

## Appendix: A2 logical schema and all other code

### A1: DDL

```
1  -- DDL using SQLite 3 syntax based on https://www.sqlite.org/
2
3  CREATE TABLE Guest (
4      Guest_id INTEGER PRIMARY KEY AUTOINCREMENT,
5      Post_code TEXT NOT NULL,
6      State TEXT NOT NULL,
7      City TEXT NOT NULL,
8      Country TEXT Not NULL,
9      Street_name TEXT NOT NULL,
10     Street_number TEXT NOT NULL,
11     Phone_num_work TEXT,
12     Phone_num_cell TEXT,
13     Phone_num_home TEXT,
14     Email_address TEXT,
15     First_name TEXT NOT NULL,
16     Middle_name TEXT,
17     Last_name TEXT NOT NULL,
18 );
19
20 CREATE TABLE Reservation (
21     Reservation_id INTEGER PRIMARY KEY AUTOINCREMENT,
22     Guest_id INTEGER NOT NULL,
23     -- Smoking_allowed could be boolean, but it would be converted to numeric in SQLite
24     Smoking_preferred INTEGER,
25     Nr_beds_preferred INTEGER,
26     High_or_low_floor_preferred TEXT,
27     -- SQLite does not have a Date data type,
28     -- it can be stored as either Text, Real or Integer
29     -- In this case Integer is chosen for simplicity.
30     -- Date operations work on all of them. "https://www.sqlite.org/datatype3.html"
31     Arrival_date INTEGER NOT NULL,
32     Departure_date INTEGER NOT NULL,
33     Credit_card_num INTEGER NOT NULL,
34     Credit_card_expiry_year INTEGER NOT NULL,
35     credit_card_expiry_month INTEGER NOT NULL,
36     Channel_id INTEGER,
37     Channel_fee REAL,
38     FOREIGN KEY (Guest_id) REFERENCES Guest(Guest_id),
39     FOREIGN KEY (Channel_id) REFERENCES Booking_channel(Channel_id)
40 );
41
42
```

```

43 CREATE TABLE Additional_services (
44     Add_serv_id PRIMARY KEY AUTOINCREMENT,
45     Reservation_id INTEGER,
46     Service_name TEXT NOT NULL,
47     FOREIGN KEY (Reservation_id) REFERENCES Reservation(Reservation_id)
48 );
49
50 CREATE TABLE Stay (
51     Stay_id PRIMARY KEY AUTOINCREMENT,
52     Guest_id INTEGER NOT NULL,
53     -- SQLite does not have a Date data type,
54     -- it can be stored as either Text, Real or Integer
55     -- In this case Integer is chosen for simplicity.
56     -- Date operations work on all of them. "https://www.sqlite.org/datatype3.html"
57     Arrival_date INTEGER NOT NULL,
58     Departure_date INTEGER NOT NULL,
59     Channel_id TEXT,
60     Channel_fee REAL,
61     -- Assuming official invoice number cannot contain
62     -- letters of the alphabet, only numbers.
63     Invoice_number INTEGER,
64     FOREIGN KEY (Guest_id) REFERENCES Guest(Guest_id),
65     FOREIGN KEY (Channel_id) REFERENCES Booking_channel(Channel_id)
66 );
67
68
69 CREATE TABLE Invoice_charges (
70     Charge_item_id INTEGER PRIMARY KEY AUTOINCREMENT,
71     Stay_id INTEGER NOT NULL,
72     Item_name TEXT,
73     Ex_tax_amount REAL,
74     Tax_amount REAL,
75     FOREIGN KEY (Stay_id) REFERENCES Stay(Stay_id)
76 );
77
78
79 CREATE TABLE Invoice_payments (
80     Payment_item_id INTEGER PRIMARY KEY AUTOINCREMENT,
81     Stay_id INTEGER NOT NULL,
82     Payment_type TEXT NOT NULL,
83     Amount REAL NOT NULL,
84     FOREIGN KEY (Stay_id) REFERENCES Stay(Stay_id)
85 );
86
87 CREATE TABLE Booking_channel (
88     Channel_id PRIMARY KEY AUTOINCREMENT,
89     Channel_name TEXT
90 );
91
92 CREATE TABLE Hotel (
93     Hotel_id INTEGER PRIMARY KEY AUTOINCREMENT,
94     Name TEXT NOT NULL,
95     Home_page TEXT NOT NULL,

```

```

96 Post_code TEXT NOT NULL,
97 State TEXT NOT NULL,
98 City TEXT NOT NULL,
99 Street_name TEXT NOT NULL,
100 Street_number TEXT NOT NULL,
101 Primary_phone_number TEXT NOT NULL
102 );
103
104 CREATE TABLE Additional_facilities (
105     Add_facility_id INTEGER PRIMARY KEY AUTOINCREMENT,
106     Add_facility_name TEXT NOT NULL,
107     Add_facility_cost float NOT NULL,
108     Hotel_id INTEGER,
109     FOREIGN KEY (Hotel_id) REFERENCES Hotel(Hotel_id)
110 );
111
112 CREATE TABLE Room (
113     Hotel_id INTEGER NOT NULL,
114     Room_name_or_number TEXT NOT NULL,
115     floor INTEGER NOT NULL,
116     Nr_beds INTEGER NOT NULL,
117     -- Smoking_allowed could be boolean, but it would be converted to numeric in SQLite
118     Smoking_allowed INTEGER NOT NULL,
119     PRIMARY KEY (Hotel_id, Room_name_or_number),
120     FOREIGN KEY (Hotel_id) REFERENCES Hotel(Hotel_id)
121 );
122
123 CREATE TABLE Room_allocation (
124     Hotel_id INTEGER NOT NULL,
125     Room_name_or_number INTEGER NOT NULL,
126     -- SQLite does not have a Date data type,
127     -- it can be stored as either Text, Real or Integer
128     -- In this case Integer is chosen for simplicity.
129     -- Date operations work on all of them. "https://www.sqlite.org/datatype3.html"
130     Date INTEGER NOT NULL,
131     Channel_id INTEGER DEFAULT NULL,
132     Reservation_id INTEGER DEFAULT NULL,
133     Stay_id INTEGER DEFAULT NULL,
134     PRIMARY KEY (Hotel_id, Room_name_or_number, Date),
135     FOREIGN KEY (Channel_id) REFERENCES Booking_channel(Channel_id),
136     FOREIGN KEY (Reservation_id) REFERENCES Reservation(Reservation_id),
137     FOREIGN KEY (Stay_id) REFERENCES Stay(Stay_id),
138     FOREIGN KEY (Hotel_id, Room_name_or_number)
139     REFERENCES Room(Hotel_id, Room_name_or_number)
140     -- Ensure a room is only allocated to one purpose.
141     CONSTRAINT Only_one_key CHECK
142     ((Channel_id NOT NULL
143     OR Reservation_id NOT NULL
144     OR Stay_id NOT NULL)
145     AND NOT (Channel_id NOT NULL AND Reservation_id NOT NULL)
146     AND NOT (Reservation_id NOT NULL AND Stay_id NOT NULL)
147     AND NOT (Channel_id NOT NULL AND Stay_id NOT NULL))
148 );

```

## A1 Queries

```
1  -- 1. The total spent for the customer for a particular stay (checkout invoice).
2
3  -- Assuming total spent includes taxes paid.
4  SELECT S.Guest_id, S.Stay_id, SUM(IP.Amount) FROM Stay S
5  INNER JOIN Invoice_payments IP ON IP.Stay_id = S.Stay_id
6  -- To specify a particular stay:
7  WHERE S.Stay_id = 12345;
8
9
10 -- 2. The most valuable customers in (a) the last two months,
11 -- (b) past year and (c) from the beginning of the records.
12
13 -- a
14
15 -- Assuming last two months means to count all
16 -- stays that started in the last two months.
17 -- Assuming value means total spent including taxes paid
18 SELECT G.Guest_id, SUM(IP.Amount) AS Total_spent FROM Stay S
19 INNER JOIN Guest G ON G.Guest_id = S.Guest_id
20 INNER JOIN Invoice_payments IP ON IP.Stay_id = S.Stay_id
21 -- Based on https://www.sqlite.org/lang\_datefunc.html
22 WHERE date(S.Arrival_date) >= date('now', "-2 months")
23 GROUP BY G.Guest_id
24 ORDER BY Total_spent DESC LIMIT 10;
25
26 -- b
27
28 SELECT G.Guest_id, SUM(IP.Amount) AS Total_spent FROM Stay S
29 INNER JOIN Guest G ON G.Guest_id = S.Guest_id
30 INNER JOIN Invoice_payments IP ON IP.Stay_id = S.Stay_id
31 -- Based on https://www.sqlite.org/lang\_datefunc.html,
32 -- Assuming by past year, the last 365 days are meant.
33 -- If all stays since the start of the year are meant,
34 -- "-1 year" would be replaced by "start of year".
35 WHERE date(S.Arrival_date) >= date('now', "-1 year")
36 GROUP BY G.Guest_id
37 ORDER BY Total_spent DESC LIMIT 10;
38
39 -- c
40
41 SELECT G.Guest_id, SUM(IP.Amount) AS Total_spent FROM Stay S
42 INNER JOIN Guest G ON G.Guest_id = S.Guest_id
43 INNER JOIN Invoice_payments IP ON IP.Stay_id = S.Stay_id
44 GROUP BY G.Guest_id
45 ORDER BY Total_spent DESC LIMIT 10;
46
47
48 -- 3. Which are the top countries where our customers come from?
49
50 SELECT Country, COUNT(Guest_id) AS Frequency FROM Guest
51 GROUP BY Country
```

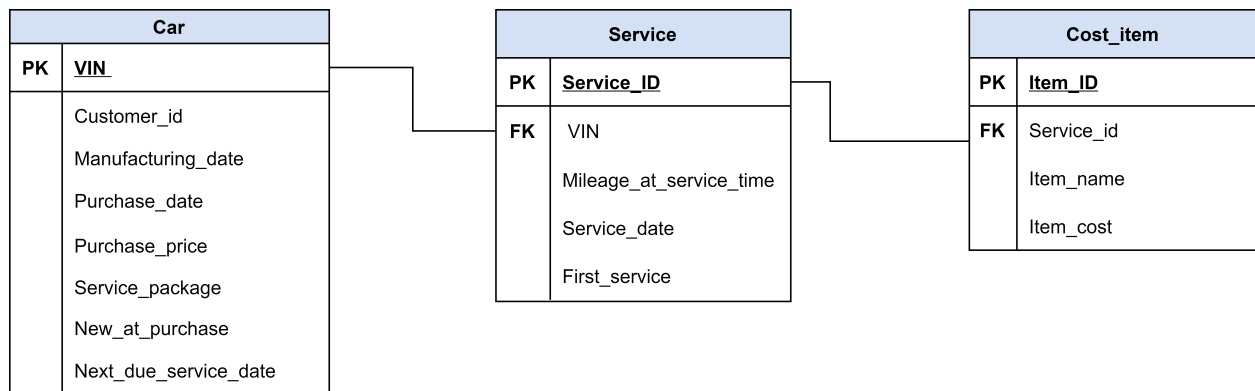
```

52 ORDER BY Frequency DESC LIMIT 10;
53
54
55 -- 4. How much did the hotel pay in referral fees for each
56 -- of the platforms that we have contracted with?
57
58 -- Assuming arrival date on the last day of the month means
59 -- channel fees are charged for that month still.
60 SELECT C.Channel_id, C.Channel_name, SUM(S.Channel_fee) AS Total_fees
61 FROM Booking_channel C
62 INNER JOIN Stay S on S.Channel_id=C.Channel_id
63 -- Only include stays that finished on or before the end of the last month
64 WHERE date(S.Arrival_date) <= date("now","start of month","-1 day")
65 GROUP BY C.Channel_name;
66
67
68 -- 5. What is the utilization rate for each hotel (that is the average
69 -- billable days of a hotel specified as the average utilization
70 -- of room bookings for the last 12 months)?
71
72 SELECT Name, Hotel_id, AVG(Utilisation) FROM
73 (
74     SELECT H.Name, R.Hotel_id, R.Room_name_or_number,
75     COUNT(RA.date)/365.0 AS Utilisation
76     FROM Room_allocation RA
77     -- Line break below to ensure fit on the page.
78     INNER JOIN Room R
79     ON RA.(Hotel_id, Room_name_or_number) = R.(Hotel_id, Room_name_or_number)
80     -- Only joining to retrieve hotel name:
81     INNER JOIN Hotel H ON R.Hotel_id = H.Hotel_id
82     WHERE date(RA.Date) >= date("now","-1 year")
83     -- To ensure that only stays count as utilised days:
84     AND Stay_id NOT NULL
85     GROUP BY R.Hotel_id, R.Room_name_or_number
86 )
87 GROUP BY Hotel_id;
88
89
90 -- 6. Calculate the Customer Value in terms of total spent
91 -- for each customer before the current booking.
92
93 -- Assuming total spent includes taxes paid
94 -- Stay only includes previous stays, so no need to filter anything.
95 SELECT S.Guest_id, SUM(IP.Amount) AS Total_spent FROM Stay S
96 INNER JOIN Invoice_payments IP ON IP.Stay_id = S.Stay_id
97 GROUP BY S.Guest_id
98 -- Only include Guest_ids that also have reservations.
99 HAVING S.Guest_id IN (SELECT DISTINCT Guest_id FROM Reservation);
100 -- Guest_id could be specified if looking for specific
101 -- HAVING S.Guest_id = 12345

```

## A2

### Logical schema:



### Queries:

```
1  -- How many customers have stopped bringing their cars
2  -- after the first encounter with the dealer?
3
4  -- Assuming Customers have only one car
5  SELECT COUNT(*) AS Churns FROM (
6      -- Subquery yields table with the number of times a customer brought
7      -- their car to service and when the next service would be due,
8      -- which can be analysed further to reveal more about
9      -- the likelihood of coming to service more times.
10     SELECT C.VIN, COUNT(S.Service_id), C.Next_due_service_date AS Total_services
11     FROM Car C
12     INNER JOIN Service S ON C.VIN=S.VIN
13     GROUP BY C.VIN
14 )
15 -- Only consider those customers churns, whose due service dates have already passed.
16 WHERE date(Next_due_service_date)<date("now")
17 AND Total_services = 1;
18
19
20 -- What is the relationship between the price of the service
21 -- and the age of the car in terms of
22 --(a)actual car age (e.g., mileage) and b) time with the current owner?
23
24 -- One query combined to extract information for both questions:
25 -- (a) can be answered via Mileage_at_service_time and
26 -- (b) can be answered with days_since_purchase
27
28
29 SELECT SUM(Costs.Item_cost), S.Mileage_at_service_time,
30 (julianday(S.service_date)-julianday(C.Purchase_date)) AS days_since_purchase
31 FROM Service S
32 INNER JOIN Car C ON C.VIN=S.VIN
```

```

33 INNER JOIN Cost_item Costs ON Costs.Service_id=S.Service_id
34 GROUP BY Service_id

```

## Part B

```

1  # Find all country directories
2  dirs <- list.dirs("./partB_data_files/")
3
4  # Drop root directory
5  dirs <- dirs[-1]
6
7  # Extract file names from every directory and save them
8  # in a named list of character vectors
9  files <- sapply(dirs,list.files,pattern="*.xlsx",USE.NAMES=T,simplify = F)
10
11 # Create empty list
12 datalist = list()
13
14
15 # Loop through directories and files within every directory
16 for (countrydir in names(files)) {
17   for (file in files[[countrydir]]) {
18
19     # Read in xlsx file, suppressing "New names" messages
20     suppressMessages(excel<-countrydir %>% paste0("/",file) %>% read_xlsx())
21     # Record sheet dimensions to use as reference points
22     rowcount <- dim(excel)[1]
23     colcount <- dim(excel)[2]
24     # Extract main table, removing empty row and column in the
25     # second row and column of the resulting table
26     data<-excel[c(5,seq(7,rowcount-3)),c(1,seq(3,colcount))]
27     # Use first row as header
28     colnames(data) <- data[1,]
29     # Drop header names from the dataframe
30     data <- data[-1,]
31     # Rename first column to year
32     colnames(data)[1] <- "year"
33     # Convert to long format
34     data <- data %>% pivot_longer(2:length(data[1,]),
35                                   names_to = "product",
36                                   values_to = "value")
37
38     # Add flow and country from the sheet
39     data <- data %>% add_column(flow=excel[2,3] %>% unlist %>% unname,
40                                   country=excel[4,3] %>% unlist %>% unname)
41     # Reorder columns to fit format requirements
42     data <- data %>% select(country,year,flow,product,value)
43     # Add resulting dataframe to the list
44     datalist[[file]] <- data
45   }
46 }

```

```

47
48 # Bind dataframes together
49 fulldata <- do.call(bind_rows, datalist)
50
51 # Change to numeric data type and simultaneously coerce "." values to NA
52 fulldata$value <- fulldata$value %>% as.numeric

```

```
## Warning in fulldata$value %>% as.numeric: NAs introduced by coercion
```

```
1 fulldata
```

```

## # A tibble: 573,950 x 5
##   country year flow product value
##   <chr>   <chr> <chr>   <chr>   <dbl>
## 1 Albania 1971 Imports Additives/blending components    0
## 2 Albania 1971 Imports Anthracite                      NA
## 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
## # ... with 573,940 more rows

```

```

1 # Count total rows in the dataframe and display
2 (totalrecords <- fulldata %>% nrow())

```

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

```

1 # Group data by product and print number of records.
2 productrecords <- fulldata %>% group_by(product) %>% summarise(records=n())
3 # In case of every product, there are 8830 records.
4 (productrecords %>% count(records))

```

```

## # A tibble: 1 x 2
##   records    n
##   <int> <int>
## 1    8830    65

```

## Part C

```

1 # Root website link
2 link<-"https://data.food.gov.uk/catalog/datasets/38dd8d6a-5ab1-4f50-b753-ab33288e3200"
3
4 # Extract data download links from html page
5 xmllinks<-read_html(link) %>% html_nodes(".o-dataset-distribution--link") %>%
6   html_attr("href")

```



```

7  # Filter to only .xml files (to remove a link that points to webpage index)
8  xmllinks<-xmllinks[xmllinks %>% str_detect(".xml$")]
9  # Filter to English language files (some were duplicates in Welsh)
10 xmllinks<-xmllinks[xmllinks %>% str_detect("en-GB")]
11
12 # Create directory to store XMLs.
13 dir.create("xmldata",showWarnings = F)
14
15 # Create empty list to store dataframes
16 xmldatalist=list()
17
18 # Create SQLite connection
19 con<-RSQLite::dbConnect(RSQLite::SQLite(),"ratings.db")
20
21 # Around 10 minutes runtime excl. download
22 if (!RSQLite::dbExistsTable(con,"fact_Rating")||!file.exists("C_output.rds")) {
23   for (i in 1:length(xmllinks)) {
24
25     # Get link based on loop variable.
26     link <- xmllinks[i]
27
28     # Create file path for file to be saved
29     filename <- paste0("xmldata/",link %>% str_extract("(?<=/)[~/]+$"))
30
31     # Download file if the file does not yet exist.
32     if (!file.exists(filename)){
33       download.file(link,filename)
34     }
35
36     # Read xml file and find all EstablishmentDetail nodes.
37     nodes <- read_xml(filename) %>% xml_find_all("//EstablishmentDetail")
38
39     # Extract ID column to be used as an index later.
40     ids<-xml_child(nodes,"FHRSID") %>% xml_text %>% trimws
41
42     # Find the amount of fields for every node that are populated
43     nodelengths<-xml_length(nodes)
44
45     # Find the names of the populated fields for every node
46     nodenames<-xml_children(nodes) %>% xml_name
47
48     # Find the values of the populated fields for every node
49     nodevalues<-xml_children(nodes) %>% xml_text %>% trimws
50
51     # Create a long dataframe with node names and values, indexed by
52     # repeating the index node-length times.
53     # After that, pivot to a wide-format.
54     # Finally, add the values of the nested fields (Geocode and Scores)
55     # separately, as they would have been merged into one column otherwise.
56     df<-tibble(ID=rep(ids,nodelengths),
57                variable=nodenames,
58                values=nodevalues) %>%
59     pivot_wider(names_from="variable",values_from="values") %>%

```

```

60     add_column(Longitude=xml_child(nodes,search="Geocode") %>%
61         xml_child("Longitude") %>% xml_text,
62         Latitude=xml_child(nodes,search="Geocode") %>%
63         xml_child("Latitude") %>% xml_text,
64         Hygiene=xml_child(nodes,search="Scores") %>%
65         xml_child("Hygiene") %>% xml_text,
66         Structural=xml_child(nodes,search="Scores") %>%
67         xml_child("Structural") %>% xml_text,
68         ConfidenceInManagement=xml_child(nodes,search="Scores") %>%
69         xml_child("ConfidenceInManagement") %>% xml_text)
70
71     # Remove duplicate and unnecessary columns
72     df$Scores<-NULL
73     df$Geocode<-NULL
74     df$ID<-NULL
75
76     # Progress tracker for the parsing process, prints a message every 10 files.
77     xmldataalist[[filename]]<-df
78     if (i %% 10 == 0){
79         paste0(i, " files processed from a total of ", length(xmllinks),".") %>%
80         print()
81     }
82
83 }
84 print("XML parsing finished.")
85
86 # Binds data together and detects column types.
87 ratingsdata<-do.call(bind_rows,xmldataalist) %>% type_convert()
88
89 # Decode HTML encoded characters and remove tags, newlines and tags.
90 ratingsdata$RightToReply<-ifelse(is.na(ratingsdata$RightToReply),
91     NA,
92     ratingsdata$RightToReply[
93         !is.na(ratingsdata$RightToReply)] %>%
94         textutils::HTMLdecode() %>%
95         str_remove_all("</?p>") %>%
96         str_remove_all("[\n\t]") %>%
97         trimws)
98
99 # Convert to dimensional model
100 dim_LocalAuthority<-ratingsdata %>% select(LocalAuthorityCode,
101     LocalAuthorityName,
102     LocalAuthorityWebSite,
103     LocalAuthorityEmailAddress) %>%
104     distinct
105 dim_Date<-ratingsdata %>%
106     select(RatingDate) %>%
107     mutate(year = lubridate::year(RatingDate),
108         month = lubridate::month(RatingDate),
109         day = lubridate::day(RatingDate)) %>%
110     distinct()
111
112 dim_BusinessType<-ratingsdata %>%

```

```

113     select(BusinessType,BusinessTypeID) %>%
114     distinct()
115
116     dim_RatingTypes<-ratingsdata %>%
117     select(RatingKey,SchemeType,RatingValue) %>%
118     distinct
119
120     fact_Rating <- ratingsdata %>% select(-c(LocalAuthorityName,
121                                             LocalAuthorityWebSite,
122                                             LocalAuthorityEmailAddress,
123                                             BusinessType,
124                                             RatingValue,
125                                             SchemeType))
126
127     # Write to SQLite database
128     RSQLite::dbWriteTable(con,"fact_Rating",fact_Rating)
129     RSQLite::dbWriteTable(con,"dim_Date",dim_Date)
130     RSQLite::dbWriteTable(con,"dim_LocalAuthority",dim_LocalAuthority)
131     RSQLite::dbWriteTable(con,"dim_BusinessType",dim_BusinessType)
132     RSQLite::dbWriteTable(con,"dim_RatingTypes",dim_RatingTypes)
133
134     # Save ratingsdata as RDS
135     saveRDS(ratingsdata,"C_output.rds")
136 }

```

```

## [1] "10 files processed from a total of 376."
## [1] "20 files processed from a total of 376."
## [1] "30 files processed from a total of 376."
## [1] "40 files processed from a total of 376."
## [1] "50 files processed from a total of 376."
## [1] "60 files processed from a total of 376."
## [1] "70 files processed from a total of 376."
## [1] "80 files processed from a total of 376."
## [1] "90 files processed from a total of 376."
## [1] "100 files processed from a total of 376."
## [1] "110 files processed from a total of 376."
## [1] "120 files processed from a total of 376."
## [1] "130 files processed from a total of 376."
## [1] "140 files processed from a total of 376."
## [1] "150 files processed from a total of 376."
## [1] "160 files processed from a total of 376."
## [1] "170 files processed from a total of 376."
## [1] "180 files processed from a total of 376."
## [1] "190 files processed from a total of 376."
## [1] "200 files processed from a total of 376."
## [1] "210 files processed from a total of 376."
## [1] "220 files processed from a total of 376."
## [1] "230 files processed from a total of 376."
## [1] "240 files processed from a total of 376."
## [1] "250 files processed from a total of 376."
## [1] "260 files processed from a total of 376."
## [1] "270 files processed from a total of 376."
## [1] "280 files processed from a total of 376."

```

```
## [1] "290 files processed from a total of 376."
## [1] "300 files processed from a total of 376."
## [1] "310 files processed from a total of 376."
## [1] "320 files processed from a total of 376."
## [1] "330 files processed from a total of 376."
## [1] "340 files processed from a total of 376."
## [1] "350 files processed from a total of 376."
## [1] "360 files processed from a total of 376."
## [1] "370 files processed from a total of 376."
## [1] "XML parsing finished."
```

```
##
## -- Column specification -----
## cols(
##   .default = col_character(),
##   FHRSID = col_double(),
##   BusinessTypeID = col_double(),
##   RatingDate = col_date(format = ""),
##   NewRatingPending = col_logical(),
##   Longitude = col_double(),
##   Latitude = col_double(),
##   Hygiene = col_double(),
##   Structural = col_double(),
##   ConfidenceInManagement = col_double()
## )
## i Use 'spec()' for the full column specifications.
```

```
1 # Disconnect from database
2 RSQLite::dbDisconnect(con)
```

## Part D

```
1 # Read in dataframe from part C
2 ratingsdata<-readRDS("C_output.rds")
3
4 paste(object.size(ratingsdata) / 1048576,"MB") %>% print
```

```
## [1] "217.903747558594 MB"
```

```
1 # As the dataframe is only about 218 MB, loading it into memory
2 # is easily possible on any modern computer
3 # Keeping the object in memory and running dplyr code
4 # proved faster than SQL in practice.
5
6 # SQL code provided for illustration throughout the code.
7 # Conencting to database
8 # db <- dbConnect(SQLite(), 'ratings.db')
9
10 # getting unique business type and Id for filters
11 BusinessTypeId <- ratingsdata %>% select(BusinessType,BusinessTypeID) %>% distinct()
```

```

12 # BusinessTypeId <- dbGetQuery(db, 'SELECT DISTINCT
13 #                               BusinessType, BusinessTypeID FROM dim_BusinessType;')
14
15
16 ui <- dashboardPage(
17
18   dashboardHeader(
19     title = "Ratings dashboard"
20   ),
21   #Sidebar definition
22   dashboardSidebar(
23     sidebarMenu(
24       menuItem("Ratings over time",
25         tabName = "Tab1",
26         icon = icon("history")),
27       menuItem("Sub-scores by ratings",
28         tabName = "Tab2",
29         icon = icon("project-diagram")),
30       menuItem("Top local authorities by rating",
31         tabName = "Tab3",
32         icon = icon("map-marked"))
33     )
34   ),
35   dashboardBody(
36     tabItems(
37
38       # First tab
39       tabItem(tabName = "Tab1",
40         fluidRow(
41           box(width=12,
42             h2("Density of specific rating values
43               over time based on business type"),
44             br(),"
45             Lower ratings (0-2) gaining more
46             weight in the last few years.")
47         ),
48         fluidRow(
49           box(title = "Rating Value slider", width=6,
50             sliderInput(inputId = 'RatingValues', 'Rating Values:', 0, 5, 2)),
51           box(title = "Business Type select", width=6,
52             selectInput(
53               inputId = 'BusinessType',
54               label = 'Business Types:',
55               choices = BusinessTypeId$BusinessType,
56               selected = "Restaurant/Cafe/Canteen"
57             )),
58         fluidRow(
59           box(width=12,
60             #Plot Output
61             plotlyOutput(
62               outputId = 'Play1'
63             )
64           )),

```

```

65
66     # Second tab
67     tabItem(tabName = "Tab2",
68         fluidRow(
69             box(width=12,h2("Sub-score (penalties)
70                 distributions given rating value"),
71             br(),
72             "Most penalties (highest sub-scores) awarded
73             for lack of confidence in management."
74         ),
75         fluidRow(
76             # Business type distribution plot
77             box(title = "Rating Value slider",width=12,
78                 sliderInput(
79                     inputId = 'RatingValues2',
80                     'Rating Values:',
81                     0,
82                     5,
83                     1,
84                 )),
85             fluidRow(
86                 box(width=12,
87                     #Plot Output
88                     plotlyOutput(outputId = 'Play2',
89                         height = "500px",
90                         width="100%")
91                 )
92             )),
93
94     # Third tab
95     tabItem(tabName = "Tab3",
96         fluidRow(
97             box(width=12,
98                 h2("Local authorities with highest
99                 proportion of a rating value"),
100             br(),
101             "Waltham Forest has the highest proportion (4%)
102             of establishments with a rating of 0,
103             nearly 4 times as many as the runner-up."
104         ),
105         fluidRow(
106             box(width=12,title="Rating Value Slider",
107                 sliderInput(
108                     inputId = 'RatingValues3',
109                     'Rating Values:', 0, 5, 0)
110             )),
111         fluidRow(
112             box(width=12,
113                 #Plot Output
114                 plotlyOutput(
115                     outputId = 'Play3',height = "500px",width="100%")
116             ))
117     )
118 )

```

```

119   )
120 )
121
122
123 server <- function(input, output, session) {
124
125   output$Play1 <- renderPlotly(
126     {
127       df <- ratingsdata %>%
128         filter(SchemeType=="FHRs"& BusinessType==input$BusinessType) %>%
129         select(RatingValue,RatingDate) %>%
130         mutate(year=lubridate::year(RatingDate)) %>%
131         count(year,RatingValue) %>%
132         left_join(ratingsdata %>%
133           filter(SchemeType=="FHRs" &
134             BusinessType==input$BusinessType) %>%
135           mutate(year=lubridate::year(RatingDate)) %>%
136           count(year) %>% rename(yearlycount=n),
137           by="year") %>%
138         mutate(ratings_percentage=n/yearlycount) %>%
139         filter(RatingValue==input$RatingValues)
140
141       # df <- dbGetQuery(db,
142       #   statement =
143       #     'SELECT D.ratingValue,
144       #     COUNT(D.ratingValue)/CAST(yearly_count AS REAL) AS ratings_percentage,
145       #     Date.Year FROM fact_rating F
146       #     INNER JOIN dim_RatingTypes D ON F.RatingKey=D.RatingKey
147       #     INNER JOIN dim_BusinessType B ON B.BusinessTypeID=F.BusinessTypeID
148       #     INNER JOIN dim_Date Date ON Date.RatingDate=F.RatingDate
149       #     LEFT JOIN (
150       #       SELECT Date.Year, COUNT(D.ratingValue) AS yearly_count
151       #       FROM fact_rating F
152       #       INNER JOIN dim_RatingTypes D ON F.RatingKey=D.RatingKey
153       #       INNER JOIN dim_BusinessType B ON B.BusinessTypeID=F.BusinessTypeID
154       #       INNER JOIN dim_Date Date ON Date.RatingDate=F.RatingDate
155       #       WHERE D.SchemeType = $Scheme
156       #       AND B.BusinessType = $Btype
157       #       GROUP BY Date.Year) DT ON DT.Year=Date.Year
158       #     WHERE D.SchemeType = $Scheme
159       #     AND B.BusinessType = $Btype
160       #     AND D.ratingValue = $Rating
161       #     GROUP BY D.ratingValue, Date.Year;'
162       # ,list(Scheme="FHRs",Btype=input$BusinessType,Rating=input$RatingValues))
163
164       plt1 <- plot_ly(df,x=~year,y=~ratings_percentage) %>%
165         layout(title="Proportion of rating value by year and Business Type") %>%
166         add_lines
167       plt1
168     }
169   )
170
171   # 2. This navigation option describes the sub-score distributions of every rating value.

```

```

172
173 output$Play2 <- renderPlotly(
174   {
175     df <- ratingsdata %>%
176       filter(RatingValue==input$RatingValues2) %>%
177       count(Hygiene) %>%
178       add_column(Score="Hygiene") %>%
179       rename(level=Hygiene) %>%
180       bind_rows(ratingsdata %>%
181         filter(RatingValue==input$RatingValues2) %>%
182         count(Structural) %>%
183         add_column(Score="Structural") %>%
184         rename(level=Structural)
185       ) %>%
186       bind_rows(ratingsdata %>%
187         filter(RatingValue==input$RatingValues2) %>%
188         count(ConfidenceInManagement) %>%
189         add_column(Score="ConfidenceInManagement") %>%
190         rename(level=ConfidenceInManagement)
191       ) %>%
192       rename(Frequency=n) %>%
193       drop_na()
194
195     # df<-dbGetQuery(db,
196     #   statement=
197     #   "SELECT D.RatingValue, 'Hygiene' AS Score,
198     #     F.Hygiene AS level, COUNT(F.Hygiene) AS Frequency
199     #   FROM fact_rating F
200     #   INNER JOIN dim_RatingTypes D ON F.RatingKey=D.RatingKey
201     #   INNER JOIN dim_BusinessType B ON B.BusinessTypeID=F.BusinessTypeID
202     #   WHERE D.RatingValue = $Rating
203     #   AND F.Hygiene NOT NULL
204     #   GROUP BY F.Hygiene
205     #   UNION
206     #   SELECT D.RatingValue, 'Structural' AS Score,
207     #     F.Structural AS level, COUNT(F.Structural) AS Frequency
208     #   FROM fact_rating F
209     #   INNER JOIN dim_RatingTypes D ON F.RatingKey=D.RatingKey
210     #   INNER JOIN dim_BusinessType B ON B.BusinessTypeID=F.BusinessTypeID
211     #   WHERE D.RatingValue = $Rating
212     #   AND F.Structural NOT NULL
213     #   GROUP BY F.Structural
214     #   UNION
215     #   SELECT D.RatingValue, 'ConfidenceInManagement' AS Score,
216     #     F.ConfidenceInManagement AS level,
217     #     COUNT(F.ConfidenceInManagement) AS Frequency
218     #   FROM fact_rating F
219     #   INNER JOIN dim_RatingTypes D ON F.RatingKey=D.RatingKey
220     #   INNER JOIN dim_BusinessType B ON B.BusinessTypeID=F.BusinessTypeID
221     #   WHERE D.RatingValue = $Rating
222     #   AND F.ConfidenceInManagement NOT NULL
223     #   GROUP BY F.ConfidenceInManagement;",
224     #   list(Rating=input$RatingValues2))
225

```



```

226
227
228
229 plt2<-plot_ly(df %>%
230               filter(Score=="Hygiene"),
231               x=~level,y=~Frequency) %>%
232   add_bars(name="Hygiene")
233
234 plt3<-plot_ly(df %>%
235               filter(Score=="Structural"),
236               x=~level, y=~Frequency) %>%
237   add_bars(name="Structural")
238
239 plt4<-plot_ly(df %>%
240               filter(Score=="ConfidenceInManagement"),
241               x=~level,y=~Frequency) %>%
242   add_bars(name="Confidence In Management")
243
244 subplot(plt2,plt3,plt4,nrows=1,shareY=T) %>%
245   layout(title="Sub-score values (lower is better)")
246 }
247 )
248
249 # 3. Top ten Authority for selected rating value
250
251 # This chart will render the plot showing top-10
252 # local Authorities based on rating at each level
253 output$Play3 <- renderPlotly(
254   {
255     df<-ratingsdata %>%
256       filter(SchemeType=="FHRS") %>%
257       select(LocalAuthorityName,RatingValue) %>%
258       group_by(LocalAuthorityName) %>%
259       count(RatingValue) %>%
260       left_join(ratingsdata %>%
261                 filter(SchemeType=="FHRS") %>%
262                 group_by(LocalAuthorityName) %>%
263                 summarize(All_reviews=n()),
264                 by="LocalAuthorityName") %>%
265       mutate(Rating_percentage=n/All_reviews) %>%
266       filter(RatingValue==input$RatingValues3) %>%
267       arrange(desc(Rating_percentage)) %>%
268       head(10)
269
270     df <- dbGetQuery(db,
271                      statement=
272                        "SELECT D.ratingValue,
273                          COUNT(D.ratingValue)/CAST(LocalAuthorityCount AS REAL)
274                          AS Rating_percentage,
275                          L.LocalAuthorityName
276                          FROM fact_rating F
277                          INNER JOIN dim_RatingTypes D
278                          ON F.RatingKey=D.RatingKey
279                          INNER JOIN dim_LocalAuthority L

```

```

280         ON L.LocalAuthorityCode=F.LocalAuthorityCode
281     LEFT JOIN (
282         SELECT L.LocalAuthorityName,
283             COUNT(D.ratingValue) AS LocalAuthorityCount
284         FROM fact_rating F
285             INNER JOIN dim_RatingTypes D
286             ON F.RatingKey=D.RatingKey
287             INNER JOIN dim_LocalAuthority L
288             ON L.LocalAuthorityCode=F.LocalAuthorityCode
289             WHERE D.SchemeType = $Scheme
290             GROUP BY L.LocalAuthorityName) DT
291     ON DT.LocalAuthorityName=L.LocalAuthorityName
292     WHERE D.SchemeType = $Scheme
293     AND D.ratingValue = $Rating
294     GROUP BY L.LocalAuthorityName, D.RatingValue
295     ORDER BY Rating_percentage DESC
296     LIMIT 10;",
297     list(Scheme="FHRS",Rating=input$RatingValues3))
298
299     plt4 <- plot_ly(df,x=~LocalAuthorityName,y=~Rating_percentage) %>%
300       add_bars()
301     plt4 %>% layout(xaxis = list(categoryorder = "total descending"),
302                     title="Local Authorities with highest
303                     percentage of given rating value")
304   }
305 )
306 }
307
308 # Complete app with UI and server components
309 # (commented out to allow for running through when knitting)
310 # shinyApp(ui, server)
311 # dbDisconnect(db)

```

## Screenshots of dashboard:

