QUEEN'S UNIVERSITY BELFAST

School of Electronics, Electrical Engineering and Computer Science

Database Design for Booking.com

David Grech

22nd November 2019

Database Design

Entity Relationship Diagram

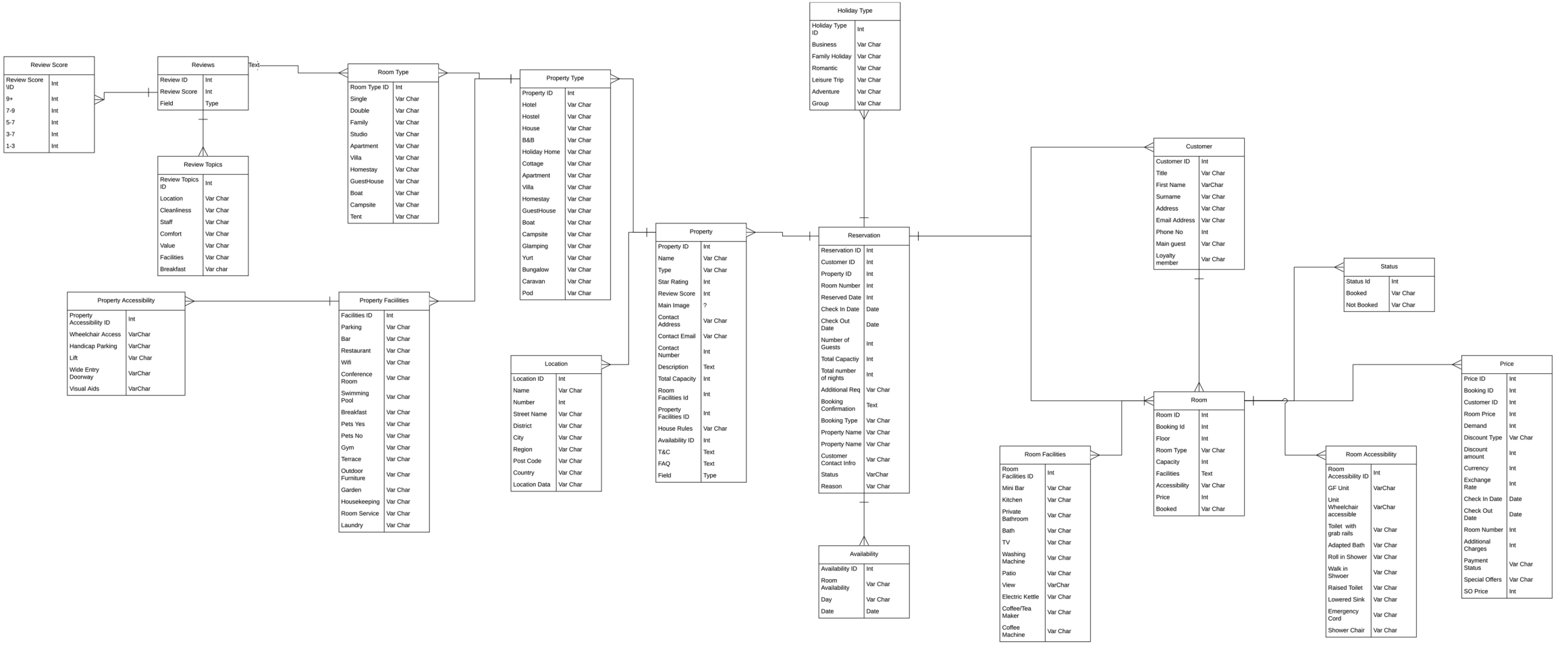
The main purpose of Booking.com is for users to find a hotel, book a room and pay money. We began entity discovery with what a user does, see Figure 1. Once we discovered the main entities, we looked at their attributes and relationships between entities. Figure 2 shows our group's first iteration of the ER diagram. The heading of each table is an entity. The list below are its attributes. The connecting lines represent the type of relationships between entities.

Figure 1: What a user does to book a hotel on Booking.com.

- Search a hotel
  - Select a location
  - Choose dates
  - Number of travellers
  - Business
- Filter initial results
  - Budget
  - Location score
  - Star rating
  - Distance from city centre
  - Activities
  - Check availability
  - Breakfast
  - Property Type
  - Review score
  - District
  - Chain
  - Accessibility
- Once property chosen - select room type
  - Double, single, twin family etc
  - Payment options
    - No Breakfast, Prepay (Non-refundable)
    - No Breakfast, Free Cancellation, No Prepay
    - Breakfast, Free Cancellation, No Prepay
  - Reserve
- Rate/ review a property
  - Leave a review
  - Rate the hotel
  - Tag
    - Business/pleasure
    - Solo/couple/family/group
    - Length of stay

Entity Relationship Diagram for Booking.com Database Version 1

Figure 2: First ER Diagram.



When experimentally adding data to the tables we had repeating data. In the Location table, the columns Country and City have repeating data. To normalise this, we added two more tables: Country and City, see the final ER diagram in Figure 3.

Now each city has its own City_ID that can be used in the Street_Address table. Each row in the City table is now unique in the database with its own unique identifier, the City_ID. It can be used as a foreign key by any other table in the database. City_ID is auto incremented in the City table to ensure every row will have a unique numeric identifier that increases by 1 integer upon each new row addition. The Street_Address table can now use City_ID as a foreign key, in its own table. The primary key is City_ID in the City table and the foreign key is City_ID in the Street_Address table.

The two tables now have a relationship based on the foreign key found in the Street_Address table. City_ID equal to 1 is Bournemouth and can now appear in the Street_Address table multiple times as the integer 1. It is not repeating data, but a reference to the relationship with the City table.

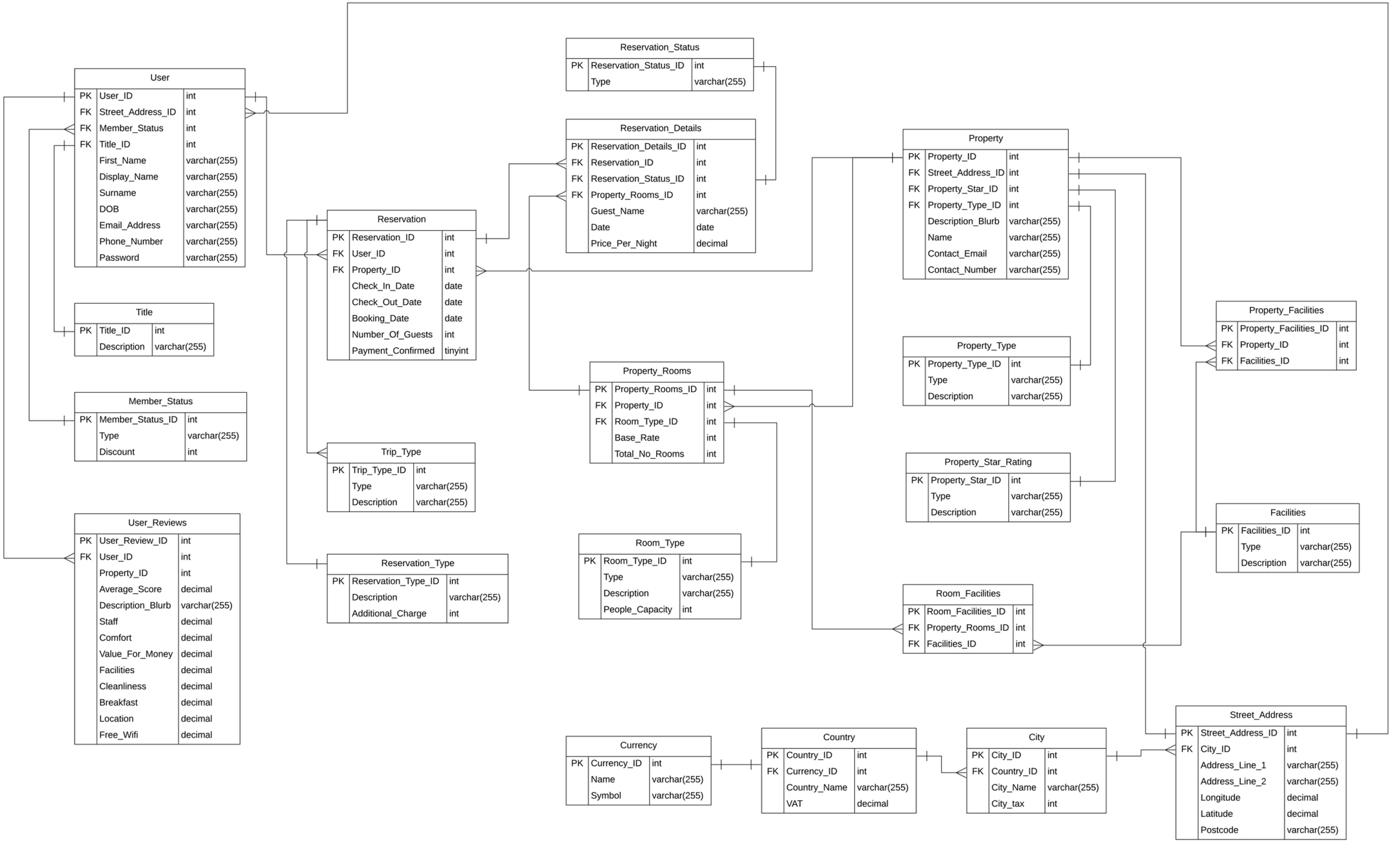
The reason we normalise data into separate tables is to have a definite database design that will not need to change. If new data is introduced, it will always be able to fit into the database design, no matter what future changes happen. Each row in the ER diagram is atomic, it is a unique element in the whole database with unique characteristics.

In the final ER diagram the Property_Facilities table has been normalised out into two different tables: the Property_Facilities table and the Facilities table, see Figure 3. In the first ER diagram a long list of facility columns would extend out to the right. A long list of hotel rows would extend below. Some hotels would not have all the facilities, as a result some entries would be null. The facilities extending out to the right makes the table more difficult to read. If the data is put into the two tables found in the final ER diagram, there is no longer a need to have null entries and the table is more readable, because you can scroll down through the data. The facilities table holds all the facilities there are, stored as unique rows with unique primary keys. The Property_Facilities table calls on the Facilities table to allocate the relevant facilities to each hotel.

In the final ER diagram, the lines between tables represent relationships. The Currency to Country table is a one to one relationship. There is one currency for one country. The Country to City table is a one to many relationship. There is one Country to many cities.

Entity Relationship Diagram for Booking.com Database Final Version

Figure 3: Final ER Diagram.



Availability

Availability is an essential part of the booking process. In the first ER Diagram, we decided the main attributes of availability would be Day, Date and Room_Availability, see Figure 4. In this model every room in every hotel has its own availability table. All dates for the next year and a half are plotted in the table. When someone makes a booking, on a certain date, Room_Availability goes down by one. Once it reaches zero there are no more rooms available on that date. There are many improvements that can be made to this model. Firstly, it is very cumbersome to have a table for every room type in every hotel. This would take extra time for the people working on the database to navigate. Secondly, the dates when someone made a booking can be found in the reservation table. This means there is duplicate data stores in the database. Finally, the day column has repeating data and could be normalised out.

Figure 4: First Availability Model (Positive).

Availability_ID	Room_Availability	Date	Day
1	12	13/10/2019	Sunday
2	12	14/10/2019	Monday
3	12	15/10/2019	Tuesday
4	12	16/10/2019	Wednesday
5	12	17/10/2019	Thursday
6	12	18/10/2019	Friday
7	12	19/10/2019	Saturday
8	12	20/10/2019	Sunday
9	12	21/10/2019	Monday

In contrast, the negative availability model only adds data when a booking is made, see Figure 5. I experimented with and created the negative model for our group to improve on the first version. The negative model is a more practical and less time-consuming design because all dates do not have to be added and every room in every hotel does not need a table: each hotel only needs one table.

Figure 5: Negative Availability Model.

Property_ID	Room_Type_ID	Day_Date	Total_No_Rooms	Rooms_Booked
7	2	Sunday 13/10/2019	12	1

The room is identified with a combination of the Property_ID and Room_Type_ID columns. When a room is booked the Rooms_Booked column goes up by one. This column could be taken out and the number of bookings could be queried from the Reservation table. Once Rooms_Booked reaches the Total_No_Rooms upper limit no more rooms can be booked. The number of rooms available is a derived attribute. It is derived from calculating Total_No_Rooms – Rooms_Booked = No_Rooms_Available. The number of rooms available does not have to be stored in the database. It can be calculated on the front end of the website and presented to the user.

Our group decided to use the negative model I created. The Availability table doesn't need to appear in the ER diagram because all information can be queried from the Property_Rooms table and the Reservation_Details table.

To improve on the design the Date column could be updated to Day_Date and a new column called Business_Day added, see Figure 6. Then the booking date could be matched with business day, or no business day. This is helpful for pricing calculations. Business days could be weekends, special events or seasonal.

Figure 6: Availability model with Day_Date and Business_Day.

Property_ID	Room_Type_ID	Day_Date	Total_No_Rooms	Rooms_Booked	Business_Day
7	2	Sunday 13/10/2019	12	1	Yes

Assumptions Made

The model assumes all hotels participate in the Genius schemes. It is assumed that all customers pay in their own currency and no currency conversions have been done. For pricing we are not including any extras such as breakfast, late check outs etc. We assumed that users would pay via a third-party payment provider.

Data Types

Primary keys are always integers. Any whole number up to 2,147,483,647 can be stored as an int: this is suitable for every int in the database model. For Payment_Confirmed the data type is tiny int. It is used like a Boolean: 1 represents paid and 0 is not paid. The decimal data type is used for any floating-point numbers,

the number of decimal places is specified for User_Reviews as (15,1), the one denotes one decimal place.

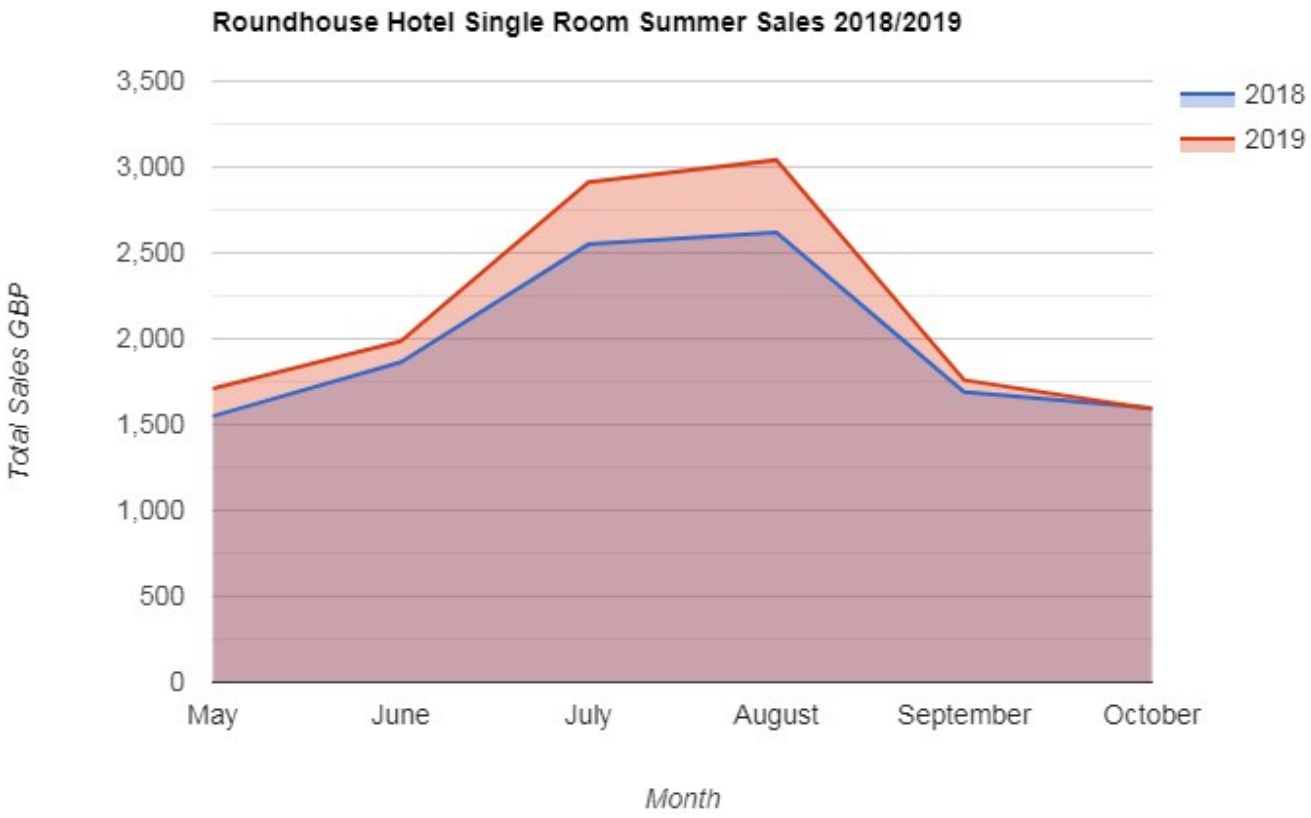
Varchar(255) allows the data store to hold 255 characters. This is enough characters to be able to write a short description. For Phone_Number, DOB and Currency varchar(28) is used. These data stores will not need more than 28 characters. Reducing the character limit reduces the amount of memory allocation. In modern computer systems this is almost never a problem, because of very large memory capacities, although in exceptionally rare cases small scale low power devices may benefit from having low memory allocation.

Microbial fuel cells (MFCs) produce a very low amount of electricity from bacteria that produce and donate electrons. MFCs have been used to power deep sea sensors on a system with 3kb of SRAM (Lee et al., 2015). The length of data types is relevant when working with such a low amount of memory, especially if the system was modified to carry out more complex functions.

Business Intelligence

Bournemouth is a seaside town with nine miles of sandy beaches. The most amount of room bookings is in the summer. Single room summer sales for the Roundhouse Hotel Bournemouth are plotted in the graph below, see Figure 7. Business professionals can use the graph to predict how much extra staff they need to employ over the next summer period and they can work out the profit they have made in years 2018/2019. They could use the graph to predict future sales growth. Once profits are worked out the business could allocate the money for other investments.

Figure 7: Roundhouse Hotel Single Room Summer Sales 2018/2019.



The command below shows total single room sales for May 2018, see Figure 8. It could be useful to see different room type sales compared with each other to see which room type is most popular and to predict room type demand next summer. For example, if double room demand is higher, single rooms can be converted to double rooms.

The command can be repeated for all room types that month to see total sales for the hotel in May 2018. This can be repeated for May to October, 2018/2019, to produce the data for the graph in Figure 7.

Figure 8: January 2018 single room sales for the Roundhouse Hotel.

```
SELECT Property_Rooms_ID, SUM(Price_Per_Night)
FROM Reservation_Details
WHERE Property_Rooms_ID = '2'
AND Date
BETWEEN '2018-05-01' AND '2018-05-31'
```

Transactions

Transactions allow queries to be completed before they are committed to disk. This is especially useful when carrying out money transactions to ensure no financial data is lost. For calculating Genius membership discount and city tax I used the transaction in Figure 9. Membership_Status.Discount is selected and stored in the @discount variable. This process is repeated to create @city_tax and @base_rate variables.

Genius membership discounts are stored in the database as decimals 0.10 and 0.25. This is easily readable for people working on the database in the future. 10% Genius discount is worked out by multiplying the base rate by (1 - 0.10). 20% City tax is stored in the same way and calculated in the same way.

Once all values are stored in variables it is useful to have a save point. At this stage If the booking is not confirmed the front end of the website will trigger a rollback and the data will not be stored in the database. If the booking is confirmed the price will update and be committed.

Figure 9: Pricing transaction.

```
START TRANSACTION;

SELECT Member_Status.Discount INTO @discount FROM User INNER JOIN Member_Status ON
User.User_ID = Member_Status.Member_Status_ID Where User_ID = '1';

SELECT City_tax INTO @city_tax FROM City WHERE City_Name = 'Bournemouth';

SELECT Base_Rate INTO @base_rate FROM Property_Rooms WHERE Property_Rooms_ID = '2';

SAVEPOINT  Before_Update;

UPDATE `Reservation_Details`
SET `Price_Per_Night` = @base_rate*(1+@city_tax)*(1-@discount)
WHERE `Reservation_Details`.`Reservation_Details_ID` = 1;

#ROLLBACK TO Before_Update;

COMMIT;
```

Cyber Security

Passwords stored in a database must not be readable in case the database is compromised. If the database is compromised and the passwords are sufficiently encrypted the passwords may remain unreadable. The password can be hashed using the message-digest algorithm (MD5), although this is now obsolete because the majority of possible hashes have been decoded and are available online. A more secure way to protect passwords is to encrypt it with a key. Advanced Encryption Standard (AES) uses a secret key to encrypt a password and requires this key to decrypt it.

Payment Card Industry Data Security Standards (PCI DSS) are a set of standards set by the PCI Security Standards Council (PCI SSC) to protect credit card data during transactions. There are a strict set of rules that all major credit card providers must meet in order to achieve PCI compliance. The rules include personal data and password encryption, having secure networks to avoid a data breach, that all computer systems that manage card holder data are secure, that card holder data is sufficiently restricted to those accessing the database and that these requirements are maintained and updated regularly (Ataya 2010). The Booking.com website must follow all PCI compliance regulations in order to have secure as possible transactions for its customers.

References

Ataya, G., 2010. PCI DSS audit and compliance. *Information Security Technical Report*, 15(4), 138-144.

Lee, I., Kim, G., Bang, S., Wolfe, A., Bell, R., Jeong, S., Kim, Y., Kagan, J., Arias-Thode, M., Chadwick, B., Sylvester, D., Blaauw, D. and Lee, Y., 2015. System-On-Mud: Ultra-Low Power Oceanic Sensing Platform Powered by Small-Scale Benthic Microbial Fuel Cells. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(4), 1-10.

Appendix A: Availability Command

Property_ID	Property_Rooms_ID	Total_No_Rooms	COUNT(Date)
1	2	8	4

Displays Property_ID, Property_Rooms_ID, the total number of rooms available and the number of rooms already booked for a chosen room.

```
Select Property_Rooms.Property_ID, Reservation_Details.Property_Rooms_ID,
Property_Rooms.Total_No_Rooms, COUNT(Date)
FROM Property_Rooms
INNER JOIN Reservation_Details
ON Property_Rooms.Property_Rooms_ID = Reservation_Details.Property_Rooms_ID
WHERE Date = '2020-01-13'
AND Property_ID = '1'
AND Property_Rooms.Property_Rooms_ID = '2'
```

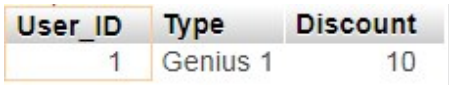
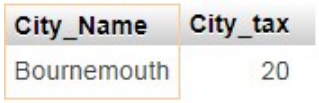
Appendix B: Room Pricing Transaction

Reservation_Details_ID	Price_Per_Night
1	55.08

Result of transaction with 20% tax and a 10% Genius discount.

```
START TRANSACTION;

SELECT Member_Status.Discount INTO @discount FROM User INNER JOIN Member_Status ON
User.User_ID = Member_Status.Member_Status_ID Where User_ID = '1';

SELECT City_tax INTO @city_tax FROM City WHERE City_Name = 'Bournemouth';

SELECT Base_Rate INTO @base_rate FROM Property_Rooms WHERE Property_Rooms_ID = '2';


SAVEPOINT  Before_Update;

UPDATE `Reservation_Details`
SET `Price_Per_Night` = @base_rate*(1+@city_tax)*(1-@discount)
WHERE `Reservation_Details`.`Reservation_Details_ID` = 1;

#ROLLBACK TO Before_Update;

COMMIT;
```

Appendix C: Room Pricing Commands

 Finds user membership type.	Select User.User_ID, Member_Status.Type, Member_Status.Discount FROM User INNER JOIN Member_Status ON User.User_ID = Member_Status.Member_Status_ID WHERE User_ID = '1'
 Finds Bournemouth city tax.	SELECT City_Name, City_tax FROM City WHERE City_Name = 'Bournemouth'
 Finds Base Rate of a chosen room.	SELECT Property_Rooms_ID, Base_Rate FROM Property_Rooms WHERE Property_Rooms_ID = '2'
 Update Price_Per_Night with 20% tax and a 10% Genius discount.	UPDATE `Reservation_Details` SET `Price_Per_Night` = '51'*1.20*0.90 WHERE `Reservation_Details`.`Reservation_Details_ID` = 1

Appendix D: User Review Command

 Gives average of all user reviews for Breakfast and Wi-Fi for chosen property.	
SELECT Property_ID, SUM(Breakfast)/COUNT(Breakfast), SUM(Free_Wifi)/COUNT(Free_Wifi) FROM User_Reviews WHERE Property_ID = '1'	

Appendix E: Facilities Command

Type	Name
Wifi	Bournemouth Highcliff Marriott Hotel
Breakfast	Bournemouth Highcliff Marriott Hotel
Room service	Bournemouth Highcliff Marriott Hotel
Air conditioning	Bournemouth Highcliff Marriott Hotel

Displays Hotels with chosen facilities.

```
SELECT Facilities.Type, Property.Name
FROM Facilities
INNER JOIN Property_Facilities
ON Facilities.Facilities_ID = Property_Facilities.Facilities_ID
INNER JOIN Property
ON Property_Facilities.Property_ID = Property.Property_ID
WHERE Facilities.Type = 'Air conditioning'
OR Facilities.Type = 'Wifi'
OR Facilities.Type = 'Breakfast'
OR Facilities.Type = 'Room Service'
```

Appendix F: Hotel Address Command

Name	Address_Line_1	Address_Line_2	City_Name	Country_Name
Roundhouse Hotel	1 Meyrick Rd	Lansdowne	Bournemouth	United Kingdom

Inner join on tables Property, Street_Address, City and Country to return Hotel name, Address line 1 and 2, City and Country.

```
SELECT Property.Name, Street_Address.Address_Line_1, Street_Address.Address_Line_2,
City.City_Name, Country.Country_Name
FROM Property
INNER JOIN Street_Address
ON Property.Street_Address_ID = Street_Address.Street_Address_ID
INNER JOIN City
ON Street_Address.City_ID = City.City_ID
INNER JOIN Country
ON City.Country_ID = Country.Country_ID
WHERE Property.Name = 'Roundhouse Hotel'
```

Appendix G: Business Intelligence Graph Commands:

Property_Rooms_ID	SUM(Price_Per_Night)
2	256.00

Command to display the sum of current sales for all single rooms in January 2020 in a chosen hotel.

```
SELECT Property_Rooms_ID, SUM(Price_Per_Night)
FROM Reservation_Details
WHERE Property_Rooms_ID = '2'
AND Date
BETWEEN '2020-01-01' AND '2020-01-31'
```

Appendix H: AES Password Encryption

Command to insert AES encrypted password.

```
INSERT into User (Password) VALUES (AES_ENCRYPT('Echo77', 'SecretKey'))
```

Appendix I: AES Password Decryption

Command to decrypt AES encrypted password.

```
SELECT AES_DECRYPT(Password, 'SecretKey') from User
```