

Data Management

IB9PHo

MSBA Group 32

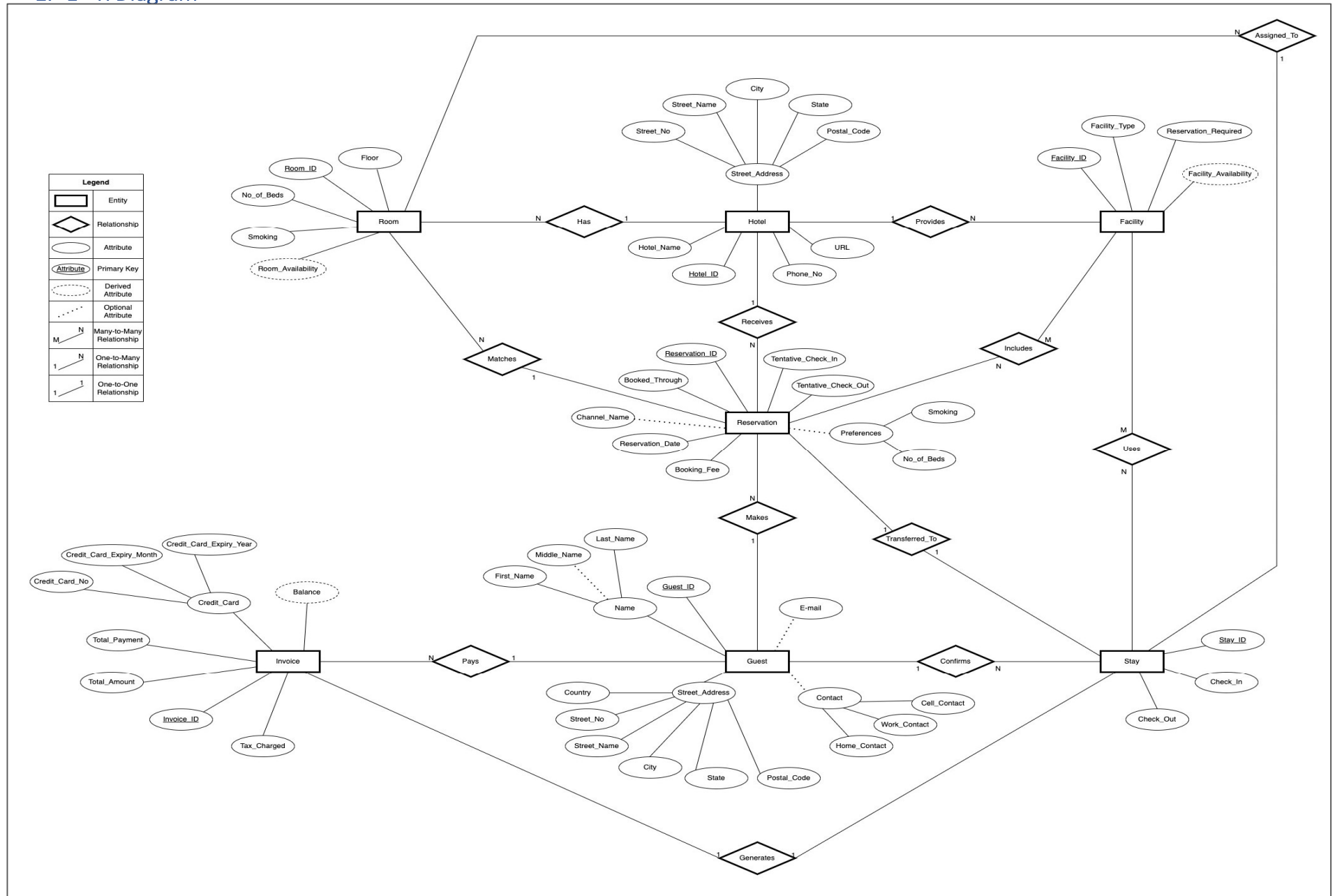
Contents

PART A1	4
1. E - R DIAGRAM	4
1.1 ASSUMPTIONS	5
1.2 ENTITIES AND ATTRIBUTES	5
1.2.1 Hotel	5
1.2.2 Guest	5
1.2.3 Facility	5
1.2.4 Reservation	6
1.2.5 Stay	6
1.2.6 Room	6
1.2.7 Invoice	6
1.3 JUNCTION TABLES	6
1.4 ENTITIES - RELATIONSHIP AND CARDINALITIES	7
1.4.1 Hotel and Room - Has	7
1.4.2 Hotel and Facility - Provides	7
1.4.3 Hotel and Reservation - Receives	7
1.4.4 Guest and Reservation - Makes	8
1.4.5 Reservation and Room - Matches	8
1.4.6 Reservation and Facility - Includes	8
1.4.7 Reservation and Stay - Transferred To	9
1.4.8 Guest and Stay - Confirms	9
1.4.9 Room and Stay - Assigned To	9
1.4.10 Stay and Facility - Uses	9
1.4.11 Stay and Invoice - Generates	10
1.4.12 Guest and Invoice - Pays	10
2. SQL	10
2.1 CREATING DATABASE TABLES - DDL QUERIES	10
2.2 QUERYING THE DATABASE	12
PART A2	15
1. E - R DIAGRAM	15
1.1 ASSUMPTIONS	16
1.2 ENTITIES AND ATTRIBUTES	16
1.2.1 Dealer	16
1.2.2 Vehicle	16
1.2.3 Customer	16
1.2.4 Servicing	16
1.3 JUNCTION TABLES	17
1.4 ENTITIES - RELATIONSHIP AND CARDINALITIES	17
2. SQL	18
2.1 Creating Database Tables - DDL queries	18
2.2 Querying the Database	19
PART D	21
1.1 DATA TAB	21
1.2 SCENARIO TAB	22
1.2.1 Scenario 1	22
1.2.2 Scenario 2	23
1.2.3 Scenario 3	24
APPENDIX	25

PART B – R CODE	25
PART C – R CODE	28
PART D – R CODE	30
1.1 Data Cleaning and SQL Queries	30
1.3 Shiny Dashboard Creation	31

PART A1

1. E - R Diagram



The objective of the Hotel Management System described above is to optimize the revenue of the WNB Hotel Group, a rapidly growing hotel chain.

1.1 Assumptions

- Facilities are assumed to be varied, from wedding hall and spas to rental phones, and all hotels are assumed to have facilities.
- While making a reservation, it is assumed that the guest can book the room without reserving any facility. Likewise, a facility, like spa or wedding hall, can be booked without booking a room.
- It is assumed that the contact phone numbers of guests do not include '+’.
- All Facility IDs and Room IDs are assumed to be unique in the DBMS, even across various hotels.
- During the stay of the guest in the hotel, the stay information is stored in a temporarily database. Once the temporary tab is closed, the data is pushed into 'Stay', the historical database.
- After the reservation is made, the data is transferred to the temporary database. In case the guest does not check in, the entry is considered cancelled & the check in & out dates are populated as 'NA'.
- Each 'Reservation_ID' corresponds to one room. Hence, if a guest makes reservation for multiple rooms during a stay, there will be multiple 'Reservation_ID's for that guest in the database.
- If a guest checks in the hotel, but the preferred room is temporarily unavailable, he is assigned a different room temporarily, and is eventually provided another room which matches the preferences. Hence, multiple rooms can be tagged to a single stay instance.
- The invoice is generated per 'Room_ID'. Hence, if multiple rooms are booked under the same 'Guest_ID', multiple invoices will be generated.

1.2 Entities and Attributes

The section describes different entities shown in the E-R diagram above.

1.2.1 Hotel

- The 'Hotel' database stores information related to all hotels such as the ID, name, phone number, URL & address.
- Each hotel has a unique 'Hotel_ID', which acts as a primary key.

Hotel									
Attribute	Hotel_ID	Hotel_Name	Phone_No	URL	Street_Name	Street_No	City	Postal_Code	State
Attribute Type	VARCHAR	VARCHAR	INT	VARCHAR	VARCHAR	VARCHAR	CHAR	INT	CHAR
Required/Not Required	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL

1.2.2 Guest

- The guest information is stored in the 'Guest' database, with 'Guest_ID' as the primary key.
- Guest name and address are mandatory fields, while the fields mentioned below are optional, and thus can be 'Null' –
 - E-mail
 - Home_Contact
 - Work_Contact
 - Cell_Contact

	Guest													
Attribute	Guest_ID	First_Name	Middle_Name	Last_Name	Street_Name	Street_No	City	Postal_Code	State	Country	E-mail	Home_Contact	Work_Contact	Cell_Contact
Attribute Type	VARCHAR	CHAR	CHAR	CHAR	VARCHAR	VARCHAR	CHAR	INT	CHAR	CHAR	VARCHAR	INT	INT	INT
Required/Not Required	NOT NULL	NOT NULL	NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NULL	NULL	NULL	NULL

1.2.3 Facility

- The DBMS stores each facility based on their unique 'Facility_ID' (primary key), its type, and whether the facility requires reservation or not.

Facility				
Attribute	Facility_ID	Facility_Type	Reservation_Required	Hotel_ID
Attribute Type	VARCHAR	CHAR	BOOLEAN	VARCHAR
Required/Not Required	NOT NULL	NOT NULL	NOT NULL	NOT NULL

1.2.4 Reservation

- Reservations are tracked through the 'Reservation' entity using a unique 'Reservation_ID' (primary key), and should include the tentative check in and check out dates, along with the date of reservation.
- The reservations can either be made directly at the hotel itself, which is labelled as 'System', or through a booking channel which is labelled as 'Channel'.
- The DBMS stores the booking fees (paid by hotel to the channel). If the reservation is made directly, the fee is entered as zero. This helps the hotel track down the charges levied by each channel.
- The reservation table also captures preferences (like floor, no. of beds, etc.) and booked facilities. Both are optional fields.

Reservation											
Attribute	Reservation_ID	Booked_Through	Channel_Name	Reservation_Date	Tentative_Check_In	Tentative_Check_Out	Booking_Fee	Smoking	No_of_Beds	Guest_ID	Hotel_ID
Attribute Type	VARCHAR	CHAR	VARCHAR	DATE	DATE	DATE	DECIMAL	BOOLEAN	INT	VARCHAR	VARCHAR
Required/Not Required	NOT NULL	NOT NULL	NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NULL	NULL	NOT NULL	NOT NULL

1.2.5 Stay

- The stay information is temporarily stored in a database. Once the temporary tab is closed, the data is pushed into 'Stay', the historical database.
- Each stay has a unique 'Stay_ID' (primary key).

Stay				
Attribute	Stay_ID	Check_In	Check_Out	Guest_ID
Attribute Type	VARCHAR	DATE	DATE	VARCHAR
Required/Not Required	NOT NULL	NOT NULL	NOT NULL	NOT NULL

1.2.6 Room

- Each Room has a unique 'Room_ID' (primary key) which can be numeric, alphanumeric characters or have a unique name.
- The DBMS stores room specifications which are utilized during allocation process according to guest preferences.

Room							
Attribute	Room_ID	Floor	No_of_beds	Smoking	Hotel_ID	Reservation_ID	Stay_ID
Attribute Type	VARCHAR	INT	INT	BOOLEAN	VARCHAR	VARCHAR	VARCHAR
Required/Not Required	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NULL	NULL

1.2.7 Invoice

- Each stay instance generates an invoice with a unique 'Invoice_ID' (primary key) which is closed during check out.
- The invoice contains line items for all services used & can be paid at any time. The invoice shows amount paid and balance due values.
- Additionally, credit card information is stored in the DBMS through this entity.

Invoice									
Attribute	Invoice_No	Total_Amount	Total_Payment	Tax_Charged	Credit_Card_No	Credit_Card_Expiry_Month	Credit_Card_Expiry_Year	Guest_ID	Stay_ID
Attribute Type	VARCHAR	NUMBER	NUMBER	NUMBER	INT (16)	INT (2)	INT (2)	VARCHAR	VARCHAR
Required/Not Required	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL

1.3 Junction Tables

To define many-to-many relationship amongst a few entities, the following junction tables have been created.

- 'Facility' and 'Reservation' entities have a many-to-many relation as same reservation can have multiple facilities tagged to it and same facility can be tagged to multiple reservations.

- Thus, the table comprises of the primary keys of both the tables –
 - Facility_ID
 - Reservation_ID

Facility Reservation		
Attribute	Facility ID	Reservation ID
Attribute Type	VARCHAR	VARCHAR
Required/Not Required	NOT NULL	NOT NULL

- Similarly, 'Facility' and 'Stay' entities have a many-to-many relation as a particular stay can have multiple facilities tagged to it, and same facility can be tagged to stay instances.
- Thus, the table comprises of the primary keys of both the tables –
 - Facility_ID
 - Stay_ID

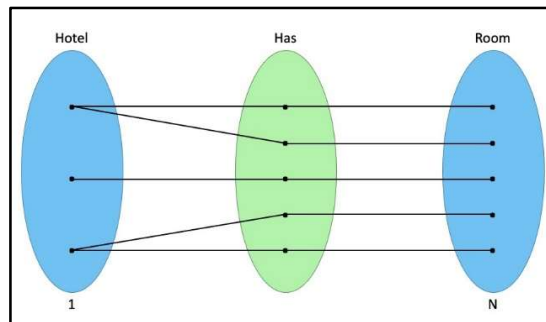
Facility_Stay		
Attribute	Facility ID	Stay ID
Attribute Type	VARCHAR	VARCHAR
Required/Not Required	NOT NULL	NOT NULL

1.4 Entities - Relationship and Cardinalities

The cardinality between various entities, derived from the E-R diagram, has been defined below.

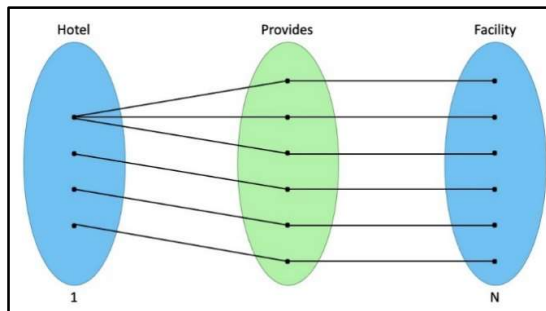
1.4.1 Hotel and Room - *Has*

- Each hotel has multiple rooms in it, and thus have a one-to-many-relationship.



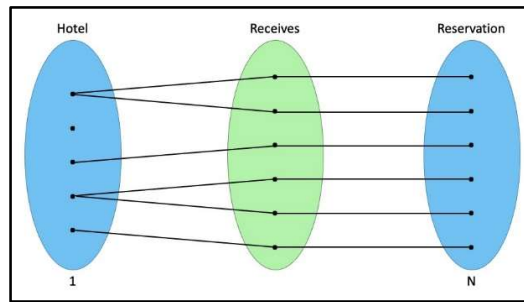
1.4.2 Hotel and Facility - *Provides*

- Each hotel provides multiple facilities to its guests, and thus have a one-to-many-relationship.



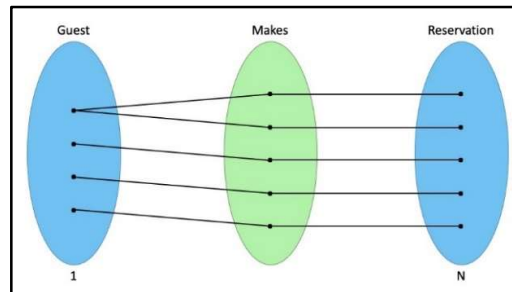
1.4.3 Hotel and Reservation - *Receives*

- Through the booking channel or via hotel website, the hotels receive multiple reservation from guests, and thus have a one-to-many-relationship.



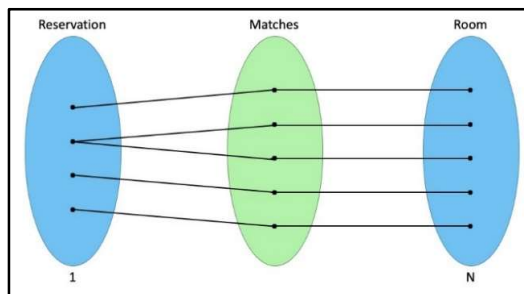
1.4.4. Guest and Reservation - *Makes*

- A guest can make reserve multiple rooms. As it is assumed that each booked room corresponds to a different reservation, a one-to-many relationship exists.



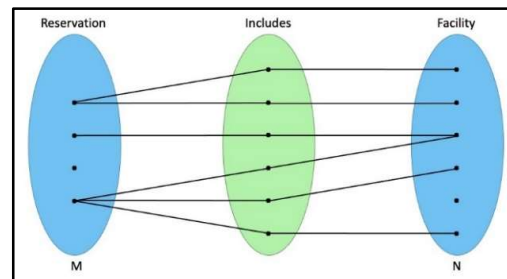
1.4.5 Reservation and Room - *Matches*

- While making a reservation the guest can provide preferences about the room (like smoking/ non-smoking, floor etc.). The reservation is matched with the room. As multiple rooms can fulfil the preference criteria, reservation and rooms have a one-to-many-relationship.



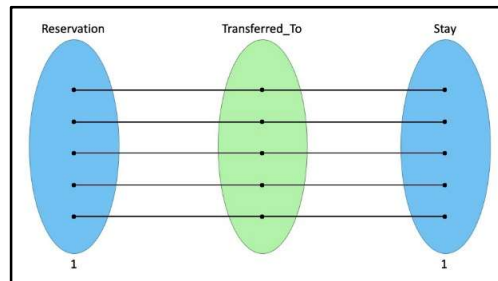
1.4.6. Reservation and Facility - *Includes*

- While making a reservation the guest can choose facilities to be utilized during the stay. A reservation can include multiple facilities, while same facility can be tagged to multiple reservations as well. Thus, it is a many-to-many relationship.



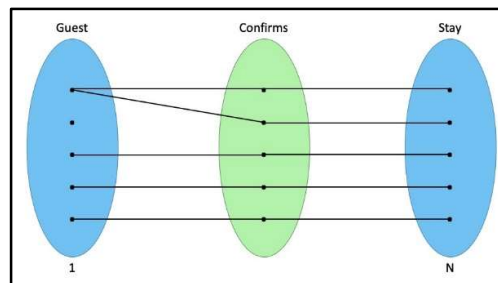
1.4.7 Reservation and Stay – *Transferred To*

- All reservations (even cancelled reservations) are passed to Stay table and are associated with separate Stay_IDs. Thus, they have a one-to-one relationship.



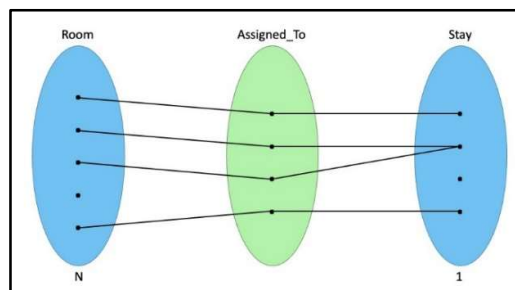
1.4.8 Guest and Stay - *Confirms*

- As each room reservation is treated as a separate stay instance, if a guest has booked multiple rooms (multiple reservations), multiple 'Stay_ID's will be tagged to them. Hence, the a one-to-many relationship exists.



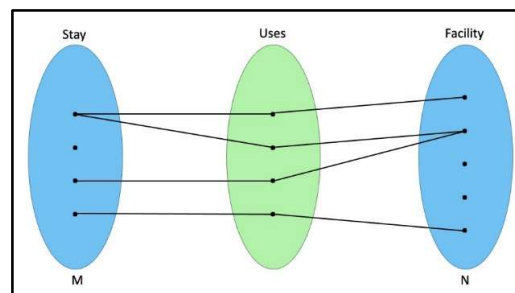
1.4.9 Room and Stay - *Assigned To*

- In case the room preferred by the guest is temporarily unavailable, they are allocated a different room on temporary basis and are later re-allocated to a room meeting their preferences. Hence, a stay instance can be associated with multiple rooms, leading to a many-to-one relationship.



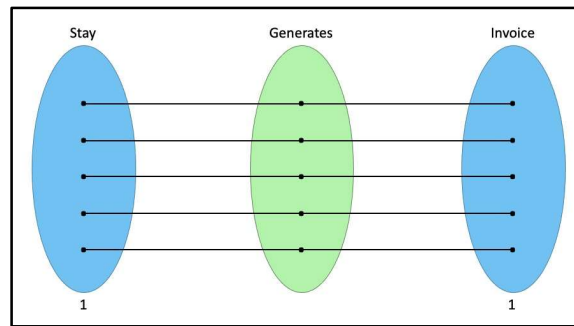
1.4.10 Stay and Facility - *Uses*

- Each stay instance can use multiple facilities and some facilities can be used in more than one stay. Hence, a many-to-many relationships exists.



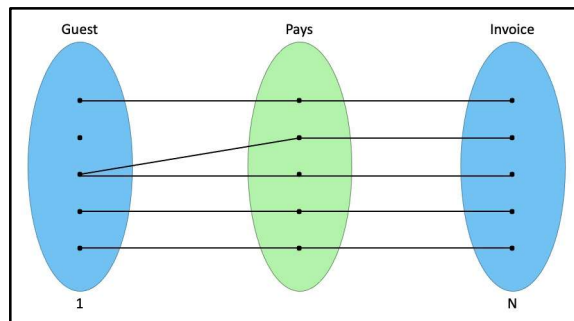
1.4.11 Stay and Invoice - *Generates*

- Each Stay generates a single Invoice which shows all the expenditures. Hence, the relationship between entities is one-to-one.



1.4.12 Guest and Invoice - *Pays*

- The invoice is generated per room. If the guest has booked multiple rooms, multiple invoices will be generated. Hence, one-to-many relationship exists.



2. SQL

2.1 Creating Database Tables - DDL queries

#Creating 'Guest' Table

```
CREATE TABLE IF NOT EXISTS 'Guest' (  
  'Guest_ID' VARCHAR PRIMARY KEY,  
  'First_Name' CHAR NOT NULL,  
  'Middle_Name' CHAR,  
  'Last_Name' CHAR NOT NULL,  
  'Street_Name' VARCHAR NOT NULL,  
  'Street_No' VARCHAR NOT NULL,  
  'City' CHAR NOT NULL,  
  'Postal_Code' INT NOT NULL,  
  'State' CHAR NOT NULL,  
  'Country' CHAR NOT NULL,  
  'E-mail' UNIQUE VARCHAR,  
  'Home_Contact' INT,  
  'Work_Contact' INT,  
  'Cell_Contact' INT  
);
```

#Assumption: Contacts do not have '+' in the beginning

#Creating 'Hotel' Table

```
CREATE TABLE IF NOT EXISTS 'Hotel' (  
  'Hotel_ID' VARCHAR PRIMARY KEY,  
  'Hotel_Name' CHAR NOT NULL,  
  'Street_Name' VARCHAR NOT NULL,  
  'Street_No' VARCHAR NOT NULL,  
  'City' CHAR NOT NULL,  
  'Postal_Code' INT NOT NULL,  
  'State' CHAR NOT NULL,  
  'Country' CHAR NOT NULL,  
  'E-mail' UNIQUE VARCHAR,  
  'Home_Contact' INT,  
  'Work_Contact' INT,  
  'Cell_Contact' INT  
);
```

```
'Hotel_ID' VARCHAR PRIMARY KEY,
'Hotel_Name' VARCHAR NOT NULL
'Phone_No' INT NOT NULL,
'URL' UNIQUE VARCHAR NOT NULL,
'Street_Name' VARCHAR NOT NULL,
'Street_No' VARCHAR NOT NULL,
'City' CHAR NOT NULL,
'Postal_Code' INT NOT NULL,
'State' CHAR NOT NULL
);
```

#Creating 'Facility' Table

```
CREATE TABLE IF NOT EXISTS 'Facility' (
'Facility_ID' VARCHAR PRIMARY KEY,
'Facility_Type' CHAR NOT NULL,
'Reservation_Required' BOOLEAN NOT NULL,
'Hotel_ID' VARCHAR NOT NULL,
FOREIGN KEY ('Hotel_ID') REFERENCES Hotel ('Hotel_ID')
);
```

#Creating 'Reservation' Table

```
CREATE TABLE IF NOT EXISTS 'Reservation' (
'Reservation_ID' VARCHAR PRIMARY KEY,
'Booked_Through' CHAR NOT NULL,
'Channel_Name' VARCHAR,
'Reservation_Date' DATE NOT NULL,
'Tentative_Check_In' DATE NOT NULL,
'Tentative_Check_Out' DATE NOT NULL,
'Booking_Fee' DECIMAL (10,2),
'Smoking' BOOLEAN NULL,
'No_of_Beds' INT NULL,
'Guest_ID' VARCHAR NOT NULL,
'Hotel_ID' VARCHAR NOT NULL,
FOREIGN KEY ('Guest_ID') REFERENCES Guest ('Guest_ID')
FOREIGN KEY ('Hotel_ID') REFERENCES Hotel ('Hotel_ID')
);
```

#Creating 'Stay' Table

```
CREATE TABLE IF NOT EXISTS 'Stay' (
'Stay_ID' VARCHAR PRIMARY KEY,
'Check_In' DATE NOT NULL,
'Check_Out' DATE NOT NULL,
'Guest_ID' VARCHAR NOT NULL,
FOREIGN KEY ('Guest_ID') REFERENCES Guest ('Guest_ID'),
);
```

#Creating 'Room' Table

```
CREATE TABLE IF NOT EXISTS 'Room' (
'Room_ID' VARCHAR PRIMARY KEY,
'Floor' INT NOT NULL,
'No_of_Beds' INT NOT NULL,
'Smoking' BOOLEAN NOT NULL,
'Hotel_ID' VARCHAR NOT NULL,
'Reservation_ID' VARCHAR,
```

```
'Stay_ID' VARCHAR,
FOREIGN KEY ('Hotel_ID') REFERENCES Hotel ('Hotel_ID')
FOREIGN KEY ('Reservation_ID') REFERENCES Reservation ('Reservation_ID')
FOREIGN KEY ('Stay_ID') REFERENCES Stay ('Stay_ID')
);
```

#Creating 'Invoice' Table

```
CREATE TABLE IF NOT EXISTS 'Invoice' (
'Invoice_No' VARCHAR PRIMARY KEY,
'Total_Amount' NUMBER NOT NULL,
'Total_Payment' NUMBER NOT NULL,
'Tax_Charged' NUMBER NOT NULL,
'Credit_Card_No' INT (16) NOT NULL,
'Credit_Card_Expiry_Month' INT (2) NOT NULL,
'Credit_Card_Expiry_Year' INT (2) NOT NULL,
'Guest_ID' VARCHAR NOT NULL,
'Stay_ID' VARCHAR NOT NULL,
FOREIGN KEY ('Guest_ID') REFERENCES Guest ('Guest_ID'),
FOREIGN KEY ('Stay_ID') REFERENCES Stay ('Stay_ID'),
);
```

2.2 Querying the Database

1. The total spent for the customer for a particular stay (checkout invoice).

#Selecting fields required to generate desired output from 'Invoice', 'Guest', & 'Stay' tables

```
SELECT i.Invoice_No, i.Guest_ID, g.First_name, g.Middle_name,g.Last_name, i.Stay_ID, s.check_in,
s.check_out, i.Total_Amount
FROM Invoice as i
```

#Using inner join to connect 'Invoice', 'Guest', & 'Stay' table using 'Guest_ID' field as the foreign key

```
INNER JOIN Guest as g
USING ('Guest_ID')
INNER JOIN Stay as s
USING ('Guest_ID'))
```

2. The most valuable customers in (a) the last two months, (b) past year and (c) from the beginning of the records.

a. Last two months -

#Selecting fields required to generate desired output from 'Invoice', 'Guest', & 'Stay' tables

```
SELECT i.Guest_ID, g.First_name, g.Middle_name, g.Last_name, SUM(i.Total_Amount) AS Total_Value,
s.Check_out
FROM Invoice AS i
```

#Using inner join to connect 'Invoice', 'Guest', & 'Stay' table using 'Guest_ID' field as the foreign key

```
INNER JOIN Stay AS s
USING ('Guest_ID')
INNER JOIN Guest AS g
USING ('Guest_ID')
```

#Adding constraint on the 'check_out' date to filter output for only last two months

```
WHERE s.Check_out >= DATE('now', '-2 months')
```

#Grouping the total spend by 'Guest ID' to display it total sum corresponding to each guest

```
GROUP BY i.Guest_ID
```

#Sorting the output to display the most valuable customer on the top

```
ORDER BY Total_Value DESC;")
```

b. Past year -

```
SELECT i.Guest_ID, g.First_name, g.Middle_name, g.Last_name, SUM(i.Total_Amount) AS Total_Value,  
s.Check_out  
FROM Invoice AS i  
INNER JOIN Stay AS s  
USING ('Guest_ID')  
INNER JOIN Guest AS g  
USING ('Guest_ID')  
#Adding constraint on the 'check_out' date to filter output for past one year  
WHERE s.Check_out >= DATE('now', '-12 months')  
GROUP BY i.Guest_ID  
ORDER BY Total_Value DESC;")
```

c. From the beginning of the records -

```
SELECT i.Guest_ID, g.First_name, g.Middle_name, g.Last_name, SUM(i.Total_Amount) AS Total_Value  
FROM Invoice AS i  
INNER JOIN Guest AS g  
USING ('Guest_ID')  
GROUP BY Guest_ID  
ORDER BY Total_Value DESC;")
```

3. Which are the top countries where our customers come from?

```
#Taking count of guests from each country from the 'Guest' table  
SELECT COUNT(*) AS 'No_of_Guests', Country  
FROM Guest  
GROUP BY Country  
#Ordering the output to display country with highest number of guests on the top  
ORDER BY No_of_Guests DESC;
```

4. How much did the hotel pay in referral fees for each of the platforms that we have contracted with?

```
#Selecting relevant fields from the 'Reservation' table and adding 'Total_Referral_Fee' column in the  
queried output to display the sum of booking fee from each channel  
SELECT Hotel_ID, Booked_through, Channel_Name, SUM(Booking_Fee) AS 'Total_Referral_Fee'  
FROM Reservation  
#Grouping by 'Hotel ID' & 'Channel Name' to display sum of referral fees paid by each hotel on each platform  
GROUP BY Hotel_ID, Channel_Name  
#Selecting only the reservations booked through a channel  
HAVING Booked_through = 'Channel'  
ORDER BY Hotel_ID, Total_Referral_Fee DESC")
```

5. What is the utilization rate for each hotel (that is the average billable days of a hotel specified as the average utilization of room bookings for the last 12 months).

```
#While calculating utilization rate for last 12 months, three cases arise-  
a. When both the 'check_in' and 'check_out' dates are older than last 12 months, and thus  
not considered in the query  
b. When the 'check_in' is before the 12-month period, however the 'check_out' date lies in the  
12-month period.  
c. When both the 'check_in' and 'check_out' dates lie within the last 12 months period
```

#To accommodate the second case, 'RoomView' view is created in sql to modify the 'check_in' date column for dates where the 'check_in' date is beyond the last 12 months period.

```
CREATE VIEW 'RoomView' AS SELECT r.Hotel_ID, r.Room_ID, s.Check_in, s.Check_out,  
CASE
```

#If both 'check_in' & 'check_out' dates are before the 12 months period, no changes made to the 'check_in' date (as it will not be a part of the queried output)

```
WHEN s.Check_in < DATE('now', '-12 months') AND s.Check_out < DATE('now', '-12 months') THEN  
s.Check_in = s.Check_in
```

#If 'check_in' date is before 12 months period but the 'check_out' date lies between the 12 months period, 'check_in' updated to oldest date of the 12-month period

```
WHEN s.Check_in < DATE('now', '-12 months') AND s.Check_out > DATE('now', '-12 months') THEN  
s.Check_in = DATE('now', '-12 months')
```

#If both 'check_in' & 'check_out' dates are within the 12 months period, no changes made to the 'check_in' date

```
ELSE s.Check_in = s.Check_in
```

```
END
```

```
FROM Stay AS s
```

```
INNER JOIN Rooms AS r
```

```
USING ('Stay_ID')
```

#Selecting relevant columns from the 'RoomView' view, adding 'Total_Days_Used' column in the queried output to display the total days the room was occupied

```
SELECT Hotel_ID, SUM(Check_out-Check_in) AS 'Total_Days_Used',
```

#Taking count of distinct 'Room_ID' and multiplying by 365 for calculating the total number of days, & storing the output in 'Total_Room_Days' column

```
(COUNT(DISTINCT(Room_ID))*365) AS 'Total_Room_Days',
```

#Calculating the 'Utilization Rate' as -

total days room was occupied (Check_out-Check_in)

*total number of days (COUNT(DISTINCT(Room_ID))*365)*

```
SUM(Check_out-Check_in)/(COUNT(DISTINCT(Room_ID))*365) AS 'Utilization_Rate'
```

```
FROM RoomView
```

#Adding constraints to 'check_in' & 'check_out' dates to get specify the 12-month period

```
WHERE (Check_in >= DATE('now', '-12 months'))
```

```
AND (Check_out >= DATE('now', '-12 months'))
```

#Grouping by 'Hotel_ID' to show utilization rate for each hotel

```
GROUP BY Hotel_ID
```

6. Calculate the Customer Value in terms of total spent for each customer before the current booking.

#Selecting fields required to generate desired output from 'Invoice', Reservation, & 'Guest' tables, & storing the sum of total amount (from 'Invoice') table as 'Total_Spnd'

```
SELECT g.Guest_ID, g.First_name, g.Middle_name, g.Last_name, sum(i.Total_amount) AS 'Total_Spend'
```

```
FROM Invoice AS i
```

#Using inner join to connect 'Invoice', 'Reservation', & 'Guest', table using 'Guest_ID' field as the foreign key

```
INNER JOIN Reservation AS r
```

```
USING ('Guest_ID')
```

```
INNER JOIN Guest AS g
```

```
USING ('Guest_ID')
```

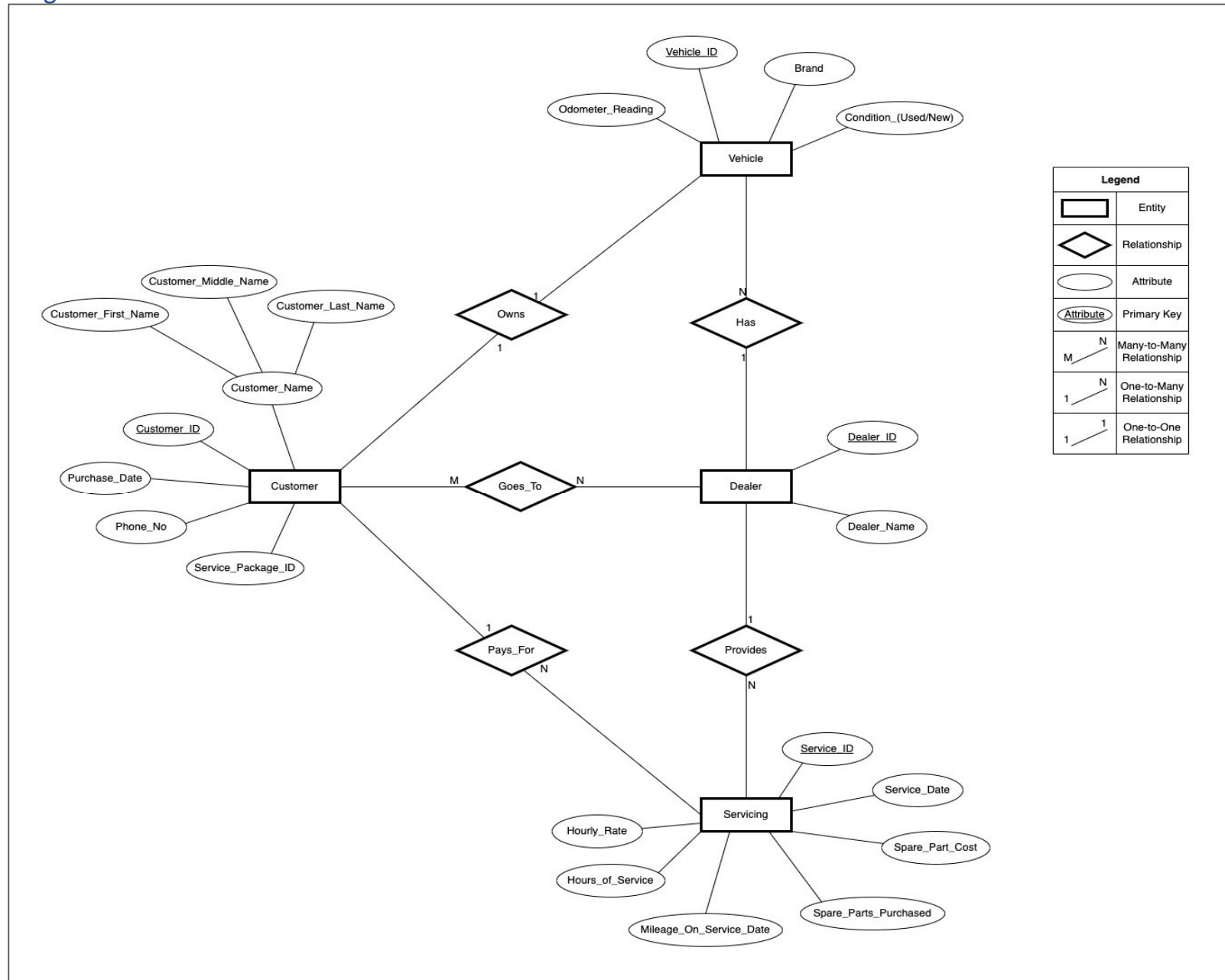
#Adding constraints on 'Reservation_Date' to ensure that the output does not contain the current booking

```
WHERE r.Reservation_Date <> Max(r.Reservation_Date)
```

```
Group by Guest_ID
```

PART A2

1. E - R Diagram



1.1 Assumptions

- It is assumed that each customer can own only one vehicle and if they buy a new one from a dealer, the previous information on the old vehicle is to be discarded.
- If a customer buys a service package, they will get a 10% discount for all services in the future.
- All the vehicle information kept in the DBMS is assumed to be up-to-date and there is no vehicle that is not being sold by the dealers.
- All of the dealers are assumed to provide at least one kind of service.

1.2 Entities and Attributes

The section describes different entities shown in the E-R diagram above.

1.2.1 Dealer

- The Dealer entity stores the dealer IDs and the first and last names of the dealers.
- Each dealer has a unique 'Dealer_ID', which acts as a primary key.

Dealer			
Attribute	Dealer_ID	Dealer_First_Name	Dealer_Last_Name
Attribute Type	VARCHAR	CHAR	CHAR
Required/Not Required	NOT NULL	NOT NULL	NOT NULL

1.2.2 Vehicle

- Each vehicle has a unique 'Vehicle_ID' (primary key) that is used to easily identify the possible needs of the customers such as the number of spare parts and their types needed, as well as the name of the brand of the vehicle.
- The condition that the vehicle in is stored so that whether it has been used or it is new when the customer buys it from the dealer is an easily accessible information.
- The mileage of the vehicle before being bought by the current customer is stored. The default value of the condition of the vehicle is "New", and the default mileage is zero.

Vehicle					
Attribute	Vehicle_ID	Brand	Condition_(Used/New)	Odometer_Reading	Dealer_ID
Attribute Type	VARCHAR	CHAR	CHAR	DECIMAL (10,2)	VARCHAR
Required/Not Required	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL

1.2.3 Customer

- Each customer has a unique 'Customer_ID' (primary key) that lets the DBMS to differentiate between each of them.
- The DBMS stores the names of the customers as well as their phone numbers to contact them when necessary.
- The purchase date of the vehicle and the service package they bought (if any) is kept in the records. The purchase date is a mandatory field while the information about the service package ID is optional. Hence, if the customer did not pay for a service package, the default value "-" does not change while Service_Package_ID stays 'Null'.

Customer								
Attribute	Customer_ID	Customer_First_Name	Customer_Middle_Name	Customer_Last_Name	Phone_No	Purchase_Date	Service_Package_ID	Vehicle_ID
Attribute Type	VARCHAR	CHAR	CHAR	CHAR	INT	DATE	VARCHAR	VARCHAR
Required/Not Required	NOT NULL	NOT NULL	NULL	NOT NULL	NOT NULL	NOT NULL	NULL	NOT NULL

1.2.4 Servicing

- Every time the customer asks for service from a dealer, the service provided is stored under the Servicing entity which acts like a transaction table.
- All the instances of services have their own unique 'Service_ID' (primary key).

- The amount of hours spent on the service and the cost of hourly labour are kept as well as whether the specific service required any spare parts being bought and if it did, the costs of the parts.
- The date the service is being provided is tracked down in addition to what the odometer reading is on that date.

Attribute	Servicing								
	Service_ID	Hours_of_Service	Hourly_Rate	Spare_Parts_Purchased	Spare_Part_Cost	Service_Date	Mileage_on_Service_Date	Dealer_ID	Customer_ID
Attribute Type	VARCHAR	DECIMAL (4,1)	DECIMAL (4,2)	VARCHAR	DECIMAL (10,2)	DATE	DECIMAL (10,2)	VARCHAR	VARCHAR
Required/Not Required	NOT NULL	NOT NULL	NOT NULL	NULL	NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL

1.3 Junction Tables

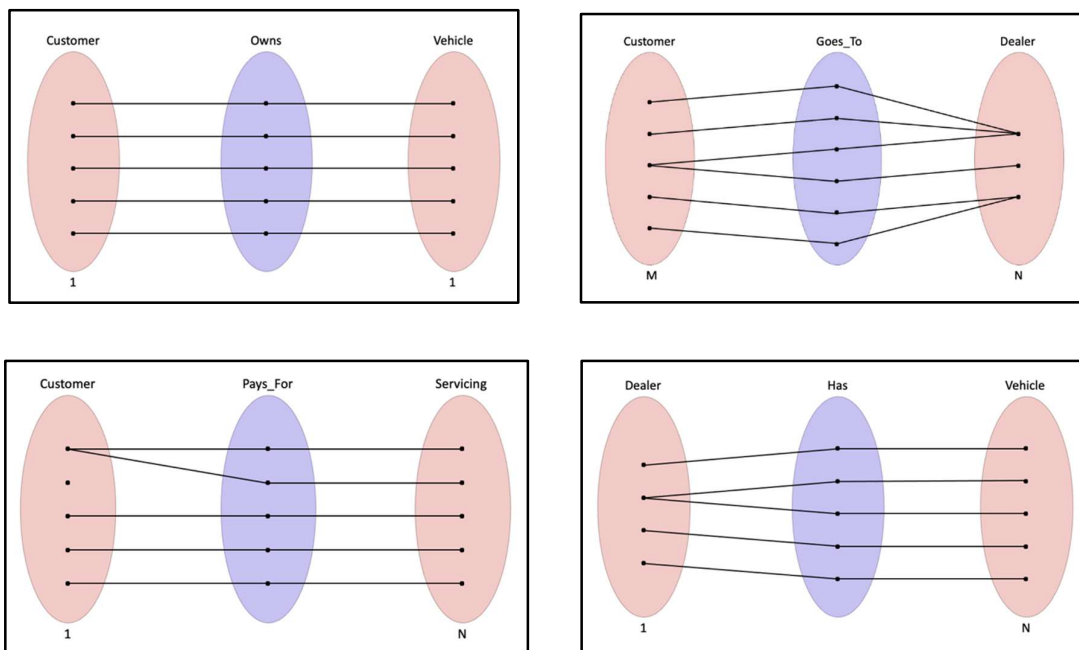
To define many-to-many relationship between Customer and Dealer entities, the following junction table have been created.

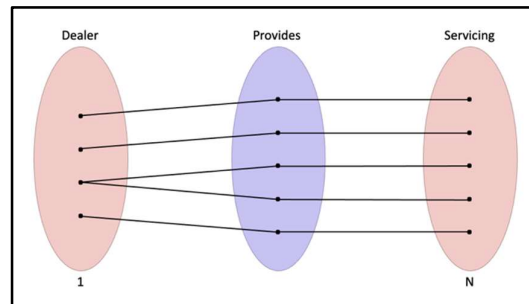
- The Customer_Dealer table is the junction table of customer and dealer entities, joined by their unique IDs which is used for the “Goes_To” relationship between them to show that a dealer can have multiple customers and a customer can go to multiple dealers.
- Hence, the table consists of the primary keys of both the tables –
 - Customer_ID
 - Dealer_ID

Attribute	Customer_Dealer	
	Customer_ID	Dealer_ID
Attribute Type	VARCHAR	VARCHAR
Required/Not Required	NOT NULL	NOT NULL

1.4 Entities - Relationship and Cardinalities

The cardinality between various entities, derived from the E-R diagram, has been defined below.





2. SQL

2.1 Creating Database Tables - DDL queries

#Creating 'Dealer' Table

```
CREATE TABLE 'Dealer' (
  'Dealer_ID' VARCHAR PRIMARY KEY,
  'Dealer_First_Name' CHAR NOT NULL,
  'Dealer_Last_Name' CHAR NOT NULL
);
```

#Creating 'Vehicle' Table

```
CREATE TABLE 'Vehicle' (
  'Vehicle_ID' VARCHAR PRIMARY KEY,
  'Brand' CHAR NOT NULL,
  'Condition'_(Used/New) CHAR DEFAULT 'New',
  'Odometer_Reading' NUMBER DEFAULT '0',
  'Dealer_ID' VARCHAR NOT NULL,
  FOREIGN KEY ('Dealer_ID') REFERENCES Dealer ('Dealer_ID')
);
```

#Creating 'Customer' Table

```
CREATE TABLE 'Customer' (
  'Customer_ID' VARCHAR PRIMARY KEY,
  'Customer_First_Name' CHAR NOT NULL,
  'Customer_Middle_Name' CHAR,
  'Customer_Last_Name' CHAR NOT NULL,
  'Phone_No' INT NOT NULL,
  'Purchase_Date' DATE NOT NULL,
  'Service_Package_ID' VARCHAR DEFAULT "-",
  'Vehicle_ID' VARCHAR NOT NULL,
  FOREIGN KEY ('Vehicle_ID') REFERENCES Vehicle ('Vehicle_ID')
);
```

#Creating 'Servicing' Table

```
CREATE TABLE 'Servicing' (
  'Service_ID' VARCHAR PRIMARY KEY,
  'Hours_of_Service' DECIMAL (4,1) NOT NULL,
  'Hourly_Rate' DECIMAL (4,2) NOT NULL,
  'Spare_Parts_Purchased' VARCHAR,
  'Spare_Part_Cost' DECIMAL (10,2),
  'Service_Date' DATE NOT NULL,
  'Mileage_on_Service_Date' DECIMAL (10,2) NOT NULL,
  'Dealer_ID' VARCHAR NOT NULL,
```

```
'Customer_ID' VARCHAR NOT NULL,
FOREIGN KEY ('Dealer_ID') REFERENCES Dealer ('Dealer_ID'),
FOREIGN KEY ('Customer_ID') REFERENCES Customer ('Customer_ID')
);
```

#Creating 'Customer_Dealership' Junction Table for 'Goes_To' Relationship

```
CREATE TABLE IF NOT EXISTS 'Customer_Dealership' (
'Customer_ID' VARCHAR NOT NULL,
'Dealership_ID' VARCHAR NOT NULL,
CONSTRAINT PK_Customer_Dealership PRIMARY KEY
(
Customer_ID,
Dealer_ID
),
FOREIGN KEY ('Customer_ID') REFERENCES Customer ('Customer_ID'),
FOREIGN KEY ('Dealer_ID') REFERENCES Dealer ('Dealer_ID')
);
```

2.2 Querying the Database

1. How many customers have stopped bringing their cars after the first encounter with the dealer?

--Counting the number of instances

```
SELECT COUNT(*) AS 'Customers leaving after the first encounter'
```

--Counting the number of customers that came for servicing once and didn't come back again

```
FROM (SELECT Count(*) AS 'Customers gone'
FROM Servicing
```

--Using left join to connect 'Servicing' and 'Customer' tables using 'Customer_ID' field as the foreign key to connect every ID from the 'Servicing' table

```
LEFT JOIN Customer USING ('Customer_ID')
```

--Using left join to connect the previously joined table and 'Customer' tables using 'Dealer_ID' field as the foreign key to connect every ID from the 'Servicing' table

```
LEFT JOIN Dealer USING ('Dealer_ID')
GROUP BY Customer_ID
```

--Specifying the instances, the customer got service from any dealer

```
HAVING COUNT(Service_Date) =1)
```

- **Output -**

1 records
Customers leaving after the first encounter
<hr/>
1000

2. What is the relationship between the price of the service and age of the car in terms of (a) actual car age (e.g., mileage) and (b) time with the current owner?

--Selecting fields required to show the relationship between the price of the service and the service and the actual car age in terms of days and mileage from 'Servicing', 'Customer' & 'Vehicle' tables

--Assuming the discount rate of service package is 10%

```
SELECT (0.9*((Hourly_Rate*Hours_of_Service)+Spare_Part_Cost)) AS 'Price_Of_Service',
```

--The difference between the mileage of the vehicle once the customer bought it and the actual mileage on the service date

```
(Mileage_on_Service_Date - Odometer_Reading) AS 'Age_Mileage',
```

--The number of years passed since the customer bought the vehicle

```
(JulianDay('now') - JulianDay(Purchase_Date))/365 AS 'Age_Time(Years)', Purchase_Date
```

```
FROM Servicing
```

--Using inner join to connect 'Servicing' and 'Customer' tables using 'Customer_ID' field as the foreign key

```
INNER JOIN Customer USING ('Customer_ID')
```

--Using inner join to connect 'Vehicle' table with the joined table of 'Servicing' and 'Customer' using 'Vehicle_ID' field as the foreign key

```
INNER JOIN Vehicle USING ('Vehicle_ID')
```

--Selecting 5 rows of the output

```
LIMIT 5;
```

- **Output -**

5 records

Price_Of_Service	Age_Mileage	Age_Time(Days)	Purchase_Date
4578.3	8052	334	08/01/2021
6114.6	12820	308	03/02/2021
5736.6	35781	632	16/03/2020
5778.9	28340	250	02/04/2021
5818.5	26337	577	10/05/2021

PART D

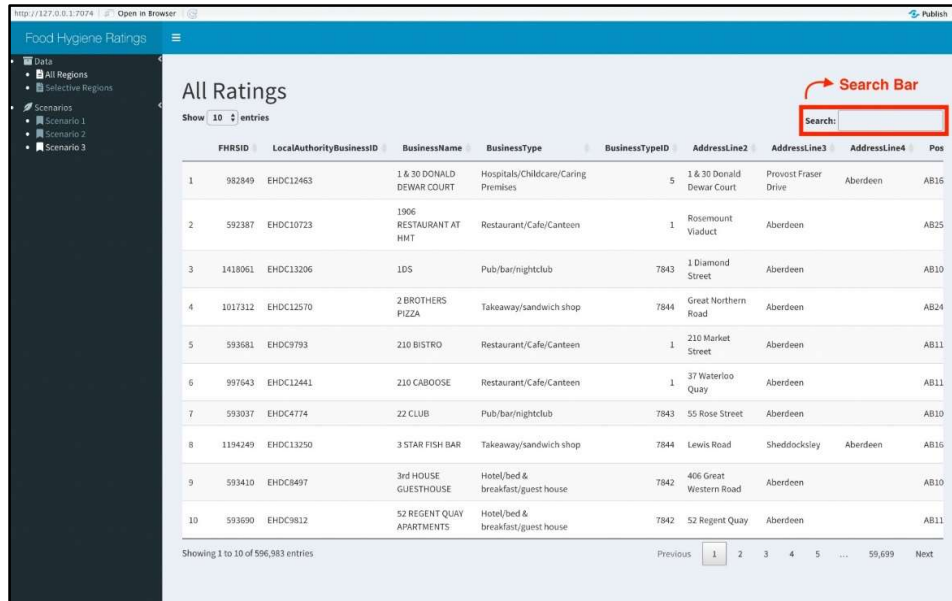
A Shiny dashboard has been created for 'Food Hygiene Ratings' data. It contains two sections, the 'Data' tab and the 'Scenario' tab.

1.1 Data Tab

The tab contains two sub-tabs –

- All Regions

The data from both UK and Scotland region is shown. The dashboard contains a search bar which can be used to search any value from the data. The number of entries shown on a page can be altered as well, using the drop-down menu.

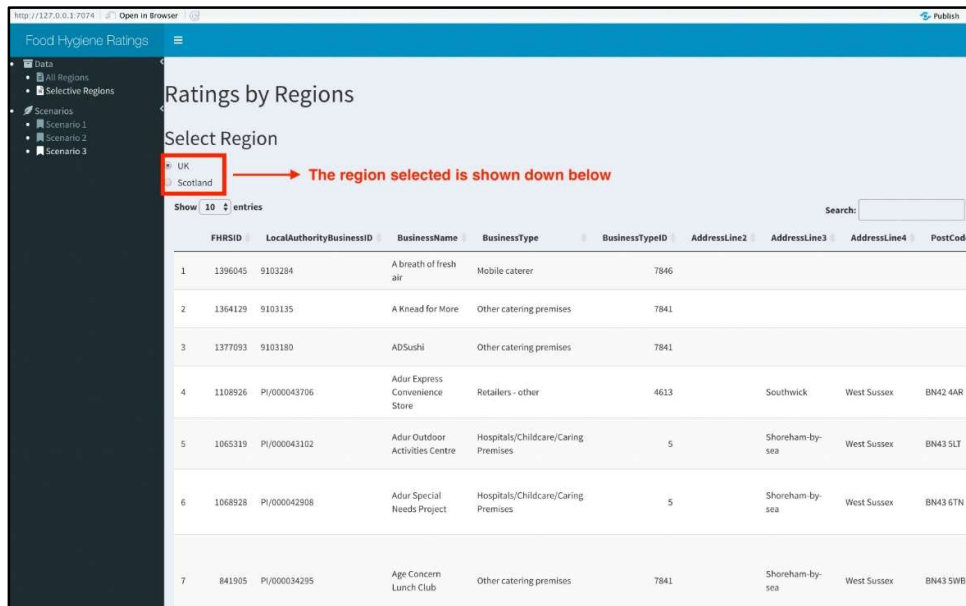


The screenshot shows the 'All Ratings' dashboard. On the left is a sidebar with 'Data' (All Regions, Selective Regions) and 'Scenarios' (Scenario 1, Scenario 2, Scenario 3). The main area is titled 'All Ratings' and shows 'Show 10 entries'. A search bar is highlighted with a red box and labeled 'Search Bar'. Below is a table with columns: FHRSID, LocalAuthorityBusinessID, BusinessName, BusinessType, BusinessTypeID, AddressLine2, AddressLine3, AddressLine4, and Pos. The table lists 10 entries. At the bottom, it says 'Showing 1 to 10 of 596,983 entries' and has pagination controls (Previous, 1, 2, 3, 4, 5, ..., 59,699, Next).

	FHRSID	LocalAuthorityBusinessID	BusinessName	BusinessType	BusinessTypeID	AddressLine2	AddressLine3	AddressLine4	Pos
1	982849	EHDC12463	1 & 30 DONALD DEWAR COURT	Hospitals/Childcare/Caring Premises	5	1 & 30 Donald Dewar Court	Provost Fraser Drive	Aberdeen	AB16
2	592387	EHDC10723	1906 RESTAURANT AT HMT	Restaurant/Cafe/Canteen	1	Rossmount Viaduct	Aberdeen		AB25
3	1418061	EHDC13206	1DS	Pub/bar/nightclub	7843	1 Diamond Street	Aberdeen		AB10
4	1017312	EHDC12570	2 BROTHERS PIZZA	Takeaway/sandwich shop	7844	Great Northern Road	Aberdeen		AB24
5	593681	EHDC9793	210 BISTRO	Restaurant/Cafe/Canteen	1	210 Market Street	Aberdeen		AB11
6	997643	EHDC12441	210 CABOOSE	Restaurant/Cafe/Canteen	1	37 Waterloo Quay	Aberdeen		AB11
7	593037	EHDC4774	22 CLUB	Pub/bar/nightclub	7843	55 Rose Street	Aberdeen		AB10
8	1194249	EHDC13250	3 STAR FISH BAR	Takeaway/sandwich shop	7844	Lewis Road	Sheddocksley	Aberdeen	AB16
9	593410	EHDC8497	3rd HOUSE GUESTHOUSE	Hotel/bed & breakfast/guest house	7842	406 Great Western Road	Aberdeen		AB10
10	593690	EHDC9812	52 REGENT QUAY APARTMENTS	Hotel/bed & breakfast/guest house	7842	52 Regent Quay	Aberdeen		AB11

- Selective Regions

Here, the data has been segregated based on regions i.e., UK and Scotland. The region can be selected using the radio button.



The screenshot shows the 'Ratings by Regions' dashboard. The sidebar is the same. The main area is titled 'Ratings by Regions' and has a 'Select Region' section with radio buttons for 'UK' (selected) and 'Scotland'. A red arrow points from the 'UK' button to the text 'The region selected is shown down below'. Below this is a search bar and a table with columns: FHRSID, LocalAuthorityBusinessID, BusinessName, BusinessType, BusinessTypeID, AddressLine2, AddressLine3, AddressLine4, and PostCode. The table lists 7 entries. At the bottom, it says 'Showing 1 to 10 of 596,983 entries' and has pagination controls (Previous, 1, 2, 3, 4, 5, ..., 59,699, Next).

	FHRSID	LocalAuthorityBusinessID	BusinessName	BusinessType	BusinessTypeID	AddressLine2	AddressLine3	AddressLine4	PostCode
1	1396045	9103284	A breath of fresh air	Mobile caterer	7846				
2	1364129	9103135	A Knead for More	Other catering premises	7841				
3	1377093	9103180	ADSunhi	Other catering premises	7841				
4	1108926	PI/000043706	Adur Express Convenience Store	Retailers - other	4613		Southwick	West Sussex	BN42 4AR
5	1065319	PI/000043102	Adur Outdoor Activities Centre	Hospitals/Childcare/Caring Premises	5		Shoreham-by-sea	West Sussex	BN43 5LT
6	1068928	PI/000042908	Adur Special Needs Project	Hospitals/Childcare/Caring Premises	5		Shoreham-by-sea	West Sussex	BN43 6TN
7	841905	PI/000034295	Age Concern Lunch Club	Other catering premises	7841		Shoreham-by-sea	West Sussex	BN43 5WB

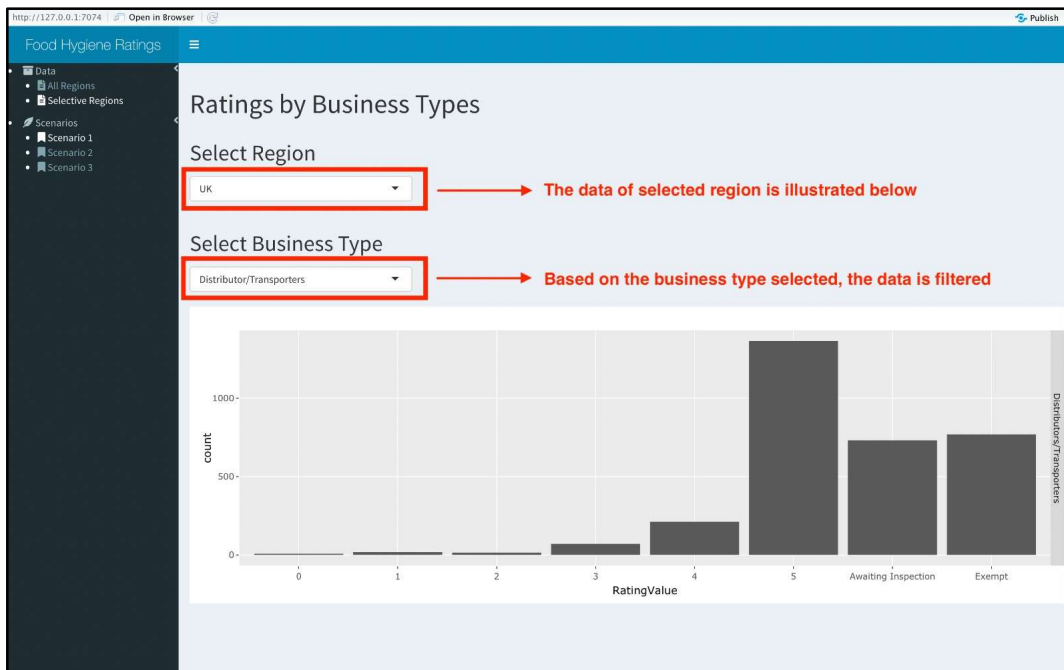
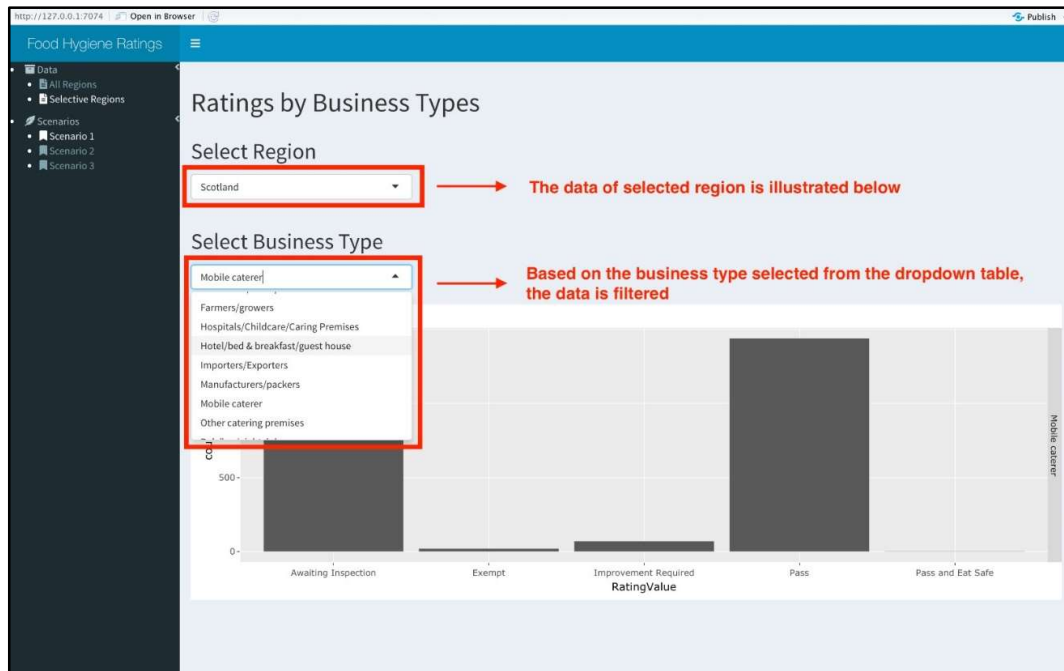
1.2 Scenario Tab

The tab illustrates three scenarios, created using SQL queries mentioned below.

1.2.1 Scenario 1

The query outputs a scenario which shows the number of ratings received corresponding to each 'rating value'. Further, the drop-down menus allow the user to select the 'Region' and the 'Business Type'.

- **SQL Query –**
SELECT COUNT(RatingValue), BusinessType, SchemeType
FROM Total_data
-- Grouping by 'SchemeType' allows the regions to be grouped, while 'BusinessType' field allows grouping by the type of businesses present in the data.
GROUP BY SchemeType, BusinessType



1.2.2 Scenario 2

The query outputs a graph showing number of ratings done per year to display the trend of food hygiene consciousness. Further, the drop-down menu allows the user to select the 'Region'.

- **SQL Query –**

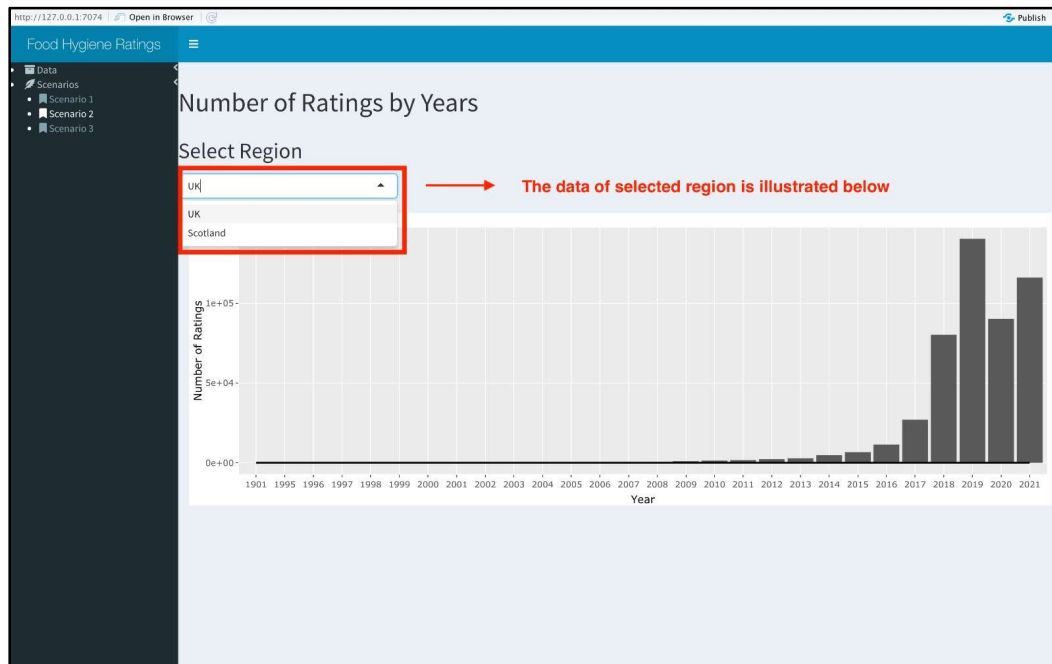
```
SELECT SchemeType, strftime('%Y', RatingDate)
```

```
AS YEAR FROM Total_data
```

```
-- Years with null value have been excluded, Scheme type selected for region segregation
```

```
WHERE YEAR IS NOT NULL AND SchemeType = 'FHRS'
```

```
ORDER BY YEAR
```

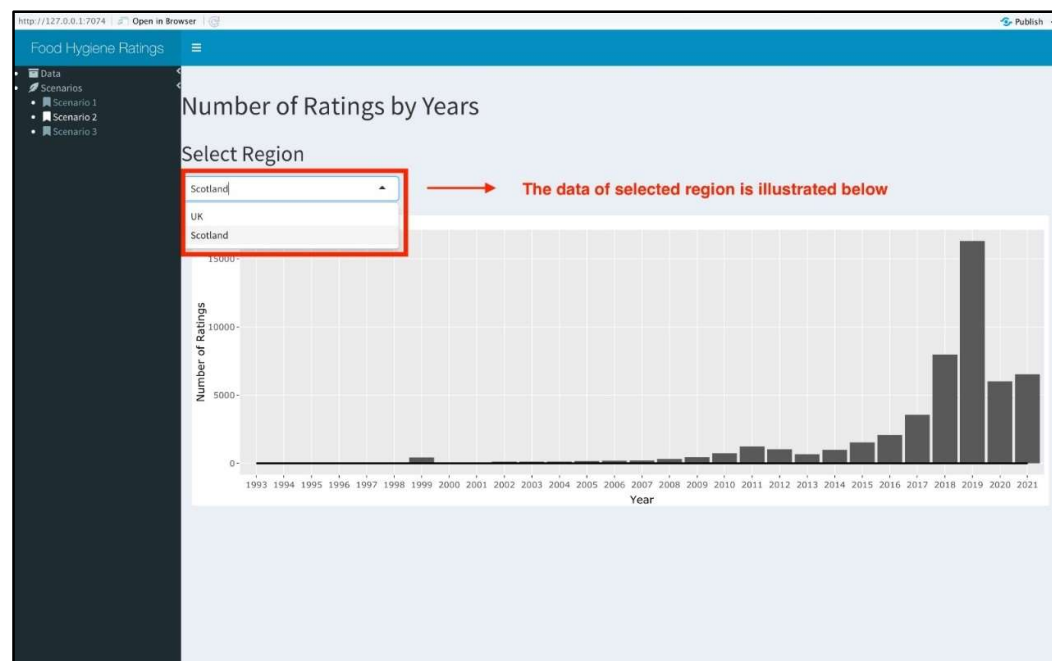


```
SELECT SchemeType, strftime('%Y', RatingDate)
```

```
AS YEAR FROM Total_data
```

```
WHERE YEAR IS NOT NULL AND SchemeType = 'FHIS'
```

```
ORDER BY YEAR
```



1.2.3 Scenario 3

The query result displays a graph showing the Non-Compliant Businesses in terms of count of bad ratings received by each business type. The purpose of this scenario is to display the performance of different businesses in terms of food hygiene ratings. Further, the drop-down menu allows the user to select the 'Region'.

- **SQL Query –**

```
SELECT RatingValue, SchemeType, BusinessType
FROM Total_data
```

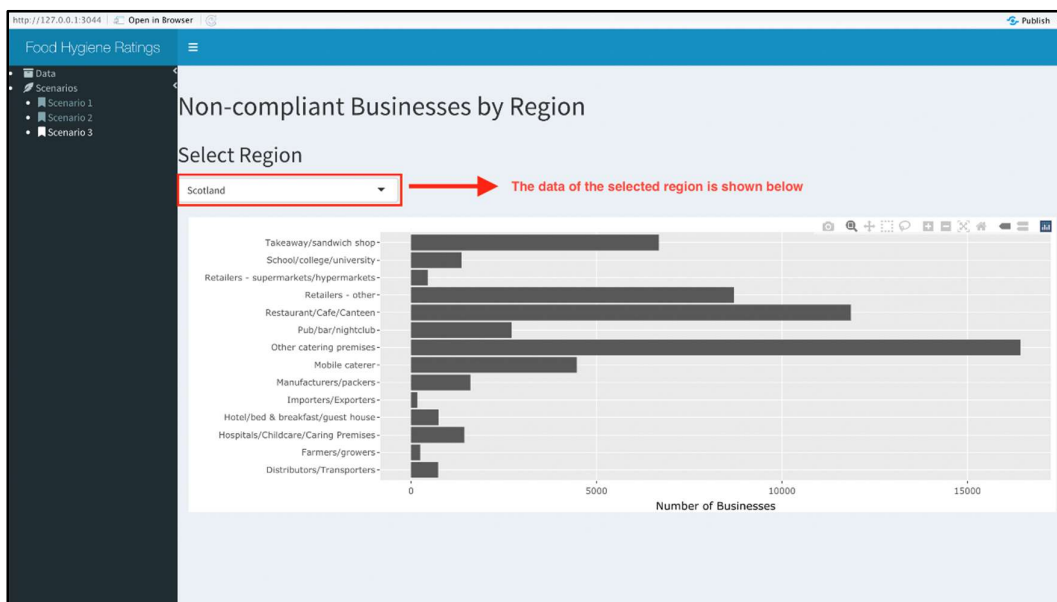
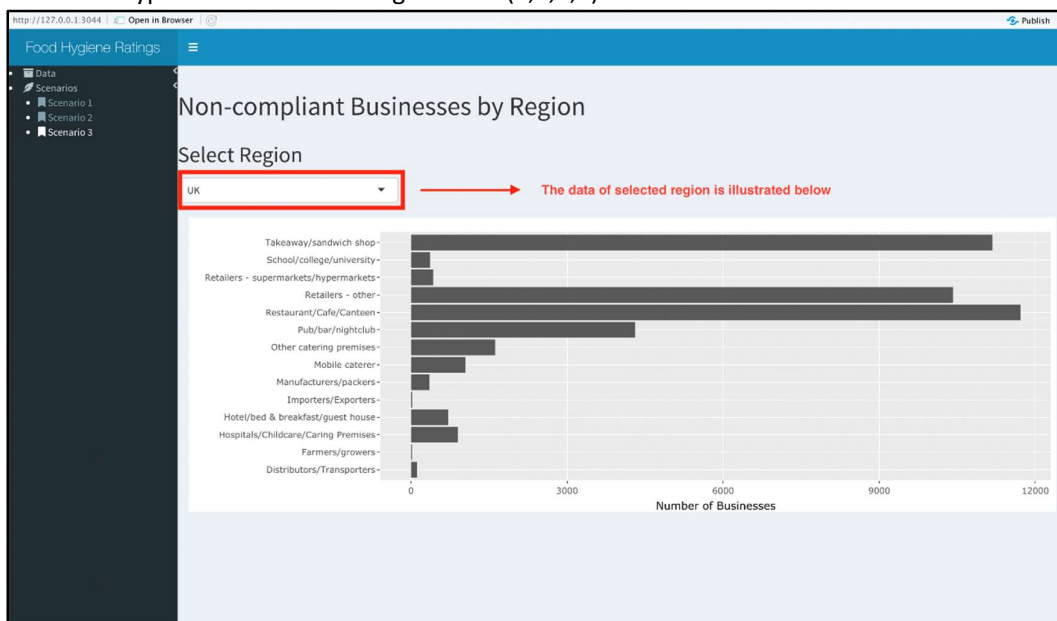
-- Selecting the non-compliant ratings only, and selecting 'schemetype' for regions segregation

```
WHERE SchemeType = 'FHIS' AND RatingValue IN ('Improvement Required', 'Awaiting Publication', 'Awaiting Inspection')
```

```
SELECT RatingValue, SchemeType, BusinessType
FROM Total_data
```

-- Selecting the non-compliant ratings only, and selecting 'schemetype' for regions segregation

```
WHERE SchemeType = 'FHRS' AND RatingValue IN (0,1,2,3)
```



APPENDIX

PART B – R Code

- **Calling the R libraries used in the code**

```
library(tidyverse)
library(dplyr)
library(readr)
library(readxl)
library(gridExtra)
```

- **Storing the *xlsx* files in 'all_files' variables using 'list.files' function**

```
all_files <- list.files("./partB_data_files/", pattern = "\\.*xlsx$", recursive = TRUE)
all_files
```

- **Creating a null data frame 'Final_df' for storing the final output**

```
Final_df = NULL
Final_df
```

- **Using for-loop to extract the data from all the *xlsx* files and storing the result in tabular form**

```
#Initiating the for-loop for reading the files in specific directory
```

```
for(i in all_files){
```

```
#Specifying a dynamic directory for reading files
```

```
excel_files <- read_excel(paste0("./PartB_data_files/", i))
```

```
#Removing the redundant rows & columns from the files
```

```
excel_files <- excel_files[-c(1,6,(nrow(excel_files)-2):nrow(excel_files)),-2]
```

```
#Assigning the country, flow & unit value in the 'info_types' tibble
```

```
info_types <- excel_files[1:3,2]
```

```
#Removing the rows which contained country, flow & unit values
```

```
excel_files <- excel_files[-c(1:3),]
```

```
#Taking column names from first row of file to make column headers
```

```
colnames(excel_files) <- excel_files[1,]
```

```
#Removing the first row which contained the headers
```

```
excel_files <- excel_files[-1,]
```

```
#Renaming first column from 'Product' to 'Year'
```

```
names(excel_files)[1]<- "Year"
```

```
#Taking pivot on 'Product' and 'Value' columns and converting the 'Values' column to numeric
```

```
excel_files <- pivot_longer(excel_files, -Year,names_to = "Product", values_to = "Value", values_transform = list(V  
alue= as.numeric))
```

```
#Adding 'Country' and 'Flow' columns to the table
```

```
excel_files <- excel_files %>% mutate(Country = as.character(info_types[3,]), Flow = as.character(info_types[1,]))
```

#Taking the for-loop out from each file and constructing data in the empty dataframe

```
Final_df <- rbind(Final_df,excel_files)
}
```

- **Rearranging columns in desired sequence**

```
Final_df <- Final_df[,c(4,1,5,2,3)]
```

#Replacing NA with zero in 'Values' column

```
Final_df[is.na(Final_df)] <- 0
```

#Displaying only first 16 records

```
Final_df1 <- head (Final_df,16)
```

	Country	Year	Flow	Product	Value
1	Albania	1971	Imports	Additives/blending components	0
2	Albania	1971	Imports	Anthracite	0
3	Albania	1971	Imports	Aviation gasoline	0
4	Albania	1971	Imports	BKB	0
5	Albania	1971	Imports	Biodiesels	0
6	Albania	1971	Imports	Biogases	0
7	Albania	1971	Imports	Biogasoline	0
8	Albania	1971	Imports	Bitumen	0
9	Albania	1971	Imports	Blast furnace gas	0
10	Albania	1971	Imports	Brown coal (if no detail)	0
11	Albania	1971	Imports	Charcoal	0
12	Albania	1971	Imports	Coal tar	0
13	Albania	1971	Imports	Coke oven coke	677
14	Albania	1971	Imports	Coke oven gas	0
15	Albania	1971	Imports	Coking coal	0
16	Albania	1971	Imports	Crude oil	0

Figure: Final Dataset - OCED Countries

- **Query 1: The total number of records in the dataset**

```
nrow(Final_df)
```

```
## [1] 573950
```

- **Query 2: The total number of records for each product across countries across years.**

```
fd <- Final_df %>% select(Product, Country, Year) %>% group_by(Product, Country, Year) %>% summarise(count = n())
```

#Displaying only first 16 records

```
fd1 <- head(fd,16)
```

	Product	Country	Year	count
1	Additives/blending components	Albania	1971	5
2	Additives/blending components	Albania	1972	5
3	Additives/blending components	Albania	1973	5
4	Additives/blending components	Albania	1974	5
5	Additives/blending components	Albania	1975	5
6	Additives/blending components	Albania	1976	5
7	Additives/blending components	Albania	1977	5
8	Additives/blending components	Albania	1978	5
9	Additives/blending components	Albania	1979	5
10	Additives/blending components	Albania	1980	5
11	Additives/blending components	Albania	1981	5
12	Additives/blending components	Albania	1982	5
13	Additives/blending components	Albania	1983	5
14	Additives/blending components	Albania	1984	5
15	Additives/blending components	Albania	1985	5
16	Additives/blending components	Albania	1986	5

Figure: Total records for each product across countries across years.

- **Retrieving download links for the XML files from the website**

```
#Assigning the 'Food Hygiene Ratings' data URL to a variable
url <- "https://data.food.gov.uk/catalog/datasets/38dd8d6a-5ab1-4f50-b753-ab33288e3200"

#Reading the website URL using 'rvest' package
sub_page <- read_html(url)

#Retrieving the names of all the cities and cleaning them
names <- sub_page %>% html_nodes(".c-dataset-element__item") %>% html_elements("h2") %>% html_text() %
>% as.vector()
names <- gsub("\n", "", names)
names <- paste0(names, ".xml")

#Retrieving the individual XML URLs of all cities
files <- sub_page %>% html_nodes(".o-dataset-distribution--link") %>% html_attr('href') %>% as.vector()

#Creating a dataframe with the retrieved links and names, and removing the unwanted files
links <- data.frame(files = files, names = names)
links <- links[which(grepl("en-", links$files)),]
```

- **Downloading all the XML files:**

```
#Initiating a for loop to obtain links and file names from the 'links' dataframe, and downloading them in a local directory
for(i in seq_len(nrow(links))) {
  file <- links[i, "files"]
  name <- links[i, "names"]

  download.file(file, name)
}
```

- **Reading the XML files into R**

```
#Getting a list of all the XML files from the local directory
all_files <- list.files(getwd())
all_files

#Initiating a function to parse all the XML files in the list by targeting the necessary XML node and storing all data in a list
data <- lapply(all_files, function(x){
  xml <- xmlParse(x)

#Parsing each individual file
  df <- xmlToDataFrame(xml, nodes = getNodeSet(xml, "//EstablishmentDetail"))

  return(df)
})

#Binding the output of the retrieved data to create a dataframe and separating the "Geocode" column into it's individual XML child nodes
df <- bind_rows(data) %>% separate(Geocode, into = c("Longitude", "Latitude"), sep = 17)
```

- **Exporting the resultant data into a csv file**

#The data is written into a csv file in the local directory

```
write.csv(df, "./Part C/df.csv")
```

1.1 Data Cleaning and SQL Queries

- **Calling the R libraries used in the code**

```
library(RSQLite)
library(tidyverse)library(dplyr)
library(DBI)
library(readr)
```

- **Cleaning the data obtained from Part C**

```
#Reading the csv file containing the data obtained from Part C
Total_data <- read_csv("df.csv") %>% mutate_all(~type.convert(., as.is = F))

#Trimming the data
Total_data$RatingValue <- stringr::str_trim(Total_data$RatingValue)
Total_data$Longitude <- strtrim(Total_data$Longitude, 8)
Total_data$Latitude <- strtrim(Total_data$Latitude, 8)

#Making changes to the data to ensure consistency
Total_data[Total_data == "AwaitingInspection"] <- "Awaiting Inspection"
Total_data[Total_data == "AwaitingPublication"] <- "Awaiting Publication"
Total_data <- Total_data[, -1]

#Converting columns into factors
Total_data$RatingValue <- as.factor(Total_data$RatingValue)
Total_data$Scores <- as.factor(Total_data$Scores)

#Creating separate dataframes for UK and Scotland regions
Scotland_ratings <- filter(Total_data, grepl('FHIS', SchemeType))
UK_ratings <- filter(Total_data, grepl('FHRS', SchemeType))

#Writing csv file to a local directory containing regional data
write.csv(UK_ratings, "./Part C/UK.csv")
write.csv(Scotland_ratings, "./Part C/Scotland.csv")

#Reading the data from the locally saved csv files
UKr <- read_csv("./Part C/UK.csv")
Scotlandr <- read_csv("./Part C/Scotland.csv")
```

- **Uploading tables to a database**

```
#Creating a database for the ratings data
connection <- dbConnect(SQLite(), "Ratings.db")

#Writing a table with the data on the database
dbWriteTable(connection, "Total_data", Total_data, overwrite == TRUE)
dbWriteTable(connection, "UKs", UKr, overwrite == TRUE)
dbWriteTable(connection, "Scotlands", Scotlandr, overwrite == TRUE)

#Checking the created table
dbListTables(connection)

## [1] "Scotlands" "Total_data" "UKs"
```

- **Scenario 1**

```
Scenario_1 <- dbGetQuery(connection,  
"SELECT COUNT(RatingValue), BusinessType, SchemeType  
FROM Total_data  
GROUP BY SchemeType, BusinessType"  
)
```

- **Scenario 2**

```
Scenario_2_UK <- dbGetQuery(connection,  
"SELECT SchemeType, strftime('%Y', RatingDate) AS YEAR  
FROM Total_data  
WHERE YEAR IS NOT NULL AND SchemeType = 'FHRS' ORDER BY YEAR"  
)
```

```
Scenario_2_Scotland <- dbGetQuery(connection,  
"SELECT SchemeType, strftime('%Y', RatingDate) AS YEAR  
FROM Total_data  
WHERE YEAR IS NOT NULL AND SchemeType = 'FHIS' ORDER BY YEAR"  
)
```

- **Scenario 3**

```
Scenario_3_Scotland <- dbGetQuery(connection,  
"SELECT RatingValue, SchemeType, BusinessType  
FROM Total_data  
WHERE SchemeType = 'FHIS' AND RatingValue IN ('Improvement Required', 'Awaiting Publication', 'Awaiting Inspection') "  
)
```

```
Scenario_3_UK <- dbGetQuery(connection,  
"SELECT RatingValue, SchemeType, BusinessType  
FROM Total_data  
WHERE SchemeType = 'FHRS' AND RatingValue IN (0,1,2,3)"  
)
```

1.3 Shiny Dashboard Creation

```
library(shiny)  
library(shinydashboard)  
library(DT)  
library(ggplot2)  
library(plotly)  
library(readr)  
library(DBI)  
library(RSQLite)  
  
# Defining the connection for retrieving data  
connection <- dbConnect(SQLite(), "Ratings.db")  
dbConnect(connection)  
  
dbWriteTable(connection, "UKs", UKr, overwrite == TRUE)  
dbWriteTable(connection, "Scotlands", Scotlandr, overwrite == TRUE)
```

```

# Reading the necessary files
UKr <- read_csv("./Part C/UK.csv")
Scotlandr <- read_csv("./Part C/Scotland.csv")

# Defining the ui part of the shiny dashboard
ui <-
  # Creating a dashboard page
  dashboardPage(
    # Putting content into the dashboard header
    dashboardHeader(title = "Food Hygiene Ratings"),
    # Putting content into the dashboard sidebar
    dashboardSidebar(
      # Creating menu in the sidebar
      menuItem("Data", icon = icon("archive"),
        # Creating sub-menu under the main menu
        menuSubItem("All Regions", tabName = "all_regions", icon = icon("file-alt")),
        menuSubItem("Selective Regions", tabName = "schemas", icon = icon("file-alt"))
      ),

      menuItem("Scenarios", icon = icon("feather-alt"),
        menuSubItem("Scenario 1", tabName = "scenario1", icon = icon("bookmark")),
        menuSubItem("Scenario 2", tabName = "scenario2", icon = icon("bookmark")),
        menuSubItem("Scenario 3", tabName = "scenario3", icon = icon("bookmark"))
      )
    )
  ,
  # Putting content into the dashboard body
  dashboardBody(
    # Creating tabs to hold contents in the body and defining the specifications for each content
    tabItems(
      # Tab for showing all data
      tabItem("all_regions",
        fluidPage(h1("All Ratings"),
          dataTableOutput("Total_data"))),
      # Tab for showing data separated by regions
      tabItem("schemas",
        fluidRow(width = 12, h1("Ratings by Regions")),
        fluidRow(width = 6,
          radioButtons("RatingsSchemaOverview", label = h2("Select Region"),
            choices = list("UK" = "FHRS",
                          "Scotland" = "FHIS"),
            selected = "FHRS")),
        fluidRow(column(width = 12, dataTableOutput("Schema_data"), style = "overflow-x:scroll;")
        )),
      # Tab for scenario 1
      tabItem("scenario1",
        fluidRow(column(width = 12, h1("Ratings by Business Types"))),
        fluidRow(column(width = 6,
          # Creating drop-down list for regions
          selectInput("RatingsSchema", label = h2("Select Region"),
            choices = list("UK" = "FHRS",
                          "Scotland" = "FHIS"),
            selected = "FHRS"))),
        fluidRow(column(width = 6,

```



```

# Creating drop-down list for business types
selectInput("BusinessType", label = h2("Select Business Type"),
  choices = list("Distributor/Transporters" = "Distributors/Transporters",
    "Farmers/growers" = "Farmers/growers",
    "Hospitals/Childcare/Caring Premises" = "Hospitals/Childcare/Caring Premises",
    "Hotel/bed & breakfast/guest house" = "Hotel/bed & breakfast/guest house",
    "Importers/Exporters" = "Importers/Exporters",
    "Manufacturers/packers" = "Manufacturers/packers",
    "Mobile caterer" = "Mobile caterer",
    "Other catering premises" = "Other catering premises",
    "Pub/bar/nightclub" = "Pub/bar/nightclub",
    "Restaurant/Cafe/Canteen" = "Restaurant/Cafe/Canteen",
    "Retailers - other" = "Retailers - other",
    "Retailers - supermarkets/hypermarkets" = "Retailers - supermarkets/hypermar
kets",
    "School/college/university" = "School/college/university",
    "Takeaway/sandwich shop" = "Takeaway/sandwich shop"),
  selected = "Distributor/Transporters"))),
fluidRow(column(width = 12, plotlyOutput("Plot1")))

),
# Tab for scenario 2
tabItem("scenario2",
  fluidRow(width = 12, h1("Number of Ratings by Years")),
  fluidRow(width = 6,
    selectInput("RatingsSchema2", label = h2("Select Region"),
      choices = list("UK" = "FHRS",
        "Scotland" = "FHIS"),
      selected = "FHRS")),
  fluidRow(column(width = 6,
    selectInput("BusinessType2", label = h2("Select Business Type"),
      choices = list("Distributor/Transporters" = "Distributors/Transporters",
        "Farmers/growers" = "Farmers/growers",
        "Hospitals/Childcare/Caring Premises" = "Hospitals/Childcare/Caring Premises",
        "Hotel/bed & breakfast/guest house" = "Hotel/bed & breakfast/guest house",
        "Importers/Exporters" = "Importers/Exporters",
        "Manufacturers/packers" = "Manufacturers/packers",
        "Mobile caterer" = "Mobile caterer",
        "Other catering premises" = "Other catering premises",
        "Pub/bar/nightclub" = "Pub/bar/nightclub",
        "Restaurant/Cafe/Canteen" = "Restaurant/Cafe/Canteen",
        "Retailers - other" = "Retailers - other",
        "Retailers - supermarkets/hypermarkets" = "Retailers - supermarkets/hypermar
kets",
        "School/college/university" = "School/college/university",
        "Takeaway/sandwich shop" = "Takeaway/sandwich shop"),
      selected = "Distributor/Transporters"))),
    fluidRow(column(width = 12, plotlyOutput("Plot2"), style = "overflow-x:scroll;")
  )),
# Tab for scenario 3
tabItem("scenario3",
  fluidRow(width = 12, h1("Non-compliant Businesses by Region")),
  fluidRow(width = 6,
    selectInput("noncompliant", label = h2("Select Region"),

```

```

        choices = list("UK" = "FHRS",
                      "Scotland" = "FHIS"),
        selected = "FHRS")),
    fluidRow(column(width = 12, plotlyOutput("Plot3"), style = "overflow-x:scroll;")
    ))

)
))
# Define server logic required to provide required plots and tables
server <- function(input, output) {
  # Rendering table for showing the entire dataset
  output$Total_data <- renderDataTable(Total_data, options = list(scrollX = TRUE))
  # Rendering table for showing data by regions
  output$Schema_data <- renderDataTable({schema_type <- input$RatingsSchemaOverview
    print(schema_type)
    if(schema_type == "FHIS"){
      selection <- Scotland_ratings
    }
    else {
      selection <- UK_ratings
    }
    selection})
  # Rendering histogram for scenario 1
  output$Plot1 <- renderPlotly({
    schema_type1 <- input$RatingsSchema
    business_type1 <- input$BusinessType
    print(schema_type1)
    print(business_type1)
    # Creating if-else statement to change output based on region selected
    if(schema_type1 == "FHIS"){
      plot1 <- ggplot(subset(Scotland_ratings, BusinessType == business_type1))+
        geom_histogram(aes(RatingValue), binwidth = 1, stat = "count") +
        facet_grid(BusinessType~.)
    }
    else {
      plot1 <- ggplot(subset(UK_ratings, BusinessType == business_type1))+
        geom_histogram(aes(RatingValue), binwidth = 1, stat = "count") +
        facet_grid(BusinessType~.)
    }
    ggplotly(plot1)
  })
  # Rendering histogram for scenario 2
  output$Plot2 <- renderPlotly({
    schema_type2 <- input$RatingsSchema2
    if(schema_type2 == "FHIS"){
      plot2 <- ggplot(data = Scenario_1_Scotland, aes(x=YEAR)) +
        geom_histogram(binwidth = 1, stat = "count") + labs(x="Year", y="Number of Ratings") +
        geom_density()
    }

    else {
      plot2 <- ggplot(data = Scenario_1_UK, aes(x=YEAR)) +
        geom_histogram(binwidth = 1, stat = "count") + labs(x="Year", y="Number of Ratings") +
        geom_density()
    }
  })
}

```

```

    }

    ggplotly(plot2)
  })

# Rendering histogram for scenario 3
output$Plot3 <- renderPlotly({
  schema_type3 <- input$noncompliant
  business_type2 <- input$BusinessType2
  print(schema_type3)
  print(business_type2)
  if(schema_type3 == "FHIS"){
    plot3 <- ggplot(Scenario_3_Scotland, aes(x=BusinessType))+
    geom_histogram(binwidth = 1, stat = "count") + coord_flip() + labs(x="", y="Number of Business")
  }
  else {
    plot3 <- ggplot(Scenario_3_UK, aes(x = BusinessType))+
    geom_histogram(binwidth = 1, stat = "count") + coord_flip() + labs(x="", y="Number of Business")
  }

  ggplotly(plot3)
})

}

# Running the application
shinyApp(ui = ui, server = server)

```