details when viewing one to book.)

With these purposes in mind, an initial ERD was created, given below in figure 2.

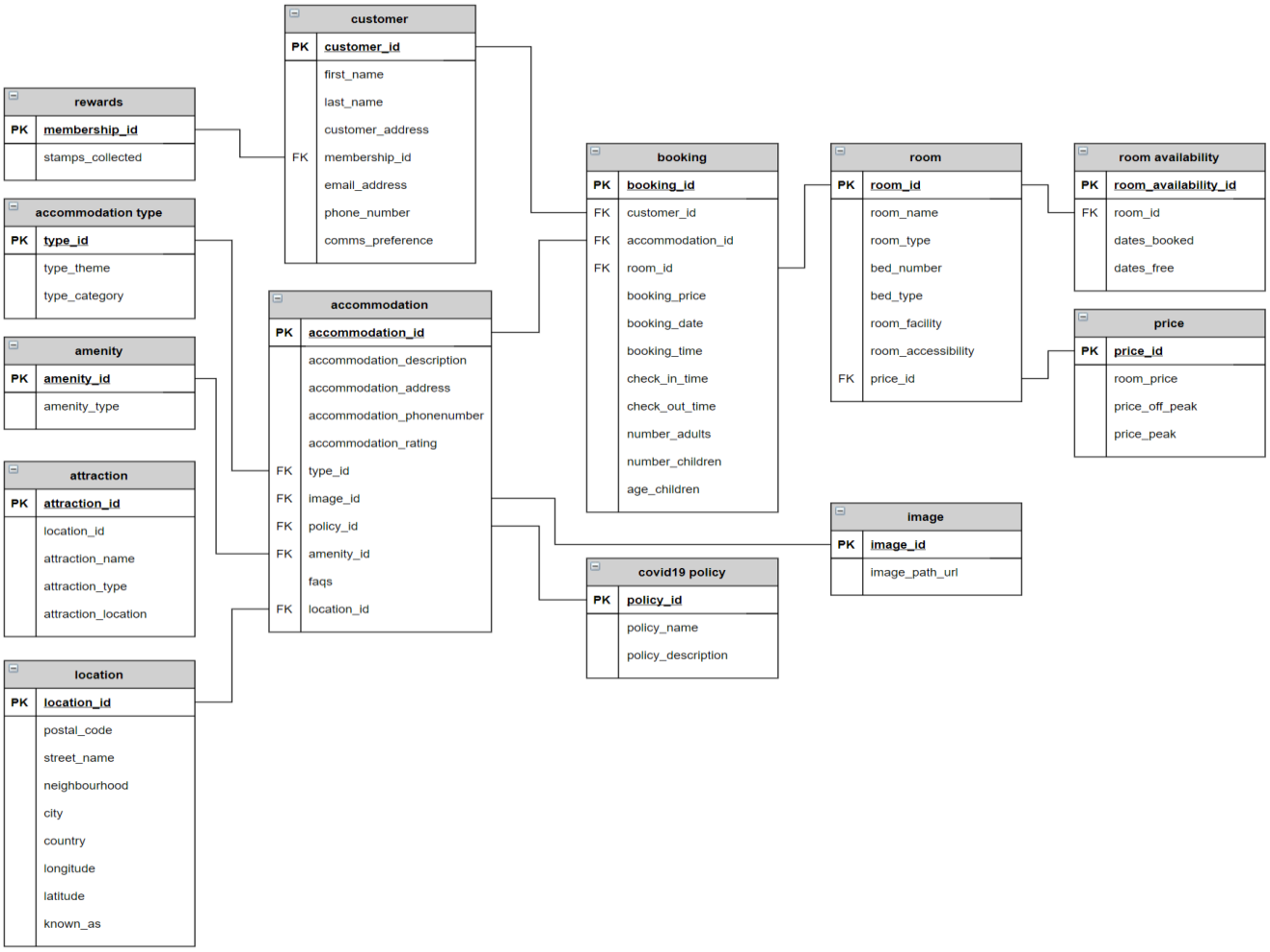

FIGURE 2: INITIAL GROUP ERD

This initial ERD was then followed by multiple iterations as the group worked through the tables and decided upon different ways for the data to be organised and which tables needed normalised. This resulted in more tables being created as more data was to be stored in order to accomplish our goals in a better fashion. This lead to the final ERD given below in figure 3.

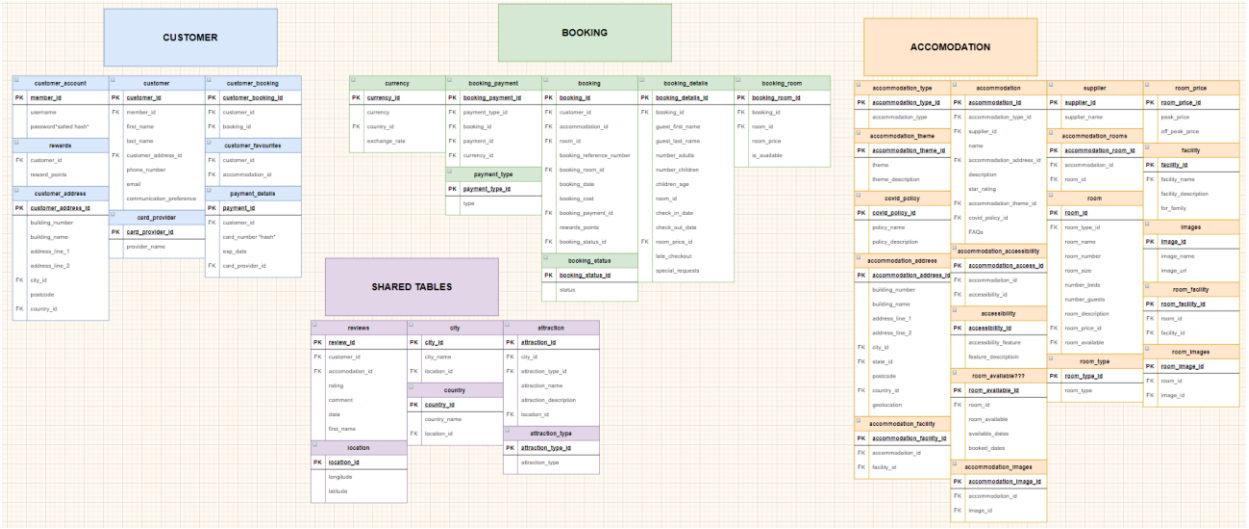

FIGURE 3: FINAL GROUP ERD

This final ERD was then developed individually and the final result used in phpMyAdmin is given in figure 4 below.

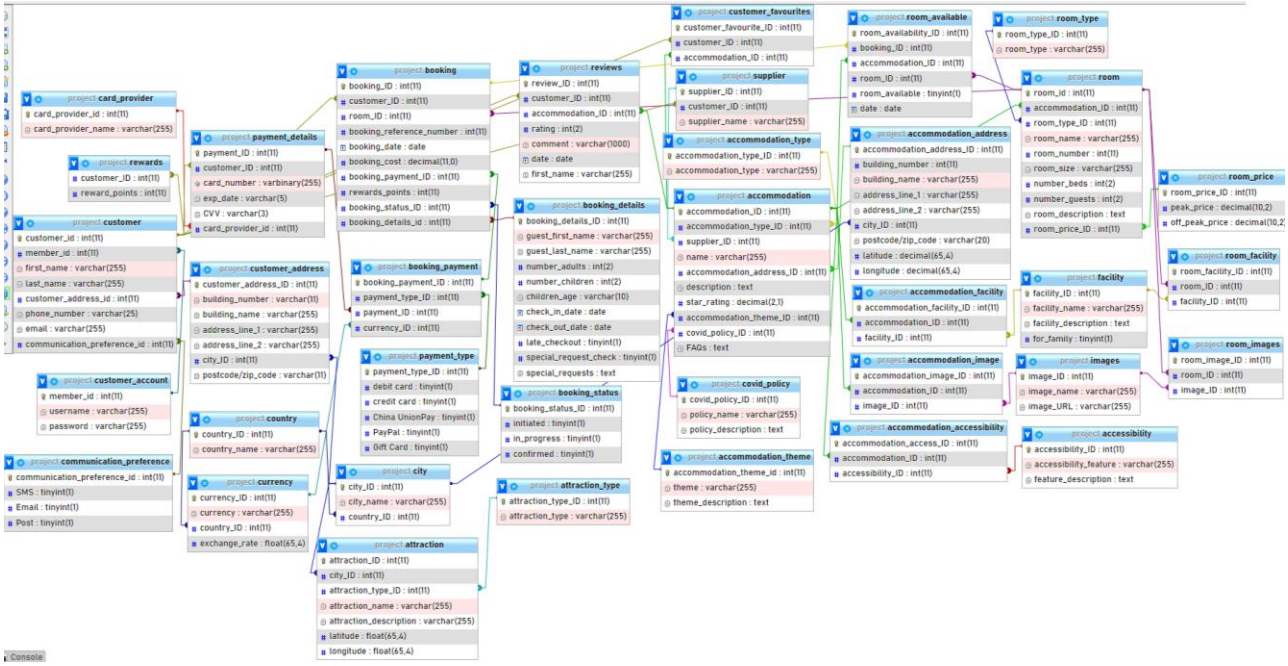

FIGURE 4: FINAL ERD DIAGRAM REACHED INDIVIDUALLY

Accommodation and rooms

The core of this database is the booking, however, as the booking section is composed partially of the accommodation and customer details, these will be gone over first. The accommodation tables went through multiple changes throughout, starting with only the accommodation, accommodation type, amenity, COVID-19 policy, image, room, pricing and room availability tables. While this may seem a lot, by the end of the group section it had then added the accommodation theme, accommodation address, accommodation facility, accommodation image, accommodation accessibility, supplier, accommodation rooms, room type, room image, room facility and review tables.

One thing that stayed the same was the primary key of both the accommodation and the room tables. This is because these were chosen as unique identifiers for the accommodations and the rooms held within them in order to have them held as unique entities, so that despite other accommodations or rooms having similar details, they can be identified as the specific room they are, which is incredibly important for a booking system as this will then help stop rooms from potentially being double booked by users. These unique ID variables are both integer types of length 11 and both are auto incremented by the system. This is because using the auto increment system means that there won't be doubling up of IDs, therefore eliminating any possible confusion on which ID matches to which property or room. This also allows them to act as foreign keys in other tables, such as the accommodation ID being used as a foreign key in the booking table in the original layout, or the room table in the final layout. Using the accommodation key as a foreign key for the room table allows for the room to be uniquely associated with a property, important for the purpose of booking these rooms out, as incorrect links between rooms and accommodations could cause a loss of money for one or both of the properties involved and confusion for the customer due to misinformation telling them that a property has a room that it actually does not. The accommodation table also stores necessary fundamental details such as name and description as these directly relate only to the accommodation. These details were stored using the varchar and text variables. These choices were made as the varchar allows for short amounts of text, up to 255 characters. This works well for shorter pieces of text such as a name, whereas the description needed a larger amount of text, which works well with the text variable, allowing up to 1000 characters. The FAQ section was also covered by text.

Another important table that stayed the same was the COVID-19 policy table. This one is important as all accommodations are required to display the measures being taken to deal with COVID-19 such as enhanced

cleaning protocols. Room price also stayed the same in terms of structure, and this was done on the assumption that the prices would adjust for off peak and peak based on the date automatically.

The tables that were added in throughout the group process were done so to either hold more data than was originally expected to be held from the initial design or to normalise out the data to conform to the normal forms. One such example of tables being normalised to conform to the third normal form is accommodation address as "Third normal form is violated when a non-key field is a fact about another non-key field" [2] and this is the case when the full address is entered into the accommodation table as the non-key fields become facts about the others such as the city being a fact about the street the accommodation is on and so on. This meant that the accommodation address was then made its own table, in order to contain this data and maintain the third normal form. It is also assumed that the addresses would be provided by a GPS or maps service, outside of the scope of this project, and so it is best represented by a table of its own as well, representing this outside service used. It also matches the entry of the details on Hotels.com as the entry of the address is a section of its own when registering one on the website. There was later in the individual section the addition of allowing building name to be null in accommodation address as not every building has a name like that. This was also the same for address line 2 of the accommodation address table.

Other important tables added in were the tables for linking accessibility to accommodation, and facilities and images to both accommodation and rooms. These were important as these were added in to represent the many to many relationships held between the rooms and accommodations and these features, as many rooms can have many facilities or images and it's the same with accommodations and facilities, images and accessibility features. These are important to represent as on the website it is shown that what facilities are related to each room and so it must be linked. It is linked through this method as the original method of facilities being in the room table would require that the room would have to be repeated multiple times in order to list all the facilities available, and this violates the first normal form, "the domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain." [2] This meant that the facilities had to be in their own table, and with them being many to many it required the usage of a room facilities table. This is then the same for images and accessibility features. This then allows for the accurate representation of the data displayed on the site with these features. The use of one table for facilities and images, despite being related to both rooms and accommodations, was also done purposefully, as some accommodations would use pictures of the rooms on the main accommodation page, and some facilities would apply to both, such as free Wi-Fi in public areas of the accommodation as well as the rooms.

The accommodation room theme and type table was also then split out into two, with theme representing such data as "LGBTQ+ friendly" accommodations and type then representing such data as whether the accommodation was a hotel, B&B or motel etc. This was done as the data was about two separate attributes, so they needed their own tables to conform to third normal form. [2] The use of both theme and type as a filter when searching for an accommodation is shown in query 1 of the appendix.

The review table stayed the same between the final group and individual design as it covered all bases required. This table is then able to be used to link an accommodation and a customer via foreign key constraints along with a rating and comment they may wish to leave, as this is reflective of what's on the website. Creating a review is shown in query 9 of the appendix.

By the end of the individual design the accommodation rooms table was removed again. This was because it was not a many to many relationship and so this meant that they could instead be linked just by a foreign key relationship of accommodation ID in the room table, as previously mentioned.

Another change made in the individual design was the addition of the foreign key of the booking ID to the room availability table. This table was designed with the assumption of it would be filled with dates automatically by a system as it would be tedious to do so for just the sample data. The addition of the booking ID to this table allows for greater clarity with the room availability for the accommodation as it will give more detail on the bookings by connecting the booking to the room availability table, meaning that the accommodation end user can go straight from checking what rooms are available to the booking to check the details on it, in such an event as the booking coming up soon and potential preparations needing made. Another change made in the individual design was the removal of the country ID and location ID from the address. The location ID was instead changed to geolocation coordinates. These changes were made as the

country ID can instead be placed within the city table and so the country the city is in can be checked via that table instead, connected to the accommodation address table by a foreign key constraint on city ID. The location ID was also removed as the location ID table was removed in its entirety, due to an ID per individual coordinate seeming redundant with how specific coordinates are themselves. This instead was replaced with geolocation, something to be kept in the event of an accommodation being somewhere like out in the countryside, where street names may be less shown, allowing users to instead use the coordinates to find the precise location of the accommodation. State ID was also removed as this was never implemented in the database.

Another addition to the accommodation tables was the addition of the foreign key constraint on customer ID in the supplier table. This was added in as this then allowed for users to be considered suppliers of accommodations on the website, a fundamental part of the website as it allows for users to set up their own properties. This foreign key then meant that it would tie the user to the accommodation as its supplier, whether that be as a large corporation or someone renting out an individual property, which would in turn allow them access to the details of the properties they had put up on Hotels.com. Accessing these details and also adding them in are shown in SQL queries 14 and 15 of the appendix. Query 15 was given in a transaction to show how the accommodation details entry can be cancelled and the details won't be saved.

Customer

As previously stated, before getting to the core of this database, the booking, the customer and customer related tables must also be gone over as they are key in forming the data of the booking tables. The customer tables started out small, with only a customer table and a rewards table. By the end of the group section there had been plenty more added in with customer account, customer address, payment details, card provider, customer favourites and customer booking.

Only some of the attributes of the customer table and rewards table stayed the same, as these were fundamental details such as the users name, contact details and their preference for contact (which would later change in the individual design.) The customer ID stayed the same too, as an auto incremented integer for the primary key. This was done for the same reason as the accommodation ID where it is to give an unique ID to the user that their account may be identified solely with it. This customer ID would then also be used as a foreign key for multiple tables such as in bookings (however the way it was used with bookings would change with further design.) The rewards table also stayed the same as it covered a simple concept of how many rewards stamps a user had and linking this to the customer table. The original had a membership ID as the primary key that linked to the customer ID as a foreign key, however this later changed to have the customer ID as a foreign key constraint in the rewards table. This was done as the membership ID was used in the customer account table and so the rewards were instead tied to the customer table via the customer ID as it is an attribute of the customer. The phone number was kept in, however the final variable type for the number was chosen as varchar due to the possibility of symbols like "+" in a phone number. The entry of customer details was shown in query 10 in the appendix. This was given in a transaction to show how the customer can cancel their account creation and have the details not save.

The tables that were added in during the group design were done so in order to normalise out some of the data or to allow for more data to be added about the users of the website. One table added to normalise out the data was the customer address table. This is because, similar to accommodation address, keeping the entire address in the customer table would be having a non-key field being a fact about another non-key field as the data relates to the address rather than the user, thus violating third normal form. Therefore the customer address table was added in to allow this to be conformed to. It also stops data being unnecessarily repeated, such as city for customers living in the same area or even address in the event two customers from the same address have signed up. This allows the table to conform to first normal form by not having the data repeated. This table then had the customer address ID as its primary key, again done using auto increment integers in order to allow the addresses to all have unique IDs. This table was also designed under the assumption that the data about addresses would be automatically filled in via a maps or GPS service, where a customer can then enter some of the data, such as post code, and have the rest be automatically filled from a choice, giving another reason as to why it would be in its own table.

One of the tables added in, the customer account table, was done so as this was used to store important details about the user used for log in, such as username and password, as well as giving a membership ID. It was assumed that the membership ID would be automatically filled in by the system, however it would not be auto incremented as it is desired for it to be different from the customer ID generated by the database. This is because the user should not have information that relates directly to the structure or content of the database that they themselves are not directly related to. The membership ID would however remain as the primary key of this table, a change from the initial design where it was used as the primary key for the rewards table. This was instead changed to the customer account table as it is a detail the customer is told and can be used alongside the username as a unique identifier of the customer's account too. With the storage of passwords comes the requirement of security. Security is necessary as a password can be used to gain access to a user's account, which can then hold sensitive information such as address or some payment details. The password was protected through the usage of a salted hash. This meant that a salt was created, a 6 character long string, which was then concatenated onto the start of the password string stored for the user. This was then hashed in order to produce a string that to the human eye would look like nonsense. This then had the salt concatenated onto the start of it too. This was shown in query 10 when an account was created. To then check if the password entered by the user on the actual website was correct, the salt at the start of the long string stored can be taken and added to the password entered and hashed. This can then be compared with the string stored with the salt removed and if they are the same then the user can log in as the password entered is correct. This is shown in query 11.

Another of the tables added was the customer favourites. This was added in as the user is able to add accommodations to a list of favourites. As it is a many to many relationship this required a table of its own so that a user can add many accommodations to their favourites and an accommodation can be added by many users. This was done through two foreign key constraints, with the primary keys of the customer and accommodation tables.

Customer booking was also added in. This was done to allow a user to create multiple bookings. It would also allow for multiple users to a booking. This was later removed in the individual as a design choice because it was not necessary for a booking to have multiple users as this is not a feature of Hotels.com.

Other important tables to be added in were the payment details and card provider tables. These were made as the customer needs to be able to pay for their bookings and so card details are needed to be stored on the website via the database. Outside services such as PayPal it assumed can be used via their service and thus aren't included in this project. These tables have a primary key each of payment ID and card provider ID. These allow for the card provider to link to the payment table and for the payment details to be safely stored individually without using the actual details as identifiers as this would be an unsafe method of storage. The payment is linked to the customer via the customer ID as a foreign key constraint as this then allows each customer to be associated with their specific card details. The card provider is set in a different table as this is a drop down menu such as visa or Mastercard being available as these are limited options to choose from. These then have a unique primary key to identify each. The card number was then to be protected due to the sensitive nature of the details. This was done in case the database was somehow breached in order to protect the card details via encryption so the card numbers aren't directly available. This was done by the AES encrypt method which allows for a password to be set in order to be able to access the card details. This was shown in query 12, where the card details are inserted and the encryption is applied with a password set, and 13, where the method of decrypting the details to be able to access them is shown. The card number had to be stored as a varbinary variable as this was the method that allowed for the AES encrypt method to work. A varchar was also used for the expiration date rather than a date as the date did not allow for only month and year and required a day, a detail not included in the expiration date of a card. This meant that a varchar would be suitable for storing the data as it allowed for both numbers and the slash ("/") in the middle.

Further changes were made in the individual design to the structure of these tables. A new table was added in for the communication preference, normalising it out of the customer table. Another change was the removal of country ID from the address.

The communication preference was normalised out of the customer table so as to avoid violating first normal form [2], as to have it in the table would result in repeated details of the customer in order to show what preferences of communication they have. It was instead stored in communication preference as its own

table as this allowed for the storage of all possible combinations of communication preference with unique IDs that can then be applied to the customer table. These were stored as the tinyint variable as this type allows for either a yes or no type response, represented by 1 or 0 respectively. This meant that each form of communication preference could be shown by whether each form of communication was desired or not. This was then linked to the customer table via a foreign key constraint on the communication preference ID into the customer table, tying their preferred communication methods to their accounts.

The country ID was removed from the address tables as this was instead used in the city table, as the city will be in the address table instead and the country can then be found by this foreign key link via joins, similar to joins used in query 1 but linking customer address to city by city ID and city to country by country ID. This allows for less detail to be required in the customer address table at once. Another change made to the customer address table was to allow for the building name to be null, as not every building has a name that some users wouldn't be able to enter anything there. This was also the same for the address line 2 option as it is not always necessary when giving the address, as the first line can cover it.

Booking

As has been mentioned, the booking is the core of the database as the entire purpose of the website Hotels.com is to allow for customers to book accommodation. This means that bookings have to be stored safely and in detail so as to avoid confusion and problems with both customers and accommodations due to potential loss of money on both sides if something goes wrong with the booking details.

In the initial design the booking section was covered by just the single booking table, linking to important details such as the customer details, room and accommodations tables with other details such as date, time and number of people on the booking stored in the table. This was later changed throughout the group section. By the final group iteration of the ERD, the booking section had had extra tables added in for currency, booking payment, payment type, booking status, booking details, and booking room.

The booking table stayed partially the same, keeping the foreign key constraints with the customer, accommodation and room IDs. It also kept the booking date and cost. These are all key details about a booking and so were fundamental details from the start, hence they were kept.

There were however also changes to the booking table. The booking reference number was added in, for similar reasons to the membership ID, giving users a number to reference their booking without giving them details on the structure of the table by giving them the actual booking ID. It was assumed that the booking reference would be automatically generated, but not in the auto increment method of the primary key, and instead generating a random 8 number long integer. The check in and out times, and dates were instead moved to a different table called booking details, along with the number of people and ages of children where applicable. This was done to partially normalise out these details as the ages of children would violate the third normal form, as it would be non-key data about non-key data [2] (in this case the number of children.) It was also done as it allowed for the focus of the table to be the details of the booking while the booking table can then be used to focus on more key attributes of the booking like the foreign key restraints such as customer ID and accommodation ID. The booking table was then given more foreign key constraints, such as booking room ID, booking payment ID and booking status ID. These were all added as these were the primary keys of new tables and thus allowed for them to be linked to the main booking table. An example of the booking table being filled in is given in query 4. This query also shows off the booking details and booking payment tables. This query was done as a transaction to show how the user can take back their data if they cancel the account creation. It was also shown in queries 2 and 5 when counting the number of bookings made at a specific accommodation.

These new tables were added in to provide further detail on the bookings. One such table was the booking payment table. This was added in, along with the payment type and currency tables, in order to link payment details of customers to the bookings they were making, to ensure that the payment came from the right customer and went to the correct accommodation. This table was composed of a primary key of booking payment ID which allows for the unique identification of booking payment details, along with otherwise entirely foreign keys. These foreign keys were payment type ID, booking ID, payment ID, and currency ID. These foreign key constraints allowed for the booking payment to be linked to the payment details of the customer as well as the booking, as previously mentioned.

The other two, payment type ID and currency ID, were added specifically for this table, with payment type as a table encompassing the varying methods of payment such as credit card, debit card, or PayPal. This was represented in the payment type table by tinyint variables, by having each potential type of payment listed as 1 and the rest 0 once each in the table as a row and assigning a unique ID to each of these. This choice of tinyint to represent each was chosen in the individual design, with the original just having a type of payment box entered. This then allows for a unique ID to be placed in to the booking payment table to declare the type of payment. The other table, currency, was used in order to help with the exchange rate if the customer was paying in one currency that would be different from the currency of the country they're booking the accommodation in. This table was composed of currency, exchange rate, and a foreign key constraint with the country ID from the country table, along with a currency ID for each type. The currency attribute was used in order to give the name of the currency, while the exchange rate was given to translate that currency into GBP, due to the nature of where this project is being done from. It was also assumed that the exchange rate would be taken automatically from a service, in order to update it as the currency exchange rates change. The country ID was used in order to tie each currency to a country, which can then be linked through to the accommodation location using joins to connect to the country table, to the city table to the accommodation address table to the accommodation table, allowing for the right currency to be used that way. The currency details were placed in their own table to conform to the third normal form, that the non-key data wouldn't be about non-key data if left in the booking payment table. [2] The payment type table was created as there are limited options for the payment and the user would pick from a drop down list, similar to how the data is stored then when the payment type is stored in its own table and chosen for the booking payment table.

Another table added in was the booking status table. This table was shown in queries 3 and 6 when showing bookings in detail. This was added in for a similar reason to the payment type table as there are limited options for the booking status, with initiated, in progress and confirmed. There is a tinyint for each possible status, with a separate booking status ID for each one being 1 and the rest being 0, to represent which stage of the booking it was in. This ID was the primary key for the table and was then used as a foreign key in the booking table to show what stage each specific booking was in. It was also assumed that these would update automatically as the payment for the booking went through. The choice of tinyint to represent each was added in the individual design, as the final group design originally had just a column called status and no other detail.

The booking room was the other table added in during the group design of the ERD. This table was added in to allow for more details to be given about the room used for the booking and to show whether it was available and how much it cost. This was done with a primary key called booking room ID to represent each individual linking of room to booking. The booking and room IDs were used as foreign keys in this table in order to provide these details. Room price was then used to give a pricing to the booking and the availability column was used to then see if the room was available. This table also conflicted with the booking table that had the room ID in it already as well as having the booking payment ID in it, when the booking ID was already in the booking payment ID. This could potentially cause unnecessary issues if any data were to be entered correctly as two different links doubles the likelihood of wrong entry. This table was normalised out of the booking table as the room details were instead about the room specific to the booking and not the booking itself.

The individual design then resulted in more changes in the booking tables. The columns used in booking status and payment were changed to include the tinyint variables as previously mentioned. Another change was the addition of the column, special request check, also as a tinyint variable, as well as allowing for special requests to be null. The booking room table was also removed entirely.

The addition of the special request check and the ability for special requests to be null was in order to allow users the option of entering a special request. The special request check was entered as a tinyint variable as mentioned, allowing for a 1 or 0, representing a yes or a no respectively. If the check was a 1 then this would allow for the user to enter a special request in the text variable of special request. If it was a 0 then the special request would be null. It was assumed that the status of the special request check would be checked by the website.

The booking room table was removed entirely, as the details that were in that table were already represented by the booking table, room price table and room availability tables, and so these should either

have been linked to the booking room table, or they can just be accessed through the usage of joins instead, leaving the booking room table redundant. Query 7 shows how joins can be used with bookings to find out how much an accommodation has made.

Shared tables

In the original design of the ERD made by the group the only tables given were attractions and location. The attractions table was used in order to represent the name and location of attractions that users of the website could then look up in order to find local attractions when booking an accommodation. The location table was used to represent the address and coordinates of the accommodation and attractions.

The attractions were represented with an attraction ID for the primary key in order to give them a unique identifier on the website, as the names could possibly be the same but in different areas, therefore stopping them getting mixed up. The location table had a primary key of location ID as well to also give each address and location a unique identifier in the table.

By the final group design these had been changed, with the address, city and country part of location ID having been normalised out in to other tables, such as city, country, accommodation address, and customer address. This was done so as to have unique address tables for customers and accommodations due to them being two separate entities and so they have two different attributes. This left the location table with just the location ID primary key, longitude, and latitude as columns. Other tables added in were the reviews table and also the attraction type table.

The location table from the original design was changed a lot as mentioned. One table that came from it, city, was used in order to represent cities. This is because the city is its own entity and can be related to many addresses, and so, to conform to first normal form and avoid the unnecessary repetition of data by repeating the city each time a customer or accommodation is registered there [2], the city table was created with a city ID for each city. This was also the logic behind the country table.

The location table, as mentioned, was left with just 3 columns and was used to designate a specific ID to each set of coordinates which would then be used to apply to the address tables. The ID was chosen to be an integer of length 11 as the ID would be automatically incremented. The length of 11 was chosen, as it was for all other auto increment IDs, as this would allow for a lot of data to be stored in the table with such a high possibility for the number of IDs to go to.

The attraction table also had the attraction type normalised out to its own table as this was data that would be represented by a limited number of options. Therefore, by putting the attraction type in its own table, the option can be chosen from a drop down menu, similar to how it would be done on the website when adding an attraction to it. The attraction type has a primary key of attraction type ID, allowing for each attraction type to be unique. The attraction table has a foreign key to the attraction type ID then to allow for them to be linked, along with foreign key constraints with city ID, and location ID, allowing for valuable data such as where it is to be available when searching for the attraction on the website.

The other table added, reviews, was added in to allow for reviews to be left by customers in relation to accommodations. This was done by linking customer and accommodation ID to the table through foreign key constraints. There are also columns for rating, comment, date, and first name, as this is the data displayed when viewing a review on an accommodation page on Hotels.com.

These tables went through further development in the individual design, such as the location table being removed altogether. There was also changes such as then removing location ID from the city and country tables, and adding in country ID to the city table.

The location table was removed for reasons given before when discussing addresses, the location ID was redundant as coordinates are already so specific that the ID was unnecessary and so the coordinates were instead added to the accommodation address as well as the attraction table.

The location ID was then removed from city and country as the table no longer existed with that primary key, however the coordinates were not then added in as cities and countries cover such large areas that it would be impossible to describe them with just one coordinate. The country ID was then added to the city table as a foreign key so as to discern which country the city is in with the removal of coordinates from the city table. This then allowed for country ID to be removed from address tables as the city ID was already in them and this would create a link to the country via this new foreign key in the city table.

Improvements

As the project was developed, many different problems were sorted through normalisation and design, however, not all possible issues were sorted there are still improvements that could be implemented.

One such improvement that could be made is the addition of contact details such as email to the accommodation table. This would allow for the users of the website to contact the accommodation in order to ask any questions they may have. Another improvement related to accommodation would be the introduction of a accommodation COVID-19 policy table, as this would allow for a many to many relationship between these tables, allowing for more specific descriptions for what each accommodation does, rather than through entering each accommodations details on what they do separately. It would also allow for unnecessary repetition to be eliminated, helping the table to conform to first normal form. [2] Another many to many relationship related to accommodation that could be added in would be between themes and accommodation, as an accommodation could have multiple applied to it such as LGBTQ+ friendly and family friendly, and would avoid violating first normal form by needing multiple entries in the accommodation table in order to allow for this via adding in an accommodation/accommodation theme table for the many to many relationship to be represented in. Another improvement would be to allow for more price variation, such as for when events are on in the nearby areas as well as being peak or off peak season. This could be implemented by adding more price options to the room price table.

Other improvements related to the booking tables would be to make the payment ID potentially null for the booking payment as not every user is using card details directly on the website to pay and could instead be using PayPal. Another improvement would be to link the booking cost to the prices given in room prices, potentially based on date, so that it does not need re-entered as this can result in anomalies due to error. Another improvement to the booking section would be to allow for a many to many relationship for rooms and bookings as bookings should be able to allow for multiple rooms, and rooms can be booked many times. This can be done through an extra table to link booking and rooms, and removing room ID from the booking table then. Another improvement for booking would be to link the customer first name of the booking details to the customer table as this would allow for less anomalies to occur in such an event as a name change. Another change relating to booking, but also the shared tables, would be to add the booking ID as a necessary part of a review, as this would help to ensure that the user has actually stayed at an accommodation before leaving a review, helping to stop fake reviews being added to an accommodation on Hotels.com. The review first name column should also link to the customer table first name detail as this will again stop anomalies occurring with name changes or misspellings and avoids unnecessary extra storage of the name.

One other improvement would be to add a lot more sample data to the database when testing, so as to better test the databases capacity when holding a greater volume of data, such as more accommodations so as to show off a greater number of accommodations being booked when showing off the database in the video, or more bookings in different times, to show how the data would be displayed when the check in and check out dates vary more.

Another improvement would be to add greater encryption and password storage, however these are not possible with the current scope of the project as it is entirely focused on the database and not other programming that would be involved in such a situation.

Conclusion

The overall structure of the database works to achieve the goals stated of allowing users to book accommodation, view the accommodation details, set up their own accommodations on the website and to leave reviews for these accommodations on Hotels.com. This was through different types of variables, primary keys and foreign keys, as was talked about in this report. This was shown through the queries used in the appendix, as each goal is achieved. It has also allowed for the accommodations to hold multiple rooms which are in turn what was booked, rather than the accommodation itself. There are improvements that could be made as well, as given above in the improvements section, however these are broadly normal form issues and quality of life improvements for the users, and the database still functions as the back end of Hotels.com, mostly conforming to the normal forms and to Chen's theory on entity-relationships. The data integrity is also maintained through the usage of the foreign key constraints throughout the project.

Bibliography

[1] P. P.-S. Chen, "The entity-relationship model - toward a unified view of data," *ACM Transactions on Database Systems,* vol. 1, no. 1, p. 1, March 1976.

[2] W. Kent, "A simple guide to five normal forms in relational database theory," *Communications of the ACM,* vol. 26, no. 2, pp. 120-125, February 1983.

Table of figures

Figure 1: initial excel tables of data taken from Hotels.com 2
Figure 2: initial group ERD 3
Figure 3: Final group ERD..... 3
Figure 4: Final ERD diagram reached individually..... 4

Appendix

SQL queries

1. Wants a normal hotel with wheelchair access in New York with conference rooms

```
SELECT accommodation.name, accessibility.accessibility_feature, accommodation_type.accommodation_type,
accommodation_theme.theme, facility.facility_name
FROM city
LEFT JOIN accommodation_address
ON city.city_ID = accommodation_address.city_ID
LEFT JOIN accommodation
ON accommodation_address.accommodation_address_ID=accommodation.accommodation_address_ID
LEFT JOIN accommodation_type
ON accommodation.accommodation_type_ID = accommodation_type.accommodation_type_ID
LEFT JOIN accommodation_accessibility
ON accommodation.accommodation_ID = accommodation_accessibility.accommodation_ID
LEFT JOIN accessibility
ON accommodation_accessibility.accessibility_ID = accessibility.accessibility_ID
LEFT JOIN accommodation_theme
ON accommodation.accommodation_theme_ID = accommodation_theme.accommodation_theme_id
LEFT JOIN accommodation_facility
ON accommodation.accommodation_ID = accommodation_facility.accommodation_ID
LEFT JOIN facility
ON accommodation_facility.facility_ID = facility.facility_ID
WHERE accessibility.accessibility_feature LIKE '%wheelchair%' AND city.city_name = 'New York' AND
accommodation_type.accommodation_type = 'Hotel' AND accommodation_theme.theme = 'normal' AND
facility.facility_name LIKE '%conference%';
```

2. Count bookings made at this hotel the booking will be at in that time and the number of rooms there:

```
SELECT

(SELECT COUNT(room_ID) FROM room WHERE room.accommodation_ID = '2') AS HiltonGardenInnRooms,

(SELECT COUNT(booking_ID) FROM room
```

```
LEFT JOIN booking
ON room.room_id = booking.room_ID
LEFT JOIN booking_details
ON booking.booking_details_id = booking_details.booking_details_ID
WHERE accommodation_ID = 2 AND check_in_date BETWEEN '2021-11-23' AND '2021-11-24') AS
HiltonGardenInnBookings
```

3. Get details on the rooms and show more detail on those booked for system side:

```
SELECT
room.room_id, room.accommodation_ID, room.room_name,
booking.booking_ID, booking.customer_ID, booking.booking_reference_number, booking.booking_details_id,
booking.booking_status_ID,
booking_details.check_in_date, booking_details.check_out_date
FROM room
LEFT JOIN booking ON room.room_id = booking.room_ID
LEFT JOIN booking_details ON booking.booking_details_id = booking_details.booking_details_ID
WHERE accommodation_ID = 2 AND check_in_date BETWEEN '2021-11-23' AND '2021-11-24';
```

4. Booking details in which he has two cards details saved and chooses wrong one so goes back and changes it while booking after deciding to put it on debit and not credit:

```
START TRANSACTION;

INSERT INTO `booking_details` (`booking_details_ID`, `guest_first_name`, `guest_last_name`, `number_adults`,
`number_children`, `children_age`, `check_in_date`, `check_out_date`, `late_checkout`,
`special_request_check`, `special_requests`)
VALUES (NULL, 'Jason', 'Voorhees', '2', '2', '1, 4', '2021-11-24', '2021-11-26', '1', '1', 'Need a cot for youngest
child');
SET @last_ID_in_booking_details = LAST_INSERT_ID();

SAVEPOINT SAVEPOINT1;

INSERT INTO `booking_payment` (`booking_payment_ID`, `payment_type_ID`, `payment_ID`, `currency_ID`)
VALUES (NULL, '1', '12', '2');
SET @last_ID_in_booking_payment = LAST_INSERT_ID();

ROLLBACK TO SAVEPOINT1;

INSERT INTO `booking_payment` (`booking_payment_ID`, `payment_type_ID`, `payment_ID`, `currency_ID`)
VALUES (NULL, '2', '3', '2');
SET @last_ID_in_booking_payment = LAST_INSERT_ID();

INSERT INTO `booking` (`booking_ID`, `customer_ID`, `room_ID`, `booking_reference_number`,
`booking_date`, `booking_cost`, `booking_payment_ID`, `rewards_points`, `booking_status_ID`,
`booking_details_id`)
VALUES (NULL, '12', '5', '25984136', '2021-11-16', '404', @last_ID_in_booking_payment, '2', '1',
@last_ID_in_booking_details);

COMMIT;
```

5. Booking count at hotel after:

```
SELECT
```

```
(SELECT COUNT(room_ID) FROM room WHERE room.accommodation_ID = '2') AS HiltonGardenInnRooms,

(SELECT COUNT(booking_ID) FROM room
LEFT JOIN booking
ON room.room_id = booking.room_ID
LEFT JOIN booking_details
ON booking.booking_details_id = booking_details.booking_details_ID
WHERE accommodation_ID = 2 AND check_in_date BETWEEN '2021-11-23' AND '2021-11-24') AS
HiltonGardenInnBookings
```

6. Show new booking made in detail:

```
SELECT
room.room_id, room.accommodation_ID, room.room_name,
booking.booking_ID, booking.customer_ID, booking.booking_reference_number, booking.booking_details_id,
booking.booking_status_ID,
booking_details.check_in_date, booking_details.check_out_date
FROM room
LEFT JOIN booking ON room.room_id = booking.room_ID
LEFT JOIN booking_details ON booking.booking_details_id = booking_details.booking_details_ID
WHERE accommodation_id = 2 AND check_in_date BETWEEN '2021-11-23' AND '2021-11-24';
```

7. Show how much this has made the hotel:

```
SELECT SUM(booking_cost)
AS TotalMadeHiltonGardenInnNov24thToNov26th
FROM room
LEFT JOIN booking
ON room.room_id = booking.room_ID
LEFT JOIN booking_details
ON booking.booking_details_id = booking_details.booking_details_ID
WHERE accommodation_id = 2 AND check_in_date BETWEEN '2021-11-23' AND '2021-11-24';
```

8. Look at sights to see in the same city

```
SELECT
city.city_name,
attraction.attraction_name, attraction.attraction_description
FROM attraction
RIGHT JOIN city
ON attraction.city_ID  = city.city_ID
WHERE city_name = 'New York';
```

9. Decides to leave review but changes mind after receiving an unexpected bill from the hotel

```
START TRANSACTION;

INSERT INTO `customer_favourites` (`customer_favourite_ID`, `customer_ID`, `accommodation_ID`) VALUES
(NULL, '12', '2');
SET @last_favourite_id = LAST_INSERT_ID();

SAVEPOINT SAVEPOINT1;

INSERT INTO `reviews` (`review_ID`, `customer_ID`, `accommodation_ID`, `rating`, `comment`, `date`,
`first_name`) VALUES (NULL, '12', '2', '8', 'Very nice hotel, spacious as required', '2021-11-30', 'Jason');
```

```
ROLLBACK TO SAVEPOINT1;

DELETE FROM `customer_favourites` WHERE customer_favourite_ID = @last_favourite_ID;

INSERT INTO `reviews` (`review_ID`, `customer_ID`, `accommodation_ID`, `rating`, `comment`, `date`,
`first_name`) VALUES (NULL, '12', '2', '5', 'Alright hotel, spacious as required', '2021-11-30', 'Jason');

ROLLBACK;
```

10. Someone creating new account and goes through with a secure password

```
START TRANSACTION;

SET @member_ID = '46513198';
SET @username = 'johnnyboy413';
SET @password = 'outofbreath';

SELECT @salt := SUBSTRING(SHA1(RAND()), 1, 6);
SELECT @saltedHash := SHA1(CONCAT(@salt, @password)) AS salted_hash_value;
SELECT @storedSaltedHash := CONCAT(@salt,@saltedHash) AS password_to_be_stored;

INSERT INTO customer_account (member_id, username, password)
VALUES (@member_ID, @username, @storedSaltedHash);

INSERT INTO `customer_address` (`customer_address_ID`, `building_number`, `building_name`,
`address_line_1`, `address_line_2`, `city_ID`, `postcode/zip_code`) VALUES (NULL, '100', NULL, 'Stranmillis
Lane', NULL, '20', 'BT9 7ER');
SET @last_customer_address_ID = LAST_INSERT_ID();

INSERT INTO `customer` (`customer_id`, `member_id`, `first_name`, `last_name`, `customer_address_id`,
`phone_number`, `email`, `communication_preference_id`) VALUES (NULL, @member_ID, 'John', 'Egbert',
@last_customer_address_ID, '07584631255', 'heirofwind@hotmail.com', '4');

COMMIT;
```

11. Login then comes round and John has to enter details

```
SET @username = 'johnnyboy413';
SET @password = 'outofbreath';

SELECT @saltInUse := SUBSTRING(password, 1, 6) FROM customer_account WHERE username = @username;

SELECT @storedSaltedHashInUse := SUBSTRING(password, 7, 46) FROM customer_account WHERE username
= @username;

SELECT @saltedHash := SHA1(CONCAT(@saltInUse, @password)) AS salted_hash_value_login;
```

12. Card encryption

```
SET @customer_ID = '12';
SET @card_number = '4020658952146325';
SET @exp_date = '10/23';
SET @CVV = '564';
SET @card_provider_ID = '2';
```

```
SET @password = 'Johnscard' ;
SET @card_number = AES_ENCRYPT(@card_number, @password);

INSERT INTO `payment_details` (`payment_ID`, `customer_ID`, `card_number`, `exp_date`, `CVV`,
`card_provider_id`)
VALUES (NULL, @customer_ID, @card_number, @exp_date, @CVV, @card_provider_ID);
```

13. Showing card details

```
SET @password = 'Johnscard';

SELECT payment_id, customer_ID, AES_DECRYPT(card_number, @password) AS card_number, exp_date,
card_provider_id
FROM payment_details WHERE payment_ID = 13;
```

14. Searching for accommodation owned by user

```
SELECT
accommodation.name, supplier.supplier_name, customer.first_name, customer.last_name
FROM accommodation
INNER JOIN supplier
ON accommodation.supplier_ID = supplier.supplier_ID
INNER JOIN customer
ON supplier.customer_ID = customer.customer_ID
WHERE customer.customer_ID = 13;
```

15. Adding in accommodation details

```
START TRANSACTION;

INSERT INTO ‘supplier’(‘supplier_ID’,’customer_ID’,’supplier_name’) VALUES (NULL, ‘20’,’John\s B&Bs’);
SET @last_supplier_ID = LAST_INSERT_ID();

INSERT INTO `accommodation_address` (`accommodation_address_ID`, `building_number`, `building_name`,
`address_line_1`, `address_line_2`, `city_ID`, `postcode/zip_code`, `latitude`, `longitude`) VALUES (NULL, '125',
NULL, 'Malone Avenue', NULL, '20', 'BT9 8FR', NULL, NULL);
SET @last_accommodation_address_ID = LAST_INSERT_ID();

INSERT INTO `accommodation_address` (`accommodation_address_ID`, `building_number`, `building_name`,
`address_line_1`, `address_line_2`, `city_ID`, `postcode/zip_code`, `latitude`, `longitude`) VALUES (NULL, '150',
NULL, ‘Dublin Road', NULL, '20', 'BT8 7YT', NULL, NULL);
SET @last_accommodation_address_ID_2 = LAST_INSERT_ID();

INSERT INTO `accommodation` (`accommodation_ID`, `accommodation_type_ID`, `supplier_ID`, `name`,
`accommodation_address_ID`, `description`, `star_rating`, `accommodation_theme_ID`, `covid_policy_ID`,
`FAQs`) VALUES (NULL, '3', @last_supplier_ID, 'John\'s B&B on Malone', 'last_accommodation_address_ID',
'John\'s B&B on Malone', '5', '2', '2', 'Is there any parking available?\r\nYes there\'s a parking space set aside
for customers');
SET @last_accommodation_ID = LAST_INSERT_ID();

INSERT INTO `accommodation` (`accommodation_ID`, `accommodation_type_ID`, `supplier_ID`, `name`,
`accommodation_address_ID`, `description`, `star_rating`, `accommodation_theme_ID`, `covid_policy_ID`,
`FAQs`) VALUES (NULL, '3', @last_supplier_ID_2, 'John\'s B&B on Malone',
'last_accommodation_address_ID_2', 'John\'s B&B on Dublin Road', '5', '2', '2', 'Is there a parking space?\r\n
Unfortunately not, sorry');
```

```
SET @last_accommodation_ID_2 = LAST_INSERT_ID();

INSERT INTO `accommodation_image` (`accommodation_image_ID`, `accommodation_ID`, `image_ID`)
VALUES (NULL, @last_accommodation_ID, '1'), (NULL, @last_accommodation_ID, '2');
INSERT INTO `accommodation_image` (`accommodation_image_ID`, `accommodation_ID`, `image_ID`)
VALUES (NULL, @last_accommodation_ID_2, '1'), (NULL, @last_accommodation_ID_2, '2');

INSERT INTO `accommodation_facility` (`accommodation_facility_ID`, `accommodation_ID`, `facility_ID`)
VALUES (NULL, @last_accommodation_ID, '18'), (NULL, @last_accommodation_ID, '8');
INSERT INTO `accommodation_facility` (`accommodation_facility_ID`, `accommodation_ID`, `facility_ID`)
VALUES (NULL, @last_accommodation_ID_2, '18'), (NULL, @last_accommodation_ID_2, '8');

INSERT INTO `accommodation_accessibility` (`accommodation_access_ID`, `accommodation_ID`,
`accessibility_ID`) VALUES (NULL, @last_accommodation_ID, '2'), (NULL, @last_accommodation_ID, '3');
INSERT INTO `accommodation_accessibility` (`accommodation_access_ID`, `accommodation_ID`,
`accessibility_ID`) VALUES (NULL, @last_accommodation_ID_2, '2'), (NULL, @last_accommodation_ID_2, '3');

COMMIT;
```