

# IB9HPO\_9

5588654, 5577965, 5579058, 5587354, 5519743, 5588060

2024-03-02

## Table of contents

<b>1</b>	<b>Database Design and Implementation</b>	<b>2</b>
1.1	Assumptions . . . . .	2
1.2	Conceptual Design: Entities and Relationships . . . . .	2
1.3	Logical Design . . . . .	5
1.4	Physical Design . . . . .	6
<b>2</b>	<b>Data Generation and Management</b>	<b>11</b>
2.1	Synthethic Data Generation . . . . .	11
2.2	Data Import and Quality Assurance . . . . .	28
<b>3</b>	<b>Data Pipeline Generation</b>	<b>44</b>
3.1	Introduction . . . . .	44
3.2	GitHub Repository Structure . . . . .	44
3.3	GitHub Actions . . . . .	45
3.3.1	Trigger conditions . . . . .	45
3.3.2	Automation Tasks . . . . .	45
<b>4</b>	<b>Data Analysis and Reporting</b>	<b>46</b>
4.1	Monthly Sales Trend Analysis by Value and Volume, from 2022 to 2023 . . . . .	46
4.2	Top 10 Products and Their Rating . . . . .	48
4.3	Spending by Membership Type . . . . .	49
4.4	Customer Queries Analysis . . . . .	50
4.5	Payment Methods . . . . .	51
4.6	Insight and Recommendation . . . . .	52
4.6.1	Insights: . . . . .	52
4.6.2	Recommendation: . . . . .	53
<b>5</b>	<b>Limitation and Future Implications</b>	<b>53</b>
<b>6</b>	<b>Conclusion</b>	<b>53</b>

This project presents the components and structure of a database system that can simulate the basic functional operation of the real-world e-commerce platform. In general, the generation of e-business systems in this project consists of four parts, starting from database design and implementation to data analysis and reporting.

# 1 Database Design and Implementation

Firstly, this project designs the Entity relationship diagram (ERD) and logical diagram of e-commerce database. Later, through mapping a well-designed relationship set and using the rules of 3NF, the project utilizes SQL to convert the ERD to the physical schema of the database.

## 1.1 Assumptions

The design of database system corresponds to the below assumptions:

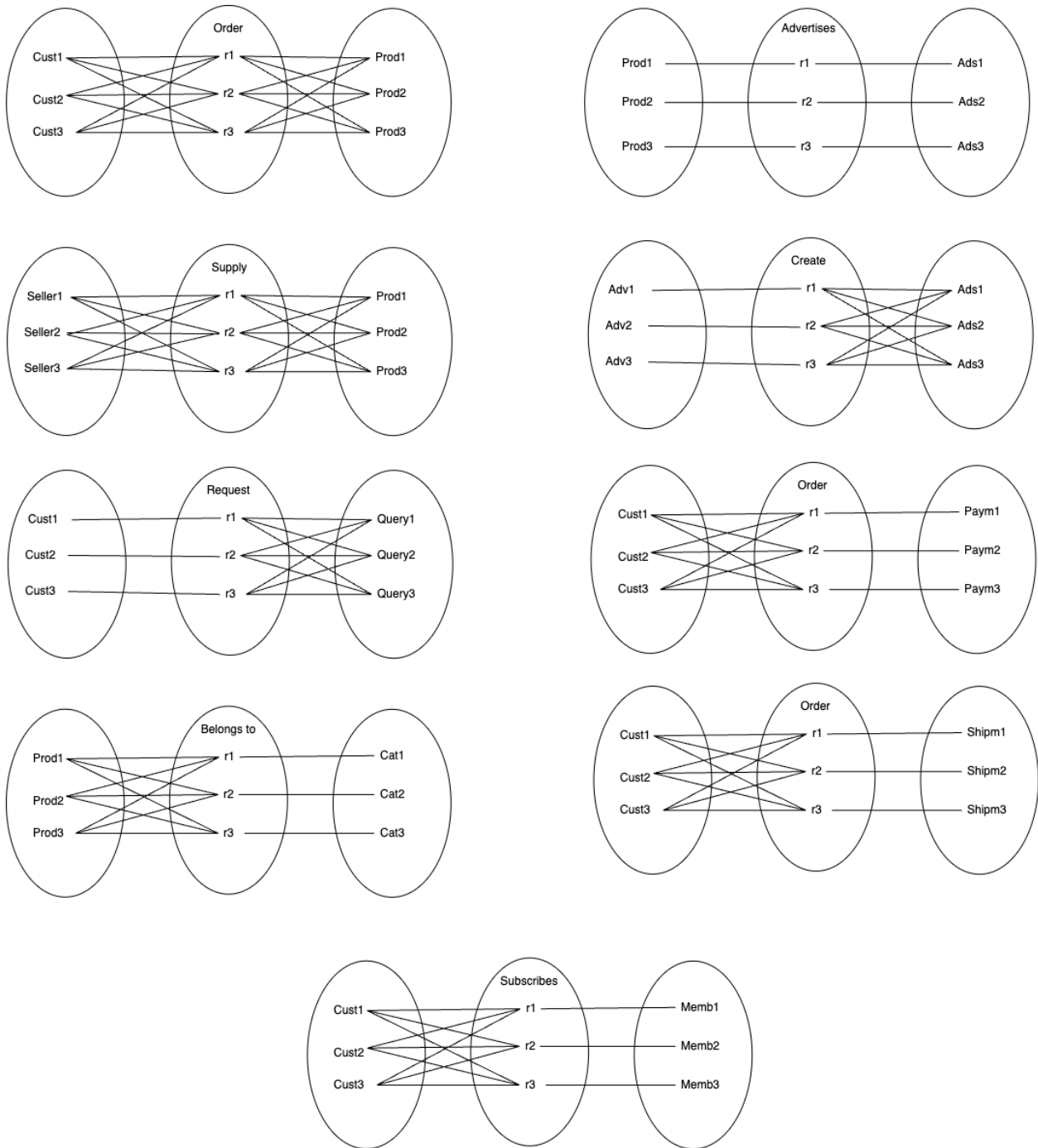
1. The customers are allowed to log in, browse and order the products simultaneously.
2. The customers can buy the different membership service if they want.
3. The customers can make a payment for their purchases.
4. The customers can track their order details through a unique order.
5. The customers can query for support if they have difficulties in cases related to any problems or delays.
6. One customer can hold just one address for both shipping and billing.
7. Every product should own a unique ID, many reviews, and belong to the corresponding category in e-commerce system.
8. The system can track each order details through a unique order ID.
9. The suppliers can supply many kinds of products that the customers may want.
10. The e-commerce platform can cooperate with multiple advertisers to create advertisements on multiple products because advertisement profit is one of the top revenue sources of e-commerce.
11. One product is only advertised in one advertisement.
12. The price of the product has already covered the required tax in the UK.
13. The date of shipment, order, and payment is automatically recorded by system, and is not manually input by human.

## 1.2 Conceptual Design: Entities and Relationships

1. Customer—Product Each customer can order many products, and each product can be ordered by many customers. Thus, cardinality is M: N.
2. Customer—Shipment Each customer can have many shipments, and each shipment can just be associated with one customer. Thus, cardinality is M: 1.
3. Customer—Purchase Each customer can own many purchases, and each purchase can be associated with one customer. Thus, cardinality is M: 1.
4. Supplier—Product Each supplier can supply many products, and each product can be from many suppliers. Thus, cardinality is M: N.

5. Category —Product Each category can have many products, and each product belongs to only 1 category. Thus, cardinality is 1: N.
6. Advertisement—product Each advertisement can advertise only 1 product, and each product can be associated with one advertisement. Thus, cardinality is 1:1.
7. Advertiser—Advertisement Each advertiser can create many advertisements, and each advertisement can be created by one advertiser. Thus, cardinality is 1: N.
8. Customer—Customer\_Query Each customer can request many queries, and each query can be linked with one customer. Thus, cardinality is 1: N.
9. Membership —Customer Each membership can include many customers, and each customer can subscribe to one membership. Thus, cardinality is 1: N.

Figure 1 shows the relationship set for the entities



**Figure 1. Relationship Sets**

The relationship of all entities and its attributes is illustrated in the Figure 2 of the Entity Relationship Diagram (ERD)

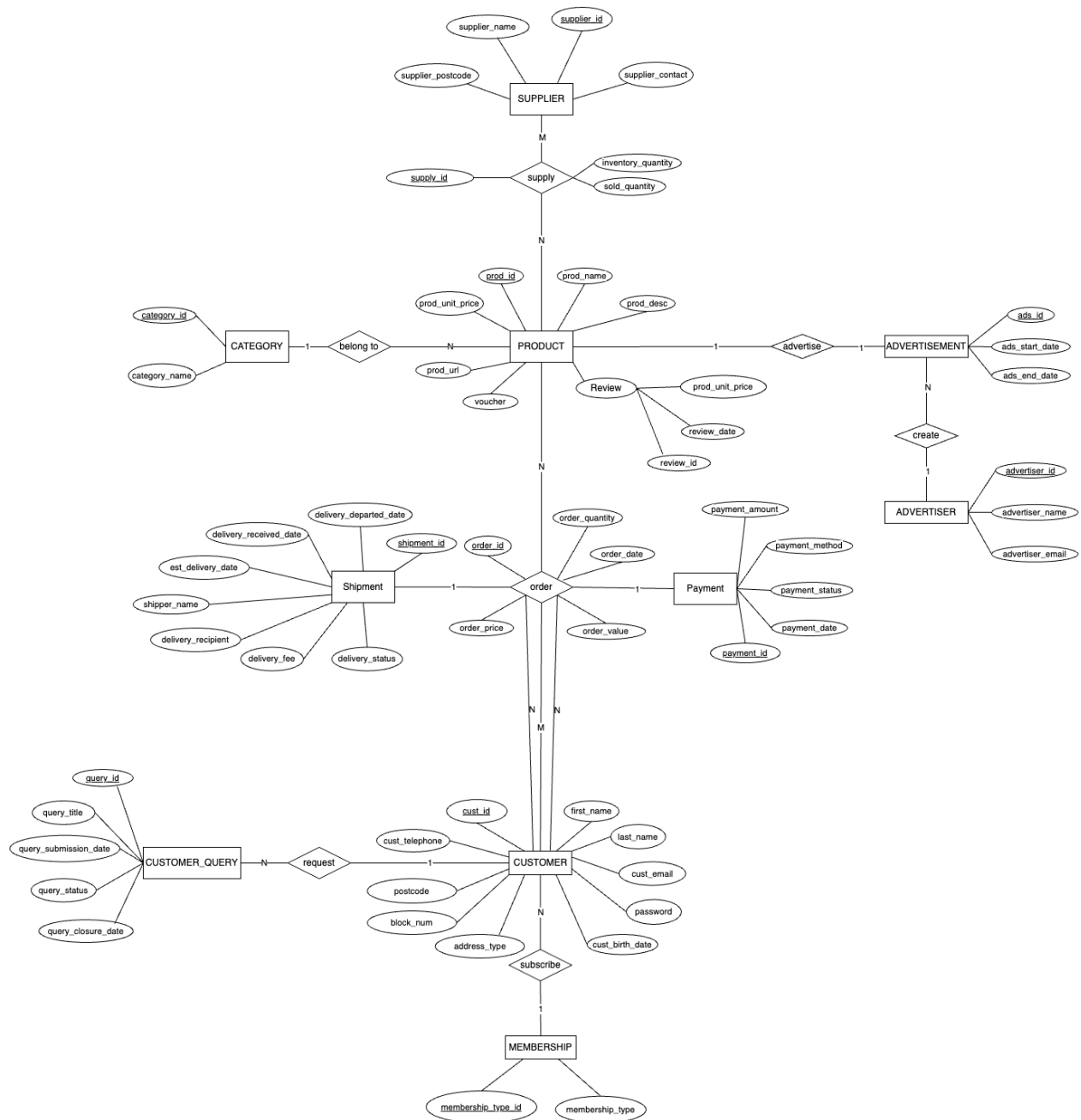


Figure 2. Entity Relationship Diagram (ERD)

### 1.3 Logical Design

Logical design is used to translate the conceptual ERD model for the application into normalised data requirements. The logical design of the e-commerce database is shown in Figure 3. The primary key is indicated using one underline while foreign key is indicated using double underline

- customer (customer\_id, cust\_name, cust\_email, password, cust\_birth\_date, postcode, address\_type, cust\_telephone)
- product (prod\_id, prod\_name, prod\_desc, review\_id, prod\_rating, review\_date, voucher, prod\_url, prod\_unit\_price)
- order (order\_id, customer\_id, prod\_id, payment\_id, shipment\_id, order\_quantity, order\_date, order\_price, order\_value)
- supply (supply\_id, prod\_id, inventory\_quantity, soldq\_uantity)
- supplier (supplier\_id, supplier\_contact, supplier\_name, supplier\_postcode)
- category (category\_id, prod\_id, category\_name)
- advertisement (ads\_id, prod\_id, advertiser\_id, ads\_start\_date, ads\_end\_date)
- advertiser (advertiser\_id, advertiser\_name, advertiser\_email)
- customer\_queries (query\_id, customer\_id, query\_title, query\_detail, query\_submission\_date, query\_closure\_date, query\_status)
- membership (membership\_type\_id, cust\_id, membership\_type)
- payment (payment\_id, payment\_method, payment\_status, payment\_date)
- shipment (shipment\_id, shipper\_name, delivery\_status, delivery\_fee, delivery\_recipient, estimated\_delivery\_date, delivery\_receive\_date, delivery\_departed\_date)

Figure 3. Logical Schema

## 1.4 Physical Design

The physical design refers to the implementation of the logical schema on the physical storage devices of a Database Management System (DBMS). For this project, SQL is used for managing and manipulating database within a DBMS.

In this step, the data type of each attributes will be defined. The decision regarding the data types are made based on various factors such as the nature of the data and the volume of the data. Choosing appropriate data types ensures data integrity, efficient storage utilization, and optimized query performance.

```

1 db_file <- "IB9HP0_9.db"
2
3 # Check if the database file exists and remove it
4 if (file.exists(db_file)) {
5   file.remove(db_file)
6 }

```

```

7
8 # Create connection to SQL database
9 db_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"IB9HP0_9.db")
10
11 # Create table for products
12 dbExecute(db_connection,
13     "CREATE TABLE IF NOT EXISTS products (
14         prod_id VARCHAR (50) PRIMARY KEY,
15         prod_name VARCHAR (50) NOT NULL,
16         prod_desc VARCHAR (100) NOT NULL,
17         voucher VARCHAR (50),
18         prod_url VARCHAR (250) NOT NULL,
19         prod_unit_price DECIMAL NOT NULL
20     )"
21 )
22
23 #Create table for reviews
24 dbExecute(db_connection,
25     "CREATE TABLE IF NOT EXISTS reviews (
26         review_id VARCHAR (50) PRIMARY KEY,
27         prod_rating DECIMAL NOT NULL,
28         review_date DATE NOT NULL,
29         prod_id VARCHAR (50),
30         FOREIGN KEY (prod_id)
31         REFERENCES products(prod_id)
32     )"
33 )
34
35 #Create table for memberships
36 dbExecute(db_connection,
37     "CREATE TABLE IF NOT EXISTS memberships (
38         membership_type_id VARCHAR (50) PRIMARY KEY,
39         membership_type VARCHAR (50) NOT NULL
40     )"
41 )
42
43 #Create table for customers
44 dbExecute(db_connection,
45     "CREATE TABLE IF NOT EXISTS customers (
46         cust_id VARCHAR (50) PRIMARY KEY,
47         first_name VARCHAR (50) NOT NULL,
48         last_name VARCHAR (50),
49         cust_email VARCHAR (50) UNIQUE,
50         password VARCHAR (50) NOT NULL,
51         cust_birth_date DATE,
52         address_type VARCHAR (50),
53         block_num VARCHAR (50),

```

```

54         postcode VARCHAR (50),
55         cust_telephone INT UNIQUE,
56         membership_type_id VARCHAR (50),
57         FOREIGN KEY (membership_type_id)
58             REFERENCES memberships(membership_type_id)
59     )"
60 )
61
62 #Create table for orders
63 dbExecute(db_connection,
64     "CREATE TABLE IF NOT EXISTS orders (
65         order_id VARCHAR (50) PRIMARY KEY,
66         cust_id VARCHAR (50),
67         FOREIGN KEY (cust_id)
68             REFERENCES customers(cust_id)
69     )"
70 )
71
72 #Create table for order details
73 dbExecute(db_connection,
74     "CREATE TABLE IF NOT EXISTS order_details (
75         order_quantity INT NOT NULL,
76         order_date DATE,
77         order_price DECIMAL NOT NULL,
78         order_value DECIMAL,
79         prod_id VARCHAR (50),
80         order_id VARCHAR (50),
81         FOREIGN KEY (prod_id)
82             REFERENCES products(prod_id),
83         FOREIGN KEY (order_id)
84             REFERENCES orders(order_id)
85     )"
86 )
87
88 #Create table for payment
89 dbExecute(db_connection,
90     "CREATE TABLE IF NOT EXISTS payments (
91         payment_id VARCHAR (50) PRIMARY KEY,
92         payment_method VARCHAR (100) NOT NULL,
93         payment_amount DECIMAL,
94         payment_status VARCHAR (100) NOT NULL,
95         payment_date DATE,
96         order_id VARCHAR (50),
97         FOREIGN KEY (order_id)
98             REFERENCES orders(order_id)
99     )"
100 )

```



```

101
102 #Create table for shipment
103 dbExecute(db_connection,
104     "CREATE TABLE IF NOT EXISTS shipments (
105         shipment_id VARCHAR (50) PRIMARY KEY,
106         delivery_status VARCHAR (50),
107         delivery_fee DECIMAL,
108         delivery_recipient VARCHAR (50),
109         shipper_name VARCHAR (50),
110         est_delivery_date DATE,
111         delivery_departed_date DATE,
112         delivery_received_date DATE,
113         prod_id VARCHAR (50),
114         order_id VARCHAR (50),
115         FOREIGN KEY (prod_id)
116             REFERENCES products(prod_id),
117         FOREIGN KEY (order_id)
118             REFERENCES orders(order_id)
119     )"
120 )
121
122 #Create table for supplier
123 dbExecute(db_connection,
124     "CREATE TABLE IF NOT EXISTS suppliers (
125         supplier_id VARCHAR (50) PRIMARY KEY,
126         supplier_name VARCHAR (50) NOT NULL UNIQUE,
127         supplier_postcode VARCHAR (100) NOT NULL UNIQUE,
128         supplier_contact INT NOT NULL UNIQUE
129     )"
130 )
131
132 #Create table for supplies
133 dbExecute(db_connection,
134     "CREATE TABLE IF NOT EXISTS supplies (
135         supply_id VARCHAR (50) PRIMARY KEY,
136         inventory_quantity INT NOT NULL,
137         sold_quantity INT NOT NULL,
138         supplier_id VARCHAR (50),
139         prod_id VARCHAR (50),
140         FOREIGN KEY (supplier_id)
141             REFERENCES suppliers(supplier_id),
142         FOREIGN KEY (prod_id)
143             REFERENCES products(prod_id)
144     )"
145 )
146
147 #Create table for customer queries

```

```

148 dbExecute(db_connection,
149     "CREATE TABLE IF NOT EXISTS customer_queries (
150         query_id VARCHAR (50) PRIMARY KEY,
151         query_title VARCHAR (50) NOT NULL,
152         query_submission_date DATE,
153         query_closure_date DATE,
154         query_status VARCHAR (50) NOT NULL,
155         cust_id VARCHAR (50),
156         FOREIGN KEY (cust_id)
157             REFERENCES customers(cust_id)
158     )"
159 )
160
161 #Create table for categories
162 dbExecute(db_connection,
163     "CREATE TABLE IF NOT EXISTS categories (
164         category_id VARCHAR (50) PRIMARY KEY,
165         category_name VARCHAR (50) NOT NULL UNIQUE
166     )"
167 )
168
169 #Create table for product categories
170 dbExecute(db_connection,
171     "CREATE TABLE IF NOT EXISTS product_categories (
172         category_id VARCHAR (50),
173         prod_id VARCHAR (50),
174         FOREIGN KEY (prod_id)
175             REFERENCES categories(category_id),
176         FOREIGN KEY (prod_id)
177             REFERENCES products(prod_id)
178     )"
179 )
180
181 #Create table for advertiser
182 dbExecute(db_connection,
183     "CREATE TABLE IF NOT EXISTS advertisers (
184         advertiser_id VARCHAR (50) PRIMARY KEY,
185         advertiser_name VARCHAR (50) NOT NULL UNIQUE,
186         advertiser_email VARCHAR (50) UNIQUE
187     )"
188 )
189
190 #Create table for advertisements
191 dbExecute(db_connection,
192     "CREATE TABLE IF NOT EXISTS advertisements (
193         ads_id VARCHAR (50) PRIMARY KEY,
194         ads_start_date DATE,

```

```

195         ads_end_date DATE,
196         prod_id VARCHAR (50) UNIQUE,
197         advertiser_id VARCHAR (50),
198         FOREIGN KEY (prod_id)
199             REFERENCES products(prod_id),
200         FOREIGN KEY (advertiser_id)
201             REFERENCES advertisers(advertiser_id)
202     )"
203 )

```

## 2 Data Generation and Management

### 2.1 Synthetic Data Generation

The data used in the project is generated using R using several packages including randomNames, dplyr, tidyr, chartlatan, stringi, lubridate, and conjurer. Additionally, AI is used to generate names, categories, and descriptions. For the address, the postcode referred to the data provided by Office for National Statistics (ONS). The data is generated at one normal form (1NF) to ensure the consistency throughout the dataset.

```

1  ## Synthetic Data Generation #1
2
3  ### 'customers' table
4  #Define parameters for customers
5  set.seed(312)
6  n_customers <- 100
7  birthdate <- sample(seq(from = as.Date(today() - years(80), "%d-%m-%Y"),
8                        to = as.Date(today() - years(18), "%d-%m-%Y"), by = "day"),
9                    n_customers)
10 cv_postcode <-
11   read.csv("data_uploads/ONSPD_AUG_2023_UK_CV.csv")[, 1] %>%
12   data.frame() %>%
13   setNames("pcd")
14 address_type <- c("Home", "Office")
15 #Create data
16 customers_data <-
17   #Create n unique customer IDs with random names
18   data.frame("cust_id" = conjurer::buildCust(n_customers),
19             "cust_name" = randomNames::randomNames(n_customers)) %>%
20   separate(cust_name, into = c("last_name", "first_name"), sep = ", ") %>%
21   #Create email column, by merging last & first name with email domain @gmail.com
22   unite(cust_email, c(last_name, first_name), sep = ".", remove = F) %>%
23   mutate(
24     "cust_email" = paste(cust_email, "gmail.com", sep = "@"),
25     #Generate user's password, using random string generation package
26     "password" =

```

```

27   stringi::stri_rand_strings(n=n_customers, length=8, pattern="[A-Za-z0-9]"),
28   #Adding customer BOD
29   "cust_birth_date" = sample(birthdate, n_customers, replace = T),
30   #Adding the phone code in UK
31   "phone_domain" = "075",
32   #create unique random strings of 7 digits
33   "cust_telephone" =
34     stringi::stri_rand_strings(n=n_customers, length=7, pattern="[0-9]"),
35   "block_num" =
36     sprintf("%s%s",
37             stri_rand_strings(n=n_customers, length=1, pattern="[A-Z]"),
38             stri_rand_strings(n=n_customers, length=2, pattern="[0-99]")),
39   #randomly assign postcode to each customer
40   "postcode" = cv_postcode[sample(nrow(cv_postcode), n_customers),],
41   #randomly assign address type to each customer
42   "address_type" = sample(address_type, n_customers, replace = T)) %>%
43   #Adding customer's telephone number by merging two phone number columns
44   unite(cust_telephone,
45         c(phone_domain, cust_telephone), sep = "", remove = T) %>%
46   #reorder the columns
47   select(1,4,3,2,5,6,8,9,10,7)
48 customers_data$cust_birth_date <- format(customers_data$cust_birth_date, "%d-%m-%Y")
49 #Save data to data file
50 write.csv(customers_data, "data_uploads/R_synth_customers_round1.csv")
51
52 ### 'products' table
53 #Getting brand and product names from Gemini
54 gemini_prods <-
55   readxl::read_excel("data_uploads/gemini_prod_cate_supplier.xlsx",
56                     .name_repair = "universal") %>%
57   setNames(c("seller_name", "category", "prod_name", "prod_desc"))
58 #Define parameters for products
59 set.seed(123)
60 n_prods <- 20
61 voucher_type <- c("10%", "20%", "50%")
62 ratings <- c(1,2,3,4,5)
63 date <- #assuming company was established on Mar 06th 2004, data here is
64   sample(seq(from = as.Date("2004/03/06"),
65             to = as.Date(lubridate::today()), by = "day"), 12)
66 #Assign product ID, and adding product names and URL
67 products_data <-
68   #generate product id
69   conjurer::buildProd(n_prods, minPrice = 1, maxPrice = 100) %>%
70   #add product name and description from gemini's file
71   mutate("prod_name" = sample(gemini_prods$prod_name, 20)) %>%
72   left_join(select(gemini_prods, -c(seller_name, category)),
73             by = join_by(prod_name)) %>%

```

```

74 #rename columns to fit schema
75 rename(prod_id = SKU, prod_unit_price = Price) %>%
76 #rename `sku` with `prod`
77 mutate("prod_id" = gsub("sku", "prod", prod_id)) %>%
78 #add product url
79 mutate("web_prefix" = "https://group9.co.uk/",
80        "prod_url1" = gsub(" ", "-", prod_name)) %>%
81 unite(prod_url, c(prod_url1, prod_id), sep = "-", remove = F) %>%
82 unite(prod_url, c(web_prefix, prod_url), sep = "", remove = T) %>%
83 mutate(
84   #Create ratings
85   "prod_rating" = sample(ratings, n_prods, replace = T),
86   #Review date
87   "review_date" = sample(format(date, "%d-%m-%Y"), n_prods, replace = T),
88   #Assign review ID
89   "review_id" =
90     conjurer::buildCust(sum(!is.na(prod_rating))),
91   "review_id" = gsub("cust", "rev", review_id)) %>%
92 #drop temp url
93 select(-prod_url1)
94 #Create vouchers -- Randomly assign voucher types to 50% of the products
95 voucher_prods <- sample_n(data.frame(products_data$prod_id),
96                           0.5*nrow(products_data)) %>% setNames("prod_id")
97 products_data <- products_data %>%
98   mutate(voucher = ifelse(products_data$prod_id %in% voucher_prods$prod_id,
99                           sample(voucher_type, nrow(voucher_prods), replace = T), NA))
100 #Finalise the table
101 products_data <-
102   products_data %>%
103   #rearrange order of columns
104   select(2,4,5,8,6,7,3,9,1)
105 #Save to .csv file
106 write.csv(products_data, "data_uploads/R_synth_products_round1.csv")
107
108 ### 'orders' table
109 #Define parameters
110 origin_date <- "1970-01-01"
111 n_orders <- 500
112 order_date <-
113   #round 1 is for orders in 2022-Mar'2024,
114   #so all orders have been paid and delivered successfully
115   sample(seq(from = as.Date("2022/01/01"),
116             to = as.Date("2024/03/01"), 12))
117 pymt_method <-
118   c("Bank Transfer", "Visa", "Mastercard", "PayPal", "GPay", "Apple Pay")
119 pymt_status <- c("Done", "Verifying")
120 shipper_lookup <-

```

```

121   data.frame("shipper_name" = c("DHL", "Group9DL", "DPD"),
122             "delivery_fee" = c(5,2,3),
123             "ETA" = c(1,5,3))
124 delivery_status <- c("Delivered", "In Progress",
125                    "Failed to contact", "Delayed")
126 orders_col_order <-
127   c("order_id", "cust_id", "prod_id", "order_quantity",
128     "order_date", "order_value", "order_price")
129 #generate n order IDs and assign customers to them, including order date
130 set.seed(122)
131 orders_data <-
132   #Create n unique order IDs
133   data.frame("order_id" = conjurer::buildCust(n_orders)) %>%
134   mutate(order_id = gsub("cust", "o", order_id),
135          payment_id = gsub("o", "pm", order_id),
136          cust_id = sample(customers_data$cust_id, n_orders, replace = T),
137          order_date = sample(order_date, n_orders, replace = T),
138          payment_method = sample(pymt_method, n_orders, replace = T),
139          payment_status = "Done",
140          delivery_recipient = randomNames::randomNames(n_orders,
141                                                         which.names = "first"))
142 #adding payment date with logic dependent on payment status
143 orders_data <- orders_data %>%
144   mutate("payment_date" = ifelse(payment_status == "Done", order_date, NA)) %>%
145   mutate("payment_date" = as.Date(payment_date,
146                                   origin = origin_date))
147 #randomly replicate certain orders to map with products
148 set.seed(122)
149 orders_data <- orders_data %>% bind_rows() %>%
150   rbind(sample_n(orders_data, 0.4*nrow(orders_data)),
151         sample_n(orders_data, 0.5*nrow(orders_data)),
152         sample_n(orders_data, 0.8*nrow(orders_data)))
153 #assign products to orders
154 orders_data <- orders_data %>%
155   mutate(
156     "prod_id" = sample(products_data$prod_id,
157                       nrow(orders_data), replace = T),
158     #generate order quantity
159     "order_quantity" = sample(seq(1,10,1), nrow(orders_data), replace = T)) %>%
160   merge(select(products_data, c(prod_id, prod_unit_price, voucher)),
161         by = "prod_id")
162 #Order value and shipper
163 orders_data <- orders_data %>%
164   #order price and value
165   mutate(
166     voucher = as.numeric(gsub("%", "", voucher))/100,
167     #product unit price is discounted in case of voucher available

```

```

168   order_price = ifelse(!is.na(voucher),
169                       prod_unit_price * voucher, prod_unit_price),
170   order_value = order_price * order_quantity,
171   #assign shippers to products
172   shipper_name =
173     sample(shipper_lookup$shipper_name, nrow(orders_data), replace = T),
174   #add delivery status
175   delivery_status = "Delivered" ) %>%
176   #lookup delivery fee
177   merge(shipper_lookup, by = "shipper_name")
178 #dates of delivery
179 orders_data <- orders_data %>%
180   #departure and ETA
181   mutate(
182     delivery_departed_date =
183       ifelse(!is.na(payment_date), (payment_date + days(2)), NA),
184     est_delivery_date = delivery_departed_date + ETA) %>%
185   #departure and ETA - format as date
186   mutate(
187     delivery_departed_date =
188       as.Date(delivery_departed_date, origin = origin_date),
189     est_delivery_date =
190       as.Date(est_delivery_date, origin = origin_date)) %>%
191   #received
192   mutate(
193     delivery_received_date =
194       ifelse(delivery_status != "Delivered", NA, est_delivery_date)) %>%
195   mutate(
196     delivery_received_date =
197       as.Date(delivery_received_date, origin = origin_date)) %>%
198   #drop ETA
199   select(-ETA)
200
201 ### generate 'shipment' from orders
202 shipment_colnames <- c("order_id", "prod_id",
203                       "delivery_departed_date",
204                       "delivery_received_date", "est_delivery_date",
205                       "shipper_name", "delivery_recipient",
206                       "delivery_fee", "delivery_status")
207 shipment_data <- select(orders_data, shipment_colnames)
208 shipment_data <- shipment_data %>%
209   mutate(shipment_id = paste("sm", rownames(shipment_data), sep = ""),
210         .before = "order_id")
211 #reformat date
212 shipment_dates <- c("delivery_departed_date",
213                    "delivery_received_date", "est_delivery_date")
214 shipment_data[shipment_dates] <- lapply(shipment_data[shipment_dates],

```

```

215         format, "%d-%m-%Y")
216 #Save data to data file
217 write.csv(shipment_data, "data_uploads/R_synth_shipment_round1.csv")
218
219 ### generate 'payment' from orders
220 payment_colnames <- c("payment_id", "payment_method", "order_id",
221                       "payment_status", "payment_date")
222 #Add payment amount
223 payment_data <- orders_data %>% group_by(payment_id) %>%
224   summarise(payment_amount = sum(order_value)) %>%
225   left_join(select(orders_data, payment_colnames), by = "payment_id")
226 #remove duplicates
227 payment_data <- distinct(payment_data, payment_id, .keep_all = T) %>%
228   select(1,4,3,2,5,6)
229 #re-format date
230 payment_data$payment_date <- format(payment_data$payment_date, "%d-%m-%Y")
231 #Save data to data file
232 write.csv(payment_data, "data_uploads/R_synth_payment_round1.csv")
233
234 #reorder the columns of 'orders'
235 orders_data <- select(orders_data, orders_col_order)
236 #Save data to data file
237 write.csv(orders_data, "data_uploads/R_synth_orders_round1.csv")
238
239 ### 'suppliers' table
240 #Define parameters for suppliers table
241 set.seed(123)
242 n_suppliers <- length(unique(gemini_prods$seller_name))
243 wc_postcode <- read.csv("data_uploads/ONSPD_AUG_2023_UK_WC.csv")[,1]
244
245 #Create suppliers table
246 suppliers_data <-
247   #Pull seller name from gemini file
248   distinct(select(gemini_prods, seller_name)) %>%
249   rename("supplier_name" = "seller_name") %>%
250   mutate("supplier_id" = seq(1, n_suppliers, 1),
251          "prefix" = "s") %>%
252   unite(supplier_id, c(prefix, supplier_id), sep = "", remove = T) %>%
253   mutate(
254     "supplier_postcode" =
255       sample(wc_postcode, n_suppliers, replace = T),
256     #Adding the phone code in UK
257     "phone_domain" = "079",
258     #create unique random strings of 7 digits
259     "supplier_contact" =
260       stringi::stri_rand_strings(n=n_suppliers, length=7, pattern="[0-9]")) %>%
261   #Adding supplier's telephone number by merging two phone number columns

```



```

262 unite(supplier_contact,
263       c(phone_domain, supplier_contact), sep = "", remove = T) %>%
264   select(2,1,4,3)
265 #Save data to data file
266 write.csv(suppliers_data, "data_uploads/R_synth_suppliers_round1.csv")
267
268 ### 'supply' table
269 #Define parameters for supply table
270 set.seed(123)
271 order_quant_by_prod <- orders_data %>%
272   group_by(prod_id) %>% summarise(sold_quantity = sum(order_quantity))
273 supply_col_order <- c("supply_id", "supplier_id", "prod_id",
274                     "inventory_quantity", "sold_quantity")
275 #Create supply table
276 supply_data <- select(products_data, c(prod_id, prod_name)) %>%
277   merge(order_quant_by_prod, by = "prod_id") %>%
278   mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
279   mutate(inventory_quantity =
280         as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
281   merge(select(gemini_prods, c(seller_name, prod_name)), by = "prod_name") %>%
282   rename("supplier_name" = "seller_name") %>%
283   merge(select(suppliers_data, c(supplier_id, supplier_name)),
284         by = "supplier_name")
285 #Create competitors for M:N relationship
286 supply_competitors <- select(products_data, c(prod_id, prod_name)) %>%
287   mutate(supplier_name =
288         sample(suppliers_data$supplier_name, n_prods, replace = T)) %>%
289   merge(select(suppliers_data, c(supplier_id, supplier_name)),
290         by = "supplier_name") %>%
291   merge(order_quant_by_prod, by = "prod_id") %>%
292   mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
293   mutate(inventory_quantity =
294         as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
295   select(2,3,1,5,6,4)
296 #Combine supply and competitors
297 supply_data <-
298   rbind(supply_data, supply_competitors) %>%
299   mutate(supply_id = paste("sp", row_number(), sep = "")) %>%
300   select(-c(supplier_name, prod_name))
301 #reorder columns
302 supply_data <- supply_data[, supply_col_order]
303 #Save data to data file
304 write.csv(supply_data, "data_uploads/R_synth_supply_round1.csv")
305
306 ### 'memberships' table
307 membership_lookup <-
308   data.frame(

```

```

309     "membership_type" = c("Student", "Trial", "Premium")) %>%
310     mutate("membership_type_id" = row_number())
311
312 #Start with the foreign key cust_id
313 set.seed(123)
314 memberships_data <- data.frame(customers_data$cust_id)
315 memberships_data <- memberships_data %>%
316     #Randomly assign membership type to all customers
317     mutate("membership_type" =
318         sample(membership_lookup$membership_type,
319             nrow(memberships_data), replace = T)) %>%
320     #Lookup membership_id
321     merge(membership_lookup, by = "membership_type") %>%
322     rename(cust_id = customers_data.cust_id) %>%
323     select(3,2,1)
324 #Save to .csv file
325 write.csv(memberships_data, "data_uploads/R_synth_memberships_round1.csv")
326
327 ### 'customer_queries' table
328 set.seed(123)
329 n_queries <- 20
330 customer_queries_data <- data.frame(
331     query_id = sprintf("Q%d", 1:n_queries),
332     cust_id = sample(customers_data$cust_id, n_queries, replace = TRUE),
333     query_title = sample(c("Delivery Issue", "Payment Issue", "Purchase Return", "Damaged Product", "Refund Request", "Complaint", "Feedback", "Inquiry", "Request", "Suggestion", "Complaint", "Feedback", "Inquiry", "Request", "Suggestion", "Complaint", "Feedback", "Inquiry", "Request", "Suggestion"), n_queries, replace = TRUE),
334     query_submission_date = sample(seq(as.Date('2023-01-01'), as.Date('2023-1-31'), by="day"), n_queries, replace = TRUE),
335     query_closure_date = sample(seq(as.Date('2023-02-01'), as.Date('2023-03-31'), by="day"), n_queries, replace = TRUE),
336     query_status = sample(c("Closed"), n_queries, replace = TRUE)
337 )
338
339 customer_queries_data$query_submission_date <- format(customer_queries_data$query_submission_date, "%Y-%m-%d")
340 customer_queries_data$query_closure_date <- format(customer_queries_data$query_closure_date, "%Y-%m-%d")
341
342 #Save to .csv file
343 write.csv(customer_queries_data, "data_uploads/R_synth_customer_queries_round1.csv", row.names = FALSE)
344
345 ### 'categories' table
346 #create lookup table for category_id and category name
347 set.seed(123)
348 category_lookup <-
349     data.frame("category_id" = seq(1, length(unique(gemini_prods$category)),1),
350         "category" = unique(gemini_prods$category),
351         "cate_code" = "cate") %>%
352     unite(category_id, c(cate_code, category_id), sep = "", remove = T)
353 #Create categories table
354 categories_data <-
355     #Pull category name and product name from gemini file

```

```

356 select(gemini_prods, c(category, prod_name)) %>%
357 #Only keep the products included in the products table
358 right_join(select(products_data, c(prod_id, prod_name)), by = "prod_name") %>%
359 #lookup category_id
360 merge(category_lookup, by = "category") %>%
361 #rename to have category_name column
362 rename(category_name = category) %>%
363 #drop product name column
364 select(-prod_name) %>%
365 #reorder the columns to match with table schema
366 select(3,2,1)
367 #Save to .csv file
368 write.csv(categories_data, "data_uploads/R_synth_categories_round1.csv")
369
370 ### 'advertisers' table
371 set.seed(123)
372 n_advertisers <- 5
373 advertisers_data <- data.frame(
374   advertiser_id = sprintf("ADV%d", 1:n_advertisers),
375   advertiser_name = c("Ads Life", "Ads Idol", "Ads is Life", "Ads Master", "Ads Expert"),
376   advertiser_email = sprintf("advertiser%d@gmail.com", 1:n_advertisers)
377 )
378 #Save to .csv file
379 write.csv(advertisers_data, "data_uploads/R_synth_advertisers_round1.csv", row.names = FALSE)
380
381 ### 'advertisements' table
382 set.seed(123)
383 n_ads <- 9
384 advertisements_data <- data.frame(
385   ads_id = sprintf("ADS%d", 1:n_ads),
386   prod_id = sample(products_data$prod_id, n_ads, replace = TRUE),
387   advertiser_id = sample(advertisers_data$advertiser_id, n_ads, replace = TRUE),
388   ads_start_date = sample(seq(as.Date('2023-01-01'), as.Date('2023-12-31'), by="day"), n_ads, replace = TRUE),
389   ads_end_date = sample(seq(as.Date('2024-01-01'), as.Date('2024-12-31'), by="day"), n_ads, replace = TRUE)
390 )
391
392 advertisements_data$ads_start_date <- format(advertisements_data$ads_start_date, "%d-%m-%Y")
393 advertisements_data$ads_end_date <- format(advertisements_data$ads_end_date, "%d-%m-%Y")
394
395 #Save to .csv file
396 write.csv(advertisements_data, "data_uploads/R_synth_advertisements_round1.csv", row.names = FALSE)
397
398 ### 'customers' table
399 #Define parameters for customers
400 set.seed(456)
401 n_customers <- 100
402 birthdate <- sample(seq(from = as.Date(today() - years(80), "%d-%m-%Y"),

```

```

403         to = as.Date(today() - years(18), "%d-%m-%Y"), by = "day"),
404         n_customers)
405 cv_postcode <-
406   read.csv("data_uploads/ONSPD_AUG_2023_UK_CV.csv")[, 1] %>%
407   data.frame() %>%
408   setNames("pcd")
409 address_type <- c("Home", "Office")
410 #Create data
411 customers_data <-
412   #Create n unique customer IDs with random names
413   data.frame("cust_id" = paste("cust", seq(101,101+n_customers-1,1), sep = ""),
414             "cust_name" = randomNames::randomNames(n_customers)) %>%
415   separate(cust_name, into = c("last_name", "first_name"), sep = ", ") %>%
416   #Create email column, by merging last & first name with email domain @gmail.com
417   unite(cust_email, c(last_name, first_name), sep = ".", remove = F) %>%
418   mutate(
419     "cust_email" = paste(cust_email,"gmail.com", sep = "@"),
420     #Generate user's password, using random string generation package
421     "password" =
422       stringi::stri_rand_strings(n=n_customers, length=8, pattern="[A-Za-z0-9]"),
423     #Adding customer BOD
424     "cust_birth_date" = sample(birthdate, n_customers, replace = T),
425     #Adding the phone code in UK
426     "phone_domain" = "075",
427     #create unique random strings of 7 digits
428     "cust_telephone" =
429       stringi::stri_rand_strings(n=n_customers, length=7, pattern="[0-9]"),
430     "block_num" =
431       sprintf("%s%s",
432             stri_rand_strings(n=n_customers, length=1, pattern="[A-Z]"),
433             stri_rand_strings(n=n_customers, length=2, pattern="[0-99]")),
434     #randomly assign postcode to each customer
435     "postcode" = cv_postcode[sample(nrow(cv_postcode), n_customers),],
436     #randomly assign address type to each customer
437     "address_type" = sample(address_type, n_customers, replace = T)) %>%
438   #Adding customer's telephone number by merging two phone number columns
439   unite(cust_telephone,
440         c(phone_domain, cust_telephone), sep = "", remove = T) %>%
441   #reorder the columns
442   select(1,4,3,2,5,6,8,9,10,7)
443 customers_data$cust_birth_date <- format(customers_data$cust_birth_date, "%d-%m-%Y")
444 #Save data to data file
445 write.csv(customers_data, "data_uploads/R_synth_customers_round2.csv")
446
447 ### 'products' table
448 #Getting brand and product names from Gemini
449 gemini_prods <-

```

```

450 readxl::read_excel("data_uploads/gemini_prod_cate_supplier_2.xlsx",
451                     .name_repair = "universal") %>%
452   setNames(c("seller_name", "category", "prod_name", "prod_desc"))
453 #Define parameters for products
454 set.seed(456)
455 n_prods <- 19
456 voucher_type <- c("10%", "20%", "50%")
457 ratings <- c(1,2,3,4,5)
458 date <- #assuming company was established on Mar 06th 2004
459   sample(seq(from = as.Date("2004/03/06"),
460             to = as.Date(lubridate::today()), by = "day"), 12)
461 #Assign product ID, and adding product names and URL
462 products_data <-
463   #generate product id
464   conjurer::buildProd(n_prods, minPrice = 1, maxPrice = 100) %>%
465   #add product name and description from gemini's file
466   mutate("prod_name" = sample(gemini_prods$prod_name, nrow(gemini_prods))) %>%
467   left_join(select(gemini_prods, -c(seller_name, category)),
468             by = join_by(prod_name)) %>%
469   #rename columns to fit schema
470   rename(prod_id = SKU, prod_unit_price = Price) %>%
471   #rename `sku` with `prod`
472   mutate("prod_id" = gsub("sku", "", prod_id)) %>%
473   mutate("prod_id" = paste("prod", as.numeric(prod_id)+20, sep = "")) %>%
474   #add product url
475   mutate("web_prefix" = "https://group9.co.uk/",
476          "prod_url1" = gsub(" ", "-", prod_name)) %>%
477   unite(prod_url, c(prod_url1, prod_id), sep = "-", remove = F) %>%
478   unite(prod_url, c(web_prefix, prod_url), sep = "", remove = T) %>%
479   mutate(
480     #Create ratings
481     "prod_rating" = sample(ratings, n_prods, replace = T),
482     #Review date
483     "review_date" = sample(format(date, "%d-%m-%Y"), n_prods, replace = T),
484     #Assign review ID
485     "review_id" = paste("rev", seq(21, 21+n_prods-1, 1), sep = ""),
486     "review_id" = gsub("cust", "rev", review_id)) %>%
487   #drop temp url
488   select(-prod_url1)
489 #Create vouchers -- Randomly assign voucher types to 50% of the products
490 voucher_prods <- sample_n(data.frame(products_data$prod_id),
491                           0.5*nrow(products_data)) %>% setNames("prod_id")
492 products_data <- products_data %>%
493   mutate(voucher = ifelse(products_data$prod_id %in% voucher_prods$prod_id,
494                           sample(voucher_type, nrow(voucher_prods), replace = T), NA))
495 #Finalise the table
496 products_data <-

```

```

497 products_data %>%
498   #rearrange order of columns
499   select(2,4,5,8,6,7,3,9,1)
500 #Save to .csv file
501 write.csv(products_data, "data_uploads/R_synth_products_round2.csv")
502
503 ### 'orders' table
504 #Define parameters
505 origin_date <- "1970-01-01"
506 n_orders <- 100
507 order_date <- #round 2 is for orders in 2024
508   sample(seq(from = as.Date("2024/03/01"),
509             to = as.Date(lubridate::today()), by = "day"), 12)
510 pymt_method <-
511   c("Bank Transfer", "Visa", "Mastercard", "PayPal", "GPay", "Apple Pay")
512 pymt_status <- c("Done", "Verifying")
513 shipper_lookup <-
514   data.frame("shipper_name" = c("DHL", "Group9DL", "DPD"),
515             "delivery_fee" = c(5,2,3),
516             "ETA" = c(1,5,3))
517 delivery_status <- c("Delivered", "In Progress",
518                    "Failed to contact", "Delayed")
519 orders_col_order <-
520   c("order_id", "cust_id", "prod_id", "order_quantity",
521     "order_date", "order_value", "order_price")
522 #generate n order IDs and assign customers to them, including order date
523 set.seed(321)
524 orders_data <-
525   #Create n unique order IDs
526   data.frame("order_id" = paste("o",seq(501, 501+n_orders-1, 1), sep = "")) %>%
527   mutate(order_id = gsub("cust", "o", order_id),
528          payment_id = gsub("o", "pm", order_id),
529          cust_id = sample(customers_data$cust_id, n_orders, replace = T),
530          order_date = sample(order_date, n_orders, replace = T),
531          payment_method = sample(pymt_method, n_orders, replace = T),
532          payment_status = sample(pymt_status, n_orders, replace = T),
533          delivery_recipient = randomNames::randomNames(n_orders,
534                                                         which.names = "first"))
535 #adding payment date with logic dependent on payment status
536 orders_data <- orders_data %>%
537   mutate("payment_date" = ifelse(payment_status == "Done", order_date, NA)) %>%
538   mutate("payment_date" = as.Date(payment_date,
539                                   origin = origin_date))
540 #randomly replicate certain orders to map with products
541 set.seed(456)
542 orders_data <- orders_data %>% bind_rows() %>%
543   rbind(sample_n(orders_data, 0.4*nrow(orders_data)),

```

```

544     sample_n(orders_data, 0.5*nrow(orders_data)),
545     sample_n(orders_data, 0.8*nrow(orders_data)))
546 #assign products to orders
547 orders_data <- orders_data %>%
548   mutate(
549     "prod_id" = sample(products_data$prod_id,
550                       nrow(orders_data), replace = T),
551     #generate order quantity
552     "order_quantity" = sample(seq(1,10,1), nrow(orders_data), replace = T)) %>%
553   merge(select(products_data, c(prod_id, prod_unit_price, voucher)),
554         by = "prod_id")
555 #Order value and shipper
556 orders_data <- orders_data %>%
557   #order price and value
558   mutate(
559     voucher = as.numeric(gsub("%", "", voucher))/100,
560     #product unit price is discounted in case of voucher available
561     order_price = ifelse(!is.na(voucher),
562                          prod_unit_price * voucher, prod_unit_price),
563     order_value = order_price * order_quantity,
564     #assign shippers to products
565     shipper_name =
566       sample(shipper_lookup$shipper_name, nrow(orders_data), replace = T),
567     #add delivery status
568     delivery_status =
569       ifelse(payment_status != "Done", "Not Started",
570             sample(delivery_status, nrow(orders_data), replace = T)) ) %>%
571   #lookup delivery fee
572   merge(shipper_lookup, by = "shipper_name")
573 #dates of delivery
574 orders_data <- orders_data %>%
575   #departure and ETA
576   mutate(
577     delivery_departed_date =
578       ifelse(!is.na(payment_date), (payment_date + days(2)), NA),
579     est_delivery_date = delivery_departed_date + ETA) %>%
580   #departure and ETA - format as date
581   mutate(
582     delivery_departed_date =
583       as.Date(delivery_departed_date, origin = origin_date),
584     est_delivery_date =
585       as.Date(est_delivery_date, origin = origin_date)) %>%
586   #received
587   mutate(
588     delivery_received_date =
589       ifelse(delivery_status != "Delivered", NA, est_delivery_date)) %>%
590   mutate(

```



```

591     delivery_received_date =
592       as.Date(delivery_received_date, origin = origin_date)) %>%
593     #drop ETA
594     select(-ETA)
595
596   ### generate 'shipment' from orders
597   shipment_colnames <- c("order_id", "prod_id",
598                         "delivery_departed_date",
599                         "delivery_received_date", "est_delivery_date",
600                         "shipper_name", "delivery_recipient",
601                         "delivery_fee", "delivery_status")
602   shipment_data <- select(orders_data, shipment_colnames)
603   shipment_data <- shipment_data %>%
604     mutate(shipment_id = paste("sm", rownames(shipment_data), sep = ""),
605           .before = "order_id")
606   #reformat date
607   shipment_dates <- c("delivery_departed_date",
608                      "delivery_received_date", "est_delivery_date")
609   shipment_data[shipment_dates] <- lapply(shipment_data[shipment_dates],
610     format, "%d-%m-%Y")
611   #Save data to data file
612   write.csv(shipment_data, "data_uploads/R_synth_shipment_round2.csv")
613
614   ### generate 'payment' from orders
615   payment_colnames <- c("payment_id", "payment_method", "order_id",
616                        "payment_status", "payment_date")
617   #Add payment amount
618   payment_data <- orders_data %>% group_by(payment_id) %>%
619     summarise(payment_amount = sum(order_value)) %>%
620     left_join(select(orders_data, payment_colnames), by = "payment_id")
621   #remove duplicates
622   payment_data <- distinct(payment_data, payment_id, .keep_all = T) %>%
623     select(1,4,3,2,5,6)
624   #re-format date
625   payment_data$payment_date <- format(payment_data$payment_date, "%d-%m-%Y")
626   #Save data to data file
627   write.csv(payment_data, "data_uploads/R_synth_payment_round2.csv")
628
629   #reorder the columns of 'orders'
630   orders_data <- select(orders_data, orders_col_order)
631   #Save data to data file
632   write.csv(orders_data, "data_uploads/R_synth_orders_round2.csv")
633
634   ### 'suppliers' table
635   #Define parameters for suppliers table
636   set.seed(456)
637   n_suppliers <- length(unique(gemini_prods$seller_name))

```



```

638 wc_postcode <- read.csv("data_uploads/ONSPD_AUG_2023_UK_WC.csv")[,1]
639
640 #Create suppliers table
641 suppliers_data <-
642   #Pull seller name from gemini file
643   distinct(select(gemini_prods, seller_name)) %>%
644   rename("supplier_name" = "seller_name") %>%
645   mutate("supplier_id" = seq(21, 21+n_suppliers-1,1),
646          "prefix" = "s") %>%
647   unite(supplier_id, c(prefix, supplier_id), sep = "", remove = T) %>%
648   mutate(
649     "supplier_postcode" =
650       sample(wc_postcode, n_suppliers, replace = T),
651     #Adding the phone code in UK
652     "phone_domain" = "079",
653     #create unique random strings of 7 digits
654     "supplier_contact" =
655       stringi::stri_rand_strings(n=n_suppliers, length=7, pattern="[0-9]")) %>%
656   #Adding supplier's telephone number by merging two phone number columns
657   unite(supplier_contact,
658         c(phone_domain, supplier_contact), sep = "", remove = T) %>%
659   select(2,1,4,3)
660 #Save data to data file
661 write.csv(suppliers_data, "data_uploads/R_synth_suppliers_round2.csv")
662
663 ### 'supply' table
664 #Define parameters for supply table
665 set.seed(456)
666 order_quant_by_prod <- orders_data %>%
667   group_by(prod_id) %>% summarise(sold_quantity = sum(order_quantity))
668 supply_col_order <- c("supply_id", "supplier_id", "prod_id",
669                      "inventory_quantity", "sold_quantity")
670 #Create supply table
671 supply_data <- select(products_data, c(prod_id, prod_name)) %>%
672   merge(order_quant_by_prod, by = "prod_id") %>%
673   mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
674   mutate(inventory_quantity =
675          as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
676   merge(select(gemini_prods, c(seller_name, prod_name)), by = "prod_name") %>%
677   rename("supplier_name" = "seller_name") %>%
678   merge(select(suppliers_data, c(supplier_id, supplier_name)),
679         by = "supplier_name")
680 #Create competitors for M:N relationship
681 supply_competitors <- select(products_data, c(prod_id, prod_name)) %>%
682   mutate(supplier_name =
683          sample(suppliers_data$supplier_name, n_prods, replace = T)) %>%
684   merge(select(suppliers_data, c(supplier_id, supplier_name)),

```



```

732 )
733
734 customer_queries_data$query_submission_date <- format(customer_queries_data$query_submission_d
735
736 #Save to .csv file
737 write.csv(customer_queries_data, "data_uploads/R_synth_customer_queries_round2.csv", row.names=
738
739 ### 'categories' table
740 #create lookup table for category_id and category name
741 set.seed(456)
742 category_lookup <-
743   data.frame("category_id" = seq(1, length(unique(gemini_prods$category)),1),
744             "category" = unique(gemini_prods$category),
745             "cate_code" = "cate") %>%
746   unite(category_id, c(cate_code, category_id), sep = "", remove = T)
747 #Create categories table
748 categories_data <-
749   #Pull category name and product name from gemini file
750   select(gemini_prods, c(category, prod_name)) %>%
751   #Only keep the products included in the products table
752   right_join(select(products_data, c(prod_id, prod_name)), by = "prod_name") %>%
753   #lookup category_id
754   merge(category_lookup, by = "category") %>%
755   #rename to have category_name column
756   rename(category_name = category) %>%
757   #drop product name column
758   select(-prod_name) %>%
759   #reorder the columns to match with table schema
760   select(3,2,1)
761 #Save to .csv file
762 write.csv(categories_data, "data_uploads/R_synth_categories_round2.csv")
763
764 ### 'advertisers' table
765 set.seed(456)
766 n_advertisers <- 5
767 advertisers_data <- data.frame(
768
769   advertiser_id = sprintf("ADV%d", 1:n_advertisers),
770   advertiser_name = c("Ads Life", "Ads Idol", "Ads is Life",
771                     "Ads Master", "Ads Expert"),
772
773   "advertiser_id" = paste("ADV",seq(6, 6+n_advertisers-1, 1), sep = ""),
774   advertiser_name = c("Ads Beauty", "Ads Power", "Ads by WBS", "Ads by MSBA", "Ads Master"),
775
776   advertiser_email = sprintf("advertiser%d@gmail.com", 1:n_advertisers)
777 )
778 #Save to .csv file

```

```

779 write.csv(advertisers_data, "data_uploads/R_synth_advertisers_round2.csv", row.names = FALSE)
780
781 ### 'advertisements' table
782 set.seed(456)
783 n_ads <- 9
784 advertisements_data <- data.frame(
785   "ads_id" = paste("ADS",seq(10, 10+n_ads-1, 1), sep = ""),
786   prod_id = sample(products_data$prod_id, n_ads, replace = TRUE),
787   advertiser_id = sample(advertisers_data$advertiser_id, n_ads, replace = TRUE),
788   ads_start_date = sample(seq(as.Date('2023-01-01'), as.Date('2023-12-31'), by="day"), n_ads, replace = TRUE),
789   ads_end_date = sample(seq(as.Date('2024-01-01'), as.Date('2024-12-31'), by="day"), n_ads, replace = TRUE),
790 )
791
792 advertisements_data$ads_start_date <- format(advertisements_data$ads_start_date, "%d-%m-%Y")
793 advertisements_data$ads_end_date <- format(advertisements_data$ads_end_date, "%d-%m-%Y")
794
795 #Save to .csv file
796 write.csv(advertisements_data, "data_uploads/R_synth_advertisements_round2.csv", row.names = FALSE)

```

Finally, all the data will be generated into csv file which are separated according to the 1NF.

## 2.2 Data Import and Quality Assurance

After generating the data, the csv file is imported. Instead of explicitly specifying the name of the data, a for loop is utilized to import the data dynamically based on the table name pattern. This approach enables the use of read.csv within the loop, facilitating the seamless addition of new data files to the existing data frame. Consequently, no manual edit is needed for new data read.

Subsequently, the data will be normalised into third normal form (3NF). Then, prior to inserting data into the database, it will undergo a two step validation process.

The first step involves assessing the quality of the data. This includes verifying aspects such as the date format of the input data. If the data does not conform to the expected format, it is reformatted according to the standardized date format in the database.

Following the initial quality check, the second step of validation involves verifying whether the new data already exists within the database. If any observations within the new data are found to be duplicates of existing records in the database, these duplicated observations will not be inserted.

The data insertion is integrated with the second step data validation simultaneously. INSERT INTO function is employed instead of utilising dbWriteTable. This choice allows for the formatting of dates as dd-mm-yyyy, unlike dbWriteTable, where dates are stored as strings. Additionally, each time the data insertion code is executed, an error log is generated. This log serves as a reference, containing information about duplicate observations and successfully stored data within the database.

```

1 # Read Data file
2 ## Read advertisements file
3 advertisement_list <- list()
4 for (ads in list.files(path = "data_uploads/", pattern = "advertisement", full.names = TRUE)) {
5   advertisements_ind <- read.csv(ads)
6   advertisement_list[[length(advertisement_list) + 1]] <- advertisements_ind
7 }
8 advertisements_file <- bind_rows(advertisement_list)
9
10 ## Read advertisers file
11 advertisers_list <- list()
12 for (adv in list.files(path = "data_uploads/", pattern = "advertiser", full.names = TRUE)) {
13   advertisers_ind <- read.csv(adv)
14   advertisers_list[[length(advertisers_list) + 1]] <- advertisers_ind
15 }
16 advertisers_file <- bind_rows(advertisers_list)
17
18 ## Read categories file
19 categories_list <- list()
20 for (cat in list.files(path = "data_uploads/", pattern = "categories", full.names = TRUE)) {
21   categories_ind <- read.csv(cat)
22   categories_list[[length(categories_list) + 1]] <- categories_ind
23 }
24 categories_file <- bind_rows(categories_list)
25
26 ## Read customer_queries file
27 customer_queries_list <- list()
28 for (cat in list.files(path = "data_uploads/", pattern = "customer_queries", full.names = TRUE)) {
29   customer_queries_ind <- read.csv(cat)
30   customer_queries_list[[length(customer_queries_list) + 1]] <- customer_queries_ind
31 }
32 customer_queries_file <- bind_rows(customer_queries_list)
33
34 ## Read customers file
35 customers_list <- list()
36 for (cust in list.files(path = "data_uploads/", pattern = "customers", full.names = TRUE)) {
37   customers_ind <- read.csv(cust)
38   customers_list[[length(customers_list) + 1]] <- customers_ind
39 }
40 customers_file <- bind_rows(customers_list)
41
42 ## Read memberships file
43 memberships_list <- list()
44 for (memb in list.files(path = "data_uploads/", pattern = "membership", full.names = TRUE)) {
45   memberships_ind <- read.csv(memb)
46   memberships_list[[length(memberships_list) + 1]] <- memberships_ind
47 }

```

```

48 memberships_file <- bind_rows(memberships_list)
49
50 ## Read orders file
51 orders_list <- list()
52 for (orders in list.files(path = "data_uploads/", pattern = "order", full.names = TRUE)) {
53   orders_ind <- read.csv(orders)
54   orders_list[[length(orders_list) + 1]] <- orders_ind
55 }
56 orders_file <- bind_rows(orders_list)
57
58 ## Read payments file
59 payments_list <- list()
60 for (payments in list.files(path = "data_uploads/", pattern = "payment", full.names = TRUE)) {
61   payments_ind <- read.csv(payments)
62   payments_list[[length(payments_list) + 1]] <- payments_ind
63 }
64 payments_file <- bind_rows(payments_list)
65
66 ## Read products file
67 products_list <- list()
68 for (products in list.files(path = "data_uploads/", pattern = "product", full.names = TRUE)) {
69   products_ind <- read.csv(products)
70   products_list[[length(products_list) + 1]] <- products_ind
71 }
72 products_file <- bind_rows(products_list)
73
74 ## Read shipments file
75 shipments_list <- list()
76 for (shipments in list.files(path = "data_uploads/", pattern = "shipment", full.names = TRUE)) {
77   shipments_ind <- read.csv(shipments)
78   shipments_list[[length(shipments_list) + 1]] <- shipments_ind
79 }
80 shipments_file <- bind_rows(shipments_list)
81
82 ## Read suppliers file
83 suppliers_list <- list()
84 for (suppliers in list.files(path = "data_uploads/", pattern = "suppliers", full.names = TRUE)) {
85   suppliers_ind <- read.csv(suppliers)
86   suppliers_list[[length(suppliers_list) + 1]] <- suppliers_ind
87 }
88 suppliers_file <- bind_rows(suppliers_list)
89
90 ## Read supplies file
91 supplies_list <- list()
92 for (supplies in list.files(path = "data_uploads/", pattern = "supply", full.names = TRUE)) {
93   supplies_ind <- read.csv(supplies)
94   supplies_list[[length(supplies_list) + 1]] <- supplies_ind

```

```

95 }
96 supplies_file <- bind_rows(supplies_list)
97
98 # Normalising the Table into 3NF
99
100 ##Normalising Products Table
101 products_table <- products_file %>%
102   select(prod_id,prod_name,prod_desc,prod_unit_price,voucher,prod_url)
103
104 ##Normalising Reviews Table
105 reviews_table <- products_file %>%
106   select(review_id,prod_id, prod_rating, review_date)
107
108 ##Normalising Memberships Table
109 memberships_table <- memberships_file %>%
110   select(membership_type_id,membership_type)
111 memberships_table <- memberships_table[!duplicated(memberships_table$membership_type_id),]
112
113 ##Normalising Customers Table
114 customers_table <- customers_file %>%
115   select(cust_id, first_name, last_name, cust_email,password, cust_birth_date, block_num, post
116 customers_table <- merge(customers_table,memberships_file, by = "cust_id")
117 customers_table$X <- NULL
118 customers_table$membership_type <- NULL
119
120 ##Normalising Orders Table
121 orders_table <- orders_file %>%
122   select(order_id, cust_id)
123 orders_table <- orders_table[!duplicated(orders_table$order_id),]
124
125 ##Normalising Order details Table
126 order_details_table <- orders_file %>%
127   select(order_id,prod_id, order_quantity, order_date, order_value, order_price)
128
129 ##Normalising Payments Table
130 payments_table <- payments_file %>%
131   select(payment_id,order_id,payment_amount,payment_method,payment_status,payment_date)
132
133 ##Normalising Shipments Table
134 shipments_table <- shipments_file %>%
135   select(shipment_id, order_id, prod_id, delivery_departed_date, delivery_received_date,est_de
136
137 ##Normalising Suppliers Table
138 suppliers_table <- suppliers_file %>%
139   select(supplier_id, supplier_name, supplier_contact, supplier_postcode)
140
141 ##Normalising Supplies Table

```



```

142 supplies_table <- supplies_file %>%
143   select(supply_id,supplier_id,prod_id,inventory_quantity,sold_quantity)
144
145 ##Normalising Customer Queries Table
146 customer_queries_table <- customer_queries_file
147
148 ##Normalising Categories Table
149 categories_table <- categories_file %>%
150   select(category_id,category_name)
151 categories_table <- categories_table[!duplicated(categories_table$category_id),]
152
153 ##Normalising Product Categories Table
154 product_categories_table <- categories_file %>%
155   select(prod_id, category_id)
156
157 ##Normalising Advertiser Table
158 advertisers_table <- advertisers_file
159
160 ##Normalising Advertisement Table
161 advertisements_table <- advertisements_file
162
163 # Data Validation
164
165 ## Advertisement table
166 ### Checking the date format for ads_start_date and ads_end_date
167 if (all(!inherits(try(as.Date(advertisements_table$ads_start_date, format = "%d-%m-%Y")), "try-error"))) {
168   print("Dates are already in the correct format")
169 } else {
170   print("Dates are not in the correct format")
171 }
172
173 if (all(!inherits(try(as.Date(advertisements_table$ads_end_date, format = "%d-%m-%Y")), "try-error"))) {
174   print("Dates are already in the correct format")
175 } else {
176   print("Dates are not in the correct format")
177 }
178
179 ## Ensuring advertisement end date is after the advertisement start date
180 for (i in 1:length(as.Date(advertisements_table$ads_start_date, format = "%d-%m-%Y"))) {
181   if (as.Date(advertisements_table$ads_end_date, format = "%d-%m-%Y")[i] > as.Date(advertisements_table$ads_start_date, format = "%d-%m-%Y")[i]) {
182     print("Ends date happened after the starts date")
183   } else {
184     print(paste("Error!", "Query", i, ": ends date happened before the starts date"))
185   }
186 }
187
188 ### Checking duplicate values for ads_id and prod_id

```



```

189 if(length(advertisements_table$ads_id[duplicated(advertisements_table$ads_id)]) > 0) {
190     print("Duplicate ads_ids found")
191 } else {
192     print("No duplicate ads_ids found")
193 }
194
195 if(length(advertisements_table$prod_id[duplicated(advertisements_table$prod_id)]) > 0) {
196     print("Duplicate prod_ids found")
197 } else {
198     print("No duplicate prod_ids found")
199 }
200
201 ## Advertisers file
202 ### Checking duplicate values for advertiser_id, advertiser_email, and advertisers name
203
204 if(length(advertisers_table$advertiser_id[duplicated(advertisers_table$advertiser_id)]) > 0) {
205     print("Duplicate advertiser_ids found")
206 } else {
207     print("No duplicate advertiser_ids found")
208 }
209
210 if(length(advertisers_table$advertiser_email[duplicated(advertisers_table$advertiser_email)]) > 0) {
211     print("Duplicate advertisers' emails found")
212 } else {
213     print("No duplicate advertisers' emails found")
214 }
215
216 if(length(advertisers_table$advertiser_name[duplicated(advertisers_table$advertiser_name)]) > 0) {
217     print("Duplicate advertisers' names found")
218 } else {
219     print("No duplicate advertisers' names found")
220 }
221
222 ## Checking the advertiser_email format
223 if(length(grep("^([A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[com$])",advertisers_table$advertiser_email)) > 0) {
224     print("All email format are correct")
225 } else {
226     print(paste("There are:", length(advertisers_table$advertiser_email) - length(grep("^([A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[com$])",advertisers_table$advertiser_email))))
227 }
228
229 ## Customer_queries file
230 ### Checking duplicate values for query_id
231
232 if(length(customer_queries_table$query_id[duplicated(customer_queries_table$query_id)]) > 0) {
233     print("Duplicate queries_ids found")
234 } else {
235     print("No duplicate queries_ids found")

```

```

236 }
237
238 ### Checking the date format for query_submission_date and query_closure_date
239 correct_date_format <- function(date_column) {
240   converted_dates <- try(as.Date(date_column, format = "%d-%m-%Y"), silent = TRUE)
241   if (inherits(converted_dates, "try-error")) {
242     return(format(mdy(date_column), "%d-%m-%Y"))
243   } else {
244     return(date_column)
245   }
246 }
247 customer_queries_table$query_submission_date <- correct_date_format(customer_queries_table$query_submission_date)
248 customer_queries_table$query_closure_date <- correct_date_format(customer_queries_table$query_closure_date)
249
250 if (all(!inherits(try(as.Date(customer_queries_table$query_submission_date, format = "%d-%m-%Y"), silent = TRUE), "try-error"))) {
251   print("Query Submission Dates are now in the correct format")
252 } else {
253   print("There was an issue with converting the Query Submission Dates")
254 }
255
256 if (all(!inherits(try(as.Date(customer_queries_table$query_closure_date, format = "%d-%m-%Y"), silent = TRUE), "try-error"))) {
257   print("Query Closure Dates are now in the correct format")
258 } else {
259   print("There was an issue with converting the Query Closure Dates")
260 }
261
262 ## Memberships file
263 ### Checking NA values inside membership_id and membership_type
264 if (any(!is.na(memberships_table))) {
265   print("There are no NA values in the dataset")
266 } else {
267   print("Error! There are NA values in the dataset")
268 }
269
270 ## Orders file
271 ### Checking NA values inside order_id, cust_id, order_quantity, order_price, prod_id
272 if (any(!is.na(order_details_table[,c("order_id", "order_quantity", "order_price", "prod_id")])) {
273   print("There are no NA values in the dataset")
274 } else {
275   print("Error! There are NA values in the dataset")
276 }
277
278 ### Checking date format for the order_date
279 correct_date_format <- function(date_column) {
280   converted_dates <- try(as.Date(date_column, format = "%d-%m-%Y"), silent = TRUE)
281   if (inherits(converted_dates, "try-error")) {
282     return(format(mdy(date_column), "%d-%m-%Y"))

```

```

283   } else {
284     return(date_column)
285   }
286 }
287 order_details_table$order_date <- correct_date_format(order_details_table$order_date)
288
289 # Print a message based on the result
290 if (all(!inherits(try(as.Date(order_details_table$order_date, format = "%d-%m-%Y")), "try-error"))) {
291   print("Dates are now in the correct format")
292 } else {
293   print("There was an issue with converting the dates")
294 }
295 ## Payment_file
296 ### Checking NA values inside payment_id, payment_method, payment_status, and order_id
297 if (any(!is.na(payments_table[,c("payment_id", "payment_method", "payment_status", "order_id")])) {
298   print("There are no NA values in the dataset")
299 } else {
300   print("Error! There are NA values in the dataset")
301 }
302
303 ### Checking date format for the payment_date
304 if (all(!inherits(try(as.Date(payments_table$payment_date, format = "%d-%m-%Y")), "try-error"))) {
305   print("Dates are already in the correct format")
306 } else {
307   print("Dates are not in the correct format")
308 }
309
310 ## Products_file
311 ### Checking duplicate values in prod_id and review_id
312 if(length(products_table$prod_id[duplicated(products_table$prod_id)]) == 0) {
313   print("No duplicate prod_ids found")
314 } else {
315   print("Duplicate ad_ids found")
316 }
317
318 if(length(reviews_table$review_id[duplicated(reviews_table$review_id)]) == 0) {
319   print("No duplicate review_ids found")
320 } else {
321   print("Duplicate review_ids found")
322 }
323
324 ### Checking NA values inside prod_id, prod_url, prod_unit_price
325 if (any(!is.na(products_table[,c("prod_id", "prod_url", "prod_unit_price")])) {
326   print("There are no NA values in the dataset")
327 } else {
328   print("Error! There are NA values in the dataset")
329 }

```

330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376

```
### Checking date format for the review_date
check_and_correct_date_format <- function(date_column_data) {
  if (all(!inherits(try(as.Date(date_column_data, format = "%d-%m-%Y")), "try-error")))) {
    return(date_column_data)
  } else {
    return(format(mdy(date_column_data), "%d-%m-%Y"))
  }
}

reviews_table$review_date <- check_and_correct_date_format(reviews_table$review_date)
if (all(!inherits(try(as.Date(reviews_table$review_date, format = "%d-%m-%Y")), "try-error")))) {
  print("Review Dates are now in the correct format")
} else {
  print("There was an issue with converting the Review Dates")
}

### Checking the URL format of the prod_url
if(length(grep(("^(http|https)://[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}(\\S*)$"),products_table$prod_url))>0){
  print("All product url format are correct")
} else {
  print(paste("There are:", length(products_table$prod_url) - length(grep(("^(http|https)://[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}(\\S*)$"),products_table$prod_url))))
}

## Shipments file
### Checking duplicate values in shipment_id
if(length(shipments_table$shipment_id[duplicated(shipments_table$shipment_id)]) == 0) {
  print("No duplicate shipment_ids found")
} else {
  print("Duplicate shipment_ids found")
}

### Checking NA values inside shipment_id, prod_id, order_id
if (any(!is.na(shipments_table[,c("prod_id", "order_id", "shipment_id")]))) {
  print("There are no NA values in the dataset")
} else {
  print("Error! There are NA values in the dataset")
}

### Checking date format for the delivery_departed_date, delivery_received_date, est_delivery_date
check_and_correct_date_format <- function(date_column_data) {
  if (all(!inherits(try(as.Date(date_column_data, format = "%d-%m-%Y")), "try-error")))) {
    return(date_column_data) # Return the original data if format is correct
  } else {
    # Correcting the format assuming the original format is mm-dd-yyyy, adjust as necessary
    return(format(mdy(date_column_data), "%d-%m-%Y"))
  }
}
```

```

377
378 shipments_table$delivery_departed_date <- check_and_correct_date_format(shipments_table$delivery_departed_date)
379 shipments_table$delivery_received_date <- check_and_correct_date_format(shipments_table$delivery_received_date)
380 shipments_table$est_delivery_date <- check_and_correct_date_format(shipments_table$est_delivery_date)
381
382 columns_to_check <- list(
383   "Delivery Departed Date" = shipments_table$delivery_departed_date,
384   "Delivery Received Date" = shipments_table$delivery_received_date,
385   "Estimated Delivery Date" = shipments_table$est_delivery_date
386 )
387
388 for (column_name in names(columns_to_check)) {
389   if (all(!inherits(try(as.Date(columns_to_check[[column_name]]), format = "%d-%m-%Y")), "try-error")) {
390     print(paste(column_name, "are now in the correct format"))
391   } else {
392     print(paste("There was an issue with converting the", column_name))
393   }
394 }
395
396 ### Checking whether the recipient names contains ' and ,
397 if (any(grepl("[',]", shipments_table$delivery_recipient))) {
398   print("Error! Some names contain invalid characters")
399 } else {
400   print("All names are valid")
401 }
402
403 ## Customer Table
404 ### Checking duplicate values in customer_id
405 if(length(customers_table$cust_id[duplicated(customers_table$cust_id)]) == 0) {
406   print("No duplicate customer_ids found")
407 } else {
408   print("Duplicate customer_ids found")
409 }
410
411 ### Checking whether the customer's first name and last name contains ' and ,
412 clean_name <- function(name) {
413   gsub("['\\", "", name)
414 }
415
416 if (any(grepl("[',]", customers_table$first_name))) {
417   print("Error! Some first names contain invalid characters")
418   customers_table$first_name <- sapply(customers_table$first_name, clean_name)
419 } else {
420   print("All first names are valid")
421 }
422
423 if (any(grepl("[',]", customers_table$last_name))) {

```

```

424   print("Error! Some last names contain invalid characters")
425   customers_table$last_name <- sapply(customers_table$last_name, clean_name)
426 } else {
427   print("All last names are valid")
428 }
429
430
431 ### Checking the customer_email format
432 if(length(grep(("^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.\com$"),customers_table$cust_email, value = TRUE)) > 0) {
433   print("All customer email format are correct")
434 } else {
435   print(paste("There are:", length(customers_table$cust_email) - length(grep(("^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.\com$"),customers_table$cust_email, value = TRUE))))
436 }
437
438 ### Checking the customer birth_date format
439 if (all(!inherits(try(as.Date(customers_table$cust_birth_date, format = "%d-%m-%Y")), "try-error"))) {
440   print("Dates are already in the correct format")
441 } else {
442   print("Dates are not in the correct format")
443 }
444
445
446 # Create connection to SQL database
447 db_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"IB9HP0_9.db")
448
449 # Inserting Dataframe into the sql database
450 write_log <- function(message, path) {
451   timestamp <- format(Sys.time(), "%Y-%m-%d %H:%M:%S")
452   message <- paste(timestamp, message, sep=" - ")
453   write(message, file = path, append = TRUE, sep = "\n")
454 }
455
456 # Path for the error log file
457 log_file_path <- "error_log.txt"
458
459 ## Inserting Products table
460 for(i in 1:nrow(products_table)){
461   tryCatch({
462     dbExecute(db_connection, paste(
463       "INSERT INTO products(prod_id, prod_name, prod_desc, voucher, prod_url, prod_unit_price)
464       values(",
465       products_table$prod_id[i], "','",
466       products_table$prod_name[i], "','",
467       products_table$prod_desc[i], "','",
468       products_table$voucher[i], "','",
469       products_table$prod_url[i], "','",
470       products_table$prod_unit_price[i], ");", sep = " ")

```

```

471     write_log(sprintf("Product %s inserted successfully.", products_table$prod_id[i]), log_file)
472 }, error = function(e) {
473     write_log(sprintf("Failed to insert product %s: %s", products_table$prod_id[i], e$message))
474 })
475 }
476
477 ## Inserting Reviews table
478 for(i in 1:nrow(reviews_table)){
479     tryCatch({
480         dbExecute(db_connection, paste(
481             "INSERT INTO reviews(review_id, prod_rating, review_date, prod_id) VALUES('",
482             reviews_table$review_id[i], "','",
483             reviews_table$prod_rating[i], "','",
484             reviews_table$review_date[i], "','",
485             reviews_table$prod_id[i], "');", sep = "")
486         )
487         write_log(sprintf("Review %s inserted successfully.", reviews_table$review_id[i]), log_file)
488     }, error = function(e) {
489         write_log(sprintf("Failed to insert review %s: %s", reviews_table$review_id[i], e$message))
490     })
491 }
492
493 ## Inserting Memberships table
494 for(i in 1:nrow(memberships_table)){
495     tryCatch({
496         dbExecute(db_connection, paste(
497             "INSERT INTO memberships(membership_type_id, membership_type) VALUES(",
498             "'", memberships_table$membership_type_id[i], "','",
499             "'", memberships_table$membership_type[i], "');", sep = "")
500         )
501         write_log(sprintf("Membership %s inserted successfully.", memberships_table$membership_type[i]), log_file)
502     }, error = function(e) {
503         write_log(sprintf("Failed to insert membership %s: %s", memberships_table$membership_type[i], e$message))
504     })
505 }
506
507 ## Inserting Customers table
508 for(i in 1:nrow(customers_table)){
509     tryCatch({
510         dbExecute(db_connection, paste(
511             "INSERT INTO customers(cust_id, first_name, last_name, cust_email, password, cust_birth_date) VALUES(",
512             "'", customers_table$cust_id[i], "','",
513             "'", customers_table$first_name[i], "','",
514             "'", customers_table$last_name[i], "','",
515             "'", customers_table$cust_email[i], "','",
516             "'", customers_table$password[i], "','",
517             "'", customers_table$cust_birth_date[i], "','",

```

```

518     "", customers_table$address_type[i], "'",",
519     "", customers_table$block_num[i], "'",",
520     "", customers_table$postcode[i], "'",",
521     "", customers_table$cust_telephone[i], "'",",
522     "", customers_table$membership_type_id[i], "');", sep = "")
523 )
524 write_log(sprintf("Customer %s inserted successfully.", customers_table$cust_id[i]), log_file)
525 }, error = function(e) {
526     write_log(sprintf("Failed to insert customer %s: %s", customers_table$cust_id[i], e$message), log_file)
527 })
528 }
529
530 ## Inserting Orders table
531 for(i in 1:nrow(orders_table)){
532     tryCatch({
533         dbExecute(db_connection, paste(
534             "INSERT INTO orders(order_id, cust_id) VALUES('",
535             orders_table$order_id[i], "'",
536             orders_table$cust_id[i], "');", sep = "")
537         )
538         write_log(sprintf("Order %s inserted successfully.", orders_table$order_id[i]), log_file)
539     }, error = function(e) {
540         write_log(sprintf("Failed to insert order %s: %s", orders_table$order_id[i], e$message), log_file)
541     })
542 }
543
544 ## Inserting Payment table
545 for(i in 1:nrow(payments_table)){
546     tryCatch({
547         dbExecute(db_connection, paste(
548             "INSERT INTO payments(payment_id, payment_method, payment_amount, payment_status, payment_date, order_id) VALUES('",
549             "", payments_table$payment_id[i], "'",
550             "", payments_table$payment_method[i], "'",
551             payments_table$payment_amount[i], ",",
552             "", payments_table$payment_status[i], "'",
553             "", payments_table$payment_date[i], "'",
554             "", payments_table$order_id[i], "');", sep = "")
555         )
556         write_log(sprintf("Payment %s inserted successfully.", payments_table$payment_id[i]), log_file)
557     }, error = function(e) {
558         write_log(sprintf("Failed to insert payment %s: %s", payments_table$payment_id[i], e$message), log_file)
559     })
560 }
561
562 ## Inserting Shipment table
563 for(i in 1:nrow(shipments_table)){
564     tryCatch({

```



```

565     dbExecute(db_connection, paste(
566         "INSERT INTO shipments(shipment_id, delivery_status, delivery_fee, delivery_recipient, sh
567 , prod_id, order_id) VALUES(",
568         "'", shipments_table$shipment_id[i], "'",",
569         "'", shipments_table$delivery_status[i], "'",",
570         shipments_table$delivery_fee[i], ",",",
571         "'", shipments_table$delivery_recipient[i], "'",",
572         "'", shipments_table$shipper_name[i], "'",",
573         "'", format(as.Date(shipments_table$est_delivery_date[i], format = "%d-%m-%Y"), "%Y-%m-%
574         "'", format(as.Date(shipments_table$delivery_departed_date[i], format = "%d-%m-%Y"), "%Y-
575         "'", format(as.Date(shipments_table$delivery_received_date[i], format = "%d-%m-%Y"), "%Y-
576         "'", shipments_table$prod_id[i], "'",",
577         "'", shipments_table$order_id[i], "');", sep = "")
578     )
579     write_log(sprintf("Shipment %s inserted successfully.", shipments_table$shipment_id[i]), 1
580 }, error = function(e) {
581     write_log(sprintf("Failed to insert shipment %s: %s", shipments_table$shipment_id[i], e$message), 1
582 })
583 }
584
585 ## Inserting Order details table
586 for(i in 1:nrow(order_details_table)){
587     tryCatch({
588         dbExecute(db_connection, paste(
589             "INSERT INTO order_details(order_quantity,order_date,order_price,order_value,prod_id,ord
590             order_details_table$order_quantity[i], ",",",
591             "'", order_details_table$order_date[i], "'",",
592             order_details_table$order_price[i], ",",",
593             order_details_table$order_value[i], ",",",
594             "'", order_details_table$prod_id[i], "'",",
595             "'", order_details_table$order_id[i], "');",sep = "")
596         )
597
598         write_log(sprintf("Order detail %s inserted successfully.", order_details_table$order_id[i]), 1
599     }, error = function(e) {
600         write_log(sprintf("Failed to insert order detail %s: %s", order_details_table$order_id[i],
601     })
602 }
603
604 ## Inserting Suppliers table
605 for(i in 1:nrow(suppliers_table)){
606     tryCatch({
607         dbExecute(db_connection, paste(
608             "INSERT INTO suppliers(supplier_id, supplier_name, supplier_postcode, supplier_contact) V
609             suppliers_table$supplier_id[i], "'",",
610             suppliers_table$supplier_name[i], "'",",
611             suppliers_table$supplier_postcode[i], "'",",

```

```

612     suppliers_table$supplier_contact[i], "');", sep = "")
613 )
614     write_log(sprintf("Supplier %s inserted successfully.", suppliers_table$supplier_id[i]), 1)
615 }, error = function(e) {
616     write_log(sprintf("Failed to insert supplier %s: %s", suppliers_table$supplier_id[i], e$message))
617 })
618 }
619
620
621 ## Inserting Supplies table
622 for(i in 1:nrow(supplies_table)){
623     tryCatch({
624         dbExecute(db_connection, paste(
625             "INSERT INTO supplies(supply_id, inventory_quantity, sold_quantity, supplier_id, prod_id)",
626             supplies_table$supply_id[i], "','",
627             supplies_table$inventory_quantity[i], ",",
628             supplies_table$sold_quantity[i], "','",
629             supplies_table$supplier_id[i], "','",
630             supplies_table$prod_id[i], "');", sep = "")
631         )
632         write_log(sprintf("Supply %s inserted successfully.", supplies_table$supply_id[i]), log_file)
633     }, error = function(e) {
634         write_log(sprintf("Failed to insert supply %s: %s", supplies_table$supply_id[i], e$message))
635     })
636 }
637
638 ## Inserting Customer queries table
639 for(i in 1:nrow(customer_queries_table)){
640     tryCatch({
641         dbExecute(db_connection, paste(
642             "INSERT INTO customer_queries(query_id, query_title, query_submission_date, query_closure_date, query_status, cust_id)",
643             "'", customer_queries_table$query_id[i], "','",
644             "'", customer_queries_table$query_title[i], "','",
645             "'", customer_queries_table$query_submission_date[i], "','",
646             "'", customer_queries_table$query_closure_date[i], "','",
647             "'", customer_queries_table$query_status[i], "','",
648             "'", customer_queries_table$cust_id[i], "');", sep = "")
649         )
650         write_log(sprintf("Customer query %s inserted successfully.", customer_queries_table$query_id[i]), log_file)
651     }, error = function(e) {
652         write_log(sprintf("Failed to insert customer query %s: %s", customer_queries_table$query_id[i], e$message))
653     })
654 }
655
656 ## Inserting Categories table
657 for(i in 1:nrow(categories_table)){
658     tryCatch({

```

```

659     dbExecute(db_connection, paste(
660         "INSERT INTO categories(category_id, category_name) VALUES('",
661         categories_table$category_id[i], "','",
662         categories_table$category_name[i], "');" , sep = "")
663     )
664     write_log(sprintf("Category %s inserted successfully.", categories_table$category_id[i]), 1)
665 }, error = function(e) {
666     write_log(sprintf("Failed to insert category %s: %s", categories_table$category_id[i], e$message))
667 })
668 }
669
670 ## Inserting Product Categories table
671 for(i in 1:nrow(product_categories_table)){
672     tryCatch({
673         dbExecute(db_connection, paste(
674             "INSERT INTO product_categories(category_id, prod_id) VALUES('",
675             product_categories_table$category_id[i], "','",
676             product_categories_table$prod_id[i], "');" , sep = "")
677         )
678         write_log(sprintf("Product category link for product %s and category %s inserted successfully.", product_categories_table$prod_id[i], product_categories_table$category_id[i]), 1)
679     }, error = function(e) {
680         write_log(sprintf("Failed to insert product category link for product %s and category %s: %s", product_categories_table$prod_id[i], product_categories_table$category_id[i], e$message))
681     })
682 }
683
684 ## Inserting Advertisers table
685 for(i in 1:nrow(advertisers_table)) {
686     tryCatch({
687         dbExecute(db_connection, paste(
688             "INSERT INTO advertisers(advertiser_id, advertiser_name, advertiser_email) VALUES(",
689             "'", advertisers_table$advertiser_id[i], "','",
690             "'", advertisers_table$advertiser_name[i], "','",
691             "'", advertisers_table$advertiser_email[i], "');" , sep = "")
692         )
693         write_log(sprintf("Advertiser %s inserted successfully.", advertisers_table$advertiser_id[i]), 1)
694     }, error = function(e) {
695         write_log(sprintf("Failed to insert advertiser %s: %s", advertisers_table$advertiser_id[i], e$message))
696     })
697 }
698
699 ## Inserting Advertisements table
700 for(i in 1:nrow(advertisements_table)) {
701     tryCatch({
702         dbExecute(db_connection, paste(
703             "INSERT INTO advertisements(ads_id, ads_start_date, ads_end_date, prod_id, advertiser_id) VALUES(",
704             "'", advertisements_table$ads_id[i], "','",
705             "'", advertisements_table$ads_start_date[i], "','",

```

```

706     "", advertisements_table$ads_end_date[i], "','",
707     "", advertisements_table$prod_id[i], "','",
708     "", advertisements_table$advertiser_id[i], "');", sep = "")
709 )
710 write_log(sprintf("Advertisement %s inserted successfully.", advertisements_table$ads_id[i]),
711 }, error = function(e) {
712     error_message <- sprintf("Failed to insert advertisement %s: %s", advertisements_table$ads_id[i], e$message)
713     write_log(error_message, log_file_path)
714 })
715 }

```

## 3 Data Pipeline Generation

### 3.1 Introduction

In this section, the project is implemented effectively through multiple streamlined workflows and seamless collaboration using GitHub. The setup of the GitHub Repository includes multiple organized folders namely R, data\_update, data\_uploads, Data Analysis Results to store the files for each stage of the project. Apart from the folders the repository includes a README file which provides a brief description of the various attributes of the repository. The next section focuses on the automating aspect of the repository to ensure the integration of the workflows by working on specific triggers and actions to perform appropriate tasks.

### 3.2 GitHub Repository Structure

Our GitHub repository (IB9HP0\_Group\_9) has a clear and logical directory structure to support efficient project management and collaboration. The directory is structured as follows:

workflows folder: stores automated workflow profiles (e.g., CI/CD processes) for automating tasks such as data validation, updating databases, etc.

data\_uploads folder: Used to store raw data files for uploads. Files in this directory follow a clear naming convention, such as R\_synth\_customers\_round1.csv and

R\_synth\_customers\_round2.csv, in order to quickly identify the source of the data. The insertion of new data also follows this rule, ensuring consistent data management.

analysis\_results folder: used to store the results of the data analysis.

docs folder : holds project documents such as the analysis report Markdown file (IB9HP0\_9.Rmd ) and the project RStudio project file (ib9hp0\_group\_9.Rproj).

R folder: contains all R scripts covering data generation, table creation, data validation and insertion, and data analysis functions.

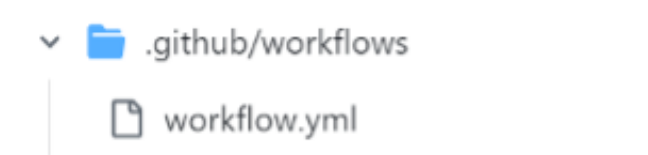
root: Includes .gitignore, README.md (the project description file), and the database file (IB9HP0\_9.db).

This structure ensures the seamless execution of multiple processes within the workflow of the repository.

### 3.3 GitHub Actions

#### 3.3.1 Trigger conditions

The GitHub repository consists of a Continuous integration Workflow. The Continuous Integration Workflow has been structured to automatically perform validation, insertion, updation and analysis parts of the project.



**Figure 4 Workflow of the E-Commerce Repository**

The Workflow consists of various functions mainly to setup the environment, install all required packages, and cache them. Similarly, all relevant dependencies after checking and restoring previously installed packages are installed. Following this, the various R scripts responsible for table creation, data validation, and data analysis are triggered.

Push to main branch: Every time a new data file or a change is committed to the repository, the workflow is triggered to run from its initial stages ensuring that all new data are run through the various above-mentioned stages of the project.

Timed run: the workflow is scheduled to run every 8 hours. This has been done to sync multiple new data points which might arise in the morning and evenings at the end of every shift.

#### 3.3.2 Automation Tasks

Check out code: Check out the latest code from the GitHub repository.

Setting up the R environment: Configure the runtime environment to use a specific version of R (in this case, version 4.3.1).

Cache R packages: Cache installed R packages for faster builds.

Install system dependencies: install system dependency packages necessary for R scripts to run, such as libcurl4-openssl-dev, libxml2-dev.

Install R dependencies: install necessary R packages, such as RSQLite, ggplot2, dplyr, etc. These packages are essential for data processing and analysis.

Execute R scripts: execute the R scripts stored in the R/ directory in order, including:

IB9HP0\_9\_synth\_1.R and IB9HP0\_9\_synth\_2.R: used to generate or process synthetic data.

IB9HP0\_9\_Table\_Creation.R: used to create database tables.

IB9HP0\_9\_Data\_Validation\_Insertion.R: for data validation and insertion into database operations.

IB9HP0\_9\_Data\_Analysis.R: to perform data analysis.

Git Operations:

Configure Git: sets the global Git username and email.

Add files: adds all changes to the Git staging area.

Commit files: commits changes to the repository, if any.

Push changes: use github-push-action to push commits back to the main branch.

Overall, a GitHub repository includes all the stages of the project from data validation, insertion, and updates. The repository can read new data files and ensure that the data is clean and suitable for the database through validation and then successfully insert and update the data frames. After successfully updating the database, the data is then run through various analysis and their respective results are stored and presented.

## 4 Data Analysis and Reporting

### 4.1 Monthly Sales Trend Analysis by Value and Volume, from 2022 to 2023

```
1 (sales_performance <-
2   dbGetQuery(db_connection,
3     "SELECT order_date AS date,
4       SUM(order_value) AS value_sales,
5       SUM(order_quantity) AS volume_sales,
6       SUM(order_value)/SUM(order_quantity) AS avg_price
7     FROM order_details
8     GROUP BY date
9     ORDER BY date"))
10
11 #### Transform data to get month and year
12 sales_performance <- sales_performance %>%
13   mutate(month = format(as.Date(date), "%m"),
14     year = format(as.Date(date), "%Y")) %>%
15   group_by(month, year) %>%
16   summarise(value_sales = sum(value_sales),
17     volume_sales = sum(volume_sales)) %>%
18   mutate(avg_price = value_sales/volume_sales)
19
20 #### Plot monthly value sales trend
21
22 p.mnth.val <- ggplot(filter(sales_performance,
23   year %in% c("2022", "2023")),
```

```

24     aes(x = month, group = year, color = year,
25         y = value_sales)) + geom_smooth(se = F, show.legend = F) +
26     labs(y = "Sales Value (GBP)", x = "Month",
27         subtitle = "Sales Value", color = "Year") +
28     theme_light() +
29     theme(plot.title = element_text(hjust = 0.5)) +
30     scale_y_continuous(labels = comma,
31         limits = c(0, 20000))
32
33 ##### Plot monthly volume sales trend
34 p.mnth.vol <- ggplot(filter(sales_performance,
35     year %in% c("2022", "2023")),
36     aes(x = month, group = year, color = year,
37         y = volume_sales)) +
38     geom_smooth(se = F, show.legend = F) +
39     labs(y = "Sales Volume (Units)", x = "Month",
40         subtitle = "Sales Volume", color = "Year") +
41     theme_light() +
42     theme(plot.title = element_text(hjust = 0.5)) +
43     scale_y_continuous(labels = comma,
44         limits = c(0, 500))
45 ##### Plot monthly avg price trend
46 p.mnth.price <- ggplot(filter(sales_performance,
47     year %in% c("2022", "2023")),
48     aes(x = month, group = year, color = year,
49         y = avg_price)) +
50     geom_smooth(se = F) +
51     labs(y = "Average Price (GBP/Unit)", x = "Month",
52         subtitle = "Average Price", color = "Year") +
53     theme_light() +
54     theme(plot.title = element_text(hjust = 0.5)) +
55     scale_y_continuous(labels = comma,
56         limits = c(0, 100))
57 ##### Combine value and volume sales graphs
58 gridExtra::grid.arrange(p.mnth.val, p.mnth.vol, p.mnth.price, ncol = 3, widths = c(0.9, 0.9, 1
59     top = ggpubr::text_grob("Monthly Sales Trend",
60         size = 15, face = "bold"))

```

Although the company's monthly sales decreased from around £12, 500 in January to around £7,500 in December during 2022, its monthly sales gradually increased from January to around £10,000 during 2013. Meanwhile, the company's overall trend of monthly sales is similar to that of monthly sales in 2022 and 2023.

## 4.2 Top 10 Products and Their Rating

```
1 ##### Get the data
2 (top_products_rating <-
3   dbGetQuery(db_connection,
4     "SELECT a.prod_name AS products,
5       SUM(b.order_quantity) AS volume_sales,
6       AVG(r.prod_rating) as avg_rating
7     FROM products a
8     JOIN order_details b ON a.prod_id = b.prod_id
9     JOIN reviews r ON a.prod_id = r.prod_id
10    GROUP BY a.prod_id
11    ORDER BY volume_sales DESC
12    LIMIT 10"))
13
14 ##### Plot product sales graph
15 p.top_prod_sales <- ggplot(top_products_rating,
16   aes(x= reorder(products, volume_sales))) +
17   geom_bar(aes(y = volume_sales), stat = "identity") + coord_flip() +
18   labs(x = "Products", y = "Volume Sales",
19     subtitle = "Volume Sales") +
20   theme_light() +
21   theme(plot.title = element_text(hjust = 0.5, face = "bold"),
22     axis.title.x = element_blank())
23 ##### Plot product ratings graph
24 p.top_prod_rating <- ggplot(top_products_rating,
25   aes(x= reorder(products, volume_sales))) +
26   geom_bar(aes(y = avg_rating), stat = "identity",
27     fill = "gray") + coord_flip() +
28   labs(y = "Product Rating",
29     subtitle = "Rating") +
30   theme_light() +
31   theme(plot.title = element_text(hjust = 0.5, face = "bold"),
32     axis.text.y = element_blank(),
33     axis.ticks.y = element_blank(),
34     axis.title = element_blank()) +
35   scale_y_continuous(limits = c(0,5))
36 ##### Combine value and volume sales graphs
37 gridExtra::grid.arrange(p.top_prod_sales, p.top_prod_rating, ncol = 2,
38   widths = c(1.1, 0.5),
39   top = ggpubr::text_grob("Top 10 Products and Their Rating",
40     size = 15, face = "bold"))
```

While the company's top 10 products have all totaled more than 400 sales, their corresponding ratings aren't the best among all products. Meanwhile, only two of these products have achieved a rating of 4 and three have a rating of only 1 (refer to lightweight backpacks, packing cubes and plush throw blankets), which warrants further improvement by the company.



### 4.3 Spending by Membership Type

```
1  ### Highest Spent based on Customer Segment
2  (membership_segmentation <-
3    dbGetQuery(db_connection,
4      "SELECT c.membership_type_id,
5        m.membership_type,
6        SUM(o.order_value) as total_spent,
7        o.order_date AS date
8      FROM customers c
9      JOIN memberships m ON c.membership_type_id = m.membership_type_id
10     JOIN orders d ON c.cust_id = d.cust_id
11     JOIN order_details o ON d.order_id = o.order_id
12     GROUP BY o.order_date, c.membership_type_id
13     ORDER BY total_spent DESC"))
14  ### Transform the data
15  membership_by_mnth_date <- membership_segmentation %>%
16    mutate("month" = format(as.Date(date), "%m"),
17          "year" = format(as.Date(date), "%Y")) %>%
18    group_by(membership_type, year) %>%
19    filter(year != 2024) %>%
20    summarise(total_spend = sum(total_spent))
21
22  ### Plot spending by membership type
23  p.membership <- ggplot(filter(membership_segmentation,
24    format(as.Date(date), "%Y") != 2024),
25    aes(x = membership_type,
26        y = total_spent)) +
27    geom_bar(stat = "identity", show.legend = F) +
28    labs(x = "Membership Type", y = "Total Spend (£)",
29      subtitle = "Spending") +
30    theme_light() +
31    theme(plot.title = element_text(hjust = 0.5, face = "bold")) +
32    scale_y_continuous(labels = comma,
33      limits = c(0, 90000))
34
35  ### Plot spending by membership type by year
36  p.membership_mnth <- ggplot(membership_by_mnth_date,
37    aes(fill = year, y = total_spend,
38        x = membership_type)) +
39    geom_col(position = "dodge", color = "white") +
40    labs(fill = "Year", x = "Membership Type", y = "Total Spend (£)",
41      subtitle = "Spending by Year") +
42    theme_light() +
43    theme(plot.title = element_text(hjust = 0.5, face = "bold"),
44      axis.title.y = element_blank(),
45      #axis.text.y = element_blank(),
```

```

46     axis.ticks.y = element_blank()) +
47     scale_y_continuous(labels = comma,
48                        limits = c(0, 90000))
49 ### Combine two membership charts
50 gridExtra::grid.arrange(p.membership, p.membership_mnth, ncol = 2,
51                          widths = c(0.5, 1.1),
52                          top = ggpubr::text_grob("Spending by Membership Type",
53                                                  size = 15, face = "bold"))

```

From the macro level, the total spent for all membership types is above 75,000 pounds between 2022 and 2023. However, in the micro view, the degree of decline in total spent for student type and premium type increases in descending order from 2022 to 2023, and only the trial type saw an increase.

## 4.4 Customer Queries Analysis

```

1  ### Most Frequent Queries - get data from db
2  (queries_frequencies <-
3    dbGetQuery(db_connection,
4               "SELECT query_title, COUNT(*) as frequencies
5               FROM customer_queries
6               GROUP BY query_title
7               ORDER BY frequencies DESC"))
8
9  ### Plot query types in terms of frequency
10 p.query_freq <- ggplot(queries_frequencies,
11                        aes(x= reorder(query_title, desc(frequencies)),
12                           y = frequencies)) +
13   geom_bar(stat = "identity") +
14   labs(x = "Query Type", y = "Frequency",
15        subtitle = "Frequency") +
16   theme_light() +
17   theme(plot.title = element_text(hjust = 0.5, face = "bold"))
18
19 ### Response Time Analysis for Customer Queries - get data from the db
20 (response_time <-
21   dbGetQuery(db_connection,
22              "SELECT query_id,
23                     query_title,
24                     query_closure_date,
25                     query_submission_date
26                     FROM customer_queries"))
27
28 ### Transform data - get turnaround time
29 response_time <- filter(response_time, query_closure_date != "NA") %>%

```

```

30 mutate(turnaround_time = round(difftime(query_closure_date, query_submission_date,
31                                         units = "weeks"),0) ) %>%
32 group_by(query_title) %>%
33 summarise(avg_turnaround_time = round(mean(turnaround_time),1)) %>%
34 merge(queries_frequencies, by = "query_title", remove = F)
35
36 ### Plot query by response time
37 h_line <- mean(response_time$avg_turnaround_time)
38 p.query_time <- ggplot(response_time,
39                        aes(x= reorder(query_title, desc(frequencies)),
40                          y = avg_turnaround_time)) +
41   geom_bar(stat = "identity") +
42   geom_hline(yintercept = h_line,
43             color = "magenta", linetype = "dashed", size = 1.1) +
44   geom_text(aes(1, h_line, label = "Avg of all types"),
45            vjust = -1, color = "magenta") +
46   labs(x = "Query Type", y = "Avg Turnaround Time (weeks)",
47        subtitle = "Turnaround Time") +
48   theme_light()
49 ### Combine frequency and turnaround time
50 gridExtra::grid.arrange(p.query_freq, p.query_time, ncol = 2,
51                        top = ggpubr::text_grob("Customer Queries",
52                                                size = 15, face = "bold"))

```

Though customers' queries include few delivery issue, the turnaround time that is the longest, which may have a significant negative effect on membership subscription.

## 4.5 Payment Methods

```

1  ### Get data from the db
2  (top_payment <-
3    dbGetQuery(db_connection,
4              "SELECT payment_method, COUNT(*) AS frequencies,
5                SUM(payment_amount) AS pymnt_amnt
6                FROM payments
7                GROUP BY payment_method
8                ORDER BY frequencies DESC"))
9  ### Plot payment method by frequency
10 p.frequency <- ggplot(top_payment, aes(x= reorder(payment_method, desc(frequencies)),
11                                         y = frequencies)) +
12   geom_bar(stat = "identity") +
13   labs(x = "Payment Method", y = "Frequency",
14        subtitle = "Frequently Used Payment Method") +
15   theme_light() +
16   theme(plot.title = element_text(hjust = 0.5, face = "bold")) +

```

```

17   scale_y_continuous(labels = comma)
18   ### Plot payment method by value
19   p.payment_amnt <- ggplot(top_payment,
20                           aes(x= reorder(payment_method, desc(frequencies)),
21                               y = pymnt_amnt)) +
22   geom_bar(stat = "identity") +
23   labs(x = "Payment Method", y = "Payment Amount (£)",
24        subtitle = "Payment Value") +
25   theme_light() +
26   theme(plot.title = element_text(hjust = 0.5, face = "bold")) +
27   scale_y_continuous(labels = comma)
28   ### Combine payment method by frequency and value
29   gridExtra::grid.arrange(p.frequency, p.payment_amnt, ncol = 2,
30                           top = ggpubr::text_grob("Payment Methods",
31                                                    size = 15, face = "bold"))
31

```

Obviously, Visa is the top 1 payment method in terms of frequency and payment amount. It's clear that although Mastercard is used by customers a little more often than Gpay, Gpay accounts for more of the payment amount. Gpay is considered to be the top 2 payment method as payment amount is a higher priority for the company, followed by Mastercard, Paypal, Bank Transfer, Apple Pay.

## 4.6 Insight and Recommendation

### 4.6.1 Insights:

It is worth noting that the increasing ratio of sales volume is higher than sales value's from July to October in 2023. It may be driven by the company's strategy of lowering prices more significantly during that period, where the average price of the product decreased from around 37.5 pounds in July to around 32 pounds in 2023, indicating price adjustment may simulate the monthly sales value.

The top 10 products cannot match the top rating degree, which can reflect the poor management of product life cycle.

Customers who try to subscribe to trial membership don't go further than subscribing to premium or student services, indicating the membership service cannot meet the real demand of the targeted customers like being wonderless, which will affect the value convey of business model for the e-commerce company.

Focusing on fastening the process of customer support re. Delivery concerns should be the priority, along with addressing payment issue given high number of concerns raised. Payment issue is also concerned because of the most customers queries compared with others, thus, it will affect the customer satisfaction directly.

#### **4.6.2 Recommendation:**

1. Reduce the price appropriately to get more orders When the market demand is weak.
2. Maintenance the product and service life cycle management like requiring suppliers to provide or create better products, improving the shipment quality even for membership.
3. Improve the design for the official website to fit the user's payment habits like putting Visa, Gpay and Mastercard at the beginning.
4. Give some promotes or discount on products in the order if they choose to use Mastercard payment method.

## **5 Limitation and Future Implications**

To be able to improved the performance of the e-commerce database, several limitations should be addressed. The current model focused primarily on capturing the key entities within the e-commerce database, yet it does not encompass the full spectrum of its complexity. Notably, supporting entities such as purchase returns have not been included in current model. Therefore, the next improvement should incorporate all the entities within the e-commerce hence the database will be centralised.

Additionally, it is important to integrate the database into the other e-commerce system. The database need to integrate seamlessly with other business systems such as inventory management, CRM, and logistics. Integration issues can lead to data silos and operational inefficiencies.

## **6 Conclusion**