IB9HPO_9

5588654, 5577965, 5579058, 5587354, 5519743, 5588060

2024-03-02

Table of contents

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

1 Database Design and Implementation

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

1.1 Assumptions

The design of database system corresponds to the below assumptions:

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

1.2 Conceptual Design: Entities and Relationships

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

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

Figure 1 shows the relationship set for the entities

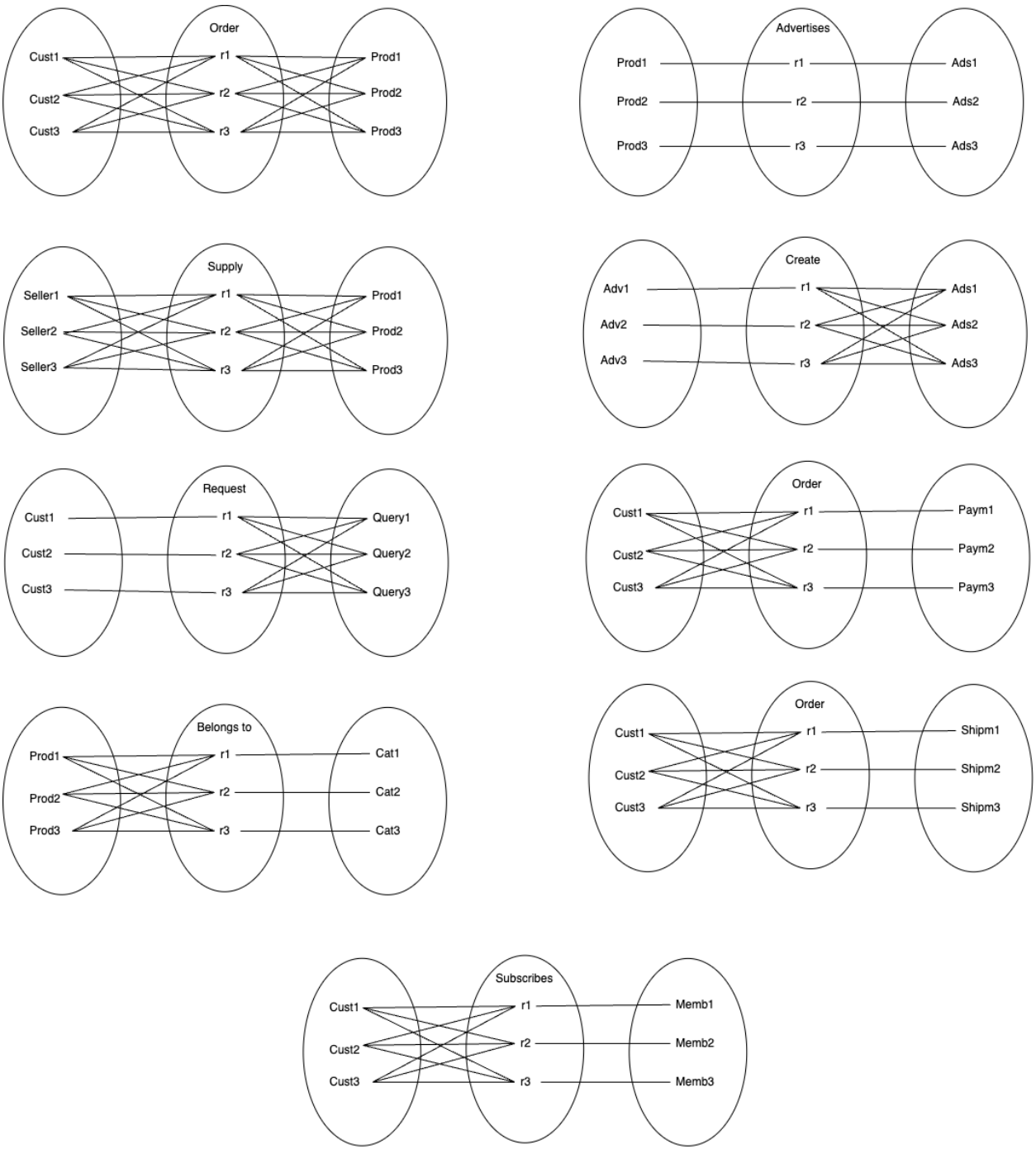

Figure 1. Relationship Sets

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

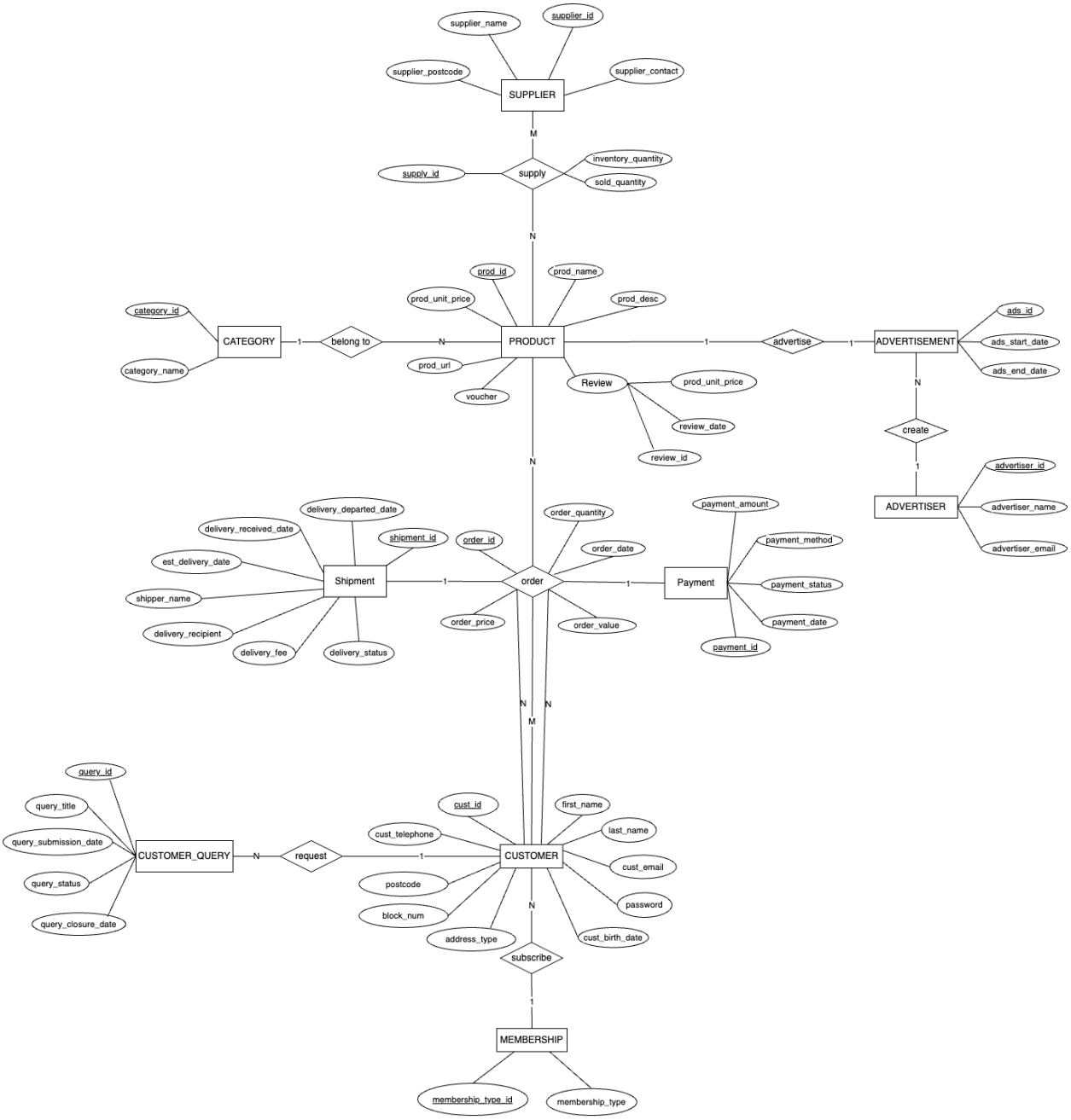

Figure 2. Entity Relationship Diagram (ERD)

1.3 Logical Design

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

- customer (customer_id, cust_name, cust_email, password, cust_birth_date, postcode,address_type, cust_telephone)
- product (prod_id, prod_name, prod_desc, review_id, prod_rating, review_date, voucher, prod_url, prod_unit_price)
- order (order_id, customer_id, prod_id, payment_id, shipment_id, order_quantity, order_date, order_price, order_value)
- supply (suppy_id, prod_id, inventory_quantity, soldq_uantity)
- supplier (supplier_id, supplier_contact, supplier_name, supplier_postcode)
- category (category_id, prod_id, category_name)
- advertisement (ads_id, prod_id, advertiser_id, ads_start_date, ads_end_date)
- advertiser (advertiser_id, advertiser_name, advertiser_email)
- customer_queries (query_id, customer_id, query_title, query_detail, query_submission_date, query_closure_date, query_status)
- membership (membership_type_id, cust_id, membership_type)
- payment (payment_id, payment_method, payment_status, payment_date)
- shipment (shipment_id, shipper_name, delivery_status, delivery_fee, delivery_recipient, estimated_delivery_date, delivery_receive_date, delivery_departed_date)

Figure 3. Logical Schema

1.4 Physical Design

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

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

```
db_file <- "IB9HP0_9.db"

# Check if the database file exists and remove it
if (file.exists(db_file)) {
  file.remove(db_file)
}
```

```
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),
```

```
                postcode VARCHAR (50),
                cust_telephone INT UNIQUE,
                membership_type_id VARCHAR (50),
                FOREIGN KEY (membership_type_id)
                  REFERENCES memberships(membership_type_id)
                )"
            )

#Create table for orders
dbExecute(db_connection,
            "CREATE TABLE IF NOT EXISTS orders (
                order_id VARCHAR (50) PRIMARY KEY,
                cust_id VARCHAR (50),
                FOREIGN KEY (cust_id)
                  REFERENCES customers(cust_id)
                )"
            )

#Create table for order details
dbExecute(db_connection,
            "CREATE TABLE IF NOT EXISTS order_details (
                order_quantity INT NOT NULL,
                order_date DATE,
                order_price DECIMAL NOT NULL,
                order_value DECIMAL,
                prod_id VARCHAR (50),
                order_id VARCHAR (50),
                FOREIGN KEY (prod_id)
                  REFERENCES products(prod_id),
                FOREIGN KEY (order_id)
                  REFERENCES orders(order_id)
                )"
            )

#Create table for payment
dbExecute(db_connection,
            "CREATE TABLE IF NOT EXISTS payments (
                payment_id VARCHAR (50) PRIMARY KEY,
                payment_method VARCHAR (100) NOT NULL,
                payment_amount DECIMAL,
                payment_status VARCHAR (100) NOT NULL,
                payment_date DATE,
                order_id VARCHAR (50),
                FOREIGN KEY (order_id)
                  REFERENCES orders(order_id)
                )"
            )
```

```
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
```

```
dbExecute(db_connection,
          "CREATE TABLE IF NOT EXISTS customer_queries (
              query_id VARCHAR (50) PRIMARY KEY,
              query_title VARCHAR (50) NOT NULL,
              query_submission_date DATE,
              query_closure_date DATE,
              query_status VARCHAR (50) NOT NULL,
              cust_id VARCHAR (50),
              FOREIGN KEY (cust_id)
                REFERENCES customers(cust_id)
            )"
          )

#Create table for categories
dbExecute(db_connection,
          "CREATE TABLE IF NOT EXISTS categories (
              category_id VARCHAR (50) PRIMARY KEY,
              category_name VARCHAR (50) NOT NULL UNIQUE
            )"
          )

#Create table for product categories
dbExecute(db_connection,
          "CREATE TABLE IF NOT EXISTS product_categories (
              category_id VARCHAR (50),
              prod_id VARCHAR (50),
              FOREIGN KEY (prod_id)
                REFERENCES categories(category_id),
              FOREIGN KEY (prod_id)
                REFERENCES products(prod_id)
            )"
          )

#Create table for advertiser
dbExecute(db_connection,
          "CREATE TABLE IF NOT EXISTS advertisers (
              advertiser_id VARCHAR (50) PRIMARY KEY,
              advertiser_name VARCHAR (50) NOT NULL UNIQUE,
              advertiser_email VARCHAR (50) UNIQUE
            )"
          )

#Create table for advertisements
dbExecute(db_connection,
          "CREATE TABLE IF NOT EXISTS advertisements (
              ads_id VARCHAR (50) PRIMARY KEY,
              ads_start_date DATE,
```

```
                ads_end_date DATE,
                prod_id VARCHAR (50) UNIQUE,
                advertiser_id VARCHAR (50),
                FOREIGN KEY (prod_id)
                  REFERENCES products(prod_id),
                FOREIGN KEY (advertiser_id)
                  REFERENCES advertisers(advertiser_id)
              )"
            )
```

2 Data Generation and Management

2.1 Synthethic Data Generation

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

```r
## Synthetic Data Generation #1

### 'customers' table
#Define parameters for customers
set.seed(312)
n_customers <- 100
birthdate <- sample(seq(from = as.Date(today() - years(80), "%d-%m-%Y"),
                        to = as.Date(today() - years(18), "%d-%m-%Y"), by = "day"),
                    n_customers)
cv_postcode <-
  read.csv("data_uploads/ONSPD_AUG_2023_UK_CV.csv")[, 1] %>%
  data.frame() %>%
  setNames("pcd")
address_type <- c("Home", "Office")
#Create data
customers_data <-
  #Create n unique customer IDs with random names
  data.frame("cust_id" = conjurer::buildCust(n_customers),
             "cust_name" = randomNames::randomNames(n_customers)) %>%
  separate(cust_name, into = c("last_name", "first_name"), sep = ", ") %>%
  #Create email column, by merging last & first name with email domain @gmail.com
  unite(cust_email, c(last_name, first_name), sep = ".", remove = F) %>%
  mutate(
    "cust_email" = paste(cust_email,"gmail.com", sep = "@"),
    #Generate user's password, using random string generation package
    "password" =
```

```r
      stringi::stri_rand_strings(n=n_customers, length=8, pattern="[A-Za-z0-9]"),
    #Adding customer BOD
    "cust_birth_date" = sample(birthdate, n_customers, replace = T),
    #Adding the phone code in UK
    "phone_domain" = "075",
    #create unique random strings of 7 digits
    "cust_telephone" =
      stringi::stri_rand_strings(n=n_customers, length=7, pattern="[0-9]"),
    "block_num" =
      sprintf("%s%s",
              stri_rand_strings(n=n_customers, length=1, pattern="[A-Z]"),
              stri_rand_strings(n=n_customers, length=2, pattern="[0-99]")),
    #randomly assign postcode to each customer
    "postcode" = cv_postcode[sample(nrow(cv_postcode), n_customers),],
    #randomly assign address type to each customer
    "address_type" = sample(address_type, n_customers, replace = T)) %>%
  #Adding customer's telephone number by merging two phone number columns
  unite(cust_telephone,
        c(phone_domain, cust_telephone), sep = "", remove = T) %>%
  #reorder the columns
  select(1,4,3,2,5,6,8,9,10,7)
customers_data$cust_birth_date <- format(customers_data$cust_birth_date, "%d-%m-%Y")
#Save data to data file
write.csv(customers_data, "data_uploads/R_synth_customers_round1.csv")

### 'products' table
#Getting brand and product names from Gemini
gemini_prods <-
  readxl::read_excel("data_uploads/gemini_prod_cate_supplier.xlsx",
                     .name_repair = "universal") %>%
  setNames(c("seller_name", "category", "prod_name", "prod_desc"))
#Define parameters for products
set.seed(123)
n_prods <- 20
voucher_type <- c("10%", "20%", "50%")
ratings <- c(1,2,3,4,5)
date <- #assuming company was established on Mar 06th 2004, data here is
  sample(seq(from = as.Date("2004/03/06"),
             to = as.Date(lubridate::today()), by = "day"), 12)
#Assign product ID, and adding product names and URL
products_data <-
  #generate product id
  conjurer::buildProd(n_prods, minPrice = 1, maxPrice = 100) %>%
  #add product name and description from gemini's file
  mutate("prod_name" = sample(gemini_prods$prod_name, 20)) %>%
  left_join(select(gemini_prods, -c(seller_name, category)),
            by = join_by(prod_name)) %>%
```

```r
  #rename columns to fit schema
  rename(prod_id = SKU, prod_unit_price = Price) %>%
  #rename `sku` with `prod`
  mutate("prod_id" = gsub("sku", "prod", prod_id)) %>%
  #add product url
  mutate("web_prefix" = "https://group9.co.uk/",
         "prod_url1" = gsub(" ", "-", prod_name)) %>%
  unite(prod_url, c(prod_url1, prod_id), sep = "-", remove = F) %>%
  unite(prod_url, c(web_prefix, prod_url), sep = "", remove = T) %>%
  mutate(
    #Create ratings
    "prod_rating" = sample(ratings, n_prods, replace = T),
    #Review date
    "review_date" = sample(format(date, "%d-%m-%Y"), n_prods, replace = T),
    #Assign review ID
    "review_id" =
      conjurer::buildCust(sum(!is.na(prod_rating))),
    "review_id" = gsub("cust", "rev", review_id)) %>%
  #drop temp url
  select(-prod_url1)
#Create vouchers -- Randomly assign voucher types to 50% of the products
voucher_prods <- sample_n(data.frame(products_data$prod_id),
                          0.5*nrow(products_data)) %>% setNames("prod_id")
products_data <- products_data %>%
  mutate(voucher = ifelse(products_data$prod_id %in% voucher_prods$prod_id,
                          sample(voucher_type, nrow(voucher_prods), replace = T), NA))
#Finalise the table
products_data <-
  products_data %>%
  #rearrange order of columns
  select(2,4,5,8,6,7,3,9,1)
#Save to .csv file
write.csv(products_data, "data_uploads/R_synth_products_round1.csv")

### 'orders' table
#Define parameters
origin_date <- "1970-01-01"
n_orders <- 500
order_date <-
  #round 1 is for orders in 2022-Mar'2024,
  #so all orders have been paid and delivered successfully
  sample(seq(from = as.Date("2022/01/01"),
             to = as.Date("2024/03/01"), 12))
pymt_method <-
  c("Bank Transfer", "Visa", "Mastercard", "PayPal", "GPay", "Apple Pay")
pymt_status <- c("Done", "Verifying")
shipper_lookup <-
```

```r
  data.frame("shipper_name" = c("DHL", "Group9DL", "DPD"),
             "delivery_fee" = c(5,2,3),
             "ETA" = c(1,5,3))
delivery_status <- c("Delivered", "In Progress",
                     "Failed to contact", "Delayed")
orders_col_order <-
  c("order_id", "cust_id", "prod_id", "order_quantity",
    "order_date", "order_value", "order_price")
#generate n order IDs and assign customers to them, including order date
set.seed(122)
orders_data <-
  #Create n unique order IDs
  data.frame("order_id" = conjurer::buildCust(n_orders)) %>%
  mutate(order_id = gsub("cust", "o", order_id),
         payment_id = gsub("o", "pm", order_id),
         cust_id = sample(customers_data$cust_id, n_orders, replace = T),
         order_date = sample(order_date, n_orders, replace = T),
         payment_method = sample(pymt_method, n_orders, replace = T),
         payment_status = "Done",
         delivery_recipient = randomNames::randomNames(n_orders,
                                                       which.names = "first"))
#adding payment date with logic dependent on payment status
orders_data <- orders_data %>%
  mutate("payment_date" = ifelse(payment_status == "Done", order_date, NA)) %>%
  mutate("payment_date" = as.Date(payment_date,
                                  origin = origin_date))
#randomly replicate certain orders to map with products
set.seed(122)
orders_data <- orders_data %>% bind_rows() %>%
  rbind(sample_n(orders_data, 0.4*nrow(orders_data)),
        sample_n(orders_data, 0.5*nrow(orders_data)),
        sample_n(orders_data, 0.8*nrow(orders_data)))
#assign products to orders
orders_data <- orders_data %>%
  mutate(
    "prod_id" = sample(products_data$prod_id,
                       nrow(orders_data), replace = T),
    #generate order quantity
    "order_quantity" = sample(seq(1,10,1), nrow(orders_data), replace = T)) %>%
  merge(select(products_data, c(prod_id, prod_unit_price, voucher)),
        by = "prod_id")
#Order value and shipper
orders_data <- orders_data %>%
  #order price and value
  mutate(
    voucher = as.numeric(gsub("%", "", voucher))/100,
    #product unit price is discounted in case of voucher available
```

```
    order_price = ifelse(!is.na(voucher),
                         prod_unit_price * voucher, prod_unit_price),
    order_value = order_price * order_quantity,
    #assign shippers to products
    shipper_name =
      sample(shipper_lookup$shipper_name, nrow(orders_data), replace = T),
    #add delivery status
    delivery_status = "Delivered" ) %>%
  #lookup delivery fee
  merge(shipper_lookup, by = "shipper_name")
#dates of delivery
orders_data <- orders_data %>%
  #departure and ETA
  mutate(
    delivery_departed_date =
      ifelse(!is.na(payment_date), (payment_date + days(2)), NA),
    est_delivery_date = delivery_departed_date + ETA) %>%
  #departure and ETA - format as date
  mutate(
    delivery_departed_date =
      as.Date(delivery_departed_date, origin = origin_date),
    est_delivery_date =
      as.Date(est_delivery_date, origin = origin_date)) %>%
  #received
  mutate(
    delivery_received_date =
      ifelse(delivery_status != "Delivered", NA, est_delivery_date)) %>%
  mutate(
    delivery_received_date =
      as.Date(delivery_received_date, origin = origin_date)) %>%
  #drop ETA
  select(-ETA)

### generate 'shipment' from orders
shipment_colnames <- c("order_id", "prod_id",
                       "delivery_departed_date",
                       "delivery_received_date", "est_delivery_date",
                       "shipper_name", "delivery_recipient",
                       "delivery_fee", "delivery_status")
shipment_data <- select(orders_data, shipment_colnames)
shipment_data <- shipment_data %>%
  mutate(shipment_id = paste("sm", rownames(shipment_data), sep = ""),
         .before = "order_id")
#reformat date
shipment_dates <- c("delivery_departed_date",
                    "delivery_received_date", "est_delivery_date")
shipment_data[shipment_dates] <- lapply(shipment_data[shipment_dates],
```

```
                                        format, "%d-%m-%Y")
#Save data to data file
write.csv(shipment_data, "data_uploads/R_synth_shipment_round1.csv")

### generate 'payment' from orders
payment_colnames <- c("payment_id", "payment_method", "order_id",
                      "payment_status", "payment_date")
#Add payment amount
payment_data <- orders_data %>% group_by(payment_id) %>%
  summarise(payment_amount = sum(order_value)) %>%
  left_join(select(orders_data,payment_colnames), by = "payment_id")
#remove duplicates
payment_data <- distinct(payment_data, payment_id, .keep_all = T) %>%
  select(1,4,3,2,5,6)
#re-format date
payment_data$payment_date <- format(payment_data$payment_date, "%d-%m-%Y")
#Save data to data file
write.csv(payment_data, "data_uploads/R_synth_payment_round1.csv")

#reorder the columns of 'orders'
orders_data <- select(orders_data, orders_col_order)
#Save data to data file
write.csv(orders_data, "data_uploads/R_synth_orders_round1.csv")

### 'suppliers' table
#Define parameters for suppliers table
set.seed(123)
n_suppliers <- length(unique(gemini_prods$seller_name))
wc_postcode <- read.csv("data_uploads/ONSPD_AUG_2023_UK_WC.csv")[,1]

#Create suppliers table
suppliers_data <-
  #Pull seller name from gemini file
  distinct(select(gemini_prods, seller_name)) %>%
  rename("supplier_name" = "seller_name") %>%
  mutate("supplier_id" = seq(1, n_suppliers,1),
         "prefix" = "s") %>%
  unite(supplier_id, c(prefix, supplier_id), sep = "", remove = T) %>%
  mutate(
    "supplier_postcode" =
      sample(wc_postcode, n_suppliers, replace = T),
    #Adding the phone code in UK
    "phone_domain" = "079",
    #create unique random strings of 7 digits
    "supplier_contact" =
      stringi::stri_rand_strings(n=n_suppliers, length=7, pattern="[0-9]")) %>%
  #Adding supplier's telephone number by merging two phone number columns
```

```r
  unite(supplier_contact,
        c(phone_domain, supplier_contact), sep = "", remove = T) %>%
  select(2,1,4,3)
#Save data to data file
write.csv(suppliers_data, "data_uploads/R_synth_suppliers_round1.csv")

### 'supply' table
#Define parameters for supply table
set.seed(123)
order_quant_by_prod <- orders_data %>%
  group_by(prod_id) %>% summarise(sold_quantity = sum(order_quantity))
supply_col_order <- c("supply_id", "supplier_id", "prod_id",
                      "inventory_quantity", "sold_quantity")
#Create supply table
supply_data <- select(products_data, c(prod_id, prod_name)) %>%
  merge(order_quant_by_prod, by = "prod_id") %>%
  mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
  mutate(inventory_quantity =
           as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
  merge(select(gemini_prods, c(seller_name, prod_name)), by = "prod_name") %>%
  rename("supplier_name" = "seller_name") %>%
  merge(select(suppliers_data, c(supplier_id, supplier_name)),
        by = "supplier_name")
#Create competitors for M:N relationship
supply_competitors <- select(products_data, c(prod_id, prod_name)) %>%
  mutate(supplier_name =
           sample(suppliers_data$supplier_name, n_prods, replace = T)) %>%
  merge(select(suppliers_data, c(supplier_id, supplier_name)),
        by = "supplier_name") %>%
  merge(order_quant_by_prod, by = "prod_id") %>%
  mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
  mutate(inventory_quantity =
           as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
  select(2,3,1,5,6,4)
#Combine supply and competitors
supply_data <-
  rbind(supply_data, supply_competitors) %>%
  mutate(supply_id = paste("sp", row_number(), sep = "")) %>%
  select(-c(supplier_name, prod_name))
#reorder columns
supply_data <- supply_data[, supply_col_order]
#Save data to data file
write.csv(supply_data, "data_uploads/R_synth_supply_round1.csv")

### 'memberships' table
membership_lookup <-
  data.frame(
```

```r
    "membership_type" =  c("Student", "Trial", "Premium")) %>%
  mutate("membership_type_id" = row_number())

#Start with the foreign key cust_id
set.seed(123)
memberships_data <- data.frame(customers_data$cust_id)
memberships_data <- memberships_data %>%
  #Randomly assign membership type to all customers
  mutate("membership_type" =
           sample(membership_lookup$membership_type,
                  nrow(memberships_data), replace = T)) %>%
  #Lookup membership_id
  merge(membership_lookup, by = "membership_type") %>%
  rename(cust_id = customers_data.cust_id) %>%
  select(3,2,1)
#Save to .csv file
write.csv(memberships_data, "data_uploads/R_synth_memberships_round1.csv")

### 'customer_queries' table
set.seed(123)
n_queries <- 20
customer_queries_data <- data.frame(
  query_id = sprintf("Q%d", 1:n_queries),
  cust_id = sample(customers_data$cust_id, n_queries, replace = TRUE),
  query_title = sample(c("Delivery Issue", "Payment Issue", "Purchase Return", "Damaged Product
  query_submission_date = sample(seq(as.Date('2023-01-01'), as.Date('2023-1-31'), by="day"), n_
  query_closure_date = sample(seq(as.Date('2023-02-01'), as.Date('2023-03-31'), by="day"), n_qu
  query_status = sample(c("Closed"), n_queries, replace = TRUE)
)

customer_queries_data$query_submission_date <- format(customer_queries_data$query_submission_da
customer_queries_data$query_closure_date <- format(customer_queries_data$query_closure_date, "%

#Save to .csv file
write.csv(customer_queries_data, "data_uploads/R_synth_customer_queries_round1.csv", row.names

### 'categories' table
#create lookup table for category_id and category name
set.seed(123)
category_lookup <-
  data.frame("category_id" = seq(1, length(unique(gemini_prods$category)),1),
             "category" = unique(gemini_prods$category),
             "cate_code" = "cate") %>%
  unite(category_id, c(cate_code, category_id), sep = "", remove = T)
#Create categories table
categories_data <-
  #Pull category name and product name from gemini file
```

```r
  select(gemini_prods, c(category, prod_name)) %>%
  #Only keep the products included in the products table
  right_join(select(products_data, c(prod_id, prod_name)), by = "prod_name") %>%
  #lookup category_id
  merge(category_lookup, by = "category") %>%
  #rename to have category_name column
  rename(category_name = category) %>%
  #drop product name column
  select(-prod_name) %>%
  #reorder the columns to match with table schema
  select(3,2,1)
#Save to .csv file
write.csv(categories_data, "data_uploads/R_synth_categories_round1.csv")

### 'advertisers' table
set.seed(123)
n_advertisers <- 5
advertisers_data <- data.frame(
  advertiser_id = sprintf("ADV%d", 1:n_advertisers),
  advertiser_name = c("Ads Life", "Ads Idol", "Ads is Life", "Ads Master", "Ads Expert"),
  advertiser_email = sprintf("advertiser%d@gmail.com", 1:n_advertisers)
)
#Save to .csv file
write.csv(advertisers_data, "data_uploads/R_synth_advertisers_round1.csv", row.names = FALSE)

### 'advertisements' table
set.seed(123)
n_ads <- 9
advertisements_data <- data.frame(
  ads_id = sprintf("ADS%d", 1:n_ads),
  prod_id = sample(products_data$prod_id, n_ads, replace = TRUE),
  advertiser_id = sample(advertisers_data$advertiser_id, n_ads, replace = TRUE),
  ads_start_date = sample(seq(as.Date('2023-01-01'), as.Date('2023-12-31'), by="day"), n_ads,
  ads_end_date = sample(seq(as.Date('2024-01-01'), as.Date('2024-12-31'), by="day"), n_ads, re
)

advertisements_data$ads_start_date <- format(advertisements_data$ads_start_date, "%d-%m-%Y")
advertisements_data$ads_end_date <- format(advertisements_data$ads_end_date, "%d-%m-%Y")

#Save to .csv file
write.csv(advertisements_data, "data_uploads/R_synth_advertisements_round1.csv", row.names = F

### 'customers' table
#Define parameters for customers
set.seed(456)
n_customers <- 100
birthdate <- sample(seq(from = as.Date(today() - years(80), "%d-%m-%Y"),
```

```
                        to = as.Date(today() - years(18), "%d-%m-%Y"), by = "day"),
                    n_customers)
cv_postcode <-
  read.csv("data_uploads/ONSPD_AUG_2023_UK_CV.csv")[, 1] %>%
  data.frame() %>%
  setNames("pcd")
address_type <- c("Home", "Office")
#Create data
customers_data <-
  #Create n unique customer IDs with random names
  data.frame("cust_id" = paste("cust", seq(101,101+n_customers-1,1), sep = ""),
             "cust_name" = randomNames::randomNames(n_customers)) %>%
  separate(cust_name, into = c("last_name", "first_name"), sep = ", ") %>%
  #Create email column, by merging last & first name with email domain @gmail.com
  unite(cust_email, c(last_name, first_name), sep = ".", remove = F) %>%
  mutate(
    "cust_email" = paste(cust_email,"gmail.com", sep = "@"),
    #Generate user's password, using random string generation package
    "password" =
      stringi::stri_rand_strings(n=n_customers, length=8, pattern="[A-Za-z0-9]"),
    #Adding customer BOD
    "cust_birth_date" = sample(birthdate, n_customers, replace = T),
    #Adding the phone code in UK
    "phone_domain" = "075",
    #create unique random strings of 7 digits
    "cust_telephone" =
      stringi::stri_rand_strings(n=n_customers, length=7, pattern="[0-9]"),
    "block_num" =
      sprintf("%s%s",
              stri_rand_strings(n=n_customers, length=1, pattern="[A-Z]"),
              stri_rand_strings(n=n_customers, length=2, pattern="[0-99]")),
    #randomly assign postcode to each customer
    "postcode" = cv_postcode[sample(nrow(cv_postcode), n_customers),],
    #randomly assign address type to each customer
    "address_type" = sample(address_type, n_customers, replace = T)) %>%
  #Adding customer's telephone number by merging two phone number columns
  unite(cust_telephone,
        c(phone_domain, cust_telephone), sep = "", remove = T) %>%
  #reorder the columns
  select(1,4,3,2,5,6,8,9,10,7)
customers_data$cust_birth_date <- format(customers_data$cust_birth_date, "%d-%m-%Y")
#Save data to data file
write.csv(customers_data, "data_uploads/R_synth_customers_round2.csv")

### 'products' table
#Getting brand and product names from Gemini
gemini_prods <-
```

```
  readxl::read_excel("data_uploads/gemini_prod_cate_supplier_2.xlsx",
                     .name_repair = "universal") %>%
  setNames(c("seller_name", "category", "prod_name", "prod_desc"))
#Define parameters for products
set.seed(456)
n_prods <- 19
voucher_type <- c("10%", "20%", "50%")
ratings <- c(1,2,3,4,5)
date <- #assuming company was established on Mar 06th 2004
  sample(seq(from = as.Date("2004/03/06"),
             to = as.Date(lubridate::today()), by = "day"), 12)
#Assign product ID, and adding product names and URL
products_data <-
  #generate product id
  conjurer::buildProd(n_prods, minPrice = 1, maxPrice = 100) %>%
  #add product name and description from gemini's file
  mutate("prod_name" = sample(gemini_prods$prod_name, nrow(gemini_prods))) %>%
  left_join(select(gemini_prods, -c(seller_name, category)),
            by = join_by(prod_name)) %>%
  #rename columns to fit schema
  rename(prod_id = SKU, prod_unit_price = Price) %>%
  #rename `sku` with `prod`
  mutate("prod_id" = gsub("sku", "", prod_id)) %>%
  mutate("prod_id" = paste("prod", as.numeric(prod_id)+20, sep = "")) %>%
  #add product url
  mutate("web_prefix" = "https://group9.co.uk/",
         "prod_url1" = gsub(" ", "-", prod_name)) %>%
  unite(prod_url, c(prod_url1, prod_id), sep = "-", remove = F) %>%
  unite(prod_url, c(web_prefix, prod_url), sep = "", remove = T) %>%
  mutate(
    #Create ratings
    "prod_rating" = sample(ratings, n_prods, replace = T),
    #Review date
    "review_date" = sample(format(date, "%d-%m-%Y"), n_prods, replace = T),
    #Assign review ID
    "review_id" = paste("rev", seq(21, 21+n_prods-1, 1), sep = ""),
    "review_id" = gsub("cust", "rev", review_id)) %>%
  #drop temp url
  select(-prod_url1)
#Create vouchers -- Randomly assign voucher types to 50% of the products
voucher_prods <- sample_n(data.frame(products_data$prod_id),
                          0.5*nrow(products_data)) %>% setNames("prod_id")
products_data <- products_data %>%
  mutate(voucher = ifelse(products_data$prod_id %in% voucher_prods$prod_id,
                          sample(voucher_type, nrow(voucher_prods), replace = T), NA))
#Finalise the table
products_data <-
```

```r
  products_data %>%
  #rearrange order of columns
  select(2,4,5,8,6,7,3,9,1)
#Save to .csv file
write.csv(products_data, "data_uploads/R_synth_products_round2.csv")

### 'orders' table
#Define parameters
origin_date <- "1970-01-01"
n_orders <- 100
order_date <- #round 2 is for orders in 2024
  sample(seq(from = as.Date("2024/03/01"),
             to = as.Date(lubridate::today()), by = "day"), 12)
pymt_method <-
  c("Bank Transfer", "Visa", "Mastercard", "PayPal", "GPay", "Apple Pay")
pymt_status <- c("Done", "Verifying")
shipper_lookup <-
  data.frame("shipper_name" = c("DHL", "Group9DL", "DPD"),
             "delivery_fee" = c(5,2,3),
             "ETA" = c(1,5,3))
delivery_status <- c("Delivered", "In Progress",
                     "Failed to contact", "Delayed")
orders_col_order <-
  c("order_id", "cust_id", "prod_id", "order_quantity",
    "order_date", "order_value", "order_price")
#generate n order IDs and assign customers to them, including order date
set.seed(321)
orders_data <-
  #Create n unique order IDs
  data.frame("order_id" = paste("o",seq(501, 501+n_orders-1, 1), sep = "")) %>%
  mutate(order_id = gsub("cust", "o", order_id),
         payment_id = gsub("o", "pm", order_id),
         cust_id = sample(customers_data$cust_id, n_orders, replace = T),
         order_date = sample(order_date, n_orders, replace = T),
         payment_method = sample(pymt_method, n_orders, replace = T),
         payment_status = sample(pymt_status, n_orders, replace = T),
         delivery_recipient = randomNames::randomNames(n_orders,
                                                       which.names = "first"))
#adding payment date with logic dependent on payment status
orders_data <- orders_data %>%
  mutate("payment_date" = ifelse(payment_status == "Done", order_date, NA)) %>%
  mutate("payment_date" = as.Date(payment_date,
                                  origin = origin_date))
#randomly replicate certain orders to map with products
set.seed(456)
orders_data <- orders_data %>% bind_rows() %>%
  rbind(sample_n(orders_data, 0.4*nrow(orders_data)),
```

```
        sample_n(orders_data, 0.5*nrow(orders_data)),
        sample_n(orders_data, 0.8*nrow(orders_data)))
#assign products to orders
orders_data <- orders_data %>%
  mutate(
    "prod_id" = sample(products_data$prod_id,
                       nrow(orders_data), replace = T),
    #generate order quantity
    "order_quantity" = sample(seq(1,10,1), nrow(orders_data), replace = T)) %>%
  merge(select(products_data, c(prod_id, prod_unit_price, voucher)),
        by = "prod_id")
#Order value and shipper
orders_data <- orders_data %>%
  #order price and value
  mutate(
    voucher = as.numeric(gsub("%", "", voucher))/100,
    #product unit price is discounted in case of voucher available
    order_price = ifelse(!is.na(voucher),
                         prod_unit_price * voucher, prod_unit_price),
    order_value = order_price * order_quantity,
    #assign shippers to products
    shipper_name =
      sample(shipper_lookup$shipper_name, nrow(orders_data), replace = T),
    #add delivery status
    delivery_status =
      ifelse(payment_status != "Done", "Not Started",
             sample(delivery_status, nrow(orders_data), replace = T)) ) %>%
  #lookup delivery fee
  merge(shipper_lookup, by = "shipper_name")
#dates of delivery
orders_data <- orders_data %>%
  #departure and ETA
  mutate(
    delivery_departed_date =
      ifelse(!is.na(payment_date), (payment_date + days(2)), NA),
    est_delivery_date = delivery_departed_date + ETA) %>%
  #departure and ETA - format as date
  mutate(
    delivery_departed_date =
      as.Date(delivery_departed_date, origin = origin_date),
    est_delivery_date =
      as.Date(est_delivery_date, origin = origin_date)) %>%
  #received
  mutate(
    delivery_received_date =
      ifelse(delivery_status != "Delivered", NA, est_delivery_date)) %>%
  mutate(
```

```
    delivery_received_date =
      as.Date(delivery_received_date, origin = origin_date)) %>%
  #drop ETA
  select(-ETA)

### generate 'shipment' from orders
shipment_colnames <- c("order_id", "prod_id",
                       "delivery_departed_date",
                       "delivery_received_date", "est_delivery_date",
                       "shipper_name", "delivery_recipient",
                       "delivery_fee", "delivery_status")
shipment_data <- select(orders_data, shipment_colnames)
shipment_data <- shipment_data %>%
  mutate(shipment_id = paste("sm", rownames(shipment_data), sep = ""),
         .before = "order_id")
#reformat date
shipment_dates <- c("delivery_departed_date",
                    "delivery_received_date", "est_delivery_date")
shipment_data[shipment_dates] <- lapply(shipment_data[shipment_dates],
                                        format, "%d-%m-%Y")
#Save data to data file
write.csv(shipment_data, "data_uploads/R_synth_shipment_round2.csv")

### generate 'payment' from orders
payment_colnames <- c("payment_id", "payment_method", "order_id",
                      "payment_status", "payment_date")
#Add payment amount
payment_data <- orders_data %>% group_by(payment_id) %>%
  summarise(payment_amount = sum(order_value)) %>%
  left_join(select(orders_data,payment_colnames), by = "payment_id")
#remove duplicates
payment_data <- distinct(payment_data, payment_id, .keep_all = T) %>%
  select(1,4,3,2,5,6)
#re-format date
payment_data$payment_date <- format(payment_data$payment_date, "%d-%m-%Y")
#Save data to data file
write.csv(payment_data, "data_uploads/R_synth_payment_round2.csv")

#reorder the columns of 'orders'
orders_data <- select(orders_data, orders_col_order)
#Save data to data file
write.csv(orders_data, "data_uploads/R_synth_orders_round2.csv")

### 'suppliers' table
#Define parameters for suppliers table
set.seed(456)
n_suppliers <- length(unique(gemini_prods$seller_name))
```

```r
wc_postcode <- read.csv("data_uploads/ONSPD_AUG_2023_UK_WC.csv")[,1]

#Create suppliers table
suppliers_data <-
  #Pull seller name from gemini file
  distinct(select(gemini_prods, seller_name)) %>%
  rename("supplier_name" = "seller_name") %>%
  mutate("supplier_id" = seq(21, 21+n_suppliers-1,1),
         "prefix" = "s") %>%
  unite(supplier_id, c(prefix, supplier_id), sep = "", remove = T) %>%
  mutate(
    "supplier_postcode" =
      sample(wc_postcode, n_suppliers, replace = T),
    #Adding the phone code in UK
    "phone_domain" = "079",
    #create unique random strings of 7 digits
    "supplier_contact" =
      stringi::stri_rand_strings(n=n_suppliers, length=7, pattern="[0-9]")) %>%
  #Adding supplier's telephone number by merging two phone number columns
  unite(supplier_contact,
        c(phone_domain, supplier_contact), sep = "", remove = T) %>%
  select(2,1,4,3)
#Save data to data file
write.csv(suppliers_data, "data_uploads/R_synth_suppliers_round2.csv")

### 'supply' table
#Define parameters for supply table
set.seed(456)
order_quant_by_prod <- orders_data %>%
  group_by(prod_id) %>% summarise(sold_quantity = sum(order_quantity))
supply_col_order <- c("supply_id", "supplier_id", "prod_id",
                      "inventory_quantity", "sold_quantity")
#Create supply table
supply_data <- select(products_data, c(prod_id, prod_name)) %>%
  merge(order_quant_by_prod, by = "prod_id") %>%
  mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
  mutate(inventory_quantity =
           as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
  merge(select(gemini_prods, c(seller_name, prod_name)), by = "prod_name") %>%
  rename("supplier_name" = "seller_name") %>%
  merge(select(suppliers_data, c(supplier_id, supplier_name)),
        by = "supplier_name")
#Create competitors for M:N relationship
supply_competitors <- select(products_data, c(prod_id, prod_name)) %>%
  mutate(supplier_name =
           sample(suppliers_data$supplier_name, n_prods, replace = T)) %>%
  merge(select(suppliers_data, c(supplier_id, supplier_name)),
```

```r
        by = "supplier_name") %>%
  merge(order_quant_by_prod, by = "prod_id") %>%
  mutate(sold_quantity = as.integer(sample(seq(0.2,1),1)*sold_quantity)) %>%
  mutate(inventory_quantity =
           as.integer(sold_quantity * sample(seq(1.1, 2.3), 1))) %>%
  select(2,3,1,5,6,4)
#Combine supply and competitors
supply_data <-
  rbind(supply_data, supply_competitors) %>%
  mutate(supply_id = paste("sp", row_number(), sep = "")) %>%
  select(-c(supplier_name, prod_name))
#reorder columns
supply_data <- supply_data[, supply_col_order]
#Save data to data file
write.csv(supply_data, "data_uploads/R_synth_supply_round2.csv")

### 'memberships' table
membership_lookup <-
  data.frame(
    "membership_type" =  c("Student", "Trial", "Premium")) %>%
  mutate("membership_type_id" = row_number())

#Start with the foreign key cust_id
set.seed(456)
memberships_data <- data.frame(customers_data$cust_id)
memberships_data <- memberships_data %>%
  #Randomly assign membership type to all customers
  mutate("membership_type" =
           sample(membership_lookup$membership_type,
                  nrow(memberships_data), replace = T)) %>%
  #Lookup membership_id
  merge(membership_lookup, by = "membership_type") %>%
  rename(cust_id = customers_data.cust_id) %>%
  select(3,2,1)
#Save to .csv file
write.csv(memberships_data, "data_uploads/R_synth_memberships_round2.csv")

### 'customer_queries' table
set.seed(456)
n_queries <- 20
customer_queries_data <- data.frame(
  "query_id" = paste("Q",seq(21, 21+n_queries-1, 1), sep = ""),
  cust_id = sample(customers_data$cust_id, n_queries, replace = TRUE),
  query_title = sample(c("Delivery Issue", "Payment Issue", "Purchase Return", "Damaged Product
  query_submission_date = sample(seq(as.Date('2024-03-15'), as.Date('2024-03-20'), by="day"), r
  query_closure_date = sample(c("NA"), n_queries, replace = TRUE),
  query_status = sample(c("On Progress", "Submitted"), n_queries, replace = TRUE)
```

```r
)

customer_queries_data$query_submission_date <- format(customer_queries_data$query_submission_da

#Save to .csv file
write.csv(customer_queries_data, "data_uploads/R_synth_customer_queries_round2.csv", row.names

### 'categories' table
#create lookup table for category_id and category name
set.seed(456)
category_lookup <-
  data.frame("category_id" = seq(1, length(unique(gemini_prods$category)),1),
             "category" = unique(gemini_prods$category),
             "cate_code" = "cate") %>%
  unite(category_id, c(cate_code, category_id), sep = "", remove = T)
#Create categories table
categories_data <-
  #Pull category name and product name from gemini file
  select(gemini_prods, c(category, prod_name)) %>%
  #Only keep the products included in the products table
  right_join(select(products_data, c(prod_id, prod_name)), by = "prod_name") %>%
  #lookup category_id
  merge(category_lookup, by = "category") %>%
  #rename to have category_name column
  rename(category_name = category) %>%
  #drop product name column
  select(-prod_name) %>%
  #reorder the columns to match with table schema
  select(3,2,1)
#Save to .csv file
write.csv(categories_data, "data_uploads/R_synth_categories_round2.csv")

### 'advertisers' table
set.seed(456)
n_advertisers <- 5
advertisers_data <- data.frame(

  advertiser_id = sprintf("ADV%d", 1:n_advertisers),
  advertiser_name = c("Ads Life", "Ads Idol", "Ads is Life",
                      "Ads Master", "Ads Expert"),

  "advertiser_id" = paste("ADV",seq(6, 6+n_advertisers-1, 1), sep = ""),
  advertiser_name = c("Ads Beauty", "Ads Power", "Ads by WBS", "Ads by MSBA", "Ads Master"),

  advertiser_email = sprintf("advertiser%d@gmail.com", 1:n_advertisers)
)
#Save to .csv file
```

```r
write.csv(advertisers_data, "data_uploads/R_synth_advertisers_round2.csv", row.names = FALSE)

### 'advertisements' table
set.seed(456)
n_ads <- 9
advertisements_data <- data.frame(
  "ads_id" = paste("ADS",seq(10, 10+n_ads-1, 1), sep = ""),
  prod_id = sample(products_data$prod_id, n_ads, replace = TRUE),
  advertiser_id = sample(advertisers_data$advertiser_id, n_ads, replace = TRUE),
  ads_start_date = sample(seq(as.Date('2023-01-01'), as.Date('2023-12-31'), by="day"), n_ads,
  ads_end_date = sample(seq(as.Date('2024-01-01'), as.Date('2024-12-31'), by="day"), n_ads, re
)

advertisements_data$ads_start_date <- format(advertisements_data$ads_start_date, "%d-%m-%Y")
advertisements_data$ads_end_date <- format(advertisements_data$ads_end_date, "%d-%m-%Y")

#Save to .csv file
write.csv(advertisements_data, "data_uploads/R_synth_advertisements_round2.csv", row.names = F
```

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

2.2 Data Import and Quality Assurance

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

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

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

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

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

```r
# Read Data file
## Read advertisements file
advertisement_list <- list()
for (ads in list.files(path = "data_uploads/", pattern = "advertisement", full.names = TRUE)) {
  advertisements_ind <- read.csv(ads)
  advertisement_list[[length(advertisement_list) + 1]] <- advertisements_ind
}
advertisements_file <- bind_rows(advertisement_list)

## Read advertisers file
advertisers_list <- list()
for (adv in list.files(path = "data_uploads/", pattern = "advertiser", full.names = TRUE)) {
  advertisers_ind <- read.csv(adv)
  advertisers_list[[length(advertisers_list) + 1]] <- advertisers_ind
}
advertisers_file <- bind_rows(advertisers_list)

## Read categories file
categories_list <- list()
for (cat in list.files(path = "data_uploads/", pattern = "categories", full.names = TRUE)) {
  categories_ind <- read.csv(cat)
  categories_list[[length(categories_list) + 1]] <- categories_ind
}
categories_file <- bind_rows(categories_list)

## Read customer_queries file
customer_queries_list <- list()
for (cat in list.files(path = "data_uploads/", pattern = "customer_queries", full.names = TRUE)
  customer_queries_ind <- read.csv(cat)
  customer_queries_list[[length(customer_queries_list) + 1]] <- customer_queries_ind
}
customer_queries_file <- bind_rows(customer_queries_list)

## Read customers file
customers_list <- list()
for (cust in list.files(path = "data_uploads/", pattern = "customers", full.names = TRUE)) {
  customers_ind <- read.csv(cust)
  customers_list[[length(customers_list) + 1]] <- customers_ind
}
customers_file <- bind_rows(customers_list)

## Read memberships file
memberships_list <- list()
for (memb in list.files(path = "data_uploads/", pattern = "membership", full.names = TRUE)) {
  memberships_ind <- read.csv(memb)
  memberships_list[[length(memberships_list) + 1]] <- memberships_ind
}
```

```r
memberships_file <- bind_rows(memberships_list)

## Read orders file
orders_list <- list()
for (orders in list.files(path = "data_uploads/", pattern = "order", full.names = TRUE)) {
  orders_ind <- read.csv(orders)
  orders_list[[length(orders_list) + 1]] <- orders_ind
}
orders_file <- bind_rows(orders_list)

## Read payments file
payments_list <- list()
for (payments in list.files(path = "data_uploads/", pattern = "payment", full.names = TRUE)) {
  payments_ind <- read.csv(payments)
  payments_list[[length(payments_list) + 1]] <- payments_ind
}
payments_file <- bind_rows(payments_list)

## Read products file
products_list <- list()
for (products in list.files(path = "data_uploads/", pattern = "product", full.names = TRUE)) {
  products_ind <- read.csv(products)
  products_list[[length(products_list) + 1]] <- products_ind
}
products_file <- bind_rows(products_list)

## Read shipments file
shipments_list <- list()
for (shipments in list.files(path = "data_uploads/", pattern = "shipment", full.names = TRUE))
  shipments_ind <- read.csv(shipments)
  shipments_list[[length(shipments_list) + 1]] <- shipments_ind
}
shipments_file <- bind_rows(shipments_list)

## Read suppliers file
suppliers_list <- list()
for (suppliers in list.files(path = "data_uploads/", pattern = "suppliers", full.names = TRUE)
  suppliers_ind <- read.csv(suppliers)
  suppliers_list[[length(suppliers_list) + 1]] <- suppliers_ind
}
suppliers_file <- bind_rows(suppliers_list)

## Read supplies file
supplies_list <- list()
for (supplies in list.files(path = "data_uploads/", pattern = "supply", full.names = TRUE)) {
  supplies_ind <- read.csv(supplies)
  supplies_list[[length(supplies_list) + 1]] <- supplies_ind
```

```r
}
supplies_file <- bind_rows(supplies_list)

# Normalising the Table into 3NF

##Normalising Products Table
products_table <- products_file %>%
  select(prod_id,prod_name,prod_desc,prod_unit_price,voucher,prod_url)

##Normalising Reviews Table
reviews_table <- products_file %>%
  select(review_id,prod_id, prod_rating, review_date)

##Normalising Memberships Table
memberships_table <- memberships_file %>%
  select(membership_type_id,membership_type)
memberships_table <- memberships_table[!duplicated(memberships_table$membership_type_id),]

##Normalising Customers Table
customers_table <- customers_file %>%
  select(cust_id, first_name, last_name, cust_email,password, cust_birth_date, block_num, poste
customers_table <- merge(customers_table,memberships_file, by = "cust_id")
customers_table$X <- NULL
customers_table$membership_type <- NULL

##Normalising Orders Table
orders_table <- orders_file %>%
  select(order_id, cust_id)
orders_table <- orders_table[!duplicated(orders_table$order_id),]

##Normalising Order details Table
order_details_table <- orders_file %>%
  select(order_id,prod_id, order_quantity, order_date, order_value, order_price)

##Normalising Payments Table
payments_table <- payments_file %>%
  select(payment_id,order_id,payment_amount,payment_method,payment_status,payment_date)

##Normalising Shipments Table
shipments_table <- shipments_file %>%
  select(shipment_id, order_id, prod_id, delivery_departed_date, delivery_received_date,est_del

##Normalising Suppliers Table
suppliers_table <- suppliers_file %>%
  select(supplier_id, supplier_name, supplier_contact, supplier_postcode)

##Normalising Supplies Table
```

```r
supplies_table <- supplies_file %>%
  select(supply_id,supplier_id,prod_id,inventory_quantity,sold_quantity)

##Normalising Customer Queries Table
customer_queries_table <- customer_queries_file

##Normalising Categories Table
categories_table <- categories_file %>%
  select(category_id,category_name)
categories_table <- categories_table[!duplicated(categories_table$category_id),]

##Normalising Product Categories Table
product_categories_table <- categories_file %>%
  select(prod_id, category_id)

##Normalising Advertiser Table
advertisers_table <- advertisers_file

##Normalising Advertisement Table
advertisements_table <- advertisements_file

# Data Validation

## Advertisement table
### Checking the date format for ads_start_date and ads_end_date
if (all(!inherits(try(as.Date(advertisements_table$ads_start_date, format = "%d-%m-%Y")),"try-
  print("Dates are already in the correct format")
} else {
  print("Dates are not in the correct format")
}

if (all(!inherits(try(as.Date(advertisements_table$ads_end_date, format = "%d-%m-%Y")),"try-err
  print("Dates are already in the correct format")
} else {
  print("Dates are not in the correct format")
}

## Ensuring advertisement end date is after the advertisement start date
for (i in 1:length(as.Date(advertisements_table$ads_start_date, format = "%d-%m-%Y"))) {
  if (as.Date(advertisements_table$ads_end_date, format = "%d-%m-%Y")[i] > as.Date(advertisemer
    print("Ends date happened after the starts date")
  } else {
    print(paste("Error!","Query",i,": ends date happened before the starts date"))
  }
}

### Checking duplicate values for ads_id and prod_id
```

```r
if(length(advertisements_table$ads_id[duplicated(advertisements_table$ads_id)]) > 0) {
  print("Duplicate ads_ids found")
} else {
  print("No duplicate ads_ids found")
}

if(length(advertisements_table$prod_id[duplicated(advertisements_table$prod_id)]) > 0) {
  print("Duplicate prod_ids found")
} else {
  print("No duplicate prod_ids found")
}

## Advertisers file
### Checking duplicate values for advertiser_id, advertiser_email, and advertisers name

if(length(advertisers_table$advertiser_id[duplicated(advertisers_table$advertiser_id)]) > 0) {
  print("Duplicate advertiser_ids found")
} else {
  print("No duplicate advertiser_ids found")
}

if(length(advertisers_table$advertiser_email[duplicated(advertisers_table$advertiser_email)]) >
  print("Duplicate advertisers' emails found")
} else {
  print("No duplicate advertisers' emails found")
}

if(length(advertisers_table$advertiser_name[duplicated(advertisers_table$advertiser_name)]) > 0
  print("Duplicate advertisers' names found")
} else {
  print("No duplicate advertisers' names found")
}

## Checking the advertiser_email format
if(length(grep(("^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.com$"),advertisers_table$advertiser_email
  print("All email format are correct")
} else {
  print(paste("There are:", length(advertisers_table$advertiser_email) - length(grep(("^[A-Za-z
}

## Customer_queries file
### Checking duplicate values for query_id

if(length(customer_queries_table$query_id[duplicated(customer_queries_table$query_id)]) > 0) {
  print("Duplicate queries_ids found")
} else {
  print("No duplicate queries_ids found")
```

```r
}

### Checking the date format for query_submission_date and query_closure_date
correct_date_format <- function(date_column) {
  converted_dates <- try(as.Date(date_column, format = "%d-%m-%Y"), silent = TRUE)
  if (inherits(converted_dates, "try-error")) {
    return(format(mdy(date_column), "%d-%m-%Y"))
  } else {
    return(date_column)
  }
}
customer_queries_table$query_submission_date <- correct_date_format(customer_queries_table$quer
customer_queries_table$query_closure_date <- correct_date_format(customer_queries_table$query_c

if (all(!inherits(try(as.Date(customer_queries_table$query_submission_date, format = "%d-%m-%Y'
  print("Query Submission Dates are now in the correct format")
} else {
  print("There was an issue with converting the Query Submission Dates")
}

if (all(!inherits(try(as.Date(customer_queries_table$query_closure_date, format = "%d-%m-%Y"))
  print("Query Closure Dates are now in the correct format")
} else {
  print("There was an issue with converting the Query Closure Dates")
}

## Memberships file
### Checking NA values inside membership_id and membership_type
if (any(!is.na(memberships_table))) {
  print("There are no NA values in the dataset")
} else {
  print("Error! There are NA values in the dataset")
}

## Orders file
### Checking NA values inside order_id, cust_id, order_quantity, order_price, prod_id
if (any(!is.na(order_details_table[,c("order_id", "order_quantity", "order_price", "prod_id")])
  print("There are no NA values in the dataset")
} else {
  print("Error! There are NA values in the dataset")
}

### Checking date format for the order_date
correct_date_format <- function(date_column) {
  converted_dates <- try(as.Date(date_column, format = "%d-%m-%Y"), silent = TRUE)
  if (inherits(converted_dates, "try-error")) {
    return(format(mdy(date_column), "%d-%m-%Y"))
```

```
  } else {
    return(date_column)
  }
}
order_details_table$order_date <- correct_date_format(order_details_table$order_date)

# Print a message based on the result
if (all(!inherits(try(as.Date(order_details_table$order_date, format = "%d-%m-%Y")), "try-erro
  print("Dates are now in the correct format")
} else {
  print("There was an issue with converting the dates")
}
## Payment_file
### Checking NA values inside payment_id, payment_method, payment_status, and order_id
if (any(!is.na(payments_table[,c("payment_id", "payment_method", "payment_status", "order_id")]
  print("There are no NA values in the dataset")
} else {
  print("Error! There are NA values in the dataset")
}

### Checking date format for the payment_date
if (all(!inherits(try(as.Date(payments_table$payment_date, format = "%d-%m-%Y")),"try-error")))
  print("Dates are already in the correct format")
} else {
  print("Dates are not in the correct format")
}

## Products_file
### Checking duplicate values in prod_id and review_id
if(length(products_table$prod_id[duplicated(products_table$prod_id)]) == 0) {
  print("No duplicate prod_ids found")
} else {
  print("Duplicate ad_ids found")
}

if(length(reviews_table$review_id[duplicated(reviews_table$review_id)]) == 0) {
  print("No duplicate review_ids found")
} else {
  print("Duplicate review_ids found")
}

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

```r

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

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

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

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

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

```r

shipments_table$delivery_departed_date <- check_and_correct_date_format(shipments_table$delive
shipments_table$delivery_received_date <- check_and_correct_date_format(shipments_table$delive
shipments_table$est_delivery_date <- check_and_correct_date_format(shipments_table$est_deliver

columns_to_check <- list(
  "Delivery Departed Date" = shipments_table$delivery_departed_date,
  "Delivery Received Date" = shipments_table$delivery_received_date,
  "Estimated Delivery Date" = shipments_table$est_delivery_date
)

for (column_name in names(columns_to_check)) {
  if (all(!inherits(try(as.Date(columns_to_check[[column_name]], format = "%d-%m-%Y")), "try-e
    print(paste(column_name, "are now in the correct format"))
  } else {
    print(paste("There was an issue with converting the", column_name))
  }
}

### Checking whether the recipient names contains ' and ,
if (any(grepl("[',]",shipments_table$delivery_recipient))) {
  print("Error! Some names contain invalid characters")
} else {
  print("All names are valid")
}

## Customer Table
### Checking duplicate values in customer_id
if(length(customers_table$cust_id[duplicated(customers_table$cust_id)]) == 0) {
  print("No duplicate customer_ids found")
} else {
  print("Duplicate customer_ids found")
}

### Checking whether the customer's first name and last name contains ' and ,
clean_name <- function(name) {
  gsub("['\",]", "-", name)
}

if (any(grepl("[',]", customers_table$first_name))) {
  print("Error! Some first names contain invalid characters")
  customers_table$first_name <- sapply(customers_table$first_name, clean_name)
} else {
  print("All first names are valid")
}

if (any(grepl("[',]", customers_table$last_name))) {
```

```r
  print("Error! Some last names contain invalid characters")
  customers_table$last_name <- sapply(customers_table$last_name, clean_name)
} else {
  print("All last names are valid")
}


### Checking the customer_email format
if(length(grep(("^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.com$"),customers_table$cust_email, value =
  print("All customer email format are correct")
} else {
  print(paste("There are:", length(customers_table$cust_email) - length(grep(("^[A-Za-z0-9._%+
}

### Checking the customer birth_date format
if (all(!inherits(try(as.Date(customers_table$cust_birth_date, format = "%d-%m-%Y")),"try-erro
  print("Dates are already in the correct format")
} else {
  print("Dates are not in the correct format")
}


# Create connection to SQL database
db_connection <- RSQLite::dbConnect(RSQLite::SQLite(),"IB9HP0_9.db")

# Inserting Dataframe into the sql database
write_log <- function(message, path) {
  timestamp <- format(Sys.time(), "%Y-%m-%d %H:%M:%S")
  message <- paste(timestamp, message, sep=" - ")
  write(message, file = path, append = TRUE, sep = "\n")
}

# Path for the error log file
log_file_path <- "error_log.txt"

## Inserting Products table
for(i in 1:nrow(products_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO products(prod_id, prod_name, prod_desc, voucher, prod_url, prod_unit_price)
      products_table$prod_id[i], "','",
      products_table$prod_name[i], "','",
      products_table$prod_desc[i], "','",
      products_table$voucher[i], "','",
      products_table$prod_url[i], "',",
      products_table$prod_unit_price[i], ");", sep = "")
    )
```

```
    write_log(sprintf("Product %s inserted successfully.", products_table$prod_id[i]), log_file
  }, error = function(e) {
    write_log(sprintf("Failed to insert product %s: %s", products_table$prod_id[i], e$message)
  })
}

## Inserting Reviews table
for(i in 1:nrow(reviews_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO reviews(review_id, prod_rating, review_date, prod_id) VALUES('",
      reviews_table$review_id[i], "',",
      reviews_table$prod_rating[i], ",'",
      reviews_table$review_date[i], "','",
      reviews_table$prod_id[i], "');", sep = "")
    )
    write_log(sprintf("Review %s inserted successfully.", reviews_table$review_id[i]), log_file
  }, error = function(e) {
    write_log(sprintf("Failed to insert review %s: %s", reviews_table$review_id[i], e$message)
  })
}

## Inserting Memberships table
for(i in 1:nrow(memberships_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO memberships(membership_type_id, membership_type) VALUES(",
      "'", memberships_table$membership_type_id[i], "',",
      "'", memberships_table$membership_type[i], "');", sep = "")
    )
    write_log(sprintf("Membership %s inserted successfully.", memberships_table$membership_type
  }, error = function(e) {
    write_log(sprintf("Failed to insert membership %s: %s", memberships_table$membership_type_
  })
}

## Inserting Customers table
for(i in 1:nrow(customers_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO customers(cust_id, first_name, last_name, cust_email, password, cust_birth_d
      "'", customers_table$cust_id[i], "',",
      "'", customers_table$first_name[i], "',",
      "'", customers_table$last_name[i], "',",
      "'", customers_table$cust_email[i], "',",
      "'", customers_table$password[i], "',",
      "'", customers_table$cust_birth_date[i], "',",
```

```
      "'", customers_table$address_type[i], "',",
      "'", customers_table$block_num[i], "',",
      "'", customers_table$postcode[i], "',",
      "'", customers_table$cust_telephone[i], "',",
      "'", customers_table$membership_type_id[i], "');", sep = "")
    )
    write_log(sprintf("Customer %s inserted successfully.", customers_table$cust_id[i]), log_fi
  }, error = function(e) {
    write_log(sprintf("Failed to insert customer %s: %s", customers_table$cust_id[i], e$message
  })
}

## Inserting Orders table
for(i in 1:nrow(orders_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO orders(order_id, cust_id) VALUES('",
      orders_table$order_id[i], "','",
      orders_table$cust_id[i], "');", sep = "")
    )
    write_log(sprintf("Order %s inserted successfully.", orders_table$order_id[i]), log_file_pa
  }, error = function(e) {
    write_log(sprintf("Failed to insert order %s: %s", orders_table$order_id[i], e$message), lo
  })
}

## Inserting Payment table
for(i in 1:nrow(payments_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO payments(payment_id, payment_method, payment_amount, payment_status, payment
      "'", payments_table$payment_id[i], "',",
      "'", payments_table$payment_method[i], "',",
      payments_table$payment_amount[i], ",",
      "'", payments_table$payment_status[i], "',",
      "'", payments_table$payment_date[i], "',",
      "'", payments_table$order_id[i], "');",sep = "")
    )
    write_log(sprintf("Payment %s inserted successfully.", payments_table$payment_id[i]), log_i
  }, error = function(e) {
    write_log(sprintf("Failed to insert payment %s: %s", payments_table$payment_id[i], e$messag
  })
}

## Inserting Shipment table
for(i in 1:nrow(shipments_table)){
  tryCatch({
```

```
    dbExecute(db_connection, paste(
      "INSERT INTO shipments(shipment_id, delivery_status, delivery_fee, delivery_recipient, sh
, prod_id, order_id) VALUES(",
      "'", shipments_table$shipment_id[i], "',",
      "'", shipments_table$delivery_status[i], "',",
      shipments_table$delivery_fee[i], ",",
      "'", shipments_table$delivery_recipient[i], "',",
      "'", shipments_table$shipper_name[i], "',",
      "'", format(as.Date(shipments_table$est_delivery_date[i], format = "%d-%m-%Y"), "%Y-%m-%d
      "'", format(as.Date(shipments_table$delivery_departed_date[i], format = "%d-%m-%Y"), "%Y-
      "'", format(as.Date(shipments_table$delivery_received_date[i], format = "%d-%m-%Y"), "%Y-
      "'", shipments_table$prod_id[i], "',",
      "'", shipments_table$order_id[i], "');", sep = "")
    )
    write_log(sprintf("Shipment %s inserted successfully.", shipments_table$shipment_id[i]), lo
  }, error = function(e) {
    write_log(sprintf("Failed to insert shipment %s: %s", shipments_table$shipment_id[i], e$mes
  })
}

## Inserting Order details table
for(i in 1:nrow(order_details_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO order_details(order_quantity,order_date,order_price,order_value,prod_id,orde
      order_details_table$order_quantity[i], ",",
      "'", order_details_table$order_date[i], "',",
      order_details_table$order_price[i], ",",
      order_details_table$order_value[i], ",",
      "'", order_details_table$prod_id[i], "',",
      "'", order_details_table$order_id[i], "');",sep = "")
    )

    write_log(sprintf("Order detail %s inserted successfully.", order_details_table$order_id[i]
  }, error = function(e) {
    write_log(sprintf("Failed to insert order detail %s: %s", order_details_table$order_id[i],
  })
}

## Inserting Suppliers table
for(i in 1:nrow(suppliers_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO suppliers(supplier_id, supplier_name, supplier_postcode, supplier_contact) V
      suppliers_table$supplier_id[i], "','",
      suppliers_table$supplier_name[i], "','",
      suppliers_table$supplier_postcode[i], "','",
```

```
      suppliers_table$supplier_contact[i], "');", sep = "")
    )
    write_log(sprintf("Supplier %s inserted successfully.", suppliers_table$supplier_id[i]), lo
  }, error = function(e) {
    write_log(sprintf("Failed to insert supplier %s: %s", suppliers_table$supplier_id[i], e$mes
  })
}


## Inserting Supplies table
for(i in 1:nrow(supplies_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO supplies(supply_id, inventory_quantity, sold_quantity, supplier_id, prod_id)
      supplies_table$supply_id[i], "',",
      supplies_table$inventory_quantity[i], ",",
      supplies_table$sold_quantity[i], ",'",
      supplies_table$supplier_id[i], "','",
      supplies_table$prod_id[i], "');", sep = "")
    )
    write_log(sprintf("Supply %s inserted successfully.", supplies_table$supply_id[i]), log_fil
  }, error = function(e) {
    write_log(sprintf("Failed to insert supply %s: %s", supplies_table$supply_id[i], e$message)
  })
}

## Inserting Customer queries table
for(i in 1:nrow(customer_queries_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO customer_queries(query_id, query_title, query_submission_date, query_closure
      "'", customer_queries_table$query_id[i], "',",
      "'", customer_queries_table$query_title[i], "',",
      "'", customer_queries_table$query_submission_date[i], "',",
      "'", customer_queries_table$query_closure_date[i], "',",
      "'", customer_queries_table$query_status[i], "',",
      "'", customer_queries_table$cust_id[i], "');", sep = "")
    )
    write_log(sprintf("Customer query %s inserted successfully.", customer_queries_table$query_
  }, error = function(e) {
    write_log(sprintf("Failed to insert customer query %s: %s", customer_queries_table$query_id
  })
}

## Inserting Categories table
for(i in 1:nrow(categories_table)){
  tryCatch({
```

```
    dbExecute(db_connection, paste(
      "INSERT INTO categories(category_id, category_name) VALUES('",
      categories_table$category_id[i], "','",
      categories_table$category_name[i], "');", sep = "")
    )
    write_log(sprintf("Category %s inserted successfully.", categories_table$category_id[i]), l
  }, error = function(e) {
    write_log(sprintf("Failed to insert category %s: %s", categories_table$category_id[i], e$me
  })
}

## Inserting Product Categories table
for(i in 1:nrow(product_categories_table)){
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO product_categories(category_id, prod_id) VALUES('",
      product_categories_table$category_id[i], "','",
      product_categories_table$prod_id[i], "');", sep = "")
    )
    write_log(sprintf("Product category link for product %s and category %s inserted successful
  }, error = function(e) {
    write_log(sprintf("Failed to insert product category link for product %s and category %s: %
  })
}

## Inserting Advertisers table
for(i in 1:nrow(advertisers_table)) {
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO advertisers(advertiser_id, advertiser_name, advertiser_email) VALUES(",
      "'", advertisers_table$advertiser_id[i], "',",
      "'", advertisers_table$advertiser_name[i], "',",
      "'", advertisers_table$advertiser_email[i], "');",sep = "")
    )
    write_log(sprintf("Advertiser %s inserted successfully.", advertisers_table$advertiser_id[i
  }, error = function(e) {
    write_log(sprintf("Failed to insert advertiser %s: %s", advertisers_table$advertiser_id[i]
  })
}

## Inserting Advertisements table
for(i in 1:nrow(advertisements_table)) {
  tryCatch({
    dbExecute(db_connection, paste(
      "INSERT INTO advertisements(ads_id, ads_start_date, ads_end_date, prod_id, advertiser_id)
      "'", advertisements_table$ads_id[i], "',",
      "'", advertisements_table$ads_start_date[i], "',",
```

```
      "'", advertisements_table$ads_end_date[i], "',",
      "'", advertisements_table$prod_id[i], "',",
      "'", advertisements_table$advertiser_id[i], "');", sep = "")
    )
    write_log(sprintf("Advertisement %s inserted successfully.", advertisements_table$ads_id[i]
  }, error = function(e) {
    error_message <- sprintf("Failed to insert advertisement %s: %s", advertisements_table$ads_
    write_log(error_message, log_file_path)
  })
}
```

3 Data Pipeline Generation

3.1 Introduction

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

3.2 GitHub Repository Structure

Our GitHub repository (IB9HP0_Group_9) has a clear and logical directory structure to support efficient project management and collaboration. The directory is structured as follows:

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

data_uploads folder: Used to store raw data files for uploads. Files in this directory follow a clear naming convention, such as R_synth_customers_round1.csv and

R_synth_customers_round2.csv, in order to quickly identify the source of the data. The insertion of new data also follows this rule, ensuring consistent data management.

analysis_results folder: used to store the results of the data analysis.

docs folder : holds project documents such as the analysis report Markdown file (IB9HP0_9.Rmd) and the project RStudio project file (ib9hp0_group_9.Rproj).

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

root: Includes .gitignore, README.md (the project description file), and the database file (IB9HP0_9.db).

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

3.3 GitHub Actions

3.3.1 Trigger conditions

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

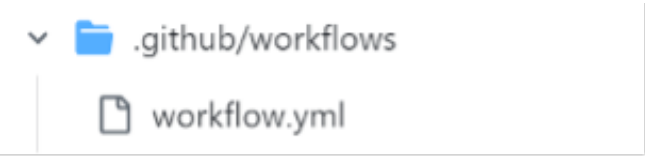

Figure 4 Workflow of the E-Commerce Repository

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

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

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

3.3.2 Automation Tasks

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

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

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

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

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

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

IB9HP0_9_synth_1.R and IB9HP0_9_synth_2.R: used to generate or process synthetic data.

IB9HP0_9_Table_Creation.R: used to create database tables.

IB9HP0_9_Data_Validation_Insertion.R: for data validation and insertion into database operations.

IB9HP0_9_Data_Analysis.R: to perform data analysis.

Git Operations:

Configure Git: sets the global Git username and email.

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

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

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

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

4 Data Analysis and Reporting

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

```
(sales_performance <-
    dbGetQuery(db_connection,
               "SELECT order_date AS date,
               SUM(order_value) AS value_sales,
               SUM(order_quantity) AS volume_sales,
               SUM(order_value)/SUM(order_quantity) AS avg_price
               FROM order_details
               GROUP BY date
               ORDER BY date"))

#### Transform data to get month and year
sales_performance <- sales_performance %>%
  mutate(month = format(as.Date(date), "%m"),
         year = format(as.Date(date), "%Y")) %>%
  group_by(month, year) %>%
  summarise(value_sales = sum(value_sales),
            volume_sales = sum(volume_sales)) %>%
  mutate(avg_price = value_sales/volume_sales)

#### Plot monthly value sales trend

p.mnth.val <- ggplot(filter(sales_performance,
                            year %in% c("2022", "2023")),
```

```
       aes(x = month, group = year, color = year,
           y = value_sales)) +geom_smooth(se = F, show.legend = F) +
  labs(y = "Sales Value (GBP)", x = "Month",
       subtitle = "Sales Value", color = "Year") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(labels = comma,
                     limits = c(0, 20000))

#### Plot monthly volume sales trend
p.mnth.vol <- ggplot(filter(sales_performance,
                            year %in% c("2022", "2023")),
                     aes(x = month, group = year, color = year,
                         y = volume_sales)) +
    geom_smooth(se = F, show.legend = F) +
    labs(y = "Sales Volume (Units)", x = "Month",
         subtitle = "Sales Volume", color = "Year") +
    theme_light() +
    theme(plot.title = element_text(hjust = 0.5)) +
    scale_y_continuous(labels = comma,
                       limits = c(0, 500))
#### Plot monthly avg price trend
p.mnth.price <- ggplot(filter(sales_performance,
                              year %in% c("2022", "2023")),
                       aes(x = month, group = year, color = year,
                           y = avg_price)) +
  geom_smooth(se = F) +
  labs(y = "Average Price (GBP/Unit)", x = "Month",
       subtitle = "Average Price", color = "Year") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(labels = comma,
                     limits = c(0, 100))
#### Combine value and volume sales graphs
gridExtra::grid.arrange(p.mnth.val, p.mnth.vol, p.mnth.price, ncol = 3, widths = c(0.9, 0.9, 1
                        top = ggpubr::text_grob("Monthly Sales Trend",
                                                size = 15, face = "bold"))
```

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

4.2 Top 10 Products and Their Rating

```
#### Get the data
(top_products_rating <-
    dbGetQuery(db_connection,
               "SELECT a.prod_name AS products,
               SUM(b.order_quantity) AS volume_sales,
               AVG(r.prod_rating) as avg_rating
               FROM products a
               JOIN order_details b ON a.prod_id = b.prod_id
               JOIN reviews r ON a.prod_id = r.prod_id
               GROUP BY a.prod_id
               ORDER BY volume_sales DESC
               LIMIT 10"))

#### Plot product sales graph
p.top_prod_sales <- ggplot(top_products_rating,
       aes(x= reorder(products, volume_sales))) +
  geom_bar(aes(y = volume_sales), stat = "identity") + coord_flip() +
  labs(x = "Products", y = "Volume Sales",
       subtitle = "Volume Sales") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"),
        axis.title.x = element_blank())
#### Plot product ratings graph
p.top_prod_rating <- ggplot(top_products_rating,
       aes(x= reorder(products, volume_sales))) +
  geom_bar(aes(y = avg_rating), stat = "identity",
           fill = "gray") + coord_flip() +
  labs(y = "Product Rating",
       subtitle = "Rating") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.title = element_blank()) +
  scale_y_continuous(limits = c(0,5))
#### Combine value and volume sales graphs
gridExtra::grid.arrange(p.top_prod_sales, p.top_prod_rating, ncol = 2,
                        widths = c(1.1, 0.5),
                        top = ggpubr::text_grob("Top 10 Products and Their Rating",
                                                size = 15, face = "bold"))
```

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

4.3 Spending by Membership Type

```
### Highest Spent based on Customer Segment
(membership_segmentation <-
    dbGetQuery(db_connection,
               "SELECT c.membership_type_id,
               m.membership_type,
               SUM(o.order_value) as total_spent,
               o.order_date AS date
               FROM customers c
               JOIN memberships m ON c.membership_type_id = m.membership_type_id
               JOIN orders d ON c.cust_id = d.cust_id
               JOIN order_details o ON d.order_id = o.order_id
               GROUP BY o.order_date, c.membership_type_id
               ORDER BY total_spent DESC"))
### Transform the data
membership_by_mnth_date <- membership_segmentation %>%
  mutate("month" = format(as.Date(date), "%m"),
         "year" = format(as.Date(date), "%Y")) %>%
  group_by(membership_type, year) %>%
  filter(year != 2024) %>%
  summarise(total_spend = sum(total_spent))

### Plot spending by membership type
p.membership <- ggplot(filter(membership_segmentation,
                              format(as.Date(date), "%Y") != 2024),
         aes(x = membership_type,
             y = total_spent)) +
  geom_bar(stat = "identity", show.legend = F) +
  labs(x = "Membership Type", y = "Total Spend (£)",
       subtitle = "Spending") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold")) +
  scale_y_continuous(labels = comma,
                     limits = c(0, 90000))

### Plot spending by membership type by year
p.membership_mnth <- ggplot(membership_by_mnth_date,
       aes(fill = year, y = total_spend,
           x = membership_type)) +
  geom_col(position = "dodge", color = "white") +
  labs(fill = "Year", x = "Membership Type", y = "Total Spend (£)",
       subtitle = "Spending by Year") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"),
        axis.title.y = element_blank(),
        #axis.text.y = element_blank(),
```

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

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

4.4 Customer Queries Analysis

```
### Most Frequent Queries - get data from db
(queries_frequencies <-
    dbGetQuery(db_connection,
               "SELECT query_title, COUNT(*) as frequencies
               FROM customer_queries
               GROUP BY query_title
               ORDER BY frequencies DESC"))

### Plot query types in terms of frequency
p.query_freq <- ggplot(queries_frequencies,
       aes(x= reorder(query_title, desc(frequencies)),
                                y = frequencies)) +
  geom_bar(stat = "identity") +
  labs(x = "Query Type", y = "Frequency",
       subtitle = "Frequency") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))

### Response Time Analyis for Customer Queries - get data from the db
(response_time <-
    dbGetQuery(db_connection,
               "SELECT query_id,
               query_title,
               query_closure_date,
               query_submission_date
               FROM customer_queries"))

### Transform data - get turnaround time
response_time <- filter(response_time, query_closure_date != "NA") %>%
```

```
  mutate(turnaround_time = round(difftime(query_closure_date, query_submission_date,
                                          units = "weeks"),0) ) %>%
  group_by(query_title) %>%
  summarise(avg_turnaround_time = round(mean(turnaround_time),1)) %>%
  merge(queries_frequencies, by = "query_title", remove = F)

### Plot query by response time
h_line <- mean(response_time$avg_turnaround_time)
p.query_time <- ggplot(response_time,
       aes(x= reorder(query_title, desc(frequencies)),
           y = avg_turnaround_time)) +
  geom_bar(stat = "identity") +
  geom_hline(yintercept = h_line,
             color = "magenta", linetype = "dashed", size = 1.1) +
  geom_text(aes(1, h_line, label = "Avg of all types"),
            vjust = -1, color = "magenta") +
  labs(x = "Query Type", y = "Avg Turnaround Time (weeks)",
       subtitle = "Turnaround Time") +
  theme_light()
### Combine frequency and turnaround time
gridExtra::grid.arrange(p.query_freq, p.query_time, ncol = 2,
                        top = ggpubr::text_grob("Customer Queries",
                                                size = 15, face = "bold"))
```

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

4.5 Payment Methods

```
### Get data from the db
(top_payment <-
    dbGetQuery(db_connection,
               "SELECT payment_method, COUNT(*) AS frequencies,
               SUM(payment_amount) AS pymnt_amnt
               FROM payments
               GROUP BY payment_method
               ORDER BY frequencies DESC"))
### Plot payment method by frequency
p.frequency <- ggplot(top_payment, aes(x= reorder(payment_method, desc(frequencies)),
                        y = frequencies)) +
  geom_bar(stat = "identity") +
  labs(x = "Payment Method", y = "Frequency",
       subtitle = "Frequently Used Payment Method") +
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold")) +
```

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

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

4.6 Insight and Recommendation

4.6.1 Insights:

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

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

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

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

4.6.2 Recommendation:

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

5 Limitation and Future Implications

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

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

6 Conclusion