Data Managment E-commerce Data Base Project

Group 8

Contents

1 Introduction **2**

2 SQL Script **2**
- 2.1 Importing Generated Data . . . 2
- 2.2 Category . . . 3
- 2.3 Supplier . . . 4
- 2.4 Review . . . 5
- 2.5 Order Details . . . 5
- 2.6 Discount . . . 5
- 2.7 Order Item . . . 5

3 Database Design and Implementation **6**
- 3.1 Entity Relationship Diagram . . . 6
- 3.2 Design Considerations . . . 8
- 3.3 Relationships and Cardinalities . . . 9
- 3.4 Logical Schema . . . 11

4 Part 2: Data Generation and Management **14**
- 4.1 Synthetic Data Generation . . . 14
- 4.2 Data Import and Quality Assurance . . . 14

5 Part 3: Data Pipeline Generation **14**

6 Part 4: Data Analysis and Reporting with Quarto in R **15**

7 Conclusion **18**

8 References **18**

9 Appendices **18**
- 9.1 Appendix 1 . . . 18

Load necessary libraries

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

```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(stringr)
```

1 Introduction

This project simulates a real-world e-commerce data environment, including all stages of data management, from database configuration to data analysis and reporting. As the database is created and managed with SQLite, GitHub Actions are used to automate processes enhancing workflow efficiency. In addition, Quarto is used in conjunction with R to conduct comprehensive data analysis and generate reports. Through this approach, technical skills and competencies necessary for effective data management are examined in-depth, mirroring the complexities and dynamics encountered in e-commerce.

2 SQL Script

2.1 Importing Generated Data

```r
con <- dbConnect(RSQLite::SQLite(), "ecommerce.db")

sql_file <- readLines("dbScript.sql")

for (sql_command in sql_file) {
  if (sql_command!=""){
    print(sql_command)
    dbExecute(con,sql_command)
    print("-------------DONE---------")
  }
}
```

```
[1] "CREATE TABLE IF NOT EXISTS 'Category'( 'Category_ID' VARCHAR(250) PRIMARY KEY, 'Category_Name' VAR
[1] "-------------DONE---------"
[1] "CREATE TABLE IF NOT EXISTS 'Suppliers'( 'Supplier_ID' VARCHAR(250) PRIMARY KEY, 'Supplier_Name' VA
[1] "-------------DONE---------"
[1] "CREATE TABLE IF NOT EXISTS 'Discounts'( 'Discount_Code' VARCHAR(50) PRIMARY KEY, 'Discount_Amount'
[1] "-------------DONE---------"
[1] "CREATE TABLE IF NOT EXISTS 'Products' ( 'Product_ID' VARCHAR(250) PRIMARY KEY, 'Product_Name' VARC
[1] "-------------DONE---------"
[1] "CREATE TABLE IF NOT EXISTS 'Customers' ( 'Cust_ID' VARCHAR(250) PRIMARY KEY, 'Cust_First_Name' VAR
[1] "-------------DONE---------"
```

```
[1] "CREATE TABLE IF NOT EXISTS 'Reviews' ( 'Review_ID' VARCHAR(50) PRIMARY KEY,'Review_Timestamp' DATE
[1] "-------------DONE---------"
[1] "CREATE TABLE IF NOT EXISTS 'Order_Items'( 'Quantity' INT NOT NULL, 'Sum_Price' INT NOT NULL, 'Orde
[1] "-------------DONE---------"
[1] "CREATE TABLE IF NOT EXISTS 'Order_Details' ( 'Order_ID'  VARCHAR(250) PRIMARY KEY, 'Order_Date' DA
[1] "-------------DONE---------"
```

```
# con <- dbConnect(RSQLite::SQLite(), "ecommerce.db")
#
# # con <- dbConnect(RSQLite::SQLite(), "ecommerce.db")
#
#
# Customers <- read_csv("Files/Customers.csv")
#
#
# Products <- read_csv("Files/Products.csv")
#
#
# Order_details <- read_csv("Files/Order_Details.csv",
#     skip = 1)
#
# Reviews <- read_csv("Files/Reviews.csv",
#     skip = 1)
#
# Suppliers <- read_csv("Files/Suppliers.csv")
#
#
# Product_Discounts <- read_csv("Files/Product_Discounts.csv")
#
#
# Product_Category <- read_csv("Files/Product_Category.csv")
#
#
# Order_Item <- read_csv("Files/Order_Item.csv",
#     skip = 1)
```

2.2 Category

```
# To create empty column for the Products table
# Products <- Products %>%
#   mutate(Category_ID = NA)
#
# # To apply the foreign key into the table
# # Define a function to assign Category_ID based on keywords in Product_Name
# assign_category_id <- function(Product_Name) {
#   if (grepl("TV|Television", Product_Name, ignore.case = TRUE)) {
#     return("CAT1")
#   } else if (grepl("Laptop|Tablet|Computing|Book|Surface|Monitor", Product_Name, ignore.case = TRUE))
#     return("CAT2")
#   } else if (grepl("Phone|Galaxy|Mi|P Series|OnePlus", Product_Name, ignore.case = TRUE)) {
#     return("CAT3")
#   } else if (grepl("Refrigerator|Washing Machine|Home Appliance|Microwave|Vacuum|Dishwasher", Product_
#     return("CAT4")
#   } else if (grepl("Headphones|Speakers|Sound System|Earbuds|Speaker|Technica|Soundbar", Product_Name
#     return("CAT5")
#   } else if (grepl("Camera|Photography|GoPro|Mirrorless|Nikon|Camcorder|Compact", Product_Name, ignor
#     return("CAT6")
#   } else if (grepl("Xbox|PS|Gaming|Switch", Product_Name, ignore.case = TRUE)) {
#     return("CAT7")
#   } else if (grepl("Smart Home|Echo|Smart Lock|Steam Deck|Hue Light", Product_Name, ignore.case = TRU
#     return("CAT8")
#   } else if (grepl("Watch|Wearable|Quest|Tracker|Gear|Band|Glasses", Product_Name, ignore.case = TRUE
#     return("CAT9")
#   } else if (grepl("Keyboard|Mouse|Peripheral|Thermostat", Product_Name, ignore.case = TRUE)) {
#     return("CAT10")
#   } else {
#     return(NA) # For products that do not match any category
#   }
# }
#
# # Apply the function to assign Category_ID to each product
# Products$Category_ID <- sapply(Products$Product_Name, assign_category_id)
```

2.3 Supplier

```
# # This is to add suppliers_id in Products
# set.seed(123)
#
# Products <- Products %>%
#   mutate(Supplier_ID = NA)
#
# # Create a function to find matching supplier ID or assign randomly if no match is found
# assign_supplier_id <- function(Product_Name, Suppliers) {
#   for (i in 1:nrow(Suppliers)) {
#     if (str_detect(Product_Name, regex(Suppliers$Supplier_Name[i], ignore_case = TRUE))) {
#       return(Suppliers$Supplier_ID[i])
#     }
#   }
#   # If no match found, assign a random supplier ID
#   random_supplier_id <- sample(Suppliers$Supplier_ID, 1)
#   return(random_supplier_id)
# }
#
# Products$Supplier_ID <- sapply(Products$Product_Name, function(x) assign_supplier_id(x, Suppliers))
#
# # Adding Discount_Code column into Products
# set.seed(123) # This is to ensure reproducibility
#
# Products <- Products %>%
#   mutate(Discount_Code = NA)
#
# codes_to_assign <- sample(1:nrow(Products), 50)
#
# random_discounts <- sample(Product_Discounts$Discount_Code, 50)
#
# Products$Discount_Code[codes_to_assign] <- random_discounts
```

2.4 Review

```
# # Product_ID column for reviews table
# set.seed(123)
# Reviews <- Reviews %>%
#   mutate(Product_ID = sample(Products$Product_ID, nrow(Reviews), replace = TRUE))
```

2.5 Order Details

```
# # Adding Cust_ID column for Order_details table.
# set.seed(123)
# Order_details <- Order_details %>%
#   mutate(Cust_ID = sample(Customers$Cust_ID, nrow(Order_details), replace = TRUE))
```

2.6 Discount

```
# # Filter out the rows from Products that have a Disocunt_Code assigned
# discounted_products <- Products %>%
#   filter(!is.na(Discount_Code)) %>%
#   select(Product_ID, Discount_Code)
#
# # Do a left join to join it together
# Product_Discounts <- Product_Discounts %>%
#   left_join(discounted_products, by = "Discount_Code")
#
# # Same step for cat_id
# # Filter out the rows from Products that have a discount code assigned on the cat ID
# discounted_cat <- Products %>%
#   filter(!is.na(Category_ID)) %>%
#   select(Category_ID, Discount_Code)
#
# # Do a left join to join it together, thus we get to see which discount code assign to which category
# Product_Discounts <- Product_Discounts %>%
#   left_join(discounted_cat, by = "Discount_Code")
```

2.7 Order Item

```
# # Order_Items, product_ID
# set.seed(123)
# Order_Item <- Order_Item %>%
#   mutate(Product_ID = NA)
#
# # Assign first 150 unique Product_IDs to the first 150 rows
# Order_Item$Product_ID[1:150] <- sample(Products$Product_ID, size = 150, replace = FALSE)
#
# # For the remaining 50 rows, randomly assign Product_IDs (allowing repeats)
# Order_Item$Product_ID[151:200] <- sample(Products$Product_ID, size = 50, replace = TRUE)
#
# # Joining Order_Item with Products to get the Price for each Product_ID
# Order_Item <- merge(Order_Item, Products[, c("Product_ID", "Product_Price")], by = "Product_ID", all.
#
# Order_Item <- Order_Item %>% rename(Quantity = Order_Item)
# # Calculating Sum_Price as Price * Quantity
# Order_Item <- Order_Item %>%
#   mutate(Sum_Price = Product_Price * Quantity)
#
# # Remove the Product_Price column
# Order_Item$Product_Price <- NULL
```

```
# set.seed(123)
# Order_Item <- Order_Item %>%
#   mutate(Order_ID = NA)
#
# # Assign first 150 unique Order_IDs to the first 150 rows
# Order_Item$Order_ID[1:150] <- sample(Order_details$Order_ID, size = 150, replace = FALSE)
#
# # Function to assign unique Order_IDs avoiding duplicates for each Product_ID, it intended to assign
# assign_unique_order_ids_full_range <- function(order_item, order_details) {
#   # Get all unique Order_IDs from Order_Details
#   all_order_ids <- unique(Order_details$Order_ID)
#
#   # Iterate over each row in order_item
#   for (i in 151:nrow(Order_Item)) { # The first 150 are pre-assigned
#     product_id <- Order_Item$Product_ID[i]
#
#     # Find Order_IDs used by the same Product_ID
#     used_order_ids <- Order_Item$Order_ID[Order_Item$Product_ID == product_id]
#
#     # Available Order_IDs are those not yet used by this Product_ID
#     available_order_ids <- setdiff(all_order_ids, used_order_ids)
#
#     if (length(available_order_ids) == 0) {
#       stop("Ran out of unique Order_IDs to assign for Product_ID: ", product_id)
#     }
#
#     # Randomly select an available Order_ID for the Product_ID
#     Order_Item$Order_ID[i] <- sample(available_order_ids, 1)
#   }
#
#   return(Order_Item)
# }
#
# <!-- # To apply this function: -->
# Order_Item <- assign_unique_order_ids_full_range(Order_Item, Order_details)


# Disconnect
# dbDisconnect(con)
```

3 Database Design and Implementation

3.1 Entity Relationship Diagram

The E-R diagram above simulates a real-world e-commerce data ecosystem, capturing the detailed relationships between entities and attributes essential for facilitating online transactions. In addition, it provides a

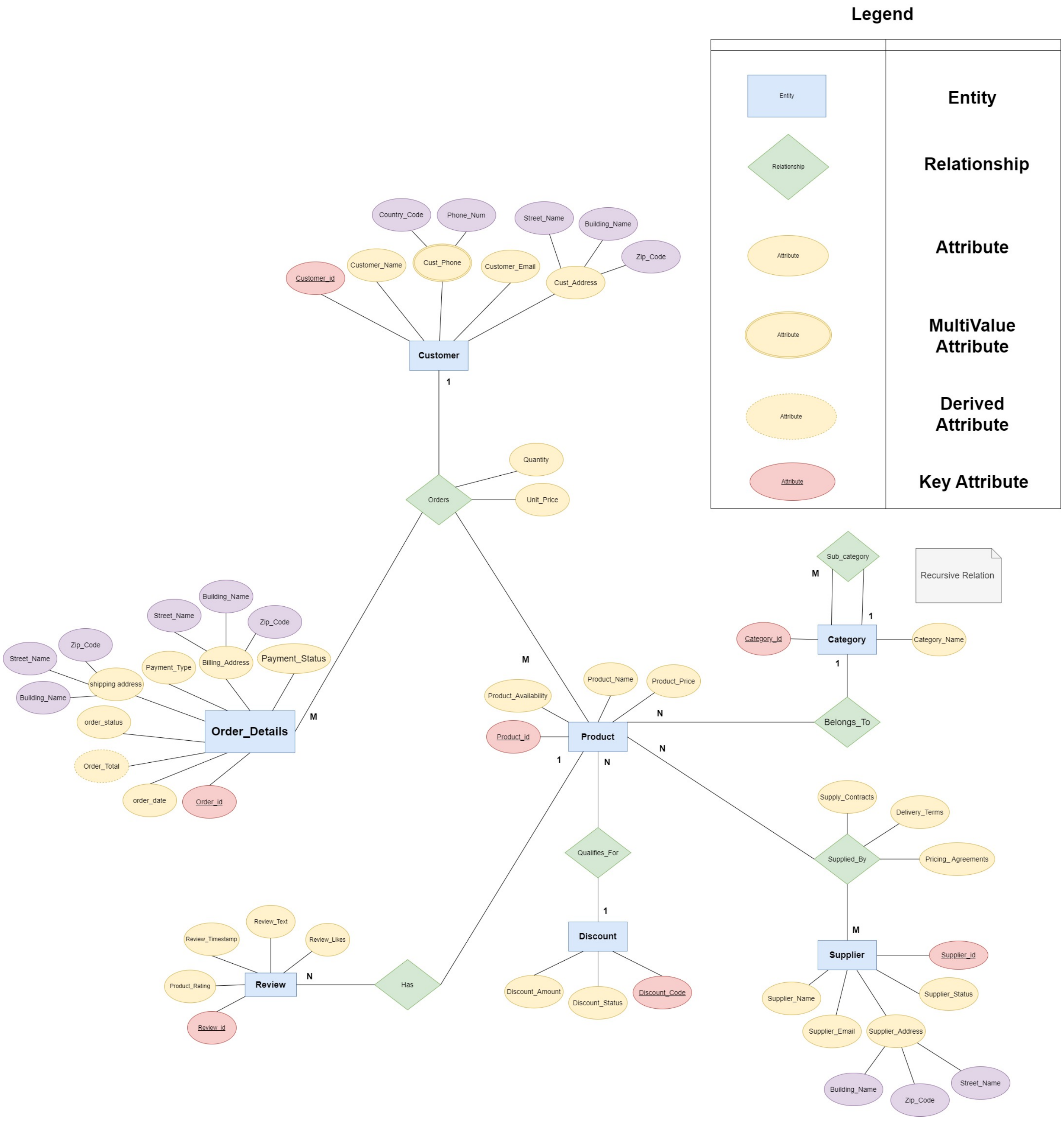

Legend
Entity
Entity
Relationship
Relationship
Attribute
Attribute
Attribute
MultiValue Attribute
Attribute
Derived Attribute
Attribute
Key Attribute
Country_Code
Phone_Num
Street_Name
Building_Name
Zip_Code
Customer_id
Customer_Name
Cust_Phone
Customer_Email
Cust_Address
Customer
1
Quantity
Orders
Unit_Price
Sub_category
M
1
Recursive Relation
Category_id
Category
Category_Name
1
Building_Name
Street_Name
Zip_Code
Zip_Code
Street_Name
Payment_Type
Billing_Address
Payment_Status
shipping address
Building_Name
order_status
Order_Details
M
Order_Total
order_date
Order_id
M
Product_Name
Product_Price
Product_Availability
N
Product_id
Product
Belongs_To
1
N
N
Supply_Contracts
Delivery_Terms
Supplied_By
Pricing_Agreements
Qualifies_For
1
Discount
M
Supplier
Supplier_id
Review_Text
Review_Timestamp
Review_Likes
Product_Rating
Review
N
Has
Review_id
Discount_Amount
Discount_Status
Discount_Code
Supplier_Name
Supplier_Status
Supplier_Email
Supplier_Address
Street_Name
Building_Name
Zip_Code

Figure 1: ERD

comprehensive view of the e-commerce system, which serves as a platform for users to browse products, make purchases, and securely complete their payments.

3.1.1 Assumptions

- The company only distributes products within the United Kingdom (UK).
- The Currency used is Pound Sterling (GBP).
- Attributes formats will be aligned with UK standard formats such as date , addresses , names . . . etc

3.1.2 Entities and Attributes

This section describes and illustrates the entities in the above ERD and their respective attributes.

3.1.2.1 Customer Shows us the users who previously have at least once purchased products and placed an order including information about their names , emails, and addresses.

3.1.2.2 Supplier Vendors who provide products. Represent the source of the product items.The entity store information about their names , addresses, emails, and status that indicates Whether the supplier is currently supplying items or not (Active/Inactive).

3.1.2.3 Product Describes all products in the stock and available for sale. Provides information about the model, price, and availability of the products.

3.1.2.4 Order_Details Emphasises all details related to placed orders including billing, shipping address, order, payment status, order date, and payment type.

3.1.2.5 Category and Sub-Category Category is the broad classification of products that share common features or are intended for a similar purpose. A sub-category is a more specific grouping of products within a category based on finer distinctions or attributes.

Sub-categories fall under a primary category and help to further organize products into narrower groups, making the product search process even more straightforward for customers.

3.1.2.6 Product_Discounts The voucher number or offer code to be applied to eligible products. The amount of discount it offers as well as the status of the discount are the main attributes.

3.1.2.7 Reviews Contains Written comments and rating of product sold by verified buyers, the likes of the top reviews as well as the time stamp of when the review was made.

3.2 Design Considerations

3.2.1 Absence of an Order Entity

The model intentionally skips direct order management. Instead, it focuses on product management and customer interactions through reviews and payment methods.Additionally, This consideration will guarantee that products purchased by customers are not tracked or stored by the system to align with privacy policies.

Order Entity not considered in this ER design in order to follow best practices by not having to include orderId as part of product table which might affect the overall performance of DB retrieval.

Customer Engagement: By including Reviews, the model emphasizes customer engagement and feedback without directly managing transactions.

Payment Information: Including Payment_Method without an Order entity suggests a pre-registration of payment preferences or a simplified wallet storage that could be expanded in the future.

3.3 Relationships and Cardinalities

3.3.1 Customer Orders Products

A Customer initiates an Order when they purchase products or services. It is considered for customer management, processing transactions, and tracking order history. One customer can place multiple orders over time, each uniquely associated with one customer.

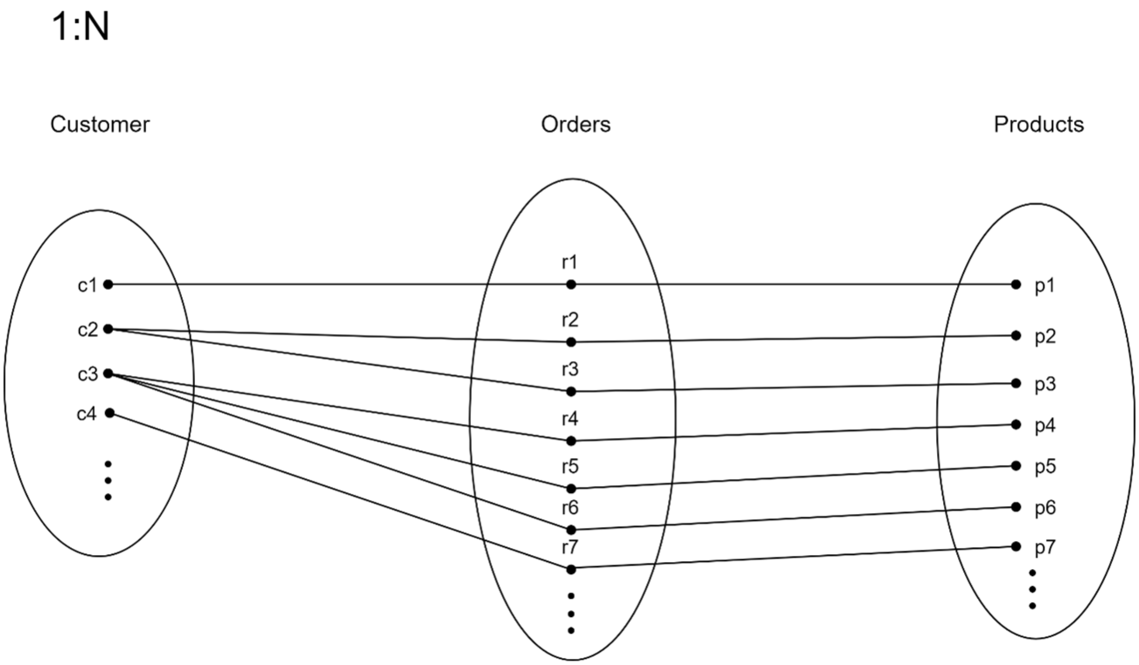


Figure 2: RMS

Associative Attributes: (Quantity: The number of units of the product ordered in this line item.), (Unit Price: The price per unit of the product at the time of the order. This is important as product prices can vary over time), (Unit_Sub_Total : The total cost for this line item (typically calculated as (Quantity * Unit Price)).

3.3.2 Customer Has Order Details

This relation will be created when customers order their first product or service. They will be linked with a particular Order Status indicating what they ordered, reflecting the current state or progression throughout the process. One customer can be associated with multiple order statuses at any given time. Moreover, it is good for tracking an order's life cycle, allowing for updates, customer notifications, and management of the order fulfillment process.

3.3.3 Product Belongs to Category

Each Product is classified under a specific Category where products can belong to only one category. This enables customers to browse products by category and helps retailers manage product listings more efficiently.

3.3.4 Category Self-Reference Relation

A category can have multiple subcategories, creating a hierarchically nested structure and making it easier for users to navigate the product catalogue. For example, the "Phones" category might have "Apple" and "Samsung" as subcategories, which in turn could have their own subcategories of different phone models.

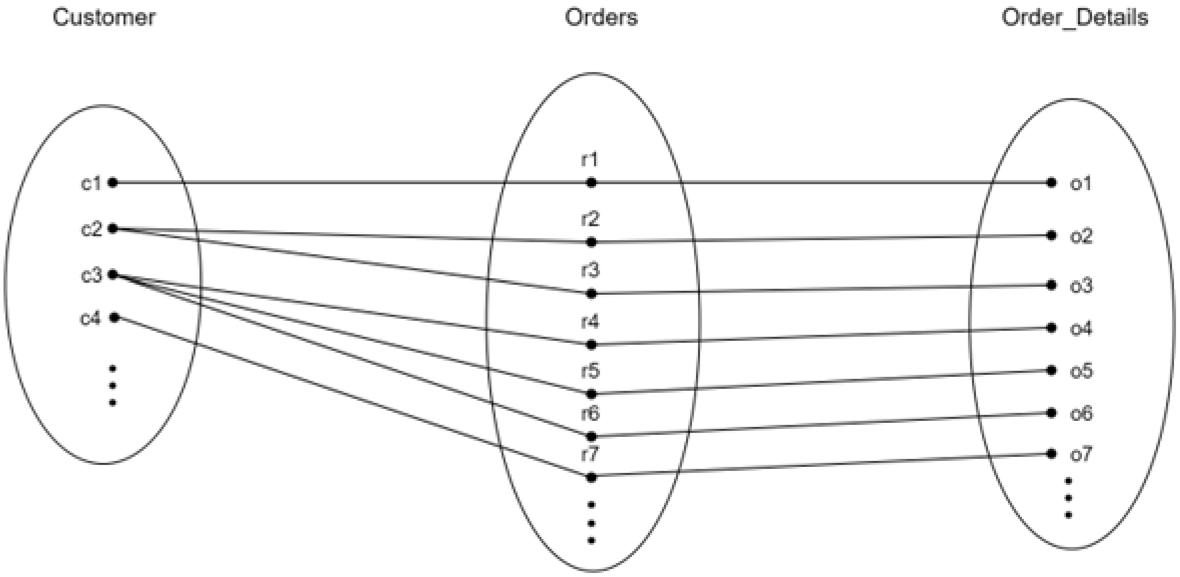


Figure 3: RMS

M:1

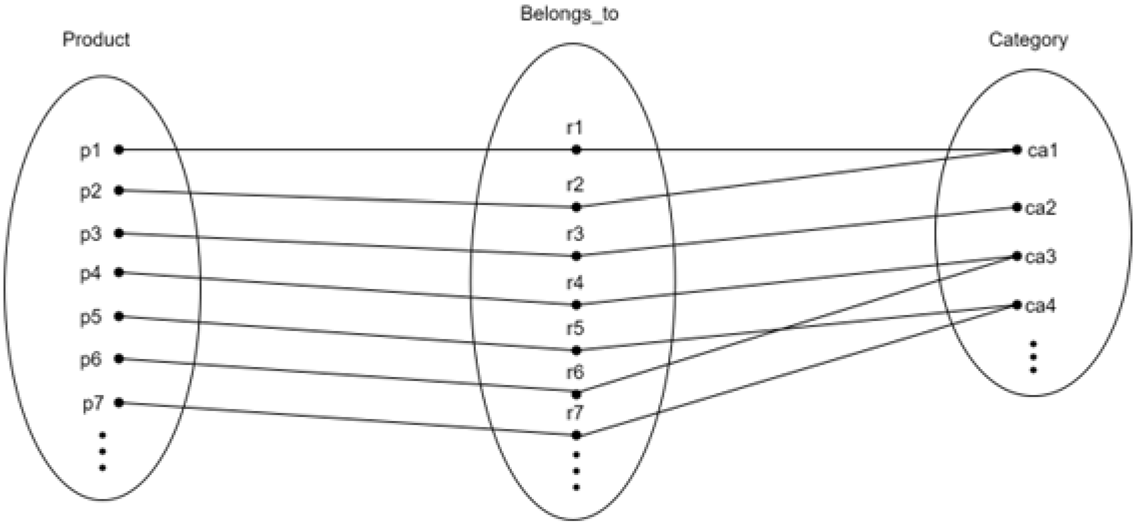


Figure 4: RMS

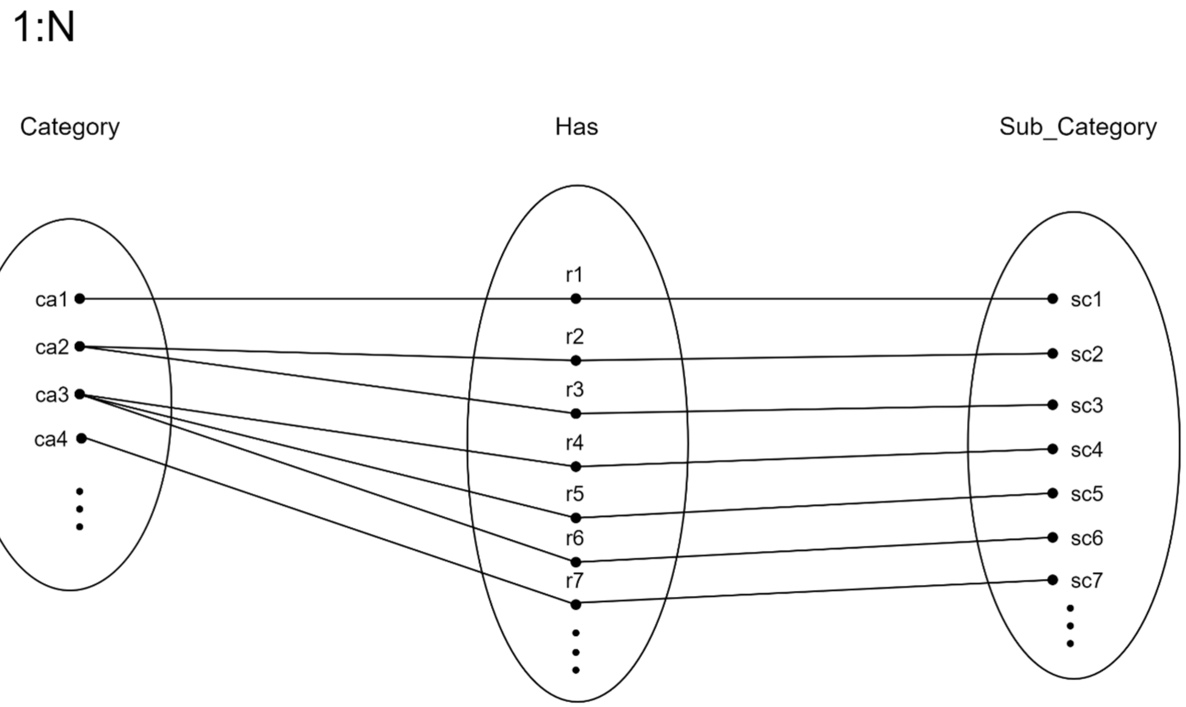


Figure 5: RMS

3.3.5 Product Supplied_By Supplier

The relationship creates a link between the products and their suppliers. Thereby indicating multiple vendors can supply a product, as well as supply multiple different products. The relation helps track inventory sources, manage supplier relationships, and ensure product availability.

3.3.6 Product Qualifies_For Discount

The relation signifying that the product is eligible for certain promotional discount enabling dynamic pricing strategies, encouraging sales, and providing customers with various savings opportunities on different products. In this context and for simplicity the relation representing one discount code or voucher that is valid to apply on multiple eligible products.

3.3.7 Product Has Reviews

A product's reviews are generated or provided by Customers reflecting the action of providing feedback or evaluation for a specific product or service to improve product offerings and customer service. However, one product can have multiple reviews over time.

3.4 Logical Schema

3.4.1 Customers

Customers (Cust_ID, Cust_Email, Cust_First_Name, Cust_Last_Name, Cust_Country_Code, Cust_Phone_Number, Cust_Street_Name, Cust_Building_Name, Cust_Zip_Code)

3.4.2 Products

Products (Product_id, *Discount_Code*, *Category_id*, Product_Name, Product_Price, Product_Availability)

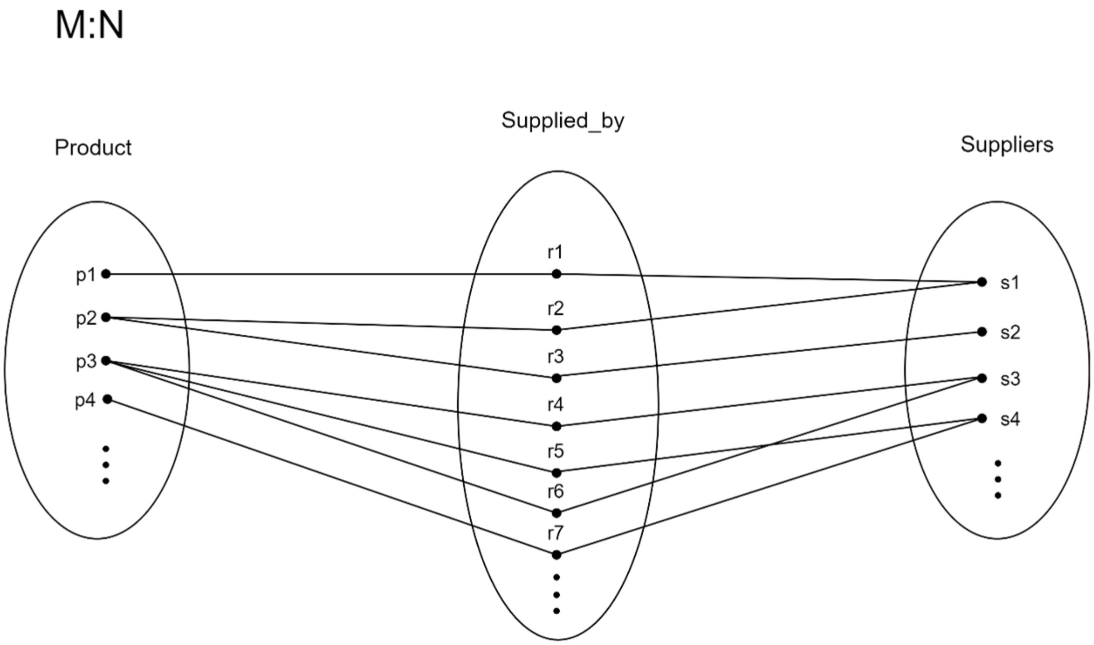


Figure 6: RMS

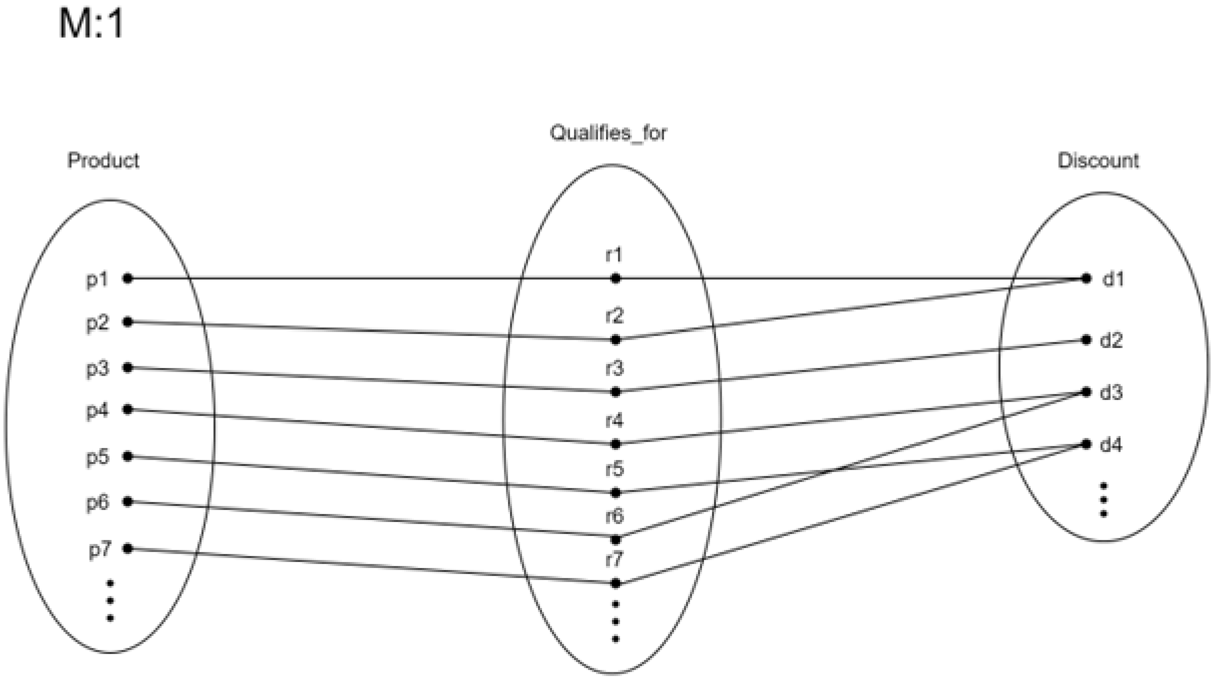


Figure 7: RMS

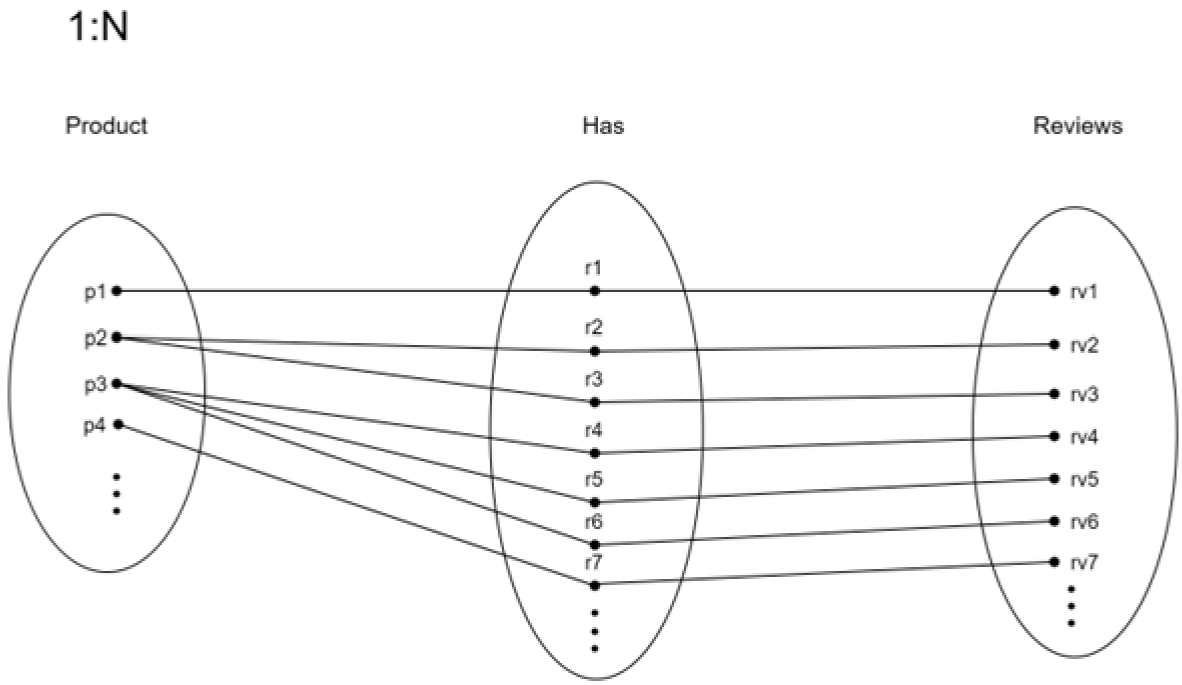


Figure 8: RMS

3.4.3 Suppliers

Suppliers ($\underline{Supplier_id}$, Supplier_Email, Supplier_Name, Supplier_Status, Sup_Building_Name, Sup_Street_Name, Sup_Zip_Code)

3.4.4 Order_Details

Order_Details ($\underline{Order_id}$, *Customer_id*, Order_Date, Order_Total, Order_Status, S_Building_Name, S_Street_Name, S_Zip_Code, Street_Name, B_Building_Name, B_Street_Name, B_Zip_Code, Payment_Type, Payment_Status)

3.4.5 Discounts

Discounts ($\underline{Discount_Code}$, Discount_Status, Discount_Amount)

3.4.6 Reviews

Reviews ($\underline{Review_id}$, *Product_id*, Review_Rating, Review_Timestamp, Review_Text, Review_Likes)

3.4.7 Categories

Categories ($\underline{Category_id}$, Category_Name)

3.4.8 SupplierProduct

SupplierProduct ($\underline{Supplier_id, Product_id}$, Supply_Contracts, Delivery_Terms, Pricing_Agreements)

3.4.9 Many To Many : Supplier - Product

SupplierProduct ($\underline{Supplier_id, Product_id}$)

3.4.10 Many to Many : Order_details - Product

ProductOrder_details ($\underline{Order_id, Product_id}$, Quantity, Unit_Price)

4 Part 2: Data Generation and Management

4.1 Synthetic Data Generation

After the agreement on the schema mentioned in the previous section, the team started to generate synthetic data that to some extent, imitated realistic e-commerce as much as possible.

ChatGPT has been used as the main tool for this step as an alternative to Mockaroo, as the former produces more structural and logical data than the latter. [**Appendix 1,2**]

4.2 Data Import and Quality Assurance

The process was done manually to the very few cells that still don't make any sense related to e-commerce context or were left blank and AI tools missed to produce.

5 Part 3: Data Pipeline Generation

The pipeline generation process consists of two phases. In the first phase, github repository has been created[1] followed by integrating the posit cloud project. Additionally, all teams members have been added as collaborators.

Second phase was the automation process utilising workflow that trigger on specific events like push and pull requests to perform data updates, validation, and analysis. This will pick up any new data added to the data base and subsequently perform the required validation and analysis as illustrated in Figures.

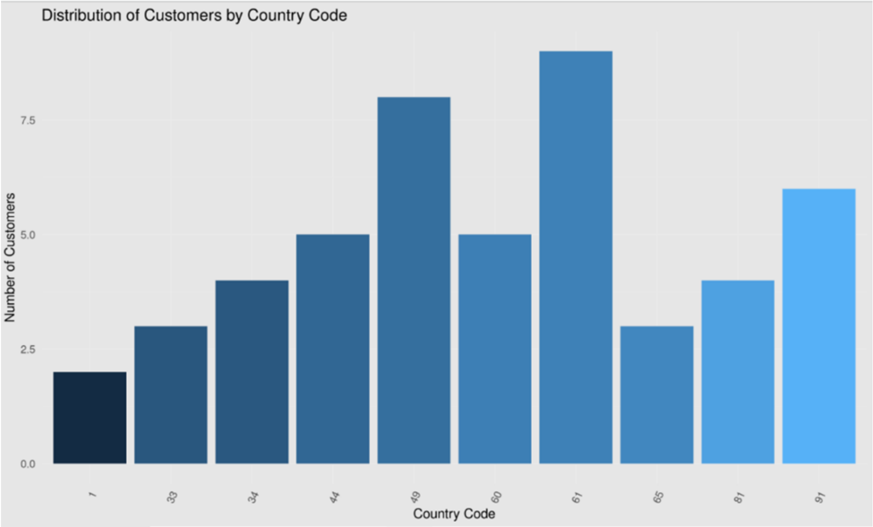
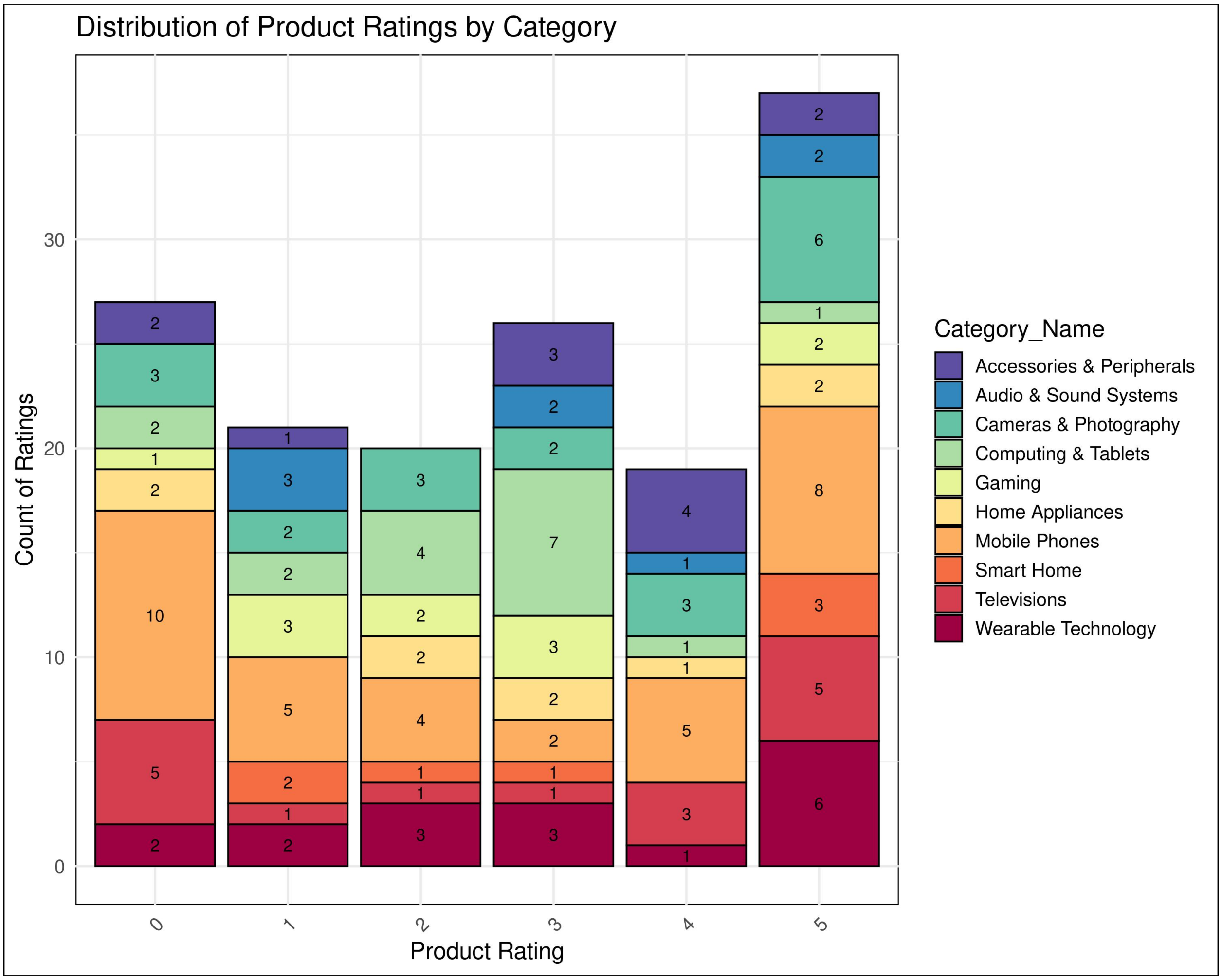


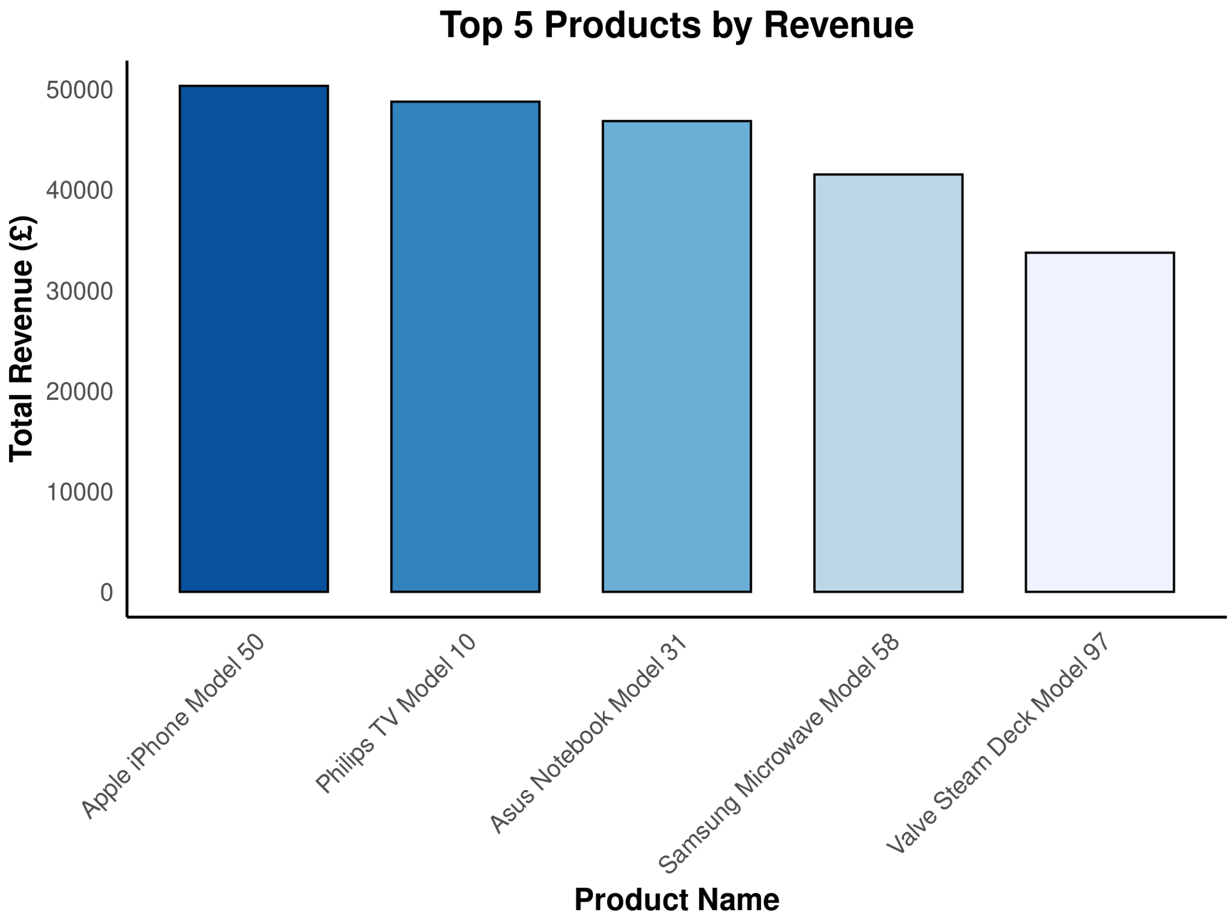
Figure 9: WorkFlow

```
# We need to rename and delete columns like building number, as it does not match or does not exist in
# Suppliers amendment
# Suppliers$Supplier_Building_Number <- NULL
# Suppliers <- Suppliers %>% rename(Supplier_Zip = Supplier_Zip_Code)

# Customers amendment
# Customers <- Customers %>% rename(Cust_Zip = Cust_Zip_Code)
# Customers <- Customers %>% rename(Cust_Country_Code = Phone_Country_Code)


# Order_details
# Don't have these in the database
# Order_details$Billing_Building_Number <- NULL
# Order_details$Shipping_Building_Number <- NULL

#This is one of the ways of doing it , havent do order_details pending for changes from abigail.
# RSQLite::dbWriteTable(con, "Category", Product_Category, overwrite=TRUE)
# RSQLite::dbWriteTable(con, "Suppliers", Suppliers, overwrite=TRUE)
# RSQLite::dbWriteTable(con, "Discounts", Product_Discounts, overwrite=TRUE)
#
#
# RSQLite::dbWriteTable(con, "Reviews", Reviews, overwrite=TRUE)
# RSQLite::dbWriteTable(con, "Order_Items", Order_Item, overwrite=TRUE)
# RSQLite::dbWriteTable(con, "Order_Details", Order_details, overwrite=TRUE)
```

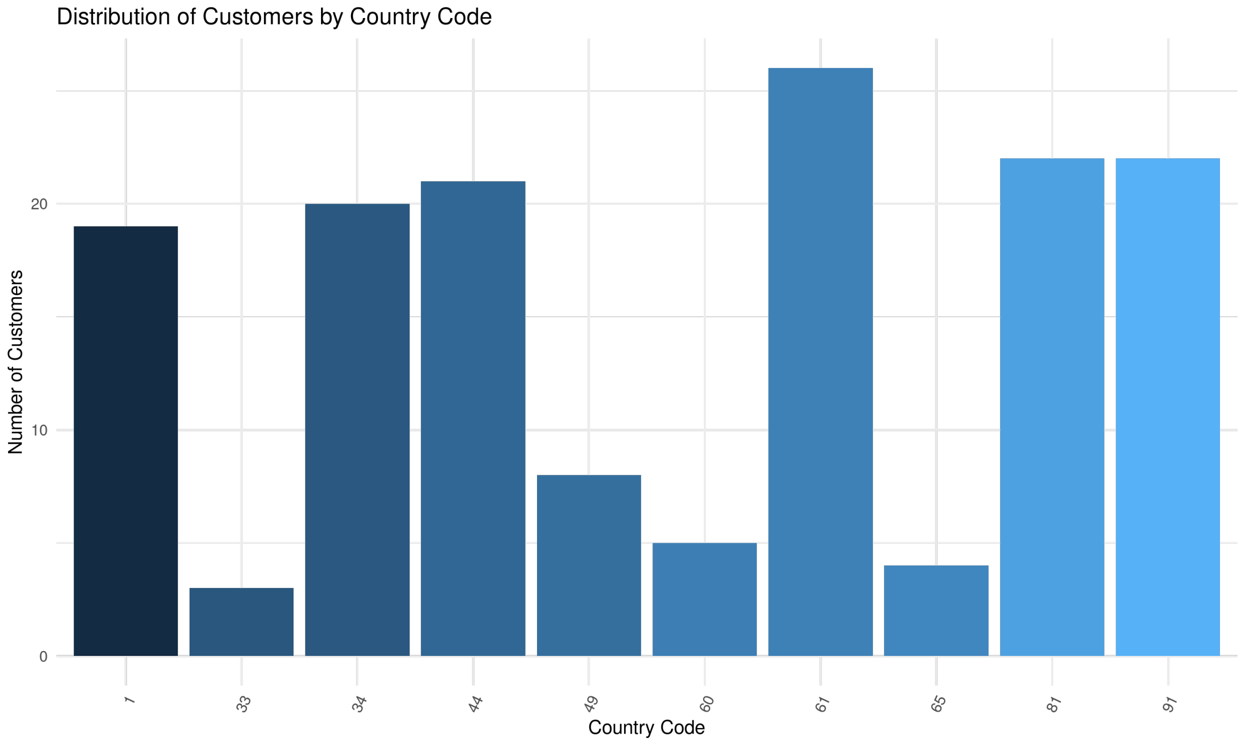
6 Part 4: Data Analysis and Reporting with Quarto in R



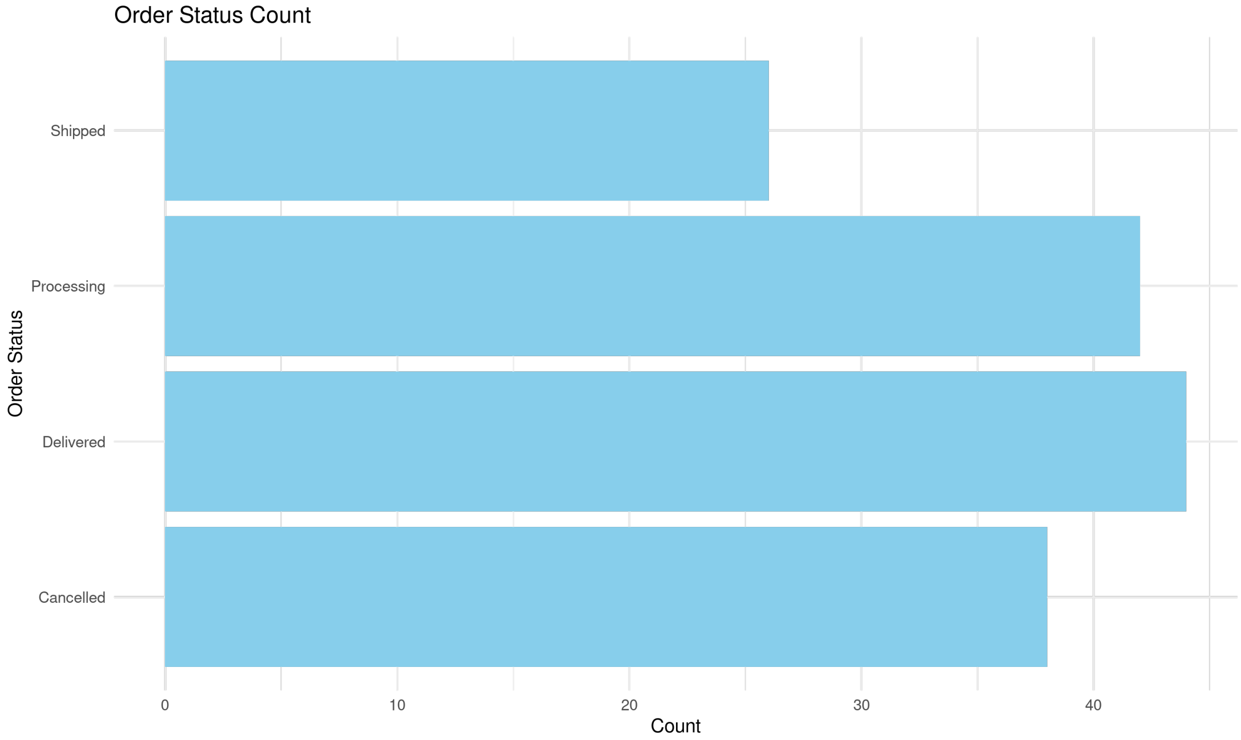
The graph above shows the distribution of product ratings from 0 to 5 coloured by category name.



The graph above shows the top 5 products in terms of revenue.



The graph above shows the distribution of customers by country code.



The graph above shows the order status count of shipped, processing, delivered and cancelled orders.

7 Conclusion

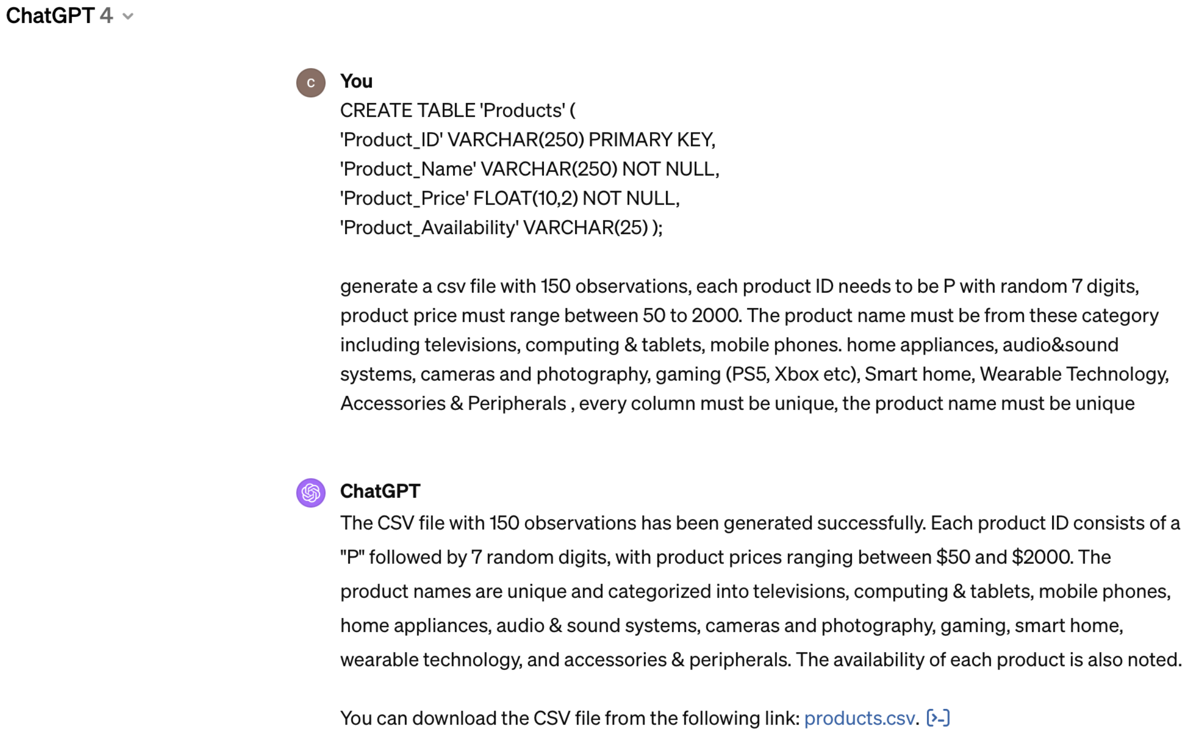
Upon completion of this project, valuable insights are gained regarding the challenges and methodologies associated with an e-commerce context, including the design of databases, analysis of data, and presentation of findings in a clear and impactful manner. Ultimately, this project serves as a reference for future e-commerce projects in particular and data-driven projects in general, providing valuable insight into the modern data management landscape.

8 References

[1] Github Repo

9 Appendices

9.1 Appendix 1



You

```
CREATE TABLE 'Products' (
'Product_ID' VARCHAR(250) PRIMARY KEY,
'Product_Name' VARCHAR(250) NOT NULL,
'Product_Price' FLOAT(10,2) NOT NULL,
'Product_Availability' VARCHAR(25) );
```

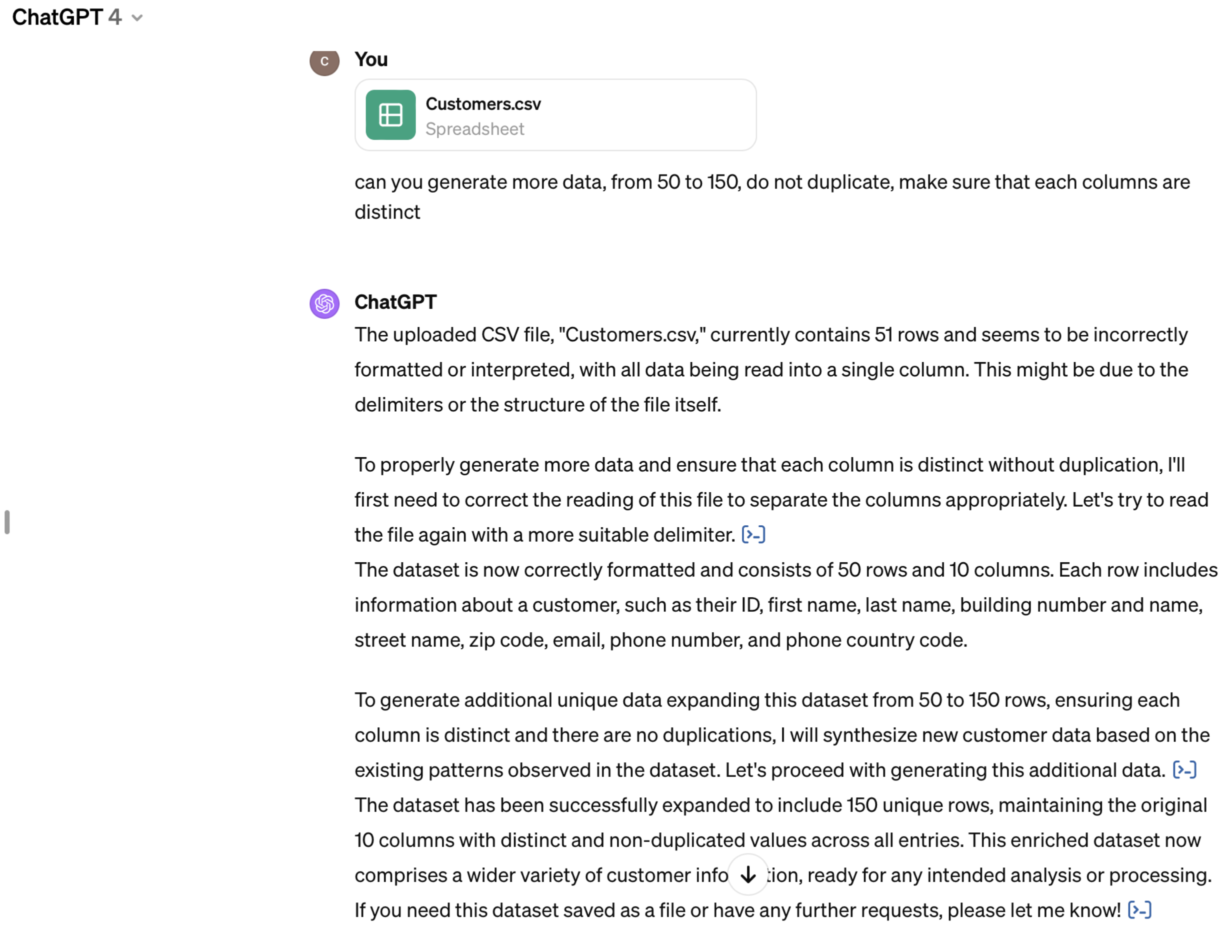
generate a csv file with 150 observations, each product ID needs to be P with random 7 digits, product price must range between 50 to 2000. The product name must be from these category including televisions, computing & tablets, mobile phones. home appliances, audio&sound systems, cameras and photography, gaming (PS5, Xbox etc), Smart home, Wearable Technology, Accessories & Peripherals , every column must be unique, the product name must be unique

ChatGPT

The CSV file with 150 observations has been generated successfully. Each product ID consists of a "P" followed by 7 random digits, with product prices ranging between $50 and $2000. The product names are unique and categorized into televisions, computing & tablets, mobile phones, home appliances, audio & sound systems, cameras and photography, gaming, smart home, wearable technology, and accessories & peripherals. The availability of each product is also noted.

You can download the CSV file from the following link: products.csv.

## Appendix 2



can you generate more data, from 50 to 150, do not duplicate, make sure that each columns are distinct

The uploaded CSV file, "Customers.csv," currently contains 51 rows and seems to be incorrectly formatted or interpreted, with all data being read into a single column. This might be due to the delimiters or the structure of the file itself.

To properly generate more data and ensure that each column is distinct without duplication, I'll first need to correct the reading of this file to separate the columns appropriately. Let's try to read the file again with a more suitable delimiter.

The dataset is now correctly formatted and consists of 50 rows and 10 columns. Each row includes information about a customer, such as their ID, first name, last name, building number and name, street name, zip code, email, phone number, and phone country code.

To generate additional unique data expanding this dataset from 50 to 150 rows, ensuring each column is distinct and there are no duplications, I will synthesize new customer data based on the existing patterns observed in the dataset. Let's proceed with generating this additional data.

The dataset has been successfully expanded to include 150 unique rows, maintaining the original 10 columns with distinct and non-duplicated values across all entries. This enriched dataset now comprises a wider variety of customer info↓tion, ready for any intended analysis or processing. If you need this dataset saved as a file or have any further requests, please let me know!

Figure 10: ChatGPT