

IB9HPO_9

5588654, 5577965, 5579058, 5587354, 5519743, 5588060

2024-03-02

Table of contents

1	Assignment Cover Sheet	2
2	Database Design and Implementation	3
2.1	Assumptions	3
2.2	Conceptual Design: Entities and Relationships	4
2.3	Logical Design	6
2.4	Physical Design	7
3	Data Generation and Management	12
3.1	Synthetic Data Generation	12
3.2	Data Import and Quality Assurance	30
4	Data Pipeline Generation	49
4.1	Introduction	49
4.2	GitHub Repository Structure	49
4.3	GitHub Actions	50
4.3.1	Trigger conditions	50
4.3.2	Automation Tasks	51
5	Data Analysis and Reporting	51
5.1	Monthly Sales Trend Analysis by Value and Volume, from 2022 to 2023	51
5.2	Top 10 Products and Their Rating	53
5.3	Spending by Membership Type	55
5.4	Customer Queries Analysis	57
5.5	Payment Methods Analysis	59
5.6	Insight and Recommendation	60
5.6.1	Insights:	60
5.6.2	Recommendation:	61
6	Limitation and Future Implications	61
7	Conclusion	61

1 Assignment Cover Sheet

Student Number: 5588654, 5577965, 5579058, 5587354, 5519743, 5588060

Module Code: IB9HPO

Module Title: Data Management

Submission Deadline: March 20th 2024

Date Submitted: March 19th 2024

Word Counts: 1940 words

Number of pages: 54 pages

Questions Attempted: 4 questions

Academic Integrity Declaration:

We're part of an academic community at Warwick. Whether studying, teaching, or researching, we're all taking part in an expert conversation which must meet standards of academic integrity. When we all meet these standards, we can take pride in our own academic achievements, as individuals and as an academic community.

Academic integrity means committing to honesty in academic work, giving credit where we've used others' ideas and being proud of our own achievements.

In submitting my work, I confirm that:

- I have read the guidance on academic integrity provided in the Student Handbook and understand the University regulations in relation to Academic Integrity. I am aware of the potential consequences of Academic Misconduct.
- I declare that the work is all my own, except where I have stated otherwise.
- No substantial part(s) of the work submitted here has also been submitted by me in other credit bearing assessments courses of study (other than in certain cases of a resubmission of a piece of work), and I acknowledge that if this has been done this may lead to an appropriate sanction.
- Where a generative Artificial Intelligence such as ChatGPT has been used I confirm I have abided by both the University guidance and specific requirements as set out in the Student Handbook and the Assessment brief. I have clearly acknowledged the use of any generative Artificial Intelligence in my submission, my reasoning for using it and which generative AI (or AIs) I have used. Except where indicated the work is otherwise entirely my own.
- I understand that should this piece of work raise concerns requiring investigation in relation to any of points above, it is possible that other work I have submitted for assessment will be checked, even if marks (provisional or confirmed) have been published.
- Where a proof-reader, paid or unpaid was used, I confirm that the proof-reader was made aware of and has complied with the University's proofreading policy.

Upon electronic submission of your assessment you will be required to agree to the statements above

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.

2 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.

2.1 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.

2.2 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 N:1.
3. Customer—Payment Each customer can make many payments, and each payment can be associated with one customer. Thus, cardinality is N: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

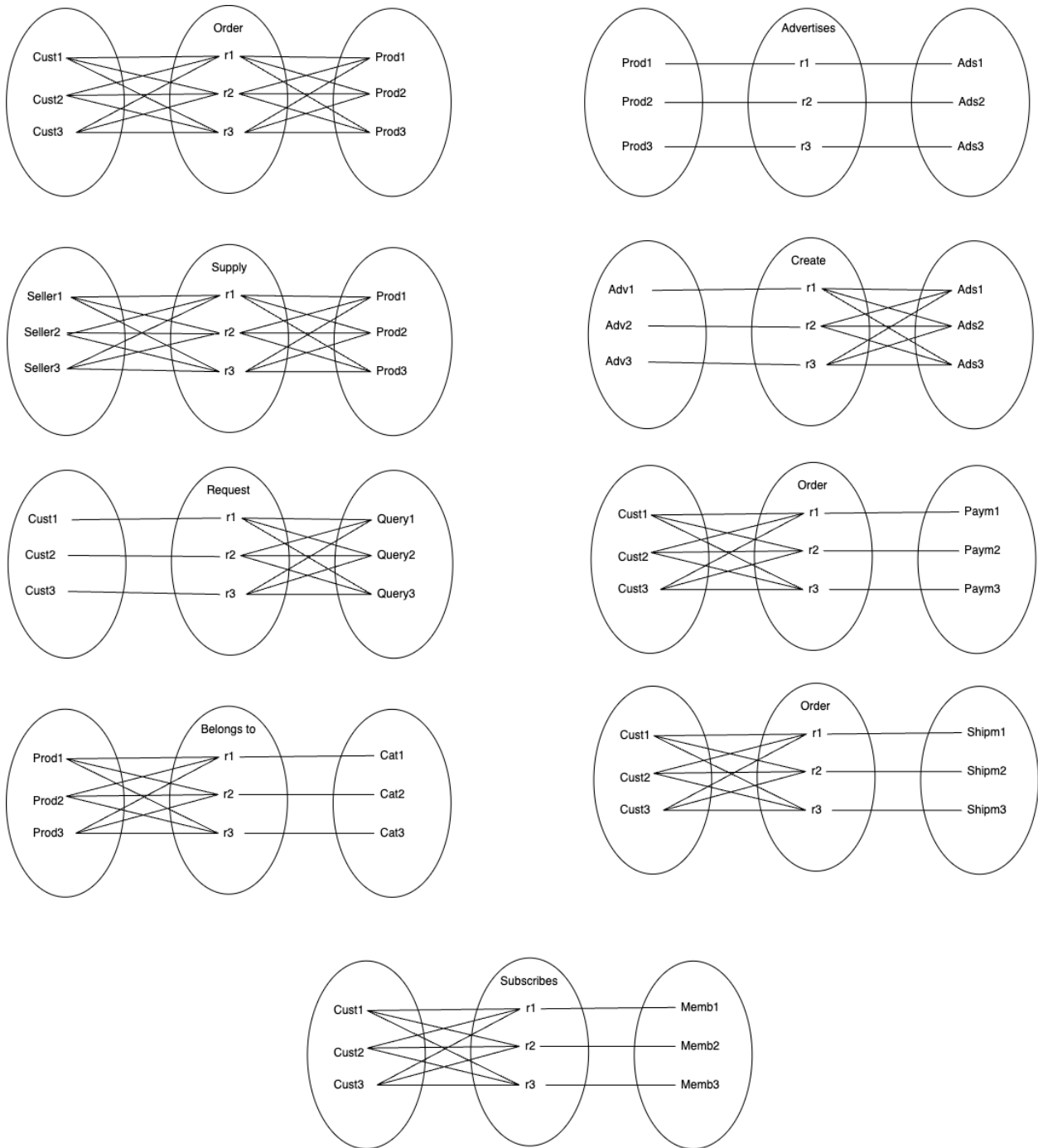


Figure 1. Relationship Sets

The relationship of all entities and its attributes is illustrated in the Figure 2 of the ERD.

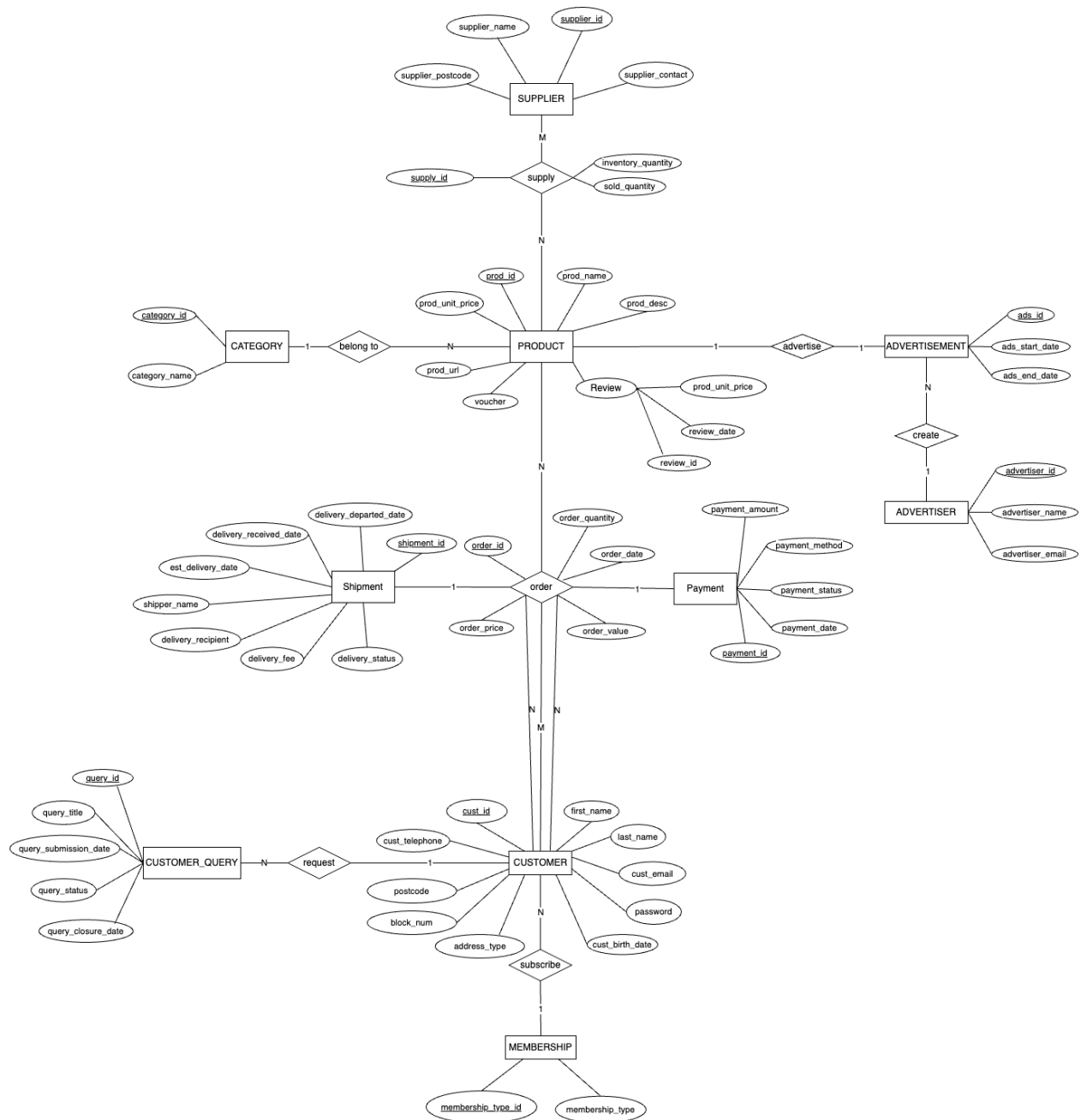


Figure 2. Entity Relationship Diagram (ERD)

2.3 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, first_name, last_name, cust_email, password, cust_birth_date, postcode, block_code, 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, order_quantity, order_date, order_price, order_value)
- supply (supply_id, supplier_id, prod_id, inventory_quantity, sold_quantity)
- 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_submission_date, query_closure_date, query_status)
- membership (membership_type_id, cust_id, membership_type)
- payment (payment_id, order_id, payment_method, payment_status, payment_date)
- shipment (shipment_id, order_id, prod_id, shipper_name, delivery_status, delivery_fee, delivery_recipient, est_delivery_date, delivery_receive_date, delivery_departed_date)

Figure 3. Logical Schema

2.4 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 decisions 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 )

```

3 Data Generation and Management

3.1 Synthetic Data Generation

The data used in the project is generated using several packages in R, including randomNames, dplyr, tidyr, charlatan, stringi, lubridate, and conjurer. Additionally, AI is used to generate names, categories, and descriptions. For the address, due to data privacy issue, the postcode is referred to the data provided by Office for National Statistics (ONS). The data is generated in one normal form (1NF) to ensure the consistency throughout the dataset. Additionally, the data is generated in two rounds, including first-time insertion and new data update for GitHub actions.

```

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"),
9                        by = "day"),
10                    n_customers)
11 cv_postcode <-
12   read.csv("data_uploads/ONSPD_AUG_2023_UK_CV.csv")[, 1] %>%
13   data.frame() %>%
14   setNames("pcd")
15 address_type <- c("Home", "Office")
16 #Create data
17 customers_data <-
18   #Create n unique customer IDs with random names
19   data.frame("cust_id" = conjurer::buildCust(n_customers),
20             "cust_name" = randomNames::randomNames(n_customers)) %>%
21   separate(cust_name, into = c("last_name", "first_name"), sep = ", ") %>%
22   #Create email column, by merging last & first name with email domain @gmail.com

```

```

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

```

```

70 #generate product id
71 conjurer::buildProd(n_prods, minPrice = 1, maxPrice = 100) %>%
72 #add product name and description from gemini's file
73 mutate("prod_name" = sample(gemini_prods$prod_name, 20)) %>%
74 left_join(select(gemini_prods, -c(seller_name, category)),
75           by = join_by(prod_name)) %>%
76 #rename columns to fit schema
77 rename(prod_id = SKU, prod_unit_price = Price) %>%
78 #rename `sku` with `prod`
79 mutate("prod_id" = gsub("sku", "prod", prod_id)) %>%
80 #add product url
81 mutate("web_prefix" = "https://group9.co.uk/",
82        "prod_url1" = gsub(" ", "-", prod_name)) %>%
83 unite(prod_url, c(prod_url1, prod_id), sep = "-", remove = F) %>%
84 unite(prod_url, c(web_prefix, prod_url), sep = "", remove = T) %>%
85 mutate(
86   #Create ratings
87   "prod_rating" = sample(ratings, n_prods, replace = T),
88   #Review date
89   "review_date" = sample(format(date, "%d-%m-%Y"), n_prods, replace = T),
90   #Assign review ID
91   "review_id" =
92     conjurer::buildCust(sum(!is.na(prod_rating))),
93   "review_id" = gsub("cust", "rev", review_id)) %>%
94 #drop temp url
95 select(-prod_url1)
96 #Create vouchers -- Randomly assign voucher types to 50% of the products
97 voucher_prods <- sample_n(data.frame(products_data$prod_id),
98                           0.5*nrow(products_data)) %>% setNames("prod_id")
99 products_data <- products_data %>%
100   mutate(voucher = ifelse(products_data$prod_id %in% voucher_prods$prod_id,
101                            sample(voucher_type, nrow(voucher_prods),
102                                    replace = T), NA))
103 #Finalise the table
104 products_data <-
105   products_data %>%
106   #rearrange order of columns
107   select(2,4,5,8,6,7,3,9,1)
108 #Save to .csv file
109 write.csv(products_data, "data_uploads/R_synth_products_round1.csv")
110
111 ### 'orders' table
112 #Define parameters
113 origin_date <- "1970-01-01"
114 n_orders <- 500
115 order_date <-
116   #round 1 is for orders in 2022-Mar'2024,

```

```

117 #so all orders have been paid and delivered successfully
118 sample(seq(from = as.Date("2022/01/01"),
119           to = as.Date("2024/03/01"), 12))
120 pymt_method <-
121   c("Bank Transfer", "Visa", "Mastercard", "PayPal", "GPay", "Apple Pay")
122 pymt_status <- c("Done", "Verifying")
123 shipper_lookup <-
124   data.frame("shipper_name" = c("DHL", "Group9DL", "DPD"),
125             "delivery_fee" = c(5,2,3),
126             "ETA" = c(1,5,3))
127 delivery_status <- c("Delivered", "In Progress",
128                    "Failed to contact", "Delayed")
129 orders_col_order <-
130   c("order_id", "cust_id", "prod_id", "order_quantity",
131     "order_date", "order_value", "order_price")
132 #generate n order IDs and assign customers to them, including order date
133 set.seed(122)
134 orders_data <-
135   #Create n unique order IDs
136   data.frame("order_id" = conjurer::buildCust(n_orders)) %>%
137   mutate(order_id = gsub("cust", "o", order_id),
138          payment_id = gsub("o", "pm", order_id),
139          cust_id = sample(customers_data$cust_id, n_orders, replace = T),
140          order_date = sample(order_date, n_orders, replace = T),
141          payment_method = sample(pymt_method, n_orders, replace = T),
142          payment_status = "Done",
143          delivery_recipient = randomNames::randomNames(n_orders,
144                                                         which.names = "first"))
145 #adding payment date with logic dependent on payment status
146 orders_data <- orders_data %>%
147   mutate("payment_date" = ifelse(payment_status == "Done", order_date, NA)) %>%
148   mutate("payment_date" = as.Date(payment_date,
149                                   origin = origin_date))
150 #randomly replicate certain orders to map with products
151 set.seed(122)
152 orders_data <- orders_data %>% bind_rows() %>%
153   rbind(sample_n(orders_data, 0.4*nrow(orders_data)),
154         sample_n(orders_data, 0.5*nrow(orders_data)),
155         sample_n(orders_data, 0.8*nrow(orders_data)))
156 #assign products to orders
157 orders_data <- orders_data %>%
158   mutate(
159     "prod_id" = sample(products_data$prod_id,
160                       nrow(orders_data), replace = T),
161     #generate order quantity
162     "order_quantity" = sample(seq(1,10,1), nrow(orders_data), replace = T)) %>%
163   merge(select(products_data, c(prod_id, prod_unit_price, voucher)),

```

```

164     by = "prod_id")
165 #Order value and shipper
166 orders_data <- orders_data %>%
167   #order price and value
168   mutate(
169     voucher = as.numeric(gsub("%", "", voucher))/100,
170     #product unit price is discounted in case of voucher available
171     order_price = ifelse(!is.na(voucher),
172                          prod_unit_price * voucher, prod_unit_price),
173     order_value = order_price * order_quantity,
174     #assign shippers to products
175     shipper_name =
176       sample(shipper_lookup$shipper_name, nrow(orders_data), replace = T),
177     #add delivery status
178     delivery_status = "Delivered" ) %>%
179     #lookup delivery fee
180     merge(shipper_lookup, by = "shipper_name")
181 #dates of delivery
182 orders_data <- orders_data %>%
183   #departure and ETA
184   mutate(
185     delivery_departed_date =
186       ifelse(!is.na(payment_date), (payment_date + days(2)), NA),
187     est_delivery_date = delivery_departed_date + ETA) %>%
188   #departure and ETA - format as date
189   mutate(
190     delivery_departed_date =
191       as.Date(delivery_departed_date, origin = origin_date),
192     est_delivery_date =
193       as.Date(est_delivery_date, origin = origin_date)) %>%
194   #received
195   mutate(
196     delivery_received_date =
197       ifelse(delivery_status != "Delivered", NA, est_delivery_date)) %>%
198   mutate(
199     delivery_received_date =
200       as.Date(delivery_received_date, origin = origin_date)) %>%
201   #drop ETA
202   select(-ETA)
203
204 ### generate 'shipment' from orders
205 shipment_colnames <- c("order_id", "prod_id",
206                       "delivery_departed_date",
207                       "delivery_received_date", "est_delivery_date",
208                       "shipper_name", "delivery_recipient",
209                       "delivery_fee", "delivery_status")
210 shipment_data <- select(orders_data, shipment_colnames)

```



```

211 shipment_data <- shipment_data %>%
212   mutate(shipment_id = paste("sm", rownames(shipment_data), sep = ""),
213     .before = "order_id")
214 #reformat date
215 shipment_dates <- c("delivery_departed_date",
216   "delivery_received_date", "est_delivery_date")
217 shipment_data[shipment_dates] <- lapply(shipment_data[shipment_dates],
218   format, "%d-%m-%Y")
219 #Save data to data file
220 write.csv(shipment_data, "data_uploads/R_synth_shipment_round1.csv")
221
222 ### generate 'payment' from orders
223 payment_colnames <- c("payment_id", "payment_method", "order_id",
224   "payment_status", "payment_date")
225 #Add payment amount
226 payment_data <- orders_data %>% group_by(payment_id) %>%
227   summarise(payment_amount = sum(order_value)) %>%
228   left_join(select(orders_data, payment_colnames), by = "payment_id")
229 #remove duplicates
230 payment_data <- distinct(payment_data, payment_id, .keep_all = T) %>%
231   select(1,4,3,2,5,6)
232 #re-format date
233 payment_data$payment_date <- format(payment_data$payment_date, "%d-%m-%Y")
234 #Save data to data file
235 write.csv(payment_data, "data_uploads/R_synth_payment_round1.csv")
236
237 #reorder the columns of 'orders'
238 orders_data <- select(orders_data, orders_col_order)
239 #Save data to data file
240 write.csv(orders_data, "data_uploads/R_synth_orders_round1.csv")
241
242 ### 'suppliers' table
243 #Define parameters for suppliers table
244 set.seed(123)
245 n_suppliers <- length(unique(gemini_prods$seller_name))
246 wc_postcode <- read.csv("data_uploads/ONSPD_AUG_2023_UK_WC.csv")[,1]
247
248 #Create suppliers table
249 suppliers_data <-
250   #Pull seller name from gemini file
251   distinct(select(gemini_prods, seller_name)) %>%
252   rename("supplier_name" = "seller_name") %>%
253   mutate("supplier_id" = seq(1, n_suppliers, 1),
254     "prefix" = "s") %>%
255   unite(supplier_id, c(prefix, supplier_id), sep = "", remove = T) %>%
256   mutate(
257     "supplier_postcode" =

```

```

258     sample(wc_postcode, n_suppliers, replace = T),
259     #Adding the phone code in UK
260     "phone_domain" = "079",
261     #create unique random strings of 7 digits
262     "supplier_contact" =
263       stringi::stri_rand_strings(n=n_suppliers, length=7, pattern="[0-9]")) %>%
264     #Adding supplier's telephone number by merging two phone number columns
265     unite(supplier_contact,
266           c(phone_domain, supplier_contact), sep = "", remove = T) %>%
267     select(2,1,4,3)
268 #Save data to data file
269 write.csv(suppliers_data, "data_uploads/R_synth_suppliers_round1.csv")
270
271 ### 'supply' table
272 #Define parameters for supply table
273 set.seed(123)
274 order_quant_by_prod <- orders_data %>%
275   group_by(prod_id) %>% summarise(sold_quantity = sum(order_quantity))
276 supply_col_order <- c("supply_id", "supplier_id", "prod_id",
277                      "inventory_quantity", "sold_quantity")
278 #Create supply table
279 supply_data <- select(products_data, c(prod_id, prod_name)) %>%
280   merge(order_quant_by_prod, by = "prod_id") %>%
281   mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
282   mutate(inventory_quantity =
283           as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
284   merge(select(gemini_prods, c(seller_name, prod_name)), by = "prod_name") %>%
285   rename("supplier_name" = "seller_name") %>%
286   merge(select(suppliers_data, c(supplier_id, supplier_name)),
287         by = "supplier_name")
288 #Create competitors for M:N relationship
289 supply_competitors <- select(products_data, c(prod_id, prod_name)) %>%
290   mutate(supplier_name =
291           sample(suppliers_data$supplier_name, n_prods, replace = T)) %>%
292   merge(select(suppliers_data, c(supplier_id, supplier_name)),
293         by = "supplier_name") %>%
294   merge(order_quant_by_prod, by = "prod_id") %>%
295   mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
296   mutate(inventory_quantity =
297           as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
298   select(2,3,1,5,6,4)
299 #Combine supply and competitors
300 supply_data <-
301   rbind(supply_data, supply_competitors) %>%
302   mutate(supply_id = paste("sp", row_number(), sep = "")) %>%
303   select(-c(supplier_name, prod_name))
304 #reorder columns

```

```

305 supply_data <- supply_data[, supply_col_order]
306 #Save data to data file
307 write.csv(supply_data, "data_uploads/R_synth_supply_round1.csv")
308
309 ### 'memberships' table
310 membership_lookup <-
311   data.frame(
312     "membership_type" = c("Student", "Trial", "Premium")) %>%
313     mutate("membership_type_id" = row_number())
314
315 #Start with the foreign key cust_id
316 set.seed(123)
317 memberships_data <- data.frame(customers_data$cust_id)
318 memberships_data <- memberships_data %>%
319   #Randomly assign membership type to all customers
320   mutate("membership_type" =
321     sample(membership_lookup$membership_type,
322           nrow(memberships_data), replace = T)) %>%
323   #Lookup membership_id
324   merge(membership_lookup, by = "membership_type") %>%
325   rename(cust_id = customers_data.cust_id) %>%
326   select(3,2,1)
327 #Save to .csv file
328 write.csv(memberships_data, "data_uploads/R_synth_memberships_round1.csv")
329
330 ### 'customer_queries' table
331 set.seed(123)
332 n_queries <- 20
333 customer_queries_data <- data.frame(
334   query_id = sprintf("Q%d", 1:n_queries),
335   cust_id = sample(customers_data$cust_id, n_queries, replace = TRUE),
336   query_title = sample(c(
337     "Delivery Issue", "Payment Issue", "Purchase Return", "Damaged Product",
338     "Wrong Delivery"), n_queries, replace = TRUE),
339   query_submission_date = sample(seq(as.Date('2023-01-01'),
340                                     as.Date('2023-1-31'), by="day"), n_queries,
341                                 replace = TRUE),
342   query_closure_date = sample(seq(as.Date('2023-02-01'),
343                                   as.Date('2023-03-31'), by="day"), n_queries,
344                               replace = TRUE),
345   query_status = sample(c("Closed"), n_queries, replace = TRUE)
346 )
347
348 customer_queries_data$query_submission_date <- format(
349   customer_queries_data$query_submission_date, "%d-%m-%Y")
350 customer_queries_data$query_closure_date <- format(
351   customer_queries_data$query_closure_date, "%d-%m-%Y")

```

```

352
353 #Save to .csv file
354 write.csv(
355     customer_queries_data, "data_uploads/R_synth_customer_queries_round1.csv",
356     row.names = FALSE)
357
358 ### 'categories' table
359 #create lookup table for category_id and category name
360 set.seed(123)
361 category_lookup <-
362     data.frame("category_id" = seq(1, length(unique(gemini_prods$category)),1),
363               "category" = unique(gemini_prods$category),
364               "cate_code" = "cate") %>%
365     unite(category_id, c(cate_code, category_id), sep = "", remove = T)
366 #Create categories table
367 categories_data <-
368     #Pull category name and product name from gemini file
369     select(gemini_prods, c(category, prod_name)) %>%
370     #Only keep the products included in the products table
371     right_join(select(products_data, c(prod_id, prod_name)),
372               by = "prod_name") %>%
373     #lookup category_id
374     merge(category_lookup, by = "category") %>%
375     #rename to have category_name column
376     rename(category_name = category) %>%
377     #drop product name column
378     select(-prod_name) %>%
379     #reorder the columns to match with table schema
380     select(3,2,1)
381 #Save to .csv file
382 write.csv(categories_data, "data_uploads/R_synth_categories_round1.csv")
383
384 ### 'advertisers' table
385 set.seed(123)
386 n_advertisers <- 5
387 advertisers_data <- data.frame(
388     advertiser_id = sprintf("ADV%d", 1:n_advertisers),
389     advertiser_name = c("Ads Life", "Ads Idol", "Ads is Life",
390                       "Ads Master", "Ads Expert"),
391     advertiser_email = sprintf("advertiser%d@gmail.com", 1:n_advertisers)
392 )
393 #Save to .csv file
394 write.csv(advertisers_data, "data_uploads/R_synth_advertisers_round1.csv",
395           row.names = FALSE)
396
397 ### 'advertisements' table
398 set.seed(123)

```

```

399 n_ads <- 9
400 advertisements_data <- data.frame(
401   ads_id = sprintf("ADS%d", 1:n_ads),
402   prod_id = sample(products_data$prod_id, n_ads, replace = TRUE),
403   advertiser_id = sample(advertisers_data$advertiser_id, n_ads, replace = TRUE),
404   ads_start_date = sample(seq(as.Date('2023-01-01'),
405                               as.Date('2023-12-31'), by="day"), n_ads,
406                           replace = TRUE),
407   ads_end_date = sample(seq(as.Date('2024-01-01'),
408                             as.Date('2024-12-31'), by="day"), n_ads,
409                           replace = TRUE)
410 )
411
412 advertisements_data$ads_start_date <- format(
413   advertisements_data$ads_start_date, "%d-%m-%Y")
414 advertisements_data$ads_end_date <- format(
415   advertisements_data$ads_end_date, "%d-%m-%Y")
416
417 #Save to .csv file
418 write.csv(advertisements_data,
419           "data_uploads/R_synth_advertisements_round1.csv", row.names = FALSE)
420
421 ### 'customers' table
422 #Define parameters for customers
423 set.seed(456)
424 n_customers <- 100
425 birthdate <- sample(seq(from = as.Date(today() - years(80), "%d-%m-%Y"),
426                           to = as.Date(today() - years(18), "%d-%m-%Y"),
427                           by = "day"),
428                     n_customers)
429
430 cv_postcode <-
431   read.csv("data_uploads/ONSPD_AUG_2023_UK_CV.csv")[, 1] %>%
432   data.frame() %>%
433   setNames("pcd")
434 address_type <- c("Home", "Office")
435 #Create data
436 customers_data <-
437   #Create n unique customer IDs with random names
438   data.frame("cust_id" = paste("cust", seq(101,101+n_customers-1,1), sep = ""),
439             "cust_name" = randomNames::randomNames(n_customers)) %>%
440   separate(cust_name, into = c("last_name", "first_name"), sep = ", ") %>%
441   #Create email column, by merging last & first name with email domain
442   unite(cust_email, c(last_name, first_name), sep = ".", remove = F) %>%
443   mutate(
444     "cust_email" = paste(cust_email,"gmail.com", sep = "@"),
445     #Generate user's password, using random string generation package
446     "password" =

```

```

446     stringi::stri_rand_strings(n=n_customers, length=8,
447                               pattern="[A-Za-z0-9]"),
448   #Adding customer BOD
449   "cust_birth_date" = sample(birthdate, n_customers, replace = T),
450   #Adding the phone code in UK
451   "phone_domain" = "075",
452   #create unique random strings of 7 digits
453   "cust_telephone" =
454     stringi::stri_rand_strings(n=n_customers, length=7, pattern="[0-9]"),
455   "block_num" =
456     sprintf("%s%s",
457             stri_rand_strings(n=n_customers, length=1, pattern="[A-Z]"),
458             stri_rand_strings(n=n_customers, length=2, pattern="[0-99]")),
459   #randomly assign postcode to each customer
460   "postcode" = cv_postcode[sample(nrow(cv_postcode), n_customers),],
461   #randomly assign address type to each customer
462   "address_type" = sample(address_type, n_customers, replace = T)) %>%
463   #Adding customer's telephone number by merging two phone number columns
464   unite(cust_telephone,
465         c(phone_domain, cust_telephone), sep = "", remove = T) %>%
466   #reorder the columns
467   select(1,4,3,2,5,6,8,9,10,7)
468 customers_data$cust_birth_date <- format(
469   customers_data$cust_birth_date, "%d-%m-%Y")
470 #Save data to data file
471 write.csv(customers_data, "data_uploads/R_synth_customers_round2.csv")
472
473 ### 'products' table
474 #Getting brand and product names from Gemini
475 gemini_prods <-
476   readxl::read_excel("data_uploads/gemini_prod_cate_supplier_2.xlsx",
477                     .name_repair = "universal") %>%
478   setNames(c("seller_name", "category", "prod_name", "prod_desc"))
479 #Define parameters for products
480 set.seed(456)
481 n_prods <- 19
482 voucher_type <- c("10%", "20%", "50%")
483 ratings <- c(1,2,3,4,5)
484 date <- #assuming company was established on Mar 06th 2004
485   sample(seq(from = as.Date("2004/03/06"),
486             to = as.Date(lubridate::today()), by = "day"), 12)
487 #Assign product ID, and adding product names and URL
488 products_data <-
489   #generate product id
490   conjurer::buildProd(n_prods, minPrice = 1, maxPrice = 100) %>%
491   #add product name and description from gemini's file
492   mutate("prod_name" = sample(gemini_prods$prod_name, nrow(gemini_prods))) %>%

```

```

493 left_join(select(gemini_prods, -c(seller_name, category)),
494           by = join_by(prod_name)) %>%
495 #rename columns to fit schema
496 rename(prod_id = SKU, prod_unit_price = Price) %>%
497 #rename `sku` with `prod`
498 mutate("prod_id" = gsub("sku", "", prod_id)) %>%
499 mutate("prod_id" = paste("prod", as.numeric(prod_id)+20, sep = "")) %>%
500 #add product url
501 mutate("web_prefix" = "https://group9.co.uk/",
502        "prod_url1" = gsub(" ", "-", prod_name)) %>%
503 unite(prod_url, c(prod_url1, prod_id), sep = "-", remove = F) %>%
504 unite(prod_url, c(web_prefix, prod_url), sep = "", remove = T) %>%
505 mutate(
506   #Create ratings
507   "prod_rating" = sample(ratings, n_prods, replace = T),
508   #Review date
509   "review_date" = sample(format(date, "%d-%m-%Y"), n_prods, replace = T),
510   #Assign review ID
511   "review_id" = paste("rev", seq(21, 21+n_prods-1, 1), sep = ""),
512   "review_id" = gsub("cust", "rev", review_id)) %>%
513 #drop temp url
514 select(-prod_url1)
515 #Create vouchers -- Randomly assign voucher types to 50% of the products
516 voucher_prods <- sample_n(data.frame(products_data$prod_id),
517                            0.5*nrow(products_data)) %>% setNames("prod_id")
518 products_data <- products_data %>%
519   mutate(voucher = ifelse(products_data$prod_id %in% voucher_prods$prod_id,
520                            sample(voucher_type, nrow(voucher_prods),
521                                   replace = T), NA))
522 #Finalise the table
523 products_data <-
524   products_data %>%
525   #rearrange order of columns
526   select(2,4,5,8,6,7,3,9,1)
527 #Save to .csv file
528 write.csv(products_data, "data_uploads/R_synth_products_round2.csv")
529
530 ### 'orders' table
531 #Define parameters
532 origin_date <- "1970-01-01"
533 n_orders <- 100
534 order_date <- #round 2 is for orders in 2024
535   sample(seq(from = as.Date("2024/03/01"),
536             to = as.Date(lubridate::today()), by = "day"), 12)
537 pymt_method <-
538   c("Bank Transfer", "Visa", "Mastercard", "PayPal", "GPay", "Apple Pay")
539 pymt_status <- c("Done", "Verifying")

```



```

540 shipper_lookup <-
541   data.frame("shipper_name" = c("DHL", "Group9DL", "DPD"),
542             "delivery_fee" = c(5,2,3),
543             "ETA" = c(1,5,3))
544 delivery_status <- c("Delivered", "In Progress",
545                     "Failed to contact", "Delayed")
546 orders_col_order <-
547   c("order_id", "cust_id", "prod_id", "order_quantity",
548     "order_date", "order_value", "order_price")
549 #generate n order IDs and assign customers to them, including order date
550 set.seed(321)
551 orders_data <-
552   #Create n unique order IDs
553   data.frame("order_id" = paste("o",seq(501, 501+n_orders-1, 1), sep = "")) %>%
554   mutate(order_id = gsub("cust", "o", order_id),
555          payment_id = gsub("o", "pm", order_id),
556          cust_id = sample(customers_data$cust_id, n_orders, replace = T),
557          order_date = sample(order_date, n_orders, replace = T),
558          payment_method = sample(pymt_method, n_orders, replace = T),
559          payment_status = sample(pymt_status, n_orders, replace = T),
560          delivery_recipient = randomNames::randomNames(n_orders,
561                                                         which.names = "first"))
562 #adding payment date with logic dependent on payment status
563 orders_data <- orders_data %>%
564   mutate("payment_date" = ifelse(payment_status == "Done", order_date, NA)) %>%
565   mutate("payment_date" = as.Date(payment_date,
566                                   origin = origin_date))
567 #randomly replicate certain orders to map with products
568 set.seed(456)
569 orders_data <- orders_data %>% bind_rows() %>%
570   rbind(sample_n(orders_data, 0.4*nrow(orders_data)),
571         sample_n(orders_data, 0.5*nrow(orders_data)),
572         sample_n(orders_data, 0.8*nrow(orders_data)))
573 #assign products to orders
574 orders_data <- orders_data %>%
575   mutate(
576     "prod_id" = sample(products_data$prod_id,
577                       nrow(orders_data), replace = T),
578     #generate order quantity
579     "order_quantity" = sample(seq(1,10,1), nrow(orders_data), replace = T)) %>%
580   merge(select(products_data, c(prod_id, prod_unit_price, voucher)),
581         by = "prod_id")
582 #Order value and shipper
583 orders_data <- orders_data %>%
584   #order price and value
585   mutate(
586     voucher = as.numeric(gsub("%", "", voucher))/100,

```



```

587 #product unit price is discounted in case of voucher available
588 order_price = ifelse(!is.na(voucher),
589                      prod_unit_price * voucher, prod_unit_price),
590 order_value = order_price * order_quantity,
591 #assign shippers to products
592 shipper_name =
593   sample(shipper_lookup$shipper_name, nrow(orders_data), replace = T),
594 #add delivery status
595 delivery_status =
596   ifelse(payment_status != "Done", "Not Started",
597          sample(delivery_status, nrow(orders_data), replace = T)) ) %>%
598 #lookup delivery fee
599 merge(shipper_lookup, by = "shipper_name")
600 #dates of delivery
601 orders_data <- orders_data %>%
602   #departure and ETA
603   mutate(
604     delivery_departed_date =
605       ifelse(!is.na(payment_date), (payment_date + days(2)), NA),
606     est_delivery_date = delivery_departed_date + ETA) %>%
607   #departure and ETA - format as date
608   mutate(
609     delivery_departed_date =
610       as.Date(delivery_departed_date, origin = origin_date),
611     est_delivery_date =
612       as.Date(est_delivery_date, origin = origin_date)) %>%
613   #received
614   mutate(
615     delivery_received_date =
616       ifelse(delivery_status != "Delivered", NA, est_delivery_date)) %>%
617   mutate(
618     delivery_received_date =
619       as.Date(delivery_received_date, origin = origin_date)) %>%
620   #drop ETA
621   select(-ETA)
622
623 ### generate 'shipment' from orders
624 shipment_colnames <- c("order_id", "prod_id",
625                       "delivery_departed_date",
626                       "delivery_received_date", "est_delivery_date",
627                       "shipper_name", "delivery_recipient",
628                       "delivery_fee", "delivery_status")
629 shipment_data <- select(orders_data, shipment_colnames)
630 shipment_data <- shipment_data %>%
631   mutate(shipment_id = paste("sm", rownames(shipment_data), sep = ""),
632          .before = "order_id")
633 #reformat date

```

```

634 shipment_dates <- c("delivery_departed_date",
635                     "delivery_received_date", "est_delivery_date")
636 shipment_data[shipment_dates] <- lapply(shipment_data[shipment_dates],
637                                         format, "%d-%m-%Y")
638 #Save data to data file
639 write.csv(shipment_data, "data_uploads/R_synth_shipment_round2.csv")
640
641 ### generate 'payment' from orders
642 payment_colnames <- c("payment_id", "payment_method", "order_id",
643                       "payment_status", "payment_date")
644 #Add payment amount
645 payment_data <- orders_data %>% group_by(payment_id) %>%
646   summarise(payment_amount = sum(order_value)) %>%
647   left_join(select(orders_data, payment_colnames), by = "payment_id")
648 #remove duplicates
649 payment_data <- distinct(payment_data, payment_id, .keep_all = T) %>%
650   select(1,4,3,2,5,6)
651 #re-format date
652 payment_data$payment_date <- format(payment_data$payment_date, "%d-%m-%Y")
653 #Save data to data file
654 write.csv(payment_data, "data_uploads/R_synth_payment_round2.csv")
655
656 #reorder the columns of 'orders'
657 orders_data <- select(orders_data, orders_col_order)
658 #Save data to data file
659 write.csv(orders_data, "data_uploads/R_synth_orders_round2.csv")
660
661 ### 'suppliers' table
662 #Define parameters for suppliers table
663 set.seed(456)
664 n_suppliers <- length(unique(gemini_prods$seller_name))
665 wc_postcode <- read.csv("data_uploads/ONSPD_AUG_2023_UK_WC.csv")[,1]
666
667 #Create suppliers table
668 suppliers_data <-
669   #Pull seller name from gemini file
670   distinct(select(gemini_prods, seller_name)) %>%
671   rename("supplier_name" = "seller_name") %>%
672   mutate("supplier_id" = seq(21, 21+n_suppliers-1,1),
673          "prefix" = "s") %>%
674   unite(supplier_id, c(prefix, supplier_id), sep = "", remove = T) %>%
675   mutate(
676     "supplier_postcode" =
677       sample(wc_postcode, n_suppliers, replace = T),
678     #Adding the phone code in UK
679     "phone_domain" = "079",
680     #create unique random strings of 7 digits

```

```

681   "supplier_contact" =
682     stringi::stri_rand_strings(n=n_suppliers, length=7, pattern="[0-9]")) %>%
683   #Adding supplier's telephone number by merging two phone number columns
684   unite(supplier_contact,
685     c(phone_domain, supplier_contact), sep = "", remove = T) %>%
686   select(2,1,4,3)
687 #Save data to data file
688 write.csv(suppliers_data, "data_uploads/R_synth_suppliers_round2.csv")
689
690 ### 'supply' table
691 #Define parameters for supply table
692 set.seed(456)
693 order_quant_by_prod <- orders_data %>%
694   group_by(prod_id) %>% summarise(sold_quantity = sum(order_quantity))
695 supply_col_order <- c("supply_id", "supplier_id", "prod_id",
696   "inventory_quantity", "sold_quantity")
697 #Create supply table
698 supply_data <- select(products_data, c(prod_id, prod_name)) %>%
699   merge(order_quant_by_prod, by = "prod_id") %>%
700   mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
701   mutate(inventory_quantity =
702     as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
703   merge(select(gemini_prods, c(seller_name, prod_name)), by = "prod_name") %>%
704   rename("supplier_name" = "seller_name") %>%
705   merge(select(suppliers_data, c(supplier_id, supplier_name)),
706     by = "supplier_name")
707 #Create competitors for M:N relationship
708 supply_competitors <- select(products_data, c(prod_id, prod_name)) %>%
709   mutate(supplier_name =
710     sample(suppliers_data$supplier_name, n_prods, replace = T)) %>%
711   merge(select(suppliers_data, c(supplier_id, supplier_name)),
712     by = "supplier_name") %>%
713   merge(order_quant_by_prod, by = "prod_id") %>%
714   mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
715   mutate(inventory_quantity =
716     as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
717   select(2,3,1,5,6,4)
718 #Combine supply and competitors
719 supply_data <-
720   rbind(supply_data, supply_competitors) %>%
721   mutate(supply_id = paste("sp", row_number(), sep = "")) %>%
722   select(-c(supplier_name, prod_name))
723 #reorder columns
724 supply_data <- supply_data[, supply_col_order]
725 #Save data to data file
726 write.csv(supply_data, "data_uploads/R_synth_supply_round2.csv")
727

```

```

728 ### 'memberships' table
729 membership_lookup <-
730   data.frame(
731     "membership_type" = c("Student", "Trial", "Premium")) %>%
732     mutate("membership_type_id" = row_number())
733
734 #Start with the foreign key cust_id
735 set.seed(456)
736 memberships_data <- data.frame(customers_data$cust_id)
737 memberships_data <- memberships_data %>%
738   #Randomly assign membership type to all customers
739   mutate("membership_type" =
740     sample(membership_lookup$membership_type,
741           nrow(memberships_data), replace = T)) %>%
742   #Lookup membership_id
743   merge(membership_lookup, by = "membership_type") %>%
744   rename(cust_id = customers_data.cust_id) %>%
745   select(3,2,1)
746 #Save to .csv file
747 write.csv(memberships_data, "data_uploads/R_synth_memberships_round2.csv")
748
749 ### 'customer_queries' table
750 set.seed(456)
751 n_queries <- 20
752 customer_queries_data <- data.frame(
753   "query_id" = paste("Q",seq(21, 21+n_queries-1, 1), sep = ""),
754   cust_id = sample(customers_data$cust_id, n_queries, replace = TRUE),
755   query_title = sample(c(
756     "Delivery Issue", "Payment Issue", "Purchase Return", "Damaged Product",
757     "Wrong Delivery"), n_queries, replace = TRUE),
758   query_submission_date = sample(seq(as.Date('2024-03-15'),
759                                     as.Date('2024-03-20'), by="day"),
760                                 n_queries, replace = TRUE),
761   query_closure_date = sample(c("NA"), n_queries, replace = TRUE),
762   query_status = sample(c("On Progress", "Submitted"), n_queries,
763                         replace = TRUE)
764 )
765
766 customer_queries_data$query_submission_date <- format(
767   customer_queries_data$query_submission_date, "%d-%m-%Y")
768
769 #Save to .csv file
770 write.csv(customer_queries_data,
771           "data_uploads/R_synth_customer_queries_round2.csv", row.names = FALSE)
772
773 ### 'categories' table
774 #create lookup table for category_id and category name

```

```

775 set.seed(456)
776 category_lookup <-
777   data.frame("category_id" = seq(1, length(unique(gemini_prods$category)),1),
778             "category" = unique(gemini_prods$category),
779             "cate_code" = "cate") %>%
780   unite(category_id, c(cate_code, category_id), sep = "", remove = T)
781 #Create categories table
782 categories_data <-
783   #Pull category name and product name from gemini file
784   select(gemini_prods, c(category, prod_name)) %>%
785   #Only keep the products included in the products table
786   right_join(select(products_data, c(prod_id, prod_name)), by = "prod_name") %>%
787   #lookup category_id
788   merge(category_lookup, by = "category") %>%
789   #rename to have category_name column
790   rename(category_name = category) %>%
791   #drop product name column
792   select(-prod_name) %>%
793   #reorder the columns to match with table schema
794   select(3,2,1)
795 #Save to .csv file
796 write.csv(categories_data, "data_uploads/R_synth_categories_round2.csv")
797
798 ### 'advertisers' table
799 set.seed(456)
800 n_advertisers <- 5
801 advertisers_data <- data.frame(
802
803   advertiser_id = sprintf("ADV%d", 1:n_advertisers),
804   advertiser_name = c("Ads Life", "Ads Idol", "Ads is Life",
805                      "Ads Master", "Ads Expert"),
806
807   "advertiser_id" = paste("ADV",seq(6, 6+n_advertisers-1, 1), sep = ""),
808   advertiser_name = c(
809     "Ads Beauty", "Ads Power", "Ads by WBS", "Ads by MSBA", "Ads Master"),
810
811   advertiser_email = sprintf("advertiser%d@gmail.com", 1:n_advertisers)
812 )
813 #Save to .csv file
814 write.csv(advertisers_data, "data_uploads/R_synth_advertisers_round2.csv",
815           row.names = FALSE)
816
817 ### 'advertisements' table
818 set.seed(456)
819 n_ads <- 9
820 advertisements_data <- data.frame(
821   "ads_id" = paste("ADS",seq(10, 10+n_ads-1, 1), sep = ""),

```

```

822 prod_id = sample(products_data$prod_id, n_ads, replace = TRUE),
823 advertiser_id = sample(advertisers_data$advertiser_id, n_ads, replace = TRUE),
824 ads_start_date = sample(seq(
825   as.Date('2023-01-01'), as.Date('2023-12-31'), by="day"),
826   n_ads, replace = TRUE),
827 ads_end_date = sample(seq(
828   as.Date('2024-01-01'), as.Date('2024-12-31'), by="day"),
829   n_ads, replace = TRUE)
830 )
831
832 advertisements_data$ads_start_date <- format(
833   advertisements_data$ads_start_date, "%d-%m-%Y")
834 advertisements_data$ads_end_date <- format(
835   advertisements_data$ads_end_date, "%d-%m-%Y")
836
837 #Save to .csv file
838 write.csv(advertisements_data, "data_uploads/R_synth_advertisements_round2.csv",
839           row.names = FALSE)

```

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

3.2 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(
5   path = "data_uploads/", pattern = "advertisement", full.names = TRUE)) {
6   advertisements_ind <- read.csv(ads)
7   advertisement_list[[length(advertisement_list) + 1]] <- advertisements_ind
8 }
9 advertisements_file <- bind_rows(advertisement_list)
10
11 ## Read advertisers file
12 advertisers_list <- list()
13 for (adv in list.files(
14   path = "data_uploads/", pattern = "advertiser", full.names = TRUE)) {
15   advertisers_ind <- read.csv(adv)
16   advertisers_list[[length(advertisers_list) + 1]] <- advertisers_ind
17 }
18 advertisers_file <- bind_rows(advertisers_list)
19
20 ## Read categories file
21 categories_list <- list()
22 for (cat in list.files(
23   path = "data_uploads/", pattern = "categories", full.names = TRUE)) {
24   categories_ind <- read.csv(cat)
25   categories_list[[length(categories_list) + 1]] <- categories_ind
26 }
27 categories_file <- bind_rows(categories_list)
28
29 ## Read customer_queries file
30 customer_queries_list <- list()
31 for (cat in list.files(
32   path = "data_uploads/", pattern = "customer_queries", full.names = TRUE)) {
33   customer_queries_ind <- read.csv(cat)
34   customer_queries_list[[length(customer_queries_list) + 1]] <- customer_queries_ind
35 }
36 customer_queries_file <- bind_rows(customer_queries_list)
37
38 ## Read customers file
39 customers_list <- list()
40 for (cust in list.files(
41   path = "data_uploads/", pattern = "customers", full.names = TRUE)) {
42   customers_ind <- read.csv(cust)
43   customers_list[[length(customers_list) + 1]] <- customers_ind
44 }
45 customers_file <- bind_rows(customers_list)
46
47 ## Read memberships file

```

```

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

```



```

95   path = "data_uploads/", pattern = "suppliers", full.names = TRUE)) {
96     suppliers_ind <- read.csv(suppliers)
97     suppliers_list[[length(suppliers_list) + 1]] <- suppliers_ind
98   }
99   suppliers_file <- bind_rows(suppliers_list)
100
101   ## Read supplies file
102   supplies_list <- list()
103   for (supplies in list.files(
104     path = "data_uploads/", pattern = "supply", full.names = TRUE)) {
105     supplies_ind <- read.csv(supplies)
106     supplies_list[[length(supplies_list) + 1]] <- supplies_ind
107   }
108   supplies_file <- bind_rows(supplies_list)
109
110   # Normalising the Table into 3NF
111
112   ##Normalising Products Table
113   products_table <- products_file %>%
114     select(prod_id,prod_name,prod_desc,prod_unit_price,voucher,prod_url)
115
116   ##Normalising Reviews Table
117   reviews_table <- products_file %>%
118     select(review_id,prod_id, prod_rating, review_date)
119
120   ##Normalising Memberships Table
121   memberships_table <- memberships_file %>%
122     select(membership_type_id,membership_type)
123   memberships_table <- memberships_table[!duplicated(
124     memberships_table$membership_type_id),]
125
126   ##Normalising Customers Table
127   customers_table <- customers_file %>%
128     select(
129       cust_id, first_name, last_name, cust_email,password, cust_birth_date,
130       block_num, postcode, address_type,cust_telephone)
131   customers_table <- merge(customers_table,memberships_file, by = "cust_id")
132   customers_table$X <- NULL
133   customers_table$membership_type <- NULL
134
135   ##Normalising Orders Table
136   orders_table <- orders_file %>%
137     select(order_id, cust_id)
138   orders_table <- orders_table[!duplicated(orders_table$order_id),]
139
140   ##Normalising Order details Table
141   order_details_table <- orders_file %>%

```

```

142     select(order_id,prod_id, order_quantity, order_date, order_value, order_price)
143
144   ##Normalising Payments Table
145   payments_table <- payments_file %>%
146     select(
147       payment_id,order_id,payment_amount,payment_method,payment_status,
148       payment_date)
149
150   ##Normalising Shipments Table
151   shipments_table <- shipments_file %>%
152     select(shipment_id, order_id, prod_id, delivery_departed_date,
153       delivery_received_date,est_delivery_date,
154       shipper_name, delivery_recipient,
155       delivery_fee,delivery_status)
156
157   ##Normalising Suppliers Table
158   suppliers_table <- suppliers_file %>%
159     select(supplier_id, supplier_name, supplier_contact, supplier_postcode)
160
161   ##Normalising Supplies Table
162   supplies_table <- supplies_file %>%
163     select(supply_id,supplier_id,prod_id,inventory_quantity,sold_quantity)
164
165   ##Normalising Customer Queries Table
166   customer_queries_table <- customer_queries_file
167
168   ##Normalising Categories Table
169   categories_table <- categories_file %>%
170     select(category_id,category_name)
171   categories_table <- categories_table[!duplicated(categories_table$category_id),]
172
173   ##Normalising Product Categories Table
174   product_categories_table <- categories_file %>%
175     select(prod_id, category_id)
176
177   ##Normalising Advertiser Table
178   advertisers_table <- advertisers_file
179
180   ##Normalising Advertisement Table
181   advertisements_table <- advertisements_file
182
183   # Data Validation
184
185   ## Advertisement table
186   ### Checking the date format for ads_start_date and ads_end_date
187   if (all(!inherits(try(as.Date(advertisements_table$ads_start_date,
188     format = "%d-%m-%Y")), "try-error")) {

```

```

189     print("Dates are already in the correct format")
190 } else {
191     print("Dates are not in the correct format")
192 }
193
194 if (all(!inherits(try(as.Date(advertisements_table$ads_end_date,
195                             format = "%d-%m-%Y")), "try-error")) {
196     print("Dates are already in the correct format")
197 } else {
198     print("Dates are not in the correct format")
199 }
200
201 ## Ensuring advertisement end date is after the advertisement start date
202 for (i in 1:length(as.Date(advertisements_table$ads_start_date,
203                             format = "%d-%m-%Y"))) {
204     if (
205         as.Date(
206             advertisements_table$ads_end_date, format = "%d-%m-%Y")[i] >
207             as.Date(advertisements_table$ads_start_date, format = "%d-%m-%Y")[i]) {
208         print("Ends date happened after the starts date")
209     } else {
210         print(
211             paste("Error!", "Query", i, ": ends date happened before the starts date"))
212     }
213 }
214
215 ### Checking duplicate values for ads_id and prod_id
216 if(length(
217     advertisements_table$ads_id[duplicated(advertisements_table$ads_id)]) > 0) {
218     print("Duplicate ads_ids found")
219 } else {
220     print("No duplicate ads_ids found")
221 }
222
223 if(length(
224     advertisements_table$prod_id[duplicated(advertisements_table$prod_id)]) > 0) {
225     print("Duplicate prod_ids found")
226 } else {
227     print("No duplicate prod_ids found")
228 }
229
230 ## Advertisers file
231 ### Checking duplicate values for advertisers file
232
233 if(length(
234     advertisers_table$advertiser_id[duplicated(advertisers_table$advertiser_id)])
235     > 0) {

```

```

236     print("Duplicate advertiser_ids found")
237 } else {
238     print("No duplicate advertiser_ids found")
239 }
240
241 if(length(
242     advertisers_table$advertiser_email[duplicated(
243         advertisers_table$advertiser_email)]) > 0) {
244     print("Duplicate advertisers' emails found")
245 } else {
246     print("No duplicate advertisers' emails found")
247 }
248
249 if(length(advertisers_table$advertiser_name[duplicated(
250     advertisers_table$advertiser_name)]) > 0) {
251     print("Duplicate advertisers' names found")
252 } else {
253     print("No duplicate advertisers' names found")
254 }
255
256 ## Checking the advertiser_email format
257 if(
258     length(grep((
259         "^ [A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.\com$"),
260         advertisers_table$advertiser_email,
261         value = TRUE)) ==
262     length(advertisers_table$advertiser_email)) {
263     print("All email format are correct")
264 } else {
265     print(
266         paste(
267             "There are:",
268             length(advertisers_table$advertiser_email) -
269             length(grep("^ [A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.\com$"),
270
271         )
272     )
273 ## Customer_queries file
274 ### Checking duplicate values for query_id
275
276 if(length(
277     customer_queries_table$query_id[duplicated(customer_queries_table$query_id)])
278     > 0) {
279     print("Duplicate queries_ids found")
280 } else {
281     print("No duplicate queries_ids found")
282 }

```

advertiser

```

283
284 ### Checking the date format for query_submission_date and query_closure_date
285 correct_date_format <- function(date_column) {
286   converted_dates <- try(as.Date(date_column, format = "%d-%m-%Y"),
287                           silent = TRUE)
288   if (inherits(converted_dates, "try-error")) {
289     return(format(mdy(date_column), "%d-%m-%Y"))
290   } else {
291     return(date_column)
292   }
293 }
294 customer_queries_table$query_submission_date <- correct_date_format(
295   customer_queries_table$query_submission_date)
296 customer_queries_table$query_closure_date <- correct_date_format(
297   customer_queries_table$query_closure_date)
298
299 if (all(!inherits(try(as.Date(customer_queries_table$query_submission_date,
300                             format = "%d-%m-%Y")), "try-error")))) {
301   print("Query Submission Dates are now in the correct format")
302 } else {
303   print("There was an issue with converting the Query Submission Dates")
304 }
305
306 if (all(!inherits(try(as.Date(customer_queries_table$query_closure_date,
307                             format = "%d-%m-%Y")), "try-error")))) {
308   print("Query Closure Dates are now in the correct format")
309 } else {
310   print("There was an issue with converting the Query Closure Dates")
311 }
312
313 ## Memberships file
314 ### Checking NA values inside membership_id and membership_type
315 if (any(!is.na(memberships_table))) {
316   print("There are no NA values in the dataset")
317 } else {
318   print("Error! There are NA values in the dataset")
319 }
320
321 ## Orders file
322 ### Checking NA values inside order table
323 if (any(!is.na(
324   order_details_table[,c("order_id", "order_quantity", "order_price", "prod_id")]
325 ))) {
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 ### Checking date format for the order_date
332 correct_date_format <- function(date_column) {
333   converted_dates <- try(as.Date(date_column, format = "%d-%m-%Y"),
334                           silent = TRUE)
335   if (inherits(converted_dates, "try-error")) {
336     return(format(mdy(date_column), "%d-%m-%Y"))
337   } else {
338     return(date_column)
339   }
340 }
341 order_details_table$order_date <- correct_date_format(
342   order_details_table$order_date)
343
344 # Print a message based on the result
345 if (all(!inherits(try(as.Date(order_details_table$order_date,
346                           format = "%d-%m-%Y")), "try-error")))) {
347   print("Dates are now in the correct format")
348 } else {
349   print("There was an issue with converting the dates")
350 }
351
352 ## Payment_file
353 ### Checking NA values inside payment file
354 if (any(!is.na(payments_table[,c("payment_id", "payment_method",
355                                   "payment_status", "order_id")]))) {
356   print("There are no NA values in the dataset")
357 } else {
358   print("Error! There are NA values in the dataset")
359 }
360
361 ### Checking date format for the payment_date
362 if (all(!inherits(try(as.Date(payments_table$payment_date,
363                                   format = "%d-%m-%Y")), "try-error")))) {
364   print("Dates are already in the correct format")
365 } else {
366   print("Dates are not in the correct format")
367 }
368
369 ## Products_file
370 ### Checking duplicate values in prod_id and review_id
371 if(length(products_table$prod_id[duplicated(products_table$prod_id)]) == 0) {
372   print("No duplicate prod_ids found")
373 } else {
374   print("Duplicate ad_ids found")
375 }
376

```

```

377 if(length(reviews_table$review_id[duplicated(reviews_table$review_id)]) == 0) {
378   print("No duplicate review_ids found")
379 } else {
380   print("Duplicate review_ids found")
381 }
382
383 ### Checking NA values inside prod_id, prod_url, prod_unit_price
384 if (any(!is.na(products_table[,c("prod_id", "prod_url", "prod_unit_price")]))) {
385   print("There are no NA values in the dataset")
386 } else {
387   print("Error! There are NA values in the dataset")
388 }
389
390 ### Checking date format for the review_date
391 check_and_correct_date_format <- function(date_column_data) {
392   if (all(!inherits(try(as.Date(date_column_data, format = "%d-%m-%Y")),
393     "try-error")))) {
394     return(date_column_data)
395   } else {
396     return(format(mdy(date_column_data), "%d-%m-%Y"))
397   }
398 }
399 reviews_table$review_date <- check_and_correct_date_format(
400   reviews_table$review_date)
401 if (all(!inherits(try(as.Date(reviews_table$review_date,
402   format = "%d-%m-%Y")), "try-error")))) {
403   print("Review Dates are now in the correct format")
404 } else {
405   print("There was an issue with converting the Review Dates")
406 }
407
408 ### Checking the URL format of the prod_url
409 if(length(grep(("^(http|https)://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}(\S*)$"),
410   products_table$prod_url, value = TRUE)) ==
411   length(products_table$prod_url)) {
412   print("All product url format are correct")
413 } else {
414   print(paste("There are:", length(products_table$prod_url) - length(
415     grep(("^(http|https)://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}(\S*)$"),
416     products_table$prod_url, value = TRUE)), "wrong product urls found"))
417 }
418
419 ## Shipments file
420 ### Checking duplicate values in shipment_id
421 if(length(shipments_table$shipment_id[duplicated(shipments_table$shipment_id)])
422   == 0) {
423   print("No duplicate shipment_ids found")

```

```

424 } else {
425   print("Duplicate shipment_ids found")
426 }
427
428 ### Checking NA values inside shipment_id, prod_id, order_id
429 if (any(!is.na(shipments_table[,c("prod_id", "order_id", "shipment_id")]))) {
430   print("There are no NA values in the dataset")
431 } else {
432   print("Error! There are NA values in the dataset")
433 }
434
435 ### Checking date format for the inside shipment table
436 check_and_correct_date_format <- function(date_column_data) {
437   if (all(!inherits(try(as.Date(date_column_data, format = "%d-%m-%Y")),
438                       "try-error")))) {
439     return(date_column_data) # Return the original data if format is correct
440   } else {
441     return(format(mdy(date_column_data), "%d-%m-%Y"))
442   }
443 }
444
445 shipments_table$delivery_departed_date <- check_and_correct_date_format(
446   shipments_table$delivery_departed_date)
447 shipments_table$delivery_received_date <- check_and_correct_date_format(
448   shipments_table$delivery_received_date)
449 shipments_table$est_delivery_date <- check_and_correct_date_format(
450   shipments_table$est_delivery_date)
451
452 columns_to_check <- list(
453   "Delivery Departed Date" = shipments_table$delivery_departed_date,
454   "Delivery Received Date" = shipments_table$delivery_received_date,
455   "Estimated Delivery Date" = shipments_table$est_delivery_date
456 )
457
458 for (column_name in names(columns_to_check)) {
459   if (all(!inherits(try(as.Date(columns_to_check[[column_name]]),
460                           format = "%d-%m-%Y")), "try-error")) {
461     print(paste(column_name, "are now in the correct format"))
462   } else {
463     print(paste("There was an issue with converting the", column_name))
464   }
465 }
466
467 ### Checking whether the recipient names contains ' and ,
468 if (any(grepl("[',]", shipments_table$delivery_recipient))) {
469   print("Error! Some names contain invalid characters")
470 } else {

```



```

471   print("All names are valid")
472 }
473
474 ## Customer Table
475 ### Checking duplicate values in customer_id
476 if(length(customers_table$cust_id[duplicated(customers_table$cust_id)]) == 0) {
477   print("No duplicate customer_ids found")
478 } else {
479   print("Duplicate customer_ids found")
480 }
481
482 ### Checking whether the customer's first name and last name contains ' and ,
483 clean_name <- function(name) {
484   gsub("['\",]", "-", name)
485 }
486
487 if (any(grepl("[',]", customers_table$first_name))) {
488   print("Error! Some first names contain invalid characters")
489   customers_table$first_name <- sapply(customers_table$first_name, clean_name)
490 } else {
491   print("All first names are valid")
492 }
493
494 if (any(grepl("[',]", customers_table$last_name))) {
495   print("Error! Some last names contain invalid characters")
496   customers_table$last_name <- sapply(customers_table$last_name, clean_name)
497 } else {
498   print("All last names are valid")
499 }
500
501
502 ### Checking the customer_email format
503 if(length(grep(("^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.com$"),
504             customers_table$cust_email, value = TRUE)) ==
505     length(customers_table$cust_email)) {
506   print("All customer email format are correct")
507 } else {
508   print(paste("There are:", length(customers_table$cust_email) - length(
509     grep(("^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.com$"),
510         customers_table$cust_email, value = TRUE)), "wrong emails found"))
511 }
512
513 ### Checking the customer birth_date format
514 if (all(!inherits(try(as.Date(customers_table$cust_birth_date,
515                             format = "%d-%m-%Y")), "try-error")))) {
516   print("Dates are already in the correct format")
517 } else {

```

```

518     print("Dates are not in the correct format")
519 }
520
521
522 # Create connection to SQL database
523 db_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"IB9HP0_9.db")
524
525 # Inserting Dataframe into the sql database
526 write_log <- function(message, path) {
527     timestamp <- format(Sys.time(), "%Y-%m-%d %H:%M:%S")
528     message <- paste(timestamp, message, sep=" - ")
529     write(message, file = path, append = TRUE, sep = "\n")
530 }
531
532 # Path for the error log file
533 log_file_path <- "error_log.txt"
534
535 ## Inserting Products table
536 for(i in 1:nrow(products_table)){
537     tryCatch({
538         dbExecute(db_connection, paste(
539             "INSERT INTO products(prod_id, prod_name, prod_desc, voucher, prod_url,
540             prod_unit_price) VALUES('",
541             products_table$prod_id[i], "','",
542             products_table$prod_name[i], "','",
543             products_table$prod_desc[i], "','",
544             products_table$voucher[i], "','",
545             products_table$prod_url[i], "','",
546             products_table$prod_unit_price[i], ");", sep = "")
547         )
548         write_log(sprintf("Product %s inserted successfully.",
549             products_table$prod_id[i]), log_file_path)
550     }, error = function(e) {
551         write_log(sprintf("Failed to insert product %s: %s",
552             products_table$prod_id[i], e$message), log_file_path)
553     })
554 }
555
556 ## Inserting Reviews table
557 for(i in 1:nrow(reviews_table)){
558     tryCatch({
559         dbExecute(db_connection, paste(
560             "INSERT INTO reviews(review_id, prod_rating, review_date, prod_id)
561             VALUES('",
562             reviews_table$review_id[i], "','",
563             reviews_table$prod_rating[i], "','",
564             reviews_table$review_date[i], "','",

```

```

565     reviews_table$prod_id[i], "');", sep = "")
566 )
567 write_log(sprintf("Review %s inserted successfully.",
568                 reviews_table$review_id[i]), log_file_path)
569 }, error = function(e) {
570     write_log(sprintf("Failed to insert review %s: %s",
571                     reviews_table$review_id[i], e$message), log_file_path)
572 })
573 }
574
575 ## Inserting Memberships table
576 for(i in 1:nrow(memberships_table)){
577     tryCatch({
578         dbExecute(db_connection, paste(
579             "INSERT INTO memberships(membership_type_id, membership_type)
580             VALUES(",
581             "'", memberships_table$membership_type_id[i], "'",
582             "'", memberships_table$membership_type[i], "');", sep = "")
583         )
584         write_log(sprintf("Membership %s inserted successfully.",
585                         memberships_table$membership_type_id[i]), log_file_path)
586     }, error = function(e) {
587         write_log(sprintf("Failed to insert membership %s: %s",
588                         memberships_table$membership_type_id[i], e$message),
589                     log_file_path)
590     })
591 }
592
593 ## Inserting Customers table
594 for(i in 1:nrow(customers_table)){
595     tryCatch({
596         dbExecute(db_connection, paste(
597             "INSERT INTO customers(cust_id, first_name, last_name, cust_email,
598             password, cust_birth_date, address_type, block_num, postcode,
599             cust_telephone, membership_type_id) VALUES(",
600             "'", customers_table$cust_id[i], "'",
601             "'", customers_table$first_name[i], "'",
602             "'", customers_table$last_name[i], "'",
603             "'", customers_table$cust_email[i], "'",
604             "'", customers_table$password[i], "'",
605             "'", customers_table$cust_birth_date[i], "'",
606             "'", customers_table$address_type[i], "'",
607             "'", customers_table$block_num[i], "'",
608             "'", customers_table$postcode[i], "'",
609             "'", customers_table$cust_telephone[i], "'",
610             "'", customers_table$membership_type_id[i], "');", sep = "")
611         )

```

```

612     write_log(sprintf("Customer %s inserted successfully.",
613                       customers_table$cust_id[i]), log_file_path)
614 }, error = function(e) {
615     write_log(sprintf("Failed to insert customer %s: %s",
616                       customers_table$cust_id[i], e$message), log_file_path)
617 })
618 }
619
620 ## Inserting Orders table
621 for(i in 1:nrow(orders_table)){
622     tryCatch({
623         dbExecute(db_connection, paste(
624             "INSERT INTO orders(order_id, cust_id) VALUES('",
625             orders_table$order_id[i], "','",
626             orders_table$cust_id[i], "');", sep = "")
627         )
628         write_log(sprintf("Order %s inserted successfully.",
629                           orders_table$order_id[i]), log_file_path)
630     }, error = function(e) {
631         write_log(sprintf("Failed to insert order %s: %s",
632                           orders_table$order_id[i], e$message), log_file_path)
633     })
634 }
635
636 ## Inserting Payment table
637 for(i in 1:nrow(payments_table)){
638     tryCatch({
639         dbExecute(db_connection, paste(
640             "INSERT INTO payments(payment_id, payment_method,
641             payment_amount, payment_status, payment_date, order_id) VALUES(",
642             "'", payments_table$payment_id[i], "','",
643             "'", payments_table$payment_method[i], "','",
644             payments_table$payment_amount[i], ",",
645             "'", payments_table$payment_status[i], "','",
646             "'", payments_table$payment_date[i], "','",
647             "'", payments_table$order_id[i], "');", sep = "")
648         )
649         write_log(sprintf("Payment %s inserted successfully.",
650                           payments_table$payment_id[i]), log_file_path)
651     }, error = function(e) {
652         write_log(sprintf("Failed to insert payment %s: %s",
653                           payments_table$payment_id[i], e$message), log_file_path)
654     })
655 }
656
657 ## Inserting Shipment table
658 for(i in 1:nrow(shipments_table)){

```

```

659 tryCatch({
660   dbExecute(db_connection, paste(
661     "INSERT INTO shipments(shipment_id, delivery_status,
662     delivery_fee, delivery_recipient, shipper_name, est_delivery_date,
663     delivery_departed_date, delivery_received_date, prod_id, order_id)
664     VALUES(",
665     "'", shipments_table$shipment_id[i], "'",",
666     "'", shipments_table$delivery_status[i], "'",",
667     shipments_table$delivery_fee[i], ",",
668     "'", shipments_table$delivery_recipient[i], "'",",
669     "'", shipments_table$shipper_name[i], "'",",
670     "'", format(as.Date(shipments_table$est_delivery_date[i],
671       format = "%d-%m-%Y"), "%Y-%m-%d"), "'",",
672     "'", format(as.Date(shipments_table$delivery_departed_date[i],
673       format = "%d-%m-%Y"), "%Y-%m-%d"), "'",",
674     "'", format(as.Date(shipments_table$delivery_received_date[i],
675       format = "%d-%m-%Y"), "%Y-%m-%d"), "'",",
676     "'", shipments_table$prod_id[i], "'",",
677     "'", shipments_table$order_id[i], "');", sep = "")
678   )
679   write_log(sprintf("Shipment %s inserted successfully.",
680     shipments_table$shipment_id[i]), log_file_path)
681 }, error = function(e) {
682   write_log(sprintf("Failed to insert shipment %s: %s",
683     shipments_table$shipment_id[i], e$message), log_file_path)
684 })
685 }
686
687 ## Inserting Order details table
688 for(i in 1:nrow(order_details_table)){
689   tryCatch({
690     dbExecute(db_connection, paste(
691       "INSERT INTO order_details(order_quantity,order_date,order_price,
692       order_value,prod_id,order_id) VALUES(",
693       order_details_table$order_quantity[i], ",",
694       "'", order_details_table$order_date[i], "'",",
695       order_details_table$order_price[i], ",",
696       order_details_table$order_value[i], ",",
697       "'", order_details_table$prod_id[i], "'",",
698       "'", order_details_table$order_id[i], "');",sep = "")
699     )
700
701     write_log(sprintf("Order detail %s inserted successfully.",
702       order_details_table$order_id[i]), log_file_path)
703   }, error = function(e) {
704     write_log(sprintf("Failed to insert order detail %s: %s",
705       order_details_table$order_id[i], e$message),

```

```

706         log_file_path)
707     })
708 }
709
710 ## Inserting Suppliers table
711 for(i in 1:nrow(suppliers_table)){
712     tryCatch({
713         dbExecute(db_connection, paste(
714             "INSERT INTO suppliers(supplier_id, supplier_name, supplier_postcode,
715             supplier_contact) VALUES('",
716             suppliers_table$supplier_id[i], "','",
717             suppliers_table$supplier_name[i], "','",
718             suppliers_table$supplier_postcode[i], "','",
719             suppliers_table$supplier_contact[i], "');", sep = "")
720         )
721         write_log(sprintf("Supplier %s inserted successfully.",
722             suppliers_table$supplier_id[i]), log_file_path)
723     }, error = function(e) {
724         write_log(sprintf("Failed to insert supplier %s: %s",
725             suppliers_table$supplier_id[i], e$message), log_file_path)
726     })
727 }
728
729
730 ## Inserting Supplies table
731 for(i in 1:nrow(supplies_table)){
732     tryCatch({
733         dbExecute(db_connection, paste(
734             "INSERT INTO supplies(supply_id, inventory_quantity,
735             sold_quantity, supplier_id, prod_id) VALUES('",
736             supplies_table$supply_id[i], "','",
737             supplies_table$inventory_quantity[i], "','",
738             supplies_table$sold_quantity[i], "','",
739             supplies_table$supplier_id[i], "','",
740             supplies_table$prod_id[i], "');", sep = "")
741         )
742         write_log(sprintf("Supply %s inserted successfully.",
743             supplies_table$supply_id[i]), log_file_path)
744     }, error = function(e) {
745         write_log(sprintf("Failed to insert supply %s: %s",
746             supplies_table$supply_id[i], e$message), log_file_path)
747     })
748 }
749
750 ## Inserting Customer queries table
751 for(i in 1:nrow(customer_queries_table)){
752     tryCatch({

```

```

753 dbExecute(db_connection, paste(
754     "INSERT INTO customer_queries(query_id, query_title,
755     query_submission_date, query_closure_date, query_status, cust_id) VALUES(",
756     "'", customer_queries_table$query_id[i], "'",",
757     "'", customer_queries_table$query_title[i], "'",",
758     "'", customer_queries_table$query_submission_date[i], "'",",
759     "'", customer_queries_table$query_closure_date[i], "'",",
760     "'", customer_queries_table$query_status[i], "'",",
761     "'", customer_queries_table$cust_id[i], "');", sep = "")
762 )
763 write_log(sprintf("Customer query %s inserted successfully.",
764     customer_queries_table$query_id[i]), log_file_path)
765 }, error = function(e) {
766     write_log(sprintf("Failed to insert customer query %s: %s",
767     customer_queries_table$query_id[i], e$message),
768     log_file_path)
769 })
770 }
771
772 ## Inserting Categories table
773 for(i in 1:nrow(categories_table)){
774     tryCatch({
775         dbExecute(db_connection, paste(
776             "INSERT INTO categories(category_id, category_name) VALUES('",
777             categories_table$category_id[i], "'",",
778             categories_table$category_name[i], "');", sep = "")
779         )
780         write_log(sprintf("Category %s inserted successfully.",
781             categories_table$category_id[i]), log_file_path)
782     }, error = function(e) {
783         write_log(sprintf("Failed to insert category %s: %s",
784             categories_table$category_id[i], e$message),
785             log_file_path)
786     })
787 }
788
789 ## Inserting Product Categories table
790 for(i in 1:nrow(product_categories_table)){
791     tryCatch({
792         dbExecute(db_connection, paste(
793             "INSERT INTO product_categories(category_id, prod_id) VALUES('",
794             product_categories_table$category_id[i], "'",",
795             product_categories_table$prod_id[i], "');", sep = "")
796         )
797         write_log(
798             sprintf(
799             "Product category link for product %s and

```

```

800     category %s inserted successfully.",
801     product_categories_table$prod_id[i],
802     product_categories_table$category_id[i]),
803     log_file_path)
804 }, error = function(e) {
805     write_log(
806         sprintf(
807             "Failed to insert product category link for product %s and
808             category %s: %s", product_categories_table$prod_id[i],
809             product_categories_table$category_id[i],
810             e$message), log_file_path)
811     })
812 }
813
814 ## Inserting Advertisers table
815 for(i in 1:nrow(advertisers_table)) {
816     tryCatch({
817         dbExecute(db_connection, paste(
818             "INSERT INTO advertisers(advertiser_id, advertiser_name,
819             advertiser_email) VALUES(",
820             "'", advertisers_table$advertiser_id[i], "'",
821             "'", advertisers_table$advertiser_name[i], "'",
822             "'", advertisers_table$advertiser_email[i], "');", sep = "")
823     )
824     write_log(sprintf("Advertiser %s inserted successfully.",
825         advertisers_table$advertiser_id[i]), log_file_path)
826 }, error = function(e) {
827     write_log(sprintf("Failed to insert advertiser %s: %s",
828         advertisers_table$advertiser_id[i], e$message),
829         log_file_path)
830     })
831 }
832
833 ## Inserting Advertisements table
834 for(i in 1:nrow(advertisements_table)) {
835     tryCatch({
836         dbExecute(db_connection, paste(
837             "INSERT INTO advertisements(ads_id, ads_start_date, ads_end_date,
838             prod_id, advertiser_id) VALUES(",
839             "'", advertisements_table$ads_id[i], "'",
840             "'", advertisements_table$ads_start_date[i], "'",
841             "'", advertisements_table$ads_end_date[i], "'",
842             "'", advertisements_table$prod_id[i], "'",
843             "'", advertisements_table$advertiser_id[i], "');", sep = "")
844     )
845     write_log(sprintf("Advertisement %s inserted successfully.",
846         advertisements_table$ads_id[i]), log_file_path)

```



```

847 }, error = function(e) {
848     error_message <- sprintf("Failed to insert advertisement %s: %s",
849                             advertisements_table$ads_id[i], e$message)
850     write_log(error_message, log_file_path)
851 })
852 }

```

4 Data Pipeline Generation

4.1 Introduction

In this section, the project is implemented effectively through a streamlined workflow and seamless collaboration using GitHub. The setup of the GitHub Repository includes multiple organized folders namely R, data_uploads, data_analysis_results, and docs 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.

4.2 GitHub Repository Structure

The 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, and database updates.
- ‘data_uploads’ folder: stores raw data files for database insertion. 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.
- ‘data_analysis_results’ folder: stores the results of the data analysis.
- ‘docs’ folder : holds project’s documents, such as the analysis report Quarto Markdown file (IB9HP0_9.qmd) 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.

4.3 GitHub Actions

4.3.1 Trigger conditions

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

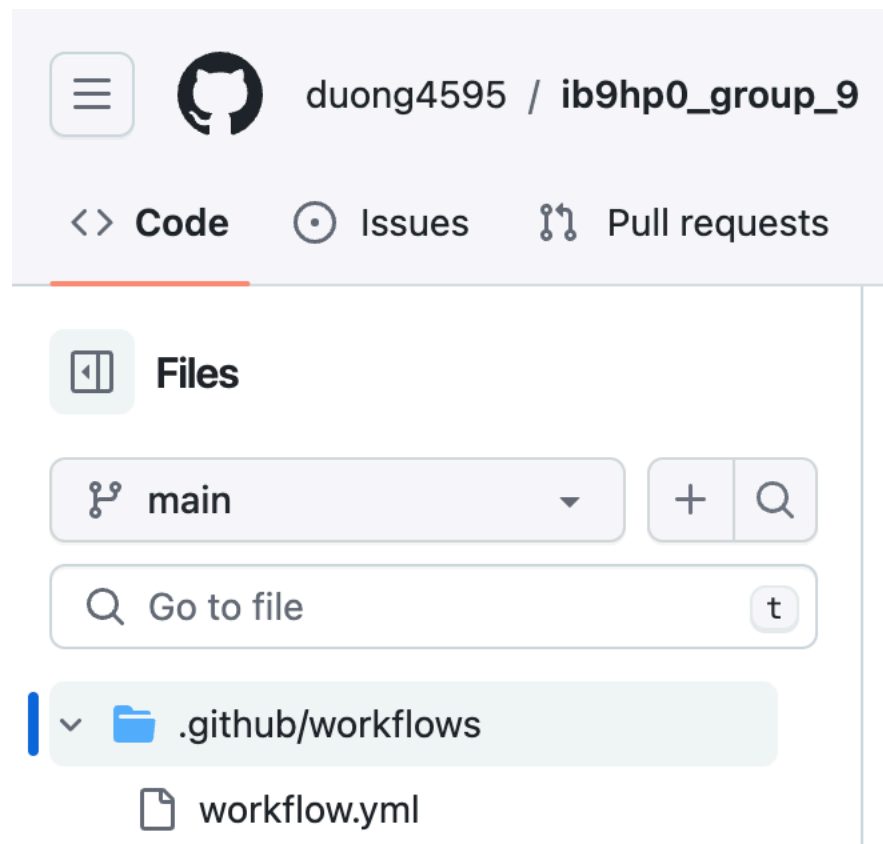


Figure 4 Workflow of the E-Commerce Repository

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.

4.3.2 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.

5 Data Analysis and Reporting

5.1 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
46 ##### Plot monthly avg price trend
47 p.mnth.price <- ggplot(filter(sales_performance,
48                             year %in% c("2022", "2023")),
49                       aes(x = month, group = year, color = year,
50                          y = avg_price)) +
51   geom_smooth(se = F) +
52   labs(y = "Average Price (GBP/Unit)", x = "Month",
53        subtitle = "Average Price", color = "Year") +
54   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,
59                           widths = c(0.9, 0.9, 1.1),
60                           top = ggpubr::text_grob("Monthly Sales Trend",
61                                                    size = 15, face = "bold")))

```

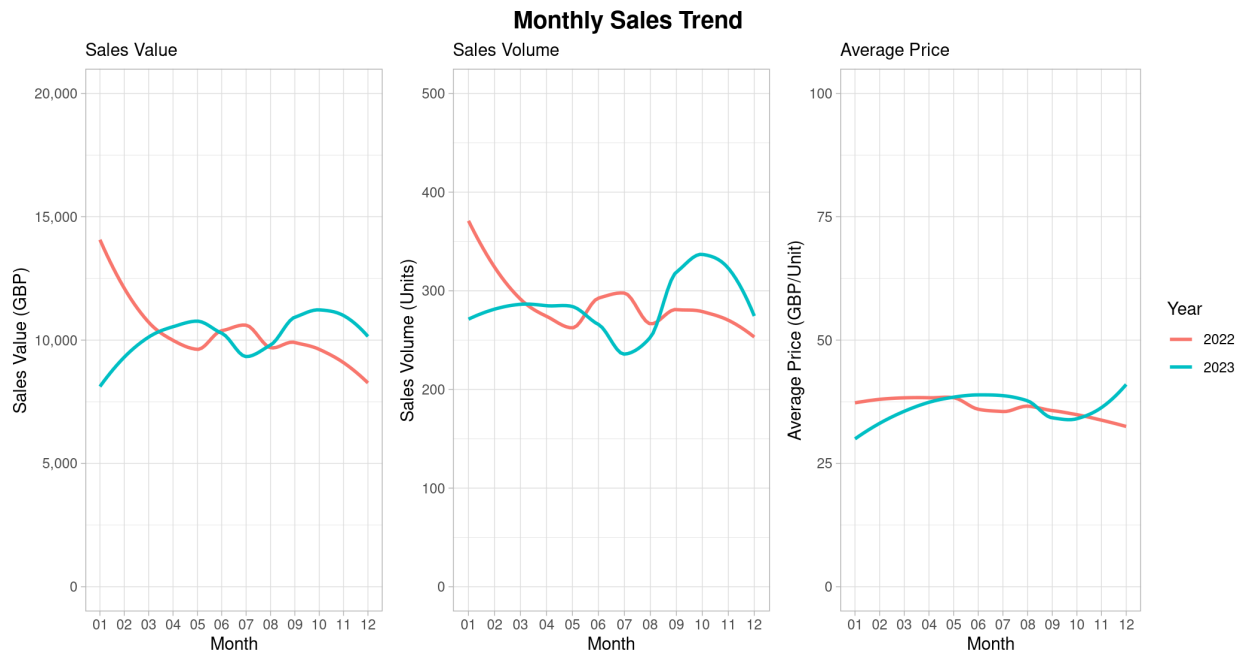


Figure 5. Monthly Sales Trend Analysis

Figure 5 shows 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 2023. Meanwhile, the company's overall trend of monthly sales is similar to that of monthly sales in 2022 and 2023.

5.2 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(
40         "Top 10 Products and Their Rating",
41         size = 15, face = "bold"))))

```

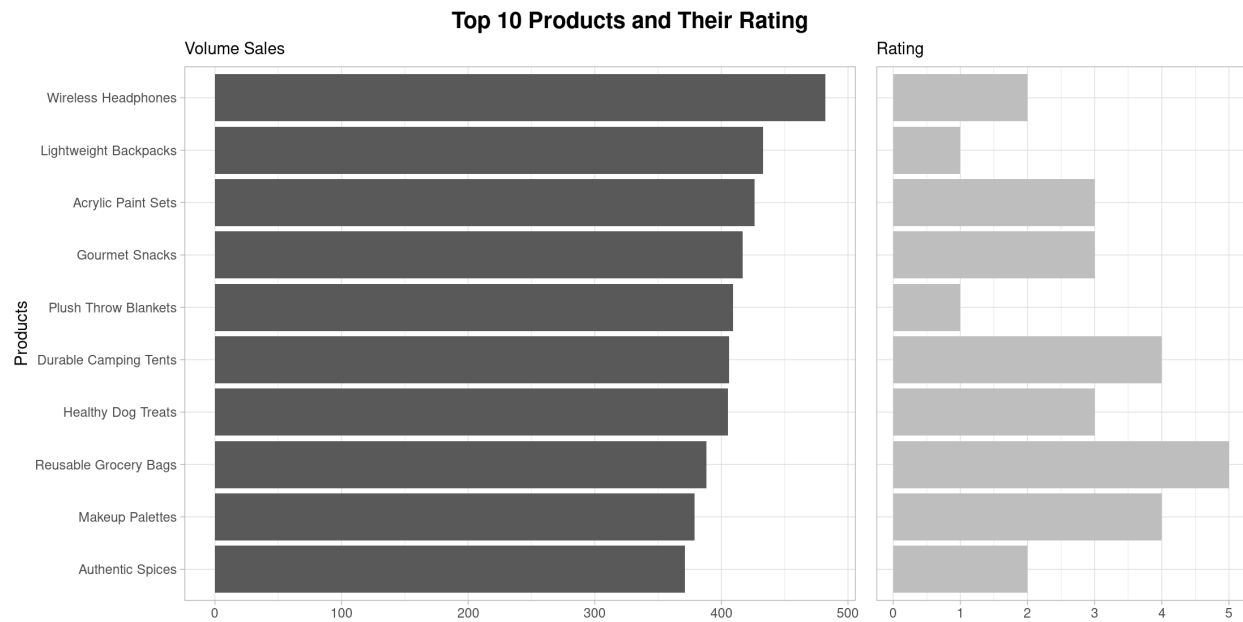


Figure 6. Top 10 Products and Rating Analysis

As illustrated in Figure 6, 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.

5.3 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")))
53

```

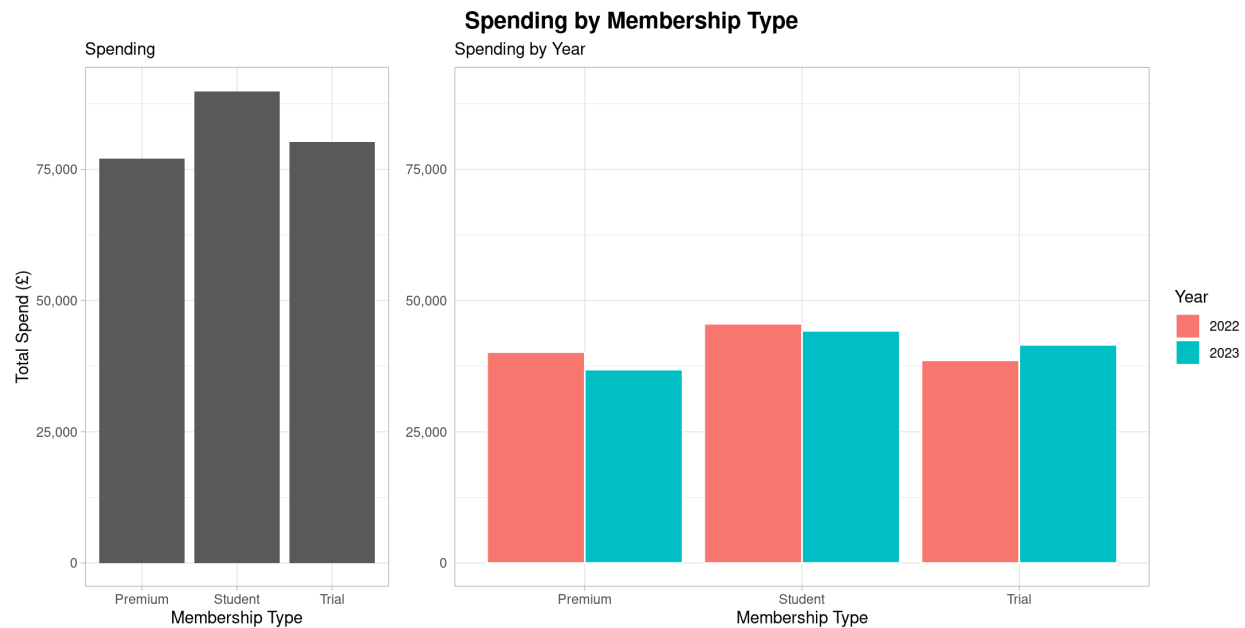



Figure 7. Total Spend by Membership Analysis

As illustrated in figure 7, 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.

5.4 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(
31     difftime(query_closure_date, query_submission_date,
32               units = "weeks"),0) ) %>%
33   group_by(query_title) %>%
34   summarise(avg_turnaround_time = round(mean(turnaround_time),1)) %>%
35   merge(queries_frequencies, by = "query_title", remove = F)
36
37 ### Plot query by response time
38 h_line <- mean(response_time$avg_turnaround_time)
39 p.query_time <- ggplot(response_time,
40   aes(x= reorder(query_title, desc(frequencies)),
41       y = avg_turnaround_time)) +
42   geom_bar(stat = "identity") +
43   geom_hline(yintercept = h_line,
44             color = "magenta", linetype = "dashed", size = 1.1) +
45   geom_text(aes(1, h_line, label = "Avg of all types"),
46             vjust = -1, color = "magenta") +
47   labs(x = "Query Type", y = "Avg Turnaround Time (weeks)",
48        subtitle = "Turnaround Time") +
49   theme_light()
50 ### Combine frequency and turnaround time
51 (gridExtra::grid.arrange(p.query_freq, p.query_time, ncol = 2,
52   top = ggpubr::text_grob("Customer Queries",
53   size = 15, face = "bold")))

```

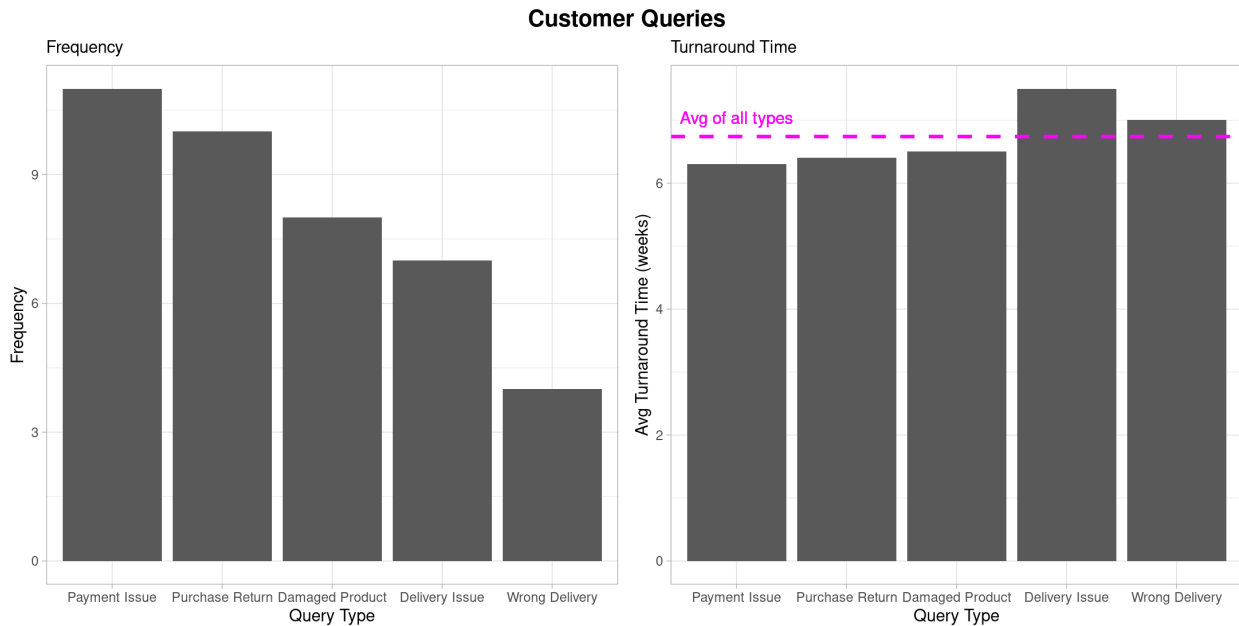


Figure 8. Customer Queries Analysis

Figure 8 shows though customers' queries include few delivery issue, the turnaround time that is the longest, which may have a significant negative effect on membership subscription.

5.5 Payment Methods Analysis

```

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,
11                                                  desc(frequencies)),
12                                         y = frequencies)) +
13    geom_bar(stat = "identity") +
14    labs(x = "Payment Method", y = "Frequency",
15         subtitle = "Frequently Used Payment Method") +
16    theme_light() +
17    theme(plot.title = element_text(hjust = 0.5, face = "bold")) +
18    scale_y_continuous(labels = comma)
19  ### Plot payment method by value
20 p.payment_amnt <- ggplot(top_payment,
21                          aes(x= reorder(payment_method, desc(frequencies)),

```

```

22                                     y = pymnt_amnt)) +
23 geom_bar(stat = "identity") +
24 labs(x = "Payment Method", y = "Payment Amount (£)",
25      subtitle = "Payment Value") +
26 theme_light() +
27 theme(plot.title = element_text(hjust = 0.5, face = "bold")) +
28 scale_y_continuous(labels = comma)
29 ### Combine payment method by frequency and value
30 (gridExtra::grid.arrange(p.frequency, p.payment_amnt, ncol = 2,
31                          top = ggpubr::text_grob("Payment Methods",
32                                                  size = 15, face = "bold"))))
32

```

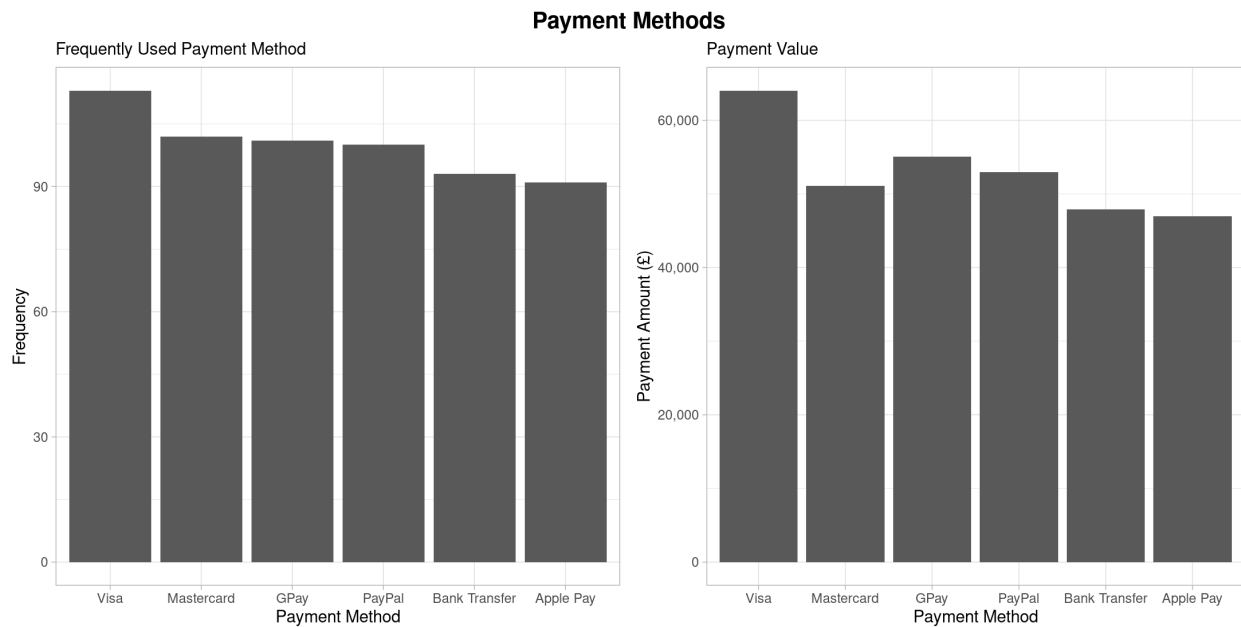


Figure 9. Payment Method Analysis

Based on Figure 9, 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.

5.6 Insight and Recommendation

5.6.1 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.

5.6.2 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.

6 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.

7 Conclusion

This project has demonstrated a comprehensive approach to database design and management for an e-commerce platform. Initiating with a detailed database design, this project carefully considered various entities and their relationships, adhering strictly to the third normal form to ensure data integrity and optimize query performance. Additionally, the use of synthetic data generation provided a simulation of e-commerce activities, along with quality assurance and validation process processes to guarante the accuracy and reliability of this data.

A significant advancement was made in automating the data pipeline through GitHub Actions. This automation not only streamlined processes but also enhanced the system's responsiveness to changing data needs.

The data analysis component of the project offered deep insights into customer behavior, product performance, and sales trends, providing valuable inputs for strategic business decision-making. By integrating sophisticated database design, advanced automation techniques, and thorough data analysis, this project shows the effectiveness of combining technical precision with practical business insight.