Group-10

Table of contents

Introduction

This report reveals the database system of an e-commerce company called "Papa's Collection" and explains its automated integration inside GitHub. The system is designed with a comprehensive database structure that comprises seven crucial entities. Each entity is crafted with various attributes and relationships to facilitate an efficient, user-friendly experience. Hence, this report will focus on database design, schema creation, data generation, pipeline, and advanced data analysis.

Part 1: Database Design and Implementation

1.1 E-R Diagram

Entities & Attributes

Our e-commerce database has the following entities: customer, product, category, supplier, order detail, transaction, and ad. The E-R diagram can be seen in Figure 1.

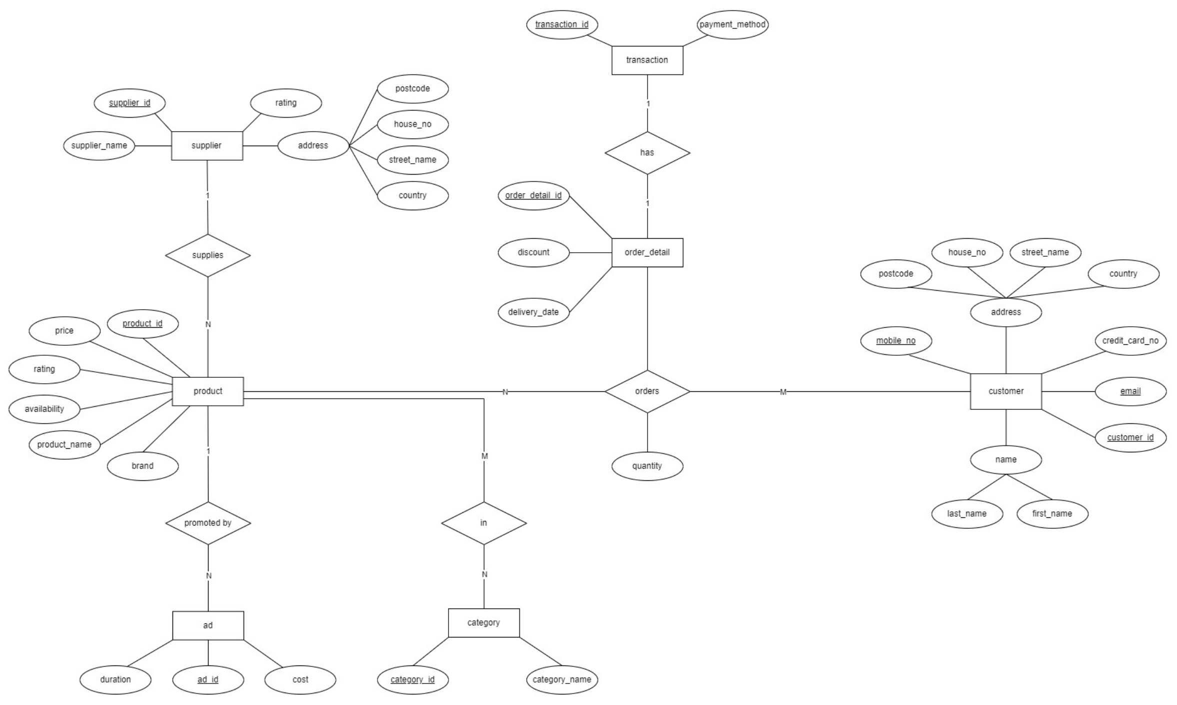


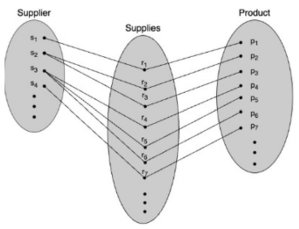
Figure 1: ER Diagram

- For the customer entity, each customer has a unique customer ID as its primary key, customer name, unique mobile number, home address, credit card number, and unique email address. The name attribute is a composite of two attributes named first name and last name. The address attribute is also a composite of four attributes: postcode, house no, street name, and country. We assume that customers can only save one address and one credit card number simultaneously.
- For the product entity, each product has a unique product ID as its primary key, product name, price, brand, rating, and availability. Product entity has one foreign key, supplier ID, denoting which supplier it is sold by.
- For the category entity, each category has a unique category ID as its primary key and category name.
- For the order detail entity, each order detail has a unique order detail ID as its primary key, discount, and delivery date. The order detail entity has one foreign key, transaction ID, to ensure order detail can be correctly traced back to their related transactions.
- For the supplier entity, each supplier has a unique supplier ID as its primary key, supplier name, address, and rating. The address attribute is also a composite of four attributes: postcode, house no, street name, and country.
- For the transaction entity, each transaction has a unique transaction ID as its primary key and payment method. The transaction entity has one foreign key order detail ID, which means each transaction will correspond to one or more order details.
- For the ad entity, each ad has a unique ad ID as its primary key, duration, and cost. The ad entity has one foreign key, product ID, showing that each ad promotes a specific product.

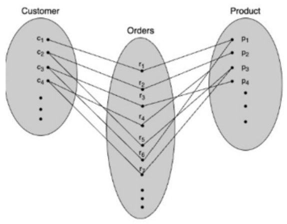
Relationships

In an e-commerce data environment, various relationships are essential for showing interactions. The relationships between different entities in our database are as follows:

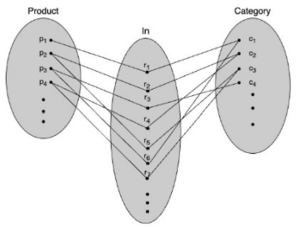
1. Each supplier can supply multiple products, but only one supplier can provide each product (1 to N).



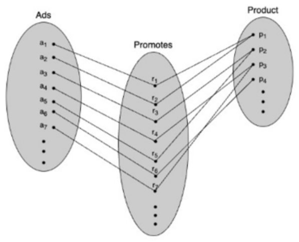
2. Each customer can order multiple products, and each product can be bought by different customers (M to N).



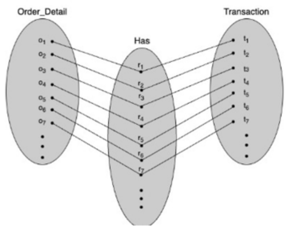
3. Each category can have multiple products, and each product can be under multiple categories, such as blender under kitchen and electrical appliances (M to N).



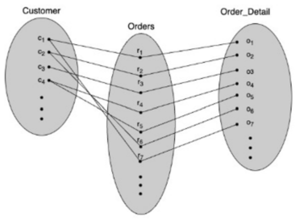
4. Each product can have multiple ads, but an ad can be for a sole product (1 to N).



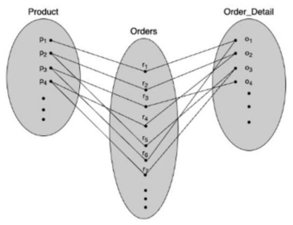
5. Each transaction can only have one order detail, and each order detail only belongs to one transaction (1 to 1).



6. Each customer can have multiple order details, but each order detail can only be under one customer (1 to N).



7. Each product can be within multiple order details, and each order detail can have many products (M to N).



1.2 SQL Database Schema Creation

Logical Schema

The logical schema given below outlines the fundamental components of our e-commerce database: customer, product, category, supplier, ad, order, order detail and transaction, as well as their attributes and relationship tables. This is the textual description of our database tables and their columns:

1. supplier (supplier_id (PK), supplier_name, rating, country, street_name, house_no, postcode)
2. customer (customer_id (PK), email, first_name, last_name, mobile_no, credit_card_no, country, street_name, house_no, postcode)
3. product (product_id (PK), supplier_id (FK), product_name, brand, price, rating, availability)
4. ad (ad_id (PK), product_id (FK), duration, cost)
5. category (category_id (PK), category_name)
6. transaction (transaction_id (PK), order_detail_id (FK), payment_method)
7. order_detail (order_detail_id (PK), transaction_id (FK), delivery_date, discount)
8. joint_in (product_id (FK), category_id (FK))
9. joint_order (order_detail_id (FK), customer_id (FK), product_id (FK), quantity)

Physical Schema

Using the logical schema, the physical schema is generated using RSQLite in R. It can be seen as follows:

```r
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(tidyr)
library(fakir)
library(charlatan)
library(generator)
library(RSQLite)
library(ggplot2)


# Connecting to database
schema_db <- RSQLite::dbConnect(RSQLite::SQLite(), "ECommerce.db")
```

```r
# Creating supplier table
create_supplier <- '
  CREATE TABLE IF NOT EXISTS "supplier" (
    "supplier_id" INTEGER PRIMARY KEY,
    "supplier_name" VARCHAR(100) NOT NULL,
    "rating" DOUBLE DEFAULT 1,
    "country" VARCHAR(55),
    "street_name" VARCHAR(100),
    "house_no" INTEGER,
    "postcode" VARCHAR(9)
  );
'

dbExecute(schema_db, create_supplier)
```

```
[1] 0
```

```r
# Creating product table
create_product <- '
CREATE TABLE IF NOT EXISTS "product" (
  "product_id" INTEGER PRIMARY KEY,
  "supplier_id" INTEGER,
  "product_name" VARCHAR(200),
  "brand" VARCHAR(150),
  "price" DOUBLE,
  "rating" DOUBLE,
  "availability" TEXT,
  FOREIGN KEY ("supplier_id") REFERENCES supplier("supplier_id")
);
'

dbExecute(schema_db, create_product)
```

```
[1] 0
```

```
# Creating customer table
create_customer <- '
CREATE TABLE IF NOT EXISTS "customer" (
  "customer_id" INTEGER PRIMARY KEY,
  "email" VARCHAR(150) UNIQUE NOT NULL,
  "first_name" VARCHAR(150) NOT NULL,
  "last_name" VARCHAR(150) NOT NULL,
  "mobile_no" CHAR(15) UNIQUE NOT NULL,
  "credit_card_no" CHAR(20) UNIQUE NOT NULL,
  "country" VARCHAR(55) NOT NULL,
  "street_name" VARCHAR(100) NOT NULL,
  "house_no" INTEGER NOT NULL,
  "postcode" VARCHAR(9) NOT NULL
);
'

dbExecute(schema_db, create_customer)
```

```
[1] 0
```

```
# Creating ad table
create_ad <- '
CREATE TABLE IF NOT EXISTS "ad" (
  "ad_id" INTEGER PRIMARY KEY,
  "product_id" INTEGER,
  "duration" INTEGER DEFAULT 1,
  "cost" DOUBLE NOT NULL,
  FOREIGN KEY ("product_id") REFERENCES product("product_id")
);
'

dbExecute(schema_db, create_ad)
```

```
[1] 0
```

```
# Creating category table
create_category <- '
CREATE TABLE IF NOT EXISTS "category" (
  "category_id" INTEGER PRIMARY KEY,
  "category_name" VARCHAR(150)
);
'

dbExecute(schema_db, create_category)
```

```
[1] 0
```

```
# Creating joint_in table
create_joint_in <- '
CREATE TABLE IF NOT EXISTS "joint_in" (
  "product_id" INTEGER,
  "category_id" INTEGER,
  FOREIGN KEY ("product_id") REFERENCES product("product_id"),
  FOREIGN KEY ("category_id") REFERENCES category("category_id")
);
'
dbExecute(schema_db, create_joint_in)
```

```
[1] 0
```

```
# creating transaction table
create_transaction <- '
CREATE TABLE IF NOT EXISTS "transaction" (
  "transaction_id" INTEGER PRIMARY KEY,
  "order_detail_id" INTEGER,
  "payment_method" VARCHAR(30),
  FOREIGN KEY ("order_detail_id") REFERENCES order_detail("order_detail_id")
);
'

dbExecute(schema_db, create_transaction)
```

```
[1] 0
```

```
create_order_detail <- '
CREATE TABLE IF NOT EXISTS "order_detail" (
  "order_detail_id" INTEGER PRIMARY KEY,
  "transaction_id" INTEGER NOT NULL,
  "delivery_date" DATE,
  "discount" DOUBLE DEFAULT 0,
  FOREIGN KEY ("transaction_id") REFERENCES "transaction"("transaction_id")
);
'

dbExecute(schema_db, create_order_detail)
```

```
[1] 0
```

```
# Creating joint_order table
create_joint_order <- '
CREATE TABLE IF NOT EXISTS "joint_order" (
  "order_detail_id" INTEGER,
  "customer_id" INTEGER,
  "product_id" INTEGER,
  "quantity" INTEGER,
  FOREIGN KEY ("order_detail_id") REFERENCES order_detail("order_detail_id"),
  FOREIGN KEY ("customer_id") REFERENCES customer("customer_id"),
  FOREIGN KEY ("product_id") REFERENCES product("product_id")
);
'


dbExecute(schema_db, create_joint_order)
```

```
[1] 0
```

```
dbDisconnect(schema_db)
```

Part 2: Data Generation and Management

2.1: Synthetic Data Generation

Various packages such as 'charlatan', 'fakir' and 'generator' in R generate categorical variables like customer/product/supplier names and street names. The integer columns are generated using sampling from different distributions in R, which are exponential, gamma and uniform. The date attribute is also created by sampling from a date range provided by the team. The first phase is generated inside "First_Phase_Data_Load.R" and stored by adding "seed". Since all data is generated manually, all properties of the data violating 1NF are resolved during the generation stage.

For example, a standard address consists of many parts, including postcode, house number, street name, and country. When generating address data, we separate these parts into four columns to maintain a single value in each column. Similarly, we separated names into last names and first names.

```r
schema_db <- RSQLite::dbConnect(RSQLite::SQLite(), "ECommerce.db")

# 1.Customers Table

set.seed(4)

# ID
numbers_colon <- sample(110000:210000, 1000, replace = FALSE)

customer <- as_tibble(numbers_colon)
names(customer)[names(customer) == "value"] <- "customer_id"

# Name
person_provider <- PersonProvider$new()

random_first_names <- replicate(1000, person_provider$first_name(), )
random_last_names <- replicate(1000, person_provider$last_name(), )
random_full_names <- paste(random_first_names, random_last_names)

customer$name <- random_full_names
customer$name_modified <- gsub(" ", "_", customer$name)
customer$email <- paste0(customer$name_modified, "@mail.com")
customer <- customer %>%
  separate(name, into = c("First_Name", "Last_Name"), sep = " ")
customer <- customer %>% select(-name_modified)
```

```
# Fake phone numbers
lower_limit <- 4400000000000
upper_limit <- 4499999999999
fake_mobile_numbers <- runif(1000, min = 0, max = 1) * (upper_limit - lower_limit) + lower_li

customer$mobile_no <- paste("+", as.character(round(fake_mobile_numbers)), sep = "")

# Credit Card Number
customer$credit_card_no <- ch_credit_card_number(n = 1000)

# Country
customer$country <- "UK"

# Street Names
street_elements <- c("Maple", "Main", "Oak", "Elm", "Cedar", "High", "Park", "Station", "Gre
fake_street_names <- paste(sample(street_elements, 1000, replace = TRUE))

customer$street_name <- fake_street_names

# House Number
house_no <- sample(1:75, 1000, replace = TRUE)
customer$house_no <- house_no

# UK-style postcodes
fake_postcodes <- paste0(sample(LETTERS, 1000, replace = TRUE), sample(LETTERS, 1000, replac
customer$postcode <- fake_postcodes

# Adjust the columns to fit database
customer <- customer[, c(1, 4, 2, 3, 5, 6, 7, 8, 9, 10)]
```

```
# 2.Suppliers Table

set.seed(5)

# ID
supplier_id <- sample(210001:310000, 30, replace = FALSE)
Suppliers <- as_tibble(supplier_id)
names(Suppliers)[names(Suppliers) == "value"] <- "Supplier_ID"

# Name
supplier_names <- c("Astonville Electronics", 'GlobalTech', 'Estelle', 'Quantin', 'Nexius',
abbrev <- c('Group', 'Co.', 'Services')
```

```
fake_supplier_names <- replicate(30, paste(sample(supplier_names, 1), sample(abbrev, 1, repla
Suppliers$Supplier_Name <- fake_supplier_names

# Rating
rating <- rgamma(30, shape = 5, rate = 1)
rating <- pmin(rating, 5)
Suppliers$Rating <- round(rating, 1)

# Country
supplier_country <- c("UK", "Ireland")
country_elements <- paste(sample(supplier_country, 30, replace = TRUE))
Suppliers$Country <- country_elements

# Street Name
supplier_streets <- c("Maple", "Main", "Oak", "Elm", "Cedar", "High", "Park", "Station", "Gre
fake_streets <- paste(sample(supplier_streets, 30, replace = TRUE))

Suppliers$Street_Name <- fake_streets

# House No
house_noo <- sample(1:75, 30, replace = TRUE)
Suppliers$House_No <- house_noo

# Postcode
fake_postcodess <- paste0(sample(LETTERS, 30, replace = TRUE), sample(LETTERS, 30, replace =
Suppliers$Postcode <- fake_postcodess
```

```
# 3.Products Table

set.seed(6)

product<- fake_products(1000)
category_column <- c(product$category)
product <- product %>% select(-color, -body_location, -sent_from)
names(product)[names(product) == "name"] <- "product_name"
names(product)[names(product) == "price"] <- "price"
names(product)[names(product) == "id"] <- "product_id"
names(product)[names(product) == "brand"] <- "brand"
names(product)[names(product) == "category"] <- "category_name"

# Brand
```

```
split_brand <- function(x) {
  separated <- separate(data.frame(x), x, into = c("Name1", "Name2"), sep = "[-,]", remove =
  separated$Name2 <- ifelse(is.na(separated$Name2), separated$Name1, separated$Name2)
  return(separated)
}
split_data <- split_brand(product$brand)

product <- cbind(product, split_data)
product <- product %>% select(-brand, -x, -Name2)
names(product)[names(product) == "Name1"] <- "brand"

# Rating
ratingg <- rgamma(1000, shape = 5, rate = 1)
ratingg <- pmin(ratingg, 5)
product$rating <- round(ratingg, 1)

# Availability
availability <- c("Yes", "No")
availability <- paste(sample(availability, 1000, replace = TRUE))
product$availability <- availability

# Supplier id
supplierid_list <- c(Suppliers$Supplier_ID)
product$supplier_id <- sample(supplierid_list, 1000, replace = TRUE)
```

```
# 4.Ad Table

set.seed(7)

# ID
ad_id <- sample(310001:410000, 1000, replace = FALSE)

ad <- as_tibble(ad_id)
names(ad)[names(ad) == "value"] <- "ad_id"

# product_id
productid_list <- c(product$product_id)
ad$product_id <- sample(productid_list, 1000, replace = FALSE)

# Duration
duration <- runif(1000, min = 0, max = 1) * 2
ad$duration <- round(duration,2)
```

```
# Cost
cost <- runif(1000, min = 0, max = 1) * 50
ad$cost <- round(cost,2)
```

```
# 5.Category Table

set.seed(8)

# ID
category_id <- sample(410001:510000, 8, replace = FALSE)
category <- as_tibble(category_id)
names(category)[names(category) == "value"] <- "category_id"

# Name
namess <- category_column
category$category_name <- unique(namess)
```

```
# 6.Transactions Table

set.seed(10)

# ID
transaction_id <- sample(610001:710000, 1500, replace = FALSE)
transaction <- as_tibble(transaction_id)
names(transaction)[names(transaction) == "value"] <- 'transaction_id'

# Payment Method
method <- c('Credit Card', 'Transfer', 'Pay at Door', 'PayPal', 'Debit Card', 'Voucher')
types <- paste(sample(method, 1500, replace = TRUE))
transaction$payment_method <- types
```

```
# 7.Order Details Table

set.seed(9)

# ID
order_details_id <- sample(510001:610000, 1500, replace = FALSE)
order_detail <- as_tibble(order_details_id)
names(order_detail)[names(order_detail) == "value"] <- 'order_detail_id'

# Delivery Date
```

```r
start_date <- as.Date("2024-03-20")
end_date <- as.Date("2024-08-20")
date_sequence <- seq.Date(start_date, end_date, by = "1 day")

dates <- sample(date_sequence, 1500, replace = TRUE)
order_detail$delivery_date <- as.character(dates)

# Discount
lower_percentage <- seq(from = 5, to = 45, by = 5)
lower_percentages <- sample(lower_percentage, 1150, replace = TRUE)
upper_percentage <- seq(from = 50, to = 75, by = 5)
upper_percentages <- sample(upper_percentage, 350, replace = TRUE)
percentages <- c(lower_percentages, upper_percentages)
order_detail$discount <- paste(percentages, '%', sep = '')

# order_detail_id
orderid_list <- c(order_detail$order_detail_id)
transaction$order_detail_id <- sample(orderid_list, 1500, replace = FALSE)

# transaction_id added as a foreign key
order_detail <- merge(x = order_detail, y = transaction, by = 'order_detail_id') %>% select(
```

```r
# Joint table for relation 'In' including category_id and product_id as foreign keys
set.seed(647823)

joint_in <- left_join(product, category, by = 'category_name')
joint_in <- joint_in %>% select(product_id, category_id)

product <- product %>% select(-category_name)
```

```r
# Joint table for relation 'Order' including order_detail_id, customer_id, product_id and qu
set.seed(2435345)
my_function <- function(n) {
  x_t <- data.frame(order_detail_id = integer(),
                    customer_id = integer(),
                    product_id = integer(),
                    quantity = integer())

  for (i in 1:n) {
    ch_order_detail_id <- order_detail$order_detail_id[i]
    ch_customer_id <- sample(customer$customer_id, 1)
    no_of_product <- as.integer(rexp(1, rate = 1/3))
    y = product$product_id

    for (j in 1:no_of_product) {
      ch_product_id <- sample(y, 1, replace = FALSE)
      quantity <- sample(1:5, 1)

      # Append to data frame
      x_t <- rbind(x_t, data.frame(order_detail_id = ch_order_detail_id,
                                   customer_id = ch_customer_id,
                                   product_id = ch_product_id,
                                   quantity = quantity))
    }
  }

  return(as_tibble(x_t))
}

result <- my_function(1500)
```

```
# Remove any duplicate row/entry
joint_order <- distinct(result, order_detail_id, customer_id, product_id, .keep_all = TRUE)
names(joint_order)[names(joint_order) == "order_detail$order_detail_id"] <- 'order_detail_id
names(joint_order)[names(joint_order) == "customer$customer_id"] <- 'customer_id'
names(joint_order)[names(joint_order) == "product$product_id"] <- 'product_id'
```

```
# Adjust the order of columns to fit database
order_detail <- order_detail[, c(1, 4, 2, 3)]
transaction <- transaction[, c(1, 3, 2)]
product <- product[, c(3, 7, 1, 4, 2, 5, 6)]
```

Also, inside the database, total price of an order is also kept. This is a derived column calculated as sum of product_price * quantity for all products purchased inside an order.

```
# Calculation of Total Price of Orders after Discounts are applied

create_price_view <- '

CREATE VIEW IF NOT EXISTS final_price_of_order AS SELECT order_detail_id, subtotal, discount
(SELECT A.order_detail_id, COUNT(A.product_id) AS no_of_products, SUM(A.total_price_for_each
(SELECT J.order_detail_id, J.product_id, P.price, J.quantity, (P.price* J.quantity) AS total
JOIN
product p WHERE J.product_id = P.product_id) A
JOIN
order_detail O ON A.order_detail_id = O.order_detail_id GROUP BY A.order_detail_id);

'

dbExecute(schema_db, create_price_view)
```

```
[1] 0
```

```
view_query <- 'SELECT * FROM final_price_of_order'
view_result <- dbGetQuery(schema_db, view_query)
```

Since it distorts normalization, total price column is stored inside a temporary view and updated generically once new orders are placed by customers. At the end of this stage, our synthetic data is normalized and adheres to the rules of 1NF, 2NF and 3NF.

2.2: Data Import and Quality Assurance

The data generated is split into two parts. Initially, the first load is imported to the database inside "First_Phase_Data_Load.R". After import, data is validated by checking the uniqueness of all primary keys and unique values like credit card number, email and mobile number and data is also validated for duplicate checks. The format and validity of all columns in tables are checked one by one to construct data integrity. Besides these, referential integrity and missing value checks are also conducted for primary keys and all entries having NOT NULL constraint in physical schema. By rigorously validating our dataset, we ensured its reliability and suitability for subsequent analysis.

```
# # Data Integrity and Quality Check

# Testing for any duplication of primary keys or unique values

duplicate_info <- character()

unique_columns <- list(
  "Customer_ID" = customer$customer_id,
  "Email" = customer$email,
  "Mobile_No" = customer$mobile_no,
  "Supplier_ID" = Suppliers$Supplier_ID,
  "Category_ID" = category$category_id,
```

```r
  "OrderDetail_ID" = order_detail$order_detail_id,
  "ad_id" = ad$ad_id,
  "product_id" = product$product_id,
  "Transaction_ID" = transaction$transaction_id
)

for (col in names(unique_columns)) {
  has_duplicates <- any(duplicated(unique_columns[[col]]))

  if (has_duplicates) {
    duplicate_info <- c(duplicate_info, paste("Column", col, "has duplicate values"))
  }
}

if (length(duplicate_info) > 0) {
  cat("Columns with duplicate values:\n")
  cat(duplicate_info, sep = "\n")
} else {
  print("No primary keys and unique values have duplicate values.")
}
```

```
[1] "No primary keys and unique values have duplicate values."
```

```r
# Formatting Email column in Customers Table (some customers have ' in their names as defaul
customer$email <- gsub("'", "", customer$email)


## Inserting fake data into database (first load)

my_db <- RSQLite::dbConnect(RSQLite::SQLite(),"ECommerce.db")
dbWriteTable(my_db, "supplier", Suppliers, overwrite = TRUE)
dbWriteTable(my_db, "product",product, overwrite= TRUE)
dbWriteTable(my_db, "ad", ad, overwrite= TRUE)
dbWriteTable(my_db, "transaction", transaction, overwrite= TRUE)
dbWriteTable(my_db, "order_detail", order_detail, overwrite= TRUE)
dbWriteTable(my_db, "customer", customer, overwrite= TRUE)
dbWriteTable(my_db, "category", category, overwrite= TRUE)

# Joint tables
dbWriteTable(my_db, "joint_in", joint_in, overwrite= TRUE)
dbWriteTable(my_db, "joint_order", joint_order, overwrite= TRUE)
```

```
dbDisconnect(schema_db)
```

Part 3: Data Pipeline Generation

3.1: GitHub Repository and Workflow Setup

The GitHub repository “Demo_Group10” includes workflows, R scripts and a database to manage the project. Initially, the workflow installs the R packages used in R scripts. Next, R scripts, which are used in automation, are executed. There are three scripts executed in the project workflow: “Second_Phase_Data_Load.R”, “Validation.R”, and “Data_Analysis.R”, respectively. The workflow also includes the collaborator names. Ultimately, the workflow automatically commits and pushes the processing results to the GitHub repository.

3.2: GitHub Actions for Continuous Integration

In this part, we set up workflows to implement GitHub actions that perform tasks such as running data validation, updating the database, and automatically running fundamental data analysis. The new data arrives in the database by executing “Second_Phase_Data_Load.R”. The system first controls whether this has a primary key already in the corresponding table or not inside “Second_Phase_Data_Load.R” before appending the new data. If it has a new primary key and unique values for the columns specified in the schema, these entries are accepted, and the database is updated with these proper rows. The remaining entries with invalid columns are rejected and the number of them is kept inside” Invalid_Entries.txt “.

After the database is updated, our workflow automatically checks the format of these newly added entries by executing “Validation.R”. After completing these checks, our workflow updates the results inside “Validation.R” according to our updated database. Finally, the workflow automatically commits and pushes the processing results to the GitHub repository.

```r
## Running data validation
# Connect to database
my_db <- RSQLite::dbConnect(RSQLite::SQLite(),"ECommerce.db")

# 1. Customer Table

customer <- dbReadTable(my_db, "customer")

# Customer ID (6-digit)
validate_customer_id <- function(customer_id) {
  ifelse(grepl("^\\d{6}$", customer_id), TRUE, FALSE)
}

# Email format (xxx@xxx.xxx)
validate_email <- function(email) {
  email_pattern <- "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.]+\\.com$"
  ifelse(grepl(email_pattern, email), TRUE, FALSE)
}

# Mobile number format (+xxxxxxxxxxxxxx)
validate_phone <- function(phone) {
  phone_pattern <- "^\\+?[1-9]\\d{1,14}$"
  ifelse(grepl(phone_pattern, phone), TRUE, FALSE)
}


# UK postcode format
validate_uk_postcode <- function(postcode) {
  postcode_pattern <- "^([Gg][Ii][Rr] 0[Aa]{2})|((([A-Za-z][0-9]{1,2})|(([A-Za-z][A-Ha-hJ-Yj
  ifelse(grepl(postcode_pattern, postcode), TRUE, FALSE)
}


# Last Name format (only letters)
validate_last_name <- function(name) {
  ifelse(grepl("^[A-Za-z]+(['-][A-Za-z]+)*$", name), TRUE, FALSE)
}

# First Name format (only letters)
validate_first_name <- function(name) {
  ifelse(grepl("^[A-Za-z]+(['-][A-Za-z]+)*$", name), TRUE, FALSE)
}

# Credit card number (only check the length)
validate_credit_card_no <- function(credit_card_no) {
  valid_length <- nchar(as.character(credit_card_no)) >= 13 & nchar(as.character(credit_card
  return(valid_length)
}


# Street name (without number)
validate_street_name <- function(street_name) {
  ifelse(grepl("^[A-Za-z]+([ '-][A-Za-z]+)*$", street_name), TRUE, FALSE)
}

# House No (number+letter)
validate_house_no <- function(house_no) {
```

```r
  ifelse(grepl("^\\d+[A-Za-z]?$", house_no), TRUE, FALSE)
}

# Country (only UK)
validate_country_uk <- function(country) {
  ifelse(toupper(country) == "UK", TRUE, FALSE)
}


# calculate invalid data
invalid_counts_customer <- data.frame(
  invalid_customer_id = sum(!sapply(customer$customer_id, validate_customer_id)),
  invalid_emails = sum(!sapply(customer$email, validate_email)),
  invalid_first_names = sum(!sapply(customer$First_Name, validate_first_name)),
  invalid_last_names = sum(!sapply(customer$Last_Name, validate_last_name)),
  invalid_phones = sum(!sapply(customer$mobile_no, validate_phone)),
  invalid_credit_card_nos = sum(!sapply(customer$credit_card_no, validate_credit_card_no)),
  invalid_countries_uk = sum(!sapply(customer$country, validate_country_uk)),
  invalid_street_names = sum(!sapply(customer$street_name, validate_street_name)),
  invalid_house_nos = sum(!sapply(customer$house_no, validate_house_no)),
  invalid_postcodes = sum(!sapply(customer$postcode, validate_uk_postcode))
)

print(invalid_counts_customer)
```

```
  invalid_customer_id invalid_emails invalid_first_names invalid_last_names
1                   0              0                   0                  0
  invalid_phones invalid_credit_card_nos invalid_countries_uk
1              0                       0                    0
  invalid_street_names invalid_house_nos invalid_postcodes
1                    0                 0                 0
```

```r
# duplicates


customer_duplicates <- dbGetQuery(my_db, "SELECT customer_id, count(*) from customer group by
 print(paste("No:of duplicates for customer table is ",nrow(customer_duplicates)))
```

```
[1] "No:of duplicates for customer table is  0"
```

```
# 2. Supplier Table

supplier <- dbReadTable(my_db, "supplier")

# Supplier ID (6-digit)
validate_supplier_id <- function(Supplier_ID) {
  ifelse(grepl("^\\d{6}$", Supplier_ID), TRUE, FALSE)
}

# Supplier Name (only letters)
validate_supplier_name <- function(Supplier_Name) {
  name_pattern <- "^[A-Za-z]+([ '.-][A-Za-z]+)*\\.?$"
  ifelse(grepl(name_pattern, Supplier_Name), TRUE, FALSE)
}



# Rating (0.0-5.0)

validate_rating <- function(Rating) {
  if(!is.numeric(Rating)) {
    return(rep(FALSE, length(Rating)))
  }

  result <- (Rating >= 0.0 & Rating <= 5.0) & (floor(Rating * 10) == (Rating * 10))
  return(result)
}

# Country
validate_country <- function(Country) {
  country_pattern <- "^[A-Za-z ]+$"
  ifelse(grepl(country_pattern, Country), TRUE, FALSE)
}


# Street Name
validate_street_name <- function(Street_Name) {
  ifelse(grepl("^[A-Za-z]+([ '-][A-Za-z]+)*$", Street_Name), TRUE, FALSE)
}

# House No (number+letter)
validate_house_no <- function(House_No) {
  ifelse(grepl("^\\d+[A-Za-z]?$", House_No), TRUE, FALSE)
}

# Postcode
validate_postcode <- function(Postcode) {
  postcode_pattern <- "^([Gg][Ii][Rr] 0[Aa]{2})|((([A-Za-z][0-9]{1,2})|(([A-Za-z][A-Ha-hJ-Yj
  ifelse(grepl(postcode_pattern, Postcode), TRUE, FALSE)
}


# calculate invalid data
invalid_counts_supplier <- data.frame(
  invalid_supplier_ids = sum(!sapply(supplier$Supplier_ID, validate_supplier_id)),
  invalid_supplier_names = sum(!sapply(supplier$Supplier_Name, validate_supplier_name)),
  invalid_ratings = sum(!sapply(supplier$Rating, validate_rating)),
  invalid_countries = sum(!sapply(supplier$Country, validate_country)),
  invalid_street_names = sum(!sapply(supplier$Street_Name, validate_street_name)),
  invalid_house_nos = sum(!sapply(supplier$House_No, validate_house_no)),
  invalid_postcodes = sum(!sapply(supplier$Postcode, validate_postcode))
)


print(invalid_counts_supplier)
```

```
  invalid_supplier_ids invalid_supplier_names invalid_ratings invalid_countries
1                    0                      0               0                 0
  invalid_street_names invalid_house_nos invalid_postcodes
1                    0                 0                 0
```

```
# duplicates

supplier_duplicates <- dbGetQuery(my_db, "SELECT supplier_id, count(*) from supplier group by
print(paste("No:of duplicates for supplier table is ",nrow(supplier_duplicates)))
```

```
[1] "No:of duplicates for supplier table is  0"
```

```
# 3. Ad Table

ad <- dbReadTable(my_db, "ad")
```

```
# ad ID (6-digit)
validate_ad_id <- function(ad_id) {
  ifelse(grepl("^\\d{6}$", ad_id), TRUE, FALSE)
}

# product ID (1-1000)
validate_product_id <- function(product_id) {
  ifelse(product_id >= 1 & product_id <= 1000, TRUE, FALSE)
}

# duration (ad duration can be 0 to 10 minutes)
validate_duration <- function(duration) {
  if(!is.numeric(duration)) {
    return(FALSE)
  }
  ifelse(duration >= 0 && duration <= 10 && floor(duration * 100) == (duration * 100), TRUE,
}


# cost
validate_cost <- function(cost) {
  if(!is.numeric(cost)) {
    return(FALSE)
  }
  ifelse(cost >= 0 && cost <= 1000, TRUE, FALSE)
}

# duration (numeric)
validate_duration <- function(duration) {
  is.numeric(duration)
}

# cost (numeric)
validate_cost <- function(cost) {
  is.numeric(cost)
}


# calculate invalid data
invalid_counts_ad <- data.frame(
  invalid_ad_ids = sum(!sapply(ad$ad_id, validate_ad_id)),
  invalid_product_ids = sum(!sapply(ad$product_id, validate_product_id)),
  invalid_durations = sum(!sapply(ad$duration, validate_duration)),
  invalid_costs = sum(!sapply(ad$cost, validate_cost))
)

print(invalid_counts_ad)
```

```
  invalid_ad_ids invalid_product_ids invalid_durations invalid_costs
1              0                   0                 0             0
```

```
# duplicates

ad_duplicates <- dbGetQuery(my_db, "SELECT ad_id, count(*) from ad group by ad_id having cou
print(paste("No:of duplicates for ad table is ",nrow(ad_duplicates)))
```

```
[1] "No:of duplicates for ad table is  0"
```

```
# 4. Category Table

category <- dbReadTable(my_db, "category")

# Category ID (6-digit)
validate_category_id <- function(category_id) {
  ifelse(grepl("^\\d{6}$", category_id), TRUE, FALSE)
}

# Category Name (only letters)
validate_category_name <- function(category_name) {
  ifelse(grepl("^[A-Za-z ]+$", category_name), TRUE, FALSE)
}

# calculate invalid data
invalid_counts_category <- data.frame(
  invalid_category_ids = sum(!sapply(category$Category_ID, validate_category_id)),
  invalid_category_names = sum(!sapply(category$Category_Name, validate_category_name))
)

print(invalid_counts_category)
```

```
  invalid_category_ids invalid_category_names
1                    0                      0
```

```
# duplicates

category_duplicates <- dbGetQuery(my_db, "SELECT category_id, count(*) from category group by
print(paste("No:of duplicates for category table is ",nrow(category_duplicates)))
```

```
[1] "No:of duplicates for category table is  0"
```

```
# 5. Transaction Table

transaction <- dbReadTable(my_db, "transaction")

# Transaction ID (6-digit)
validate_transaction_id <- function(transaction_id) {
  ifelse(grepl("^\\d{6}$", transaction_id), TRUE, FALSE)
}

# Order detail ID (6-digit)
validate_order_detail_id <- function(order_detail_id) {
  ifelse(grepl("^\\d{6}$", order_detail_id), TRUE, FALSE)
}

# Payment method (only letters)
validate_payment_method <- function(payment_method) {
  ifelse(grepl("^[A-Za-z ]+$", payment_method), TRUE, FALSE)
}

# checking payment methods other than listed and setting it to others
pay_method  <- "
  UPDATE 'transaction' SET payment_method = 'Others' where payment_method not in ('Transfer'
"
dbExecute(my_db, pay_method)
```

```
[1] 0
```

```
# calculate invalid data
invalid_counts_transaction <- data.frame(
  invalid_transaction_ids = sum(!sapply(transaction$Transaction_ID, validate_transaction_id)
  invalid_order_detail_ids = sum(!sapply(transaction$Order_Detail_ID, validate_order_detail_
  invalid_payment_methods = sum(!sapply(transaction$Payment_Method, validate_payment_method)
)
```

```
print(invalid_counts_transaction)
```

```
  invalid_transaction_ids invalid_order_detail_ids invalid_payment_methods
1                       0                        0                       0
```

```
# duplicates

transaction_duplicates <- dbGetQuery(my_db, "SELECT transaction_id, count(*) from 'transacti
print(paste("No:of duplicates for transaction table is ",nrow(transaction_duplicates)))
```

```
[1] "No:of duplicates for transaction table is  0"
```

```
# 6. Order Details Table

order_detail <- dbReadTable(my_db, "order_detail")

# Order detail ID (6-digit)
validate_order_detail_id <- function(order_detail_id) {
  ifelse(grepl("^\\d{6}$", order_detail_id), TRUE, FALSE)
}

# transaction ID (6-digit)
validate_transaction_id <- function(transaction_id) {
  ifelse(grepl("^\\d{6}$", transaction_id), TRUE, FALSE)
}

# delivery date
validate_delivery_date <- function(delivery_date) {
  tryCatch(!is.na(as.Date(delivery_date, format = "%Y-%m-%d")),
           error = function(e) FALSE)
}

# discount (xx%)
validate_discount <- function(discount) {
  if(!is.numeric(discount)) {
    return(FALSE)
  }
  ifelse(discount >= 0 & discount <= 100, TRUE, FALSE)
}
```

```r
# calculate invalid data
invalid_counts_order_detail <- data.frame(
  invalid_order_detail_ids = sum(!sapply(order_detail$Order_Detail_ID, validate_order_detail_
  invalid_transaction_ids = sum(!sapply(order_detail$Transaction_ID, validate_transaction_id
  invalid_delivery_dates = sum(!sapply(order_detail$Delivery_Date, validate_delivery_date)),
  invalid_discounts = sum(!sapply(order_detail$Discount, validate_discount))
)

print(invalid_counts_order_detail)
```

```
  invalid_order_detail_ids invalid_transaction_ids invalid_delivery_dates
1                        0                       0                      0
  invalid_discounts
1                 0
```

```r
# duplicates

order_detail_duplicates <- dbGetQuery(my_db, "SELECT order_detail_id, count(*) from order_de
print(paste("No:of duplicates for order_detail table is ",nrow(order_detail_duplicates)))
```

```
[1] "No:of duplicates for order_detail table is  0"
```

```r
#7. Products table

product <- dbReadTable(my_db, "product")

# Validate price

validate_price <- function(price){
  ifelse(price <= 0, FALSE , TRUE )
}

validate_availability <-  function(availability) {
  if (!(availability %in% c("yes", "no"))) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}
```

```r
validate_rating_pr <- function(rating) {
  if(!is.numeric(rating)) {
    return(rep(FALSE, length(rating)))
  }

  result <- (rating >= 0.0 & rating <= 5.0) & (floor(rating * 10) == (rating * 10))
  return(result)
}

validate_prname <- function(product_name){
  if (is.null(product_name) | product_name == ""){
    print("product name is null")
    return (FALSE)
  }
  else {
    return(TRUE)
  }
}

validate_brand <- function(brand){
  if(is.null(brand) | brand == ""){
    print("product brand is null")
    return(FALSE)
  }
  else {
    return(TRUE)
  }
}

# calculate invalid data
invalid_counts_product <- data.frame(
  invalid_price = sum(!sapply(product$price, validate_price)),
  invalid_rating_pr = sum(!sapply(product$rating, validate_rating_pr)),
  invalid_availability = sum(!sapply(product$availability, validate_availability)),
  invalid_prname = sum(!sapply(product$product_name, validate_prname)),
  invalid_brand = sum(!sapply(product$brand, validate_brand))
)

print(invalid_counts_product)
```

```
  invalid_price invalid_rating_pr invalid_availability invalid_prname
1             0                 0                    0              0
```

```
  invalid_brand
1             0
```

```
# duplicates

product_duplicates <- dbGetQuery(my_db, "SELECT product_id, count(*) from product group by pr
print(paste("No:of duplicates for product table is ",nrow(product_duplicates)))
```

```
[1] "No:of duplicates for product table is  0"
```

```
#Referential Integrity

# Product Table

sup_id_query <- "UPDATE Product SET supplier_id = -1 WHERE supplier_id NOT IN (SELECT supplie
dbExecute(my_db, sup_id_query)
```

```
[1] 0
```

```
# Ad Table

pr_id_query <- "UPDATE Ad SET product_id = -1 WHERE product_id NOT IN (SELECT product_id FROM
dbExecute(my_db, pr_id_query)
```

```
[1] 0
```

```
# Transaction Table

orderdet_id_query <- "UPDATE 'transaction' SET order_detail_id = -1 WHERE order_detail_id NOT
dbExecute(my_db, orderdet_id_query)
```

```
[1] 0
```

```
# Order Detail Table

tr_id_query <- "UPDATE order_detail SET transaction_id = -1 WHERE transaction_id NOT IN (SELE
dbExecute(my_db, tr_id_query)
```

```
[1] 0
```

```
# Order Table

or_id <- "UPDATE joint_order SET order_detail_id = -1 WHERE order_detail_id NOT IN (SELECT or
dbExecute(my_db, or_id)
```

```
[1] 0
```

```
cu_id <- "UPDATE joint_order SET customer_id = -1 WHERE customer_id NOT IN (SELECT customer_i
dbExecute(my_db, cu_id)
```

```
[1] 0
```

```
pr_id <- "UPDATE joint_order SET product_id = -1 WHERE product_id NOT IN (SELECT product_id F
dbExecute(my_db, pr_id)
```

```
[1] 0
```

```
# In table

pr_id_in <- "UPDATE joint_in SET product_id = -1 WHERE product_id NOT IN (SELECT product_id F
dbExecute(my_db, pr_id_in)
```

```
[1] 0
```

```
cat_id_in <- "UPDATE joint_in SET category_id = -1 WHERE category_id NOT IN (SELECT category_
dbExecute(my_db, cat_id_in)
```

```
[1] 0
```

```
dbDisconnect(my_db)
```

Part 4: Data Analysis and Reporting with Quarto in R

4.1: Advanced Data Analysis in R

We conducted advanced data analysis based on our e-commerce data in this part. Our analysis focused on several critical areas for understanding customer behaviour, product performance, and market trends.

We started by analysing customers' payment methods to help businesses better meet customer needs and ensure that the process is user-friendly for such payments. Next, we analysed the best sellers of each category, most popular category, the final price of order distribution with respect to payment method, and which product has the highest rating in each category to identify current consumer trends and preferences. In addition, we also analysed the top 20 customers to help us monitor specific trends and metrics.

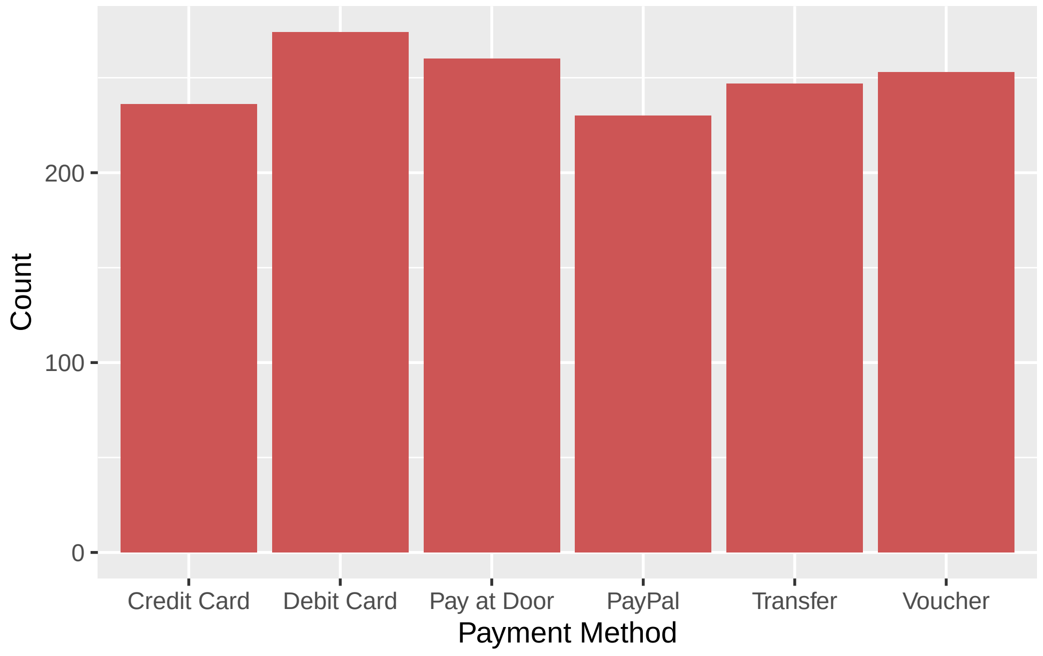
4.2: Comprehensive Reporting with Quarto

E-commerce databases contain vast amounts of data related to customer behaviour, sales trends, product performance, and more. The data in the visuals below are derived from our e-commerce database and include information about the number of products sold, the highest selling category, frequent payment methods and the most used product within each category in the latest updated database on 16th March.

```r
schema_db <- RSQLite::dbConnect(RSQLite::SQLite(), "ECommerce.db")

# Read data
table_name <- dbListTables(schema_db)
tables <- list()
for (name in table_name){
  tables[[name]] <- dbReadTable(schema_db, name)
}

## ------------------------------------------------------------------------
joint_in_table <- tables[["joint_in"]]
joint_order_table <- tables[["joint_order"]]
```

```r
# Payment methods
transaction_table <- tables[["transaction"]]
payment_plot <- ggplot(transaction_table, aes(x = payment_method))+
  geom_bar(fill = 'indianred3') +
  labs(x = "Payment Method", y = "Count")
```

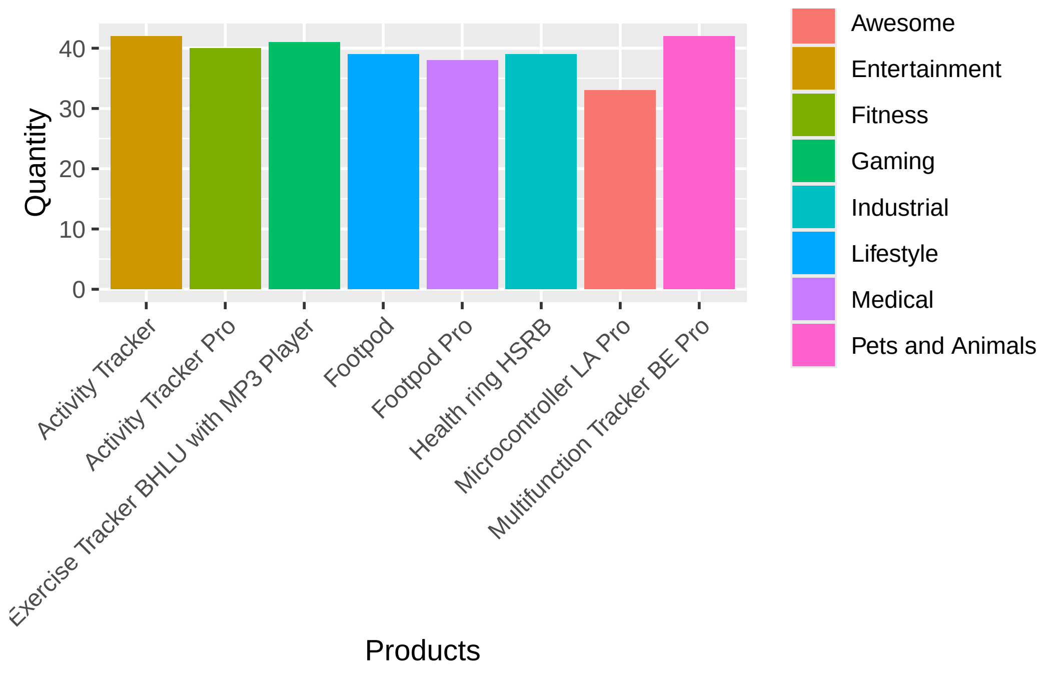
```r
payment_plot
```



The bar chart displays the payment methods, with the highest frequencies for debit Cards and the lowest for PayPal. This indicates that customers prefer to use a Debit Card while purchasing products from our site.

```r
# Best sellers for each category
product_table <- tables[["product"]]
category_table <- tables[["category"]]

bsec <- product_table %>%
  left_join(joint_in_table, by = "product_id") %>%
  left_join(category_table, by = "category_id") %>%
  left_join(joint_order_table, by = "product_id")

best_sellers <- bsec %>%
  group_by(category_name, product_name) %>%
  summarise(total_quantity = sum(quantity), .groups = 'drop') %>%
  group_by(category_name) %>%
  arrange(desc(total_quantity)) %>%
  slice(1) %>%
  ungroup()
```

```
bests <- ggplot(best_sellers) +
  geom_bar(aes(x = product_name, y = total_quantity, fill = category_name),
           stat = "identity") +
  labs(x = "Products", y = "Quantity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
bests
```



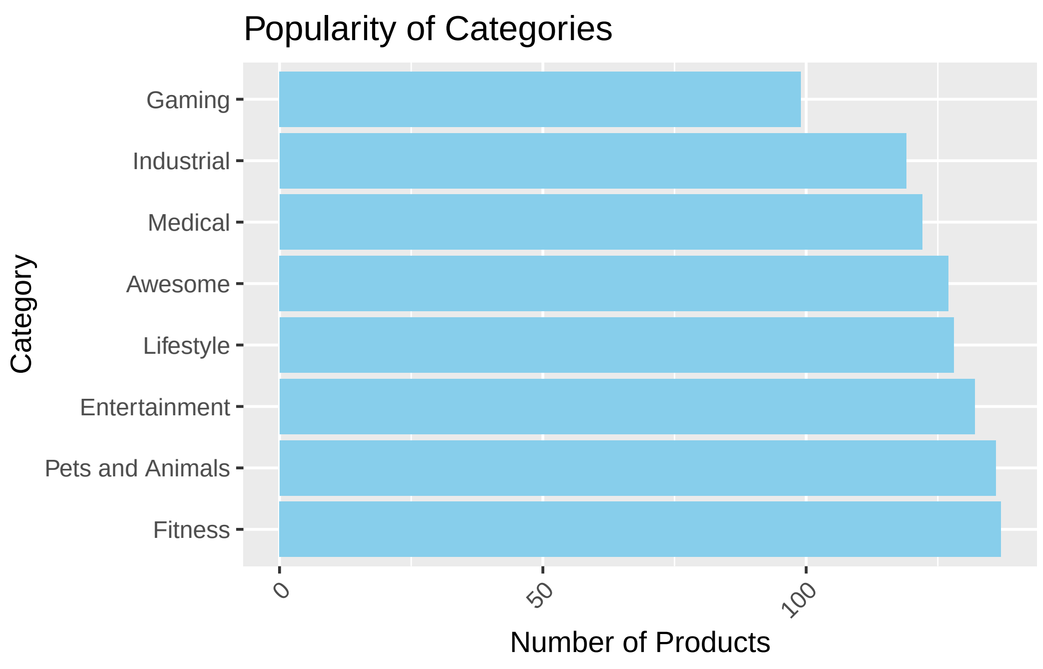
The bar chart shows quantities of products categorised by colour-coded categories, with 'Multifunction Tracker E Pro' being the most abundant in the 'Pets and Animals' category.

```
# Which category is the most popular

category_table <- tables[["category"]]
product_table <- tables[["product"]]
customer_table <- tables[["customer"]]
joint_in_table <- tables[["joint_in"]]

# Step 1: Join the product and category tables using the joint table
product_category_join <- joint_in_table%>%
  inner_join(product_table, by = "product_id") %>%
  inner_join(category_table, by = "category_id")

# Step 2: Aggregate the data to count occurrences of each category
```

```
category_counts <- product_category_join %>%
  group_by(category_name) %>%
  summarise(count = n()) %>%
  arrange(desc(count))

# Step 3: Print or visualize the results to identify the most popular category
most_popular_category <- category_counts$category_name[1]
print(paste("The most popular category is:", most_popular_category))
```

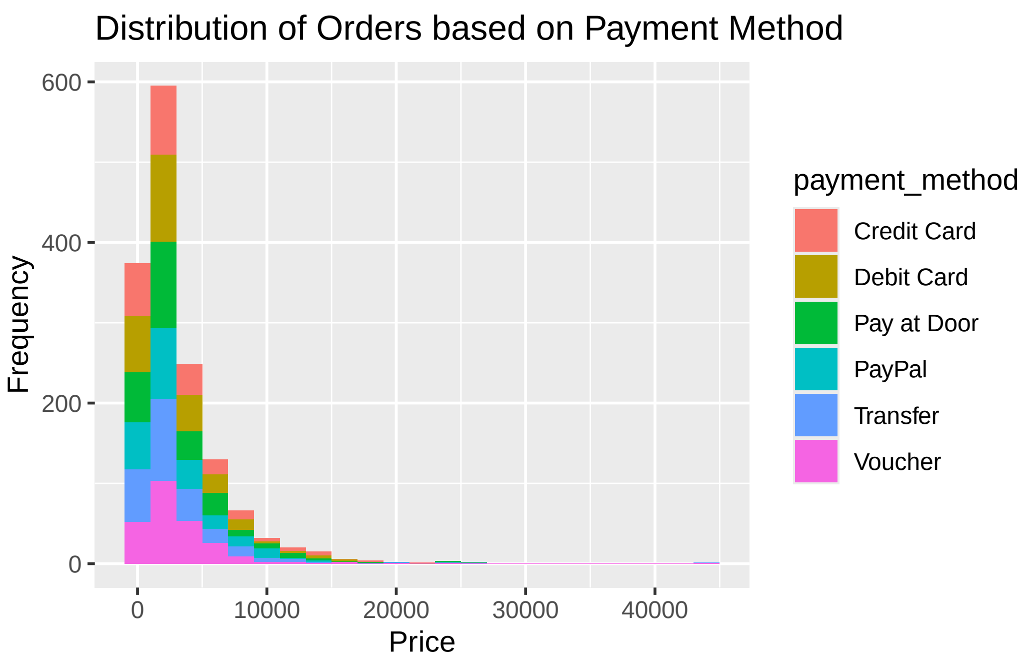
```
[1] "The most popular category is: Fitness"
```

```
# Writing it into the csv file

write.csv(most_popular_category ,"most_popular_category.csv", row.names = FALSE)


top_cat <- ggplot(category_counts, aes(x = reorder(category_name, -count), y = count)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(x = "Category", y = "Number of Products", title = "Popularity of Categories") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  coord_flip()
top_cat
```



The above visual illustrates the popularity of different categories based on our latest database analysis. We can identify that the most popular category is 'Fitness' followed by 'Pets and Animals'. The least favourite category on our site is 'Gaming'.

```
# Final price of orders distribution with respect to payment method
tot_price <- 'SELECT final_price, order_detail_id FROM final_price_of_order'
view_tot <- dbGetQuery(schema_db, tot_price)
view_tot <- as_tibble(view_tot)

transaction_table <- tables[["transaction"]]
transaction_table <- left_join(transaction_table, view_tot, by = 'order_detail_id')
basket <- ggplot(transaction_table, aes(final_price, fill = payment_method)) +
  geom_histogram(binwidth = 2000) +
  labs(title = 'Distribution of Orders based on Payment Method', x = 'Price', y = 'Frequency
basket
```



We can understand from the visual that most orders are concentrated in the lower price ranges (towards the left side of the histogram), which suggests that cheaper items are more commonly purchased.

Credit card payments (red) are the most popular across all price ranges, followed by debit cards (green) and PayPal (cyan). Payment methods such as Pay at Door (blue) and Transfer (magenta) are also used but less than credit/debit cards and PayPal. Voucher payments (yellow) are the least represented in this dataset, indicating they are the least used payment method among the customers represented in this data. The plot also shows that as the price increases, the frequency of orders decreases, which is typical in consumer behaviour as higher-priced items tend to be purchased less frequently.

We have three CSV files named – ‘top 20 customers’, ‘most_popular_category’ and ’highest_rating_per_category, which help us to monitor specific trends and metrics. This information helps us to understand the current market better.

Challenges

Ecommerce databases often involve complex data models due to the variety of entities and relationships involved. So, designing a database, choosing the right entities, and figuring out the proper relationships while maintaining data integrity was challenging. In the data generation stage, it was important to ensure quality of generated data. Since randomly generated data may not always reflect the real-world scenarios accurately and can lead to inconsistent or unrealistic data, validating and verifying the generated data was a challenge to the project. Apart from that, we have a derived column which is stored as view, so it was important to validate derived column values against the source data and ensure that the calculations accurately reflect the intended logic. Overall, addressing these challenges required proper planning and validation to ensure the integrity and accuracy of our Ecommerce database.

Conclusion

It is essential in business to analyze and track data daily to implement necessary actions