Table of contents

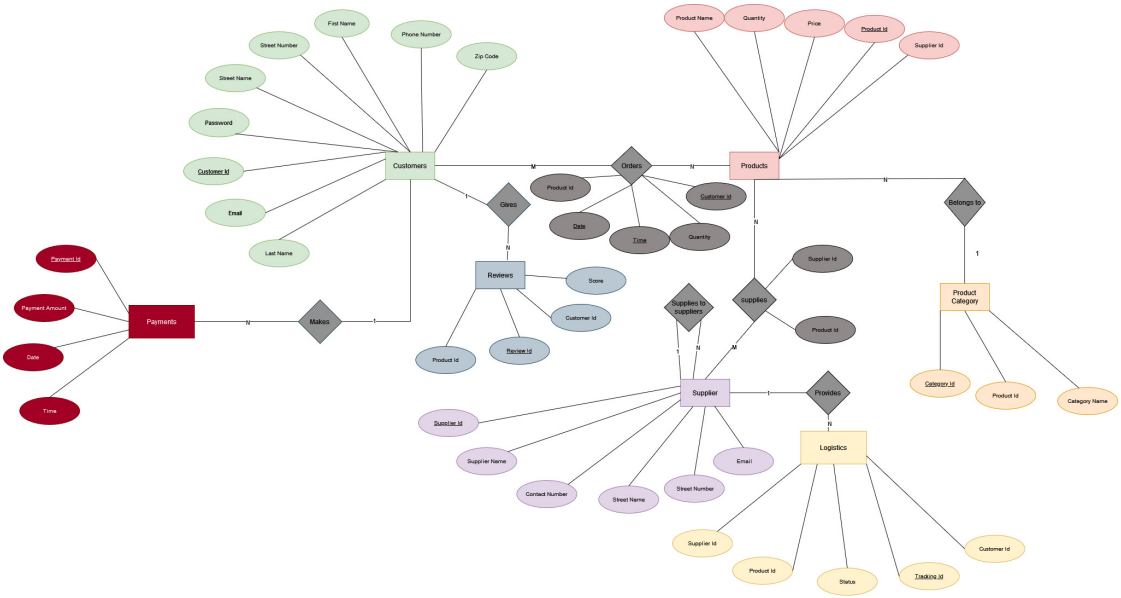
Part 1: Database Design and Implementation

1.1 E-R Diagram Design

Assumption

- Each customer has only one unique ID, default email, contact and address
- Each customer can make many orders and payments
- Each supplier supplies many products
- Each category can have many products

- Each customer can provide many reviews based on product
- Every product belongs to a product category and product category can have various products



E-R Diagram Design

Step 1: The ER model has been broken down into seven main entities namely: Customers, Products, Product category, Logistics, Supplier, Reviews and Payments.

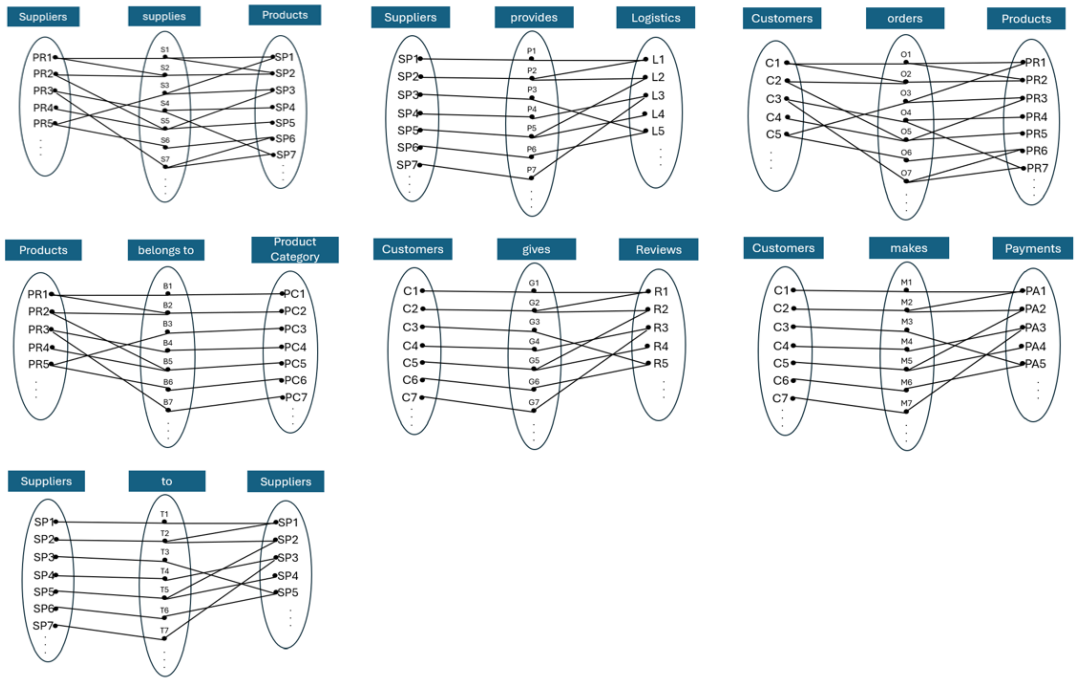
Step 2: Assigning attributes to all the entities with relevant information.

- Customer entity includes first name, last name, street number, street name, email address, contact, zip code, Customer ID(unique to every customer), password.
- Product entity includes product name, quantity, unit price, product ID(unique to every product), supplier ID(unique to every supplier).
- Product Category includes Category name, category ID, product ID.
- Logistics includes Supplier ID, product ID, status, tracking ID, Customer ID.
- Supplier includes supplier name, supplier contact number, supplier ID, supplier street name, supplier email, supplier street number.
- Reviews include score, customer ID, review ID, product ID.
- Payment includes payment ID, payment amount, payment dates, payment time.

Step 3: Defining Relationship Sets

- An ecommerce company can have a case where a single customer can M**ake** multiple payments. Hence there is one to many between Customer and Payment.
- Customers can **Order** many products at a time hence there is many to many relationship.

- One customer can **Give** many reviews for a product/ products.
- Many products can B**elong** to a similar product category, hence many to one relationship.
- Suppliers can **Provide** many logistics hence one to many relationship.
- Many suppliers can **Supplies** many products hence many to many relationship.
- **Self-referencing** for Supplies to supplier.



1.2 SQL Database Schema Creation

Logical Schema

- Customers (customer_id, first_name, last_name, email, street_name, street_number, password, phone_number, zip_code)
- Products (product_id, supplier_id, product_name, quantity, price, unit_weight)
- Payments (payment_id, payment_amount, payment_date, payment_time)
- Reviews (review_id, product_id, customer_id, score)
- Supplier (supplier_id, supplier_name, supplier_contact_number, street_name, street_number, supplier_email)
- Logistics (tracking_id, supplier_id, product_id, status, customer_id)
- Product Category (category_id, product_id, category_name)

Data Type

Entity	Attribute	Data Type
Customers	Customer_id	Integer
Customers	First_name	Character (25)
Customers	Last_name	Character (25)
Customers	Email	Character (100)
Customers	Street name	Character (100)
Customers	Street Number	Integer
Customers	Zip code	Character (6)
Customers	Phone number	Character (10)
Customers	Password	Character (500)
Products	Product name	Character (255)
Products	Quantity	Integer
Products	Price	Float
Products	Product ID	Integer
Products	Supplier ID	Integer
Product Category	Category ID	Integer
Product Category	Product ID	Integer
Product Category	Category Name	Character (100)
Logistics	Supplier ID	Integer
Logistics	Product ID	Integer
Logistics	Status	Character (50)
Logistics	Tracking ID	Integer
Logistics	Customer ID	Integer
Supplier	Supplier name	Character (255)
Supplier	Supplier ID	Integer
Supplier	Contact number	Character (20)
Supplier	Street name	Character (50)

Supplier	Street number	Integer
Supplier	Email	Character (500)
Payments	Payment ID	Integer
Payments	Payment Amount	Float
Payments	Payment Date	Date
Payments	Payment time	Integer
Reviews	Product ID	Integer
Reviews	Review ID	Integer
Reviews	Customer ID	Integer
Reviews	Score	Integer

Part 2: Data Generation and Management

2.1 Synthetic Data Generation

The project demands us to stimulate a real-world ecommerce data environment and hence requires us to generate a synthetic dataset. To maintain the integrity of original available products in market along with their exact product category with took the sample names of 1000 random products from Amazon-products which was sourced from Kaggle.com (Amazon Products Sales Dataset 2023 (kaggle.com). The rest of the entire datasets along with all its components were generated using R on posit cloud.

Customer Data Set

```
# Set seed for reproducible results
set.seed(123)

# Define first and last name functions
first_name_gen <- function() {
  sample(c("John", "Jane", "David", "Sarah", "Michael", "Jessica",
           "Robert", "Elizabeth", "Andrew", "Jennifer", "Emily", "Matthew",
           "Ryan", "Ashley", "Christopher", "Samantha", "Daniel", "Amanda",
           "William", "Stephanie"), 1)
}

last_name_gen <- function() {
  sample(c("Smith", "Johnson", "Williams", "Brown", "Jones", "Miller",
           "Davis", "Garcia", "Wilson", "Rodriguez", "Clark", "Lewis", "Walker",
           "Robinson", "Allen", "Young", "Moore", "Wright", "Nelson",
           "Scott"), 1)
```

```
}

# Generate initial random data
customer_id <- sample(1:1000, 1000, replace = FALSE)
first_name <- replicate(1000, first_name_gen())
last_name <- replicate(1000, last_name_gen())

# Generate other attributes (assuming you don't need unique values)
zip_code <- sample(100000:999999, 1000, replace = TRUE)
phone_numbers <- replicate(1000, {
  number <- sample(7000000000:7999999999, 1)
  number
})

password <- stringi::stri_rand_strings(1000, 8)

# Generate random street addresses (assuming US format)
street_number <- sample(1:100, 1000, replace = TRUE)
street_number <- as.numeric(street_number)
street_name <- sample(c("Main_St", "Elm_St", "Oak_St", "Maple_St", "Pine_St",
                        "Park_Ave", "Hilltop_Dr", "Lakeview_Blvd",
                       "Sunrise_Ave","Sunset Dr"), 1000, replace = TRUE)

# Function to generate email addresses
generate_email <- function(first_name, last_name,customer_id, domain) {
  paste(tolower(first_name), ".", tolower(last_name), ".",customer_id,
        "@", domain, sep = "")
}

domains <- c("gmail.com", "outlook.com", "yahoo.com", "hotmail.com",
             "icloud.com")

# Create the initial data frame
customer_data <- data.frame(
  customer_id,
  first_name,
  last_name,
  zip_code,
  street_number,
  street_name,
  phone_numbers,
  password,
```

```
  stringsAsFactors = FALSE
)

customer_data$email <- sapply(1:nrow(customer_data), function(i) {
  generate_email(customer_data$first_name[i],
                 customer_data$last_name[i],
                 customer_data$customer_id[i], sample(domains, 1))
})
```

Product Data Set

```
data <- read_csv("data_sample/Amazon-Products.csv")
data <- na.omit(data)
```

```
set.seed(123)
sample <- data[sample(nrow(data), size = 1000, replace = FALSE), ]
product_name <- sample$name
category_id <- sample(5000:10000, 1000, replace = FALSE)
price <- sample$actual_price
price <- as.numeric(gsub("\\D", "", price))

# Generate random data for each attribute
product_id <- sample(10000:100000, 1000, replace = FALSE)
quantity <- sample(1:1000, 1000, replace = TRUE)

supplier_id <- sample(1:100, 1000, replace = TRUE)


# Create the product_data data frame
product_data <- data.frame(
  product_id,
  product_name,
  quantity,
  price,
  supplier_id,
  stringsAsFactors = FALSE
)
```

Product Category Data Set

```r
category_name <- sample$sub_category
product_category_data <- data.frame(
  category_id,
  product_id,
  category_name
)
```

Logistic Data Set

```r
# Set seed for reproducible results
set.seed(123)

# Generate unique supplier IDs
supplier_id <- sample(1:100, 100, replace = FALSE)

# Initialize logistics_data dataframe
logistics_data <- data.frame(
  tracking_id = integer(1000),
  supplier_id = integer(1000),
  product_id = integer(1000),
  customer_id = integer(1000),
  status = character(1000),
  stringsAsFactors = FALSE
)

# Generate unique tracking IDs
unique_tracking_id_gen <- function(existing_ids = numeric(0)) {
  id <- sample(1:10000, 1)
  while (id %in% existing_ids) {
    id <- sample(1:10000, 1)
  }
  return(id)
}

existing_tracking_ids <- numeric(0)
for (i in 1:1000) {
  logistics_data$tracking_id[i] <-
    unique_tracking_id_gen(existing_tracking_ids)
  existing_tracking_ids <-
    c(existing_tracking_ids, logistics_data$tracking_id[i])
}
```

```r
# Assign product_id and supplier_id from product_data
for (i in 1:1000) {
  # Select a random product_id from product_data
  logistics_data$product_id[i] <- sample(product_data$product_id, 1)
  # Find the corresponding supplier_id for the selected product_id
  logistics_data$supplier_id[i] <-
    product_data$supplier_id[which(product_data$product_id ==
                                     logistics_data$product_id[i])[1]]
}

# Generate random customer IDs
logistics_data$customer_id <- sample(1:1000, 1000, replace = TRUE)

# Generate random status
logistics_data$status <- sample(c("In Transit", "Out for Delivery",
                                  "Delivered"),
                                1000, replace = TRUE)
```

Reviews Data Set

```r
# Set seed for reproducible results
set.seed(123)

# Function to generate unique review IDs
unique_review_id_gen <- function(existing_ids = character(0)) {
  id <- paste(sample(1:1000, 1), sep = "")
  while (id %in% existing_ids) {
    id <- paste(sample(1:1000, 1), sep = "")
  }
  return(id)
}

# Filter logistics data for purchased products by customers
purchased_products <- unique(logistics_data$customer_id)

# Generate random data for each attribute in the Reviews dataset
reviews_data <- data.frame(
  review_id = character(1000),  # Allocate space for 1000 elements
  product_id = integer(1000),
  customer_id = integer(1000),
  score = sample(1:5, 1000, replace = TRUE),  # Add score between 1 and 5
  stringsAsFactors = FALSE
```

```
)

# Generate unique review IDs
existing_review_ids <- character(0)
index <- 1
for (i in 1:length(purchased_products)) {
  customer_purchases <-logistics_data[logistics_data$customer_id ==
                                        purchased_products[i], "product_id"]
  for (j in 1:length(customer_purchases)) {
    reviews_data$review_id[index] <-
      unique_review_id_gen(existing_review_ids)
    reviews_data$product_id[index] <- customer_purchases[j]
    reviews_data$customer_id[index] <- purchased_products[i]
    existing_review_ids <-
      c(existing_review_ids, reviews_data$review_id[index])
    index <- index + 1
  }
}

# Remove extra rows if not all 1000 rows were filled
reviews_data <- reviews_data[1:index-1, ]
```

Supplier Data Set

```
set.seed(123)

supplier_names <- c(
  "ABC Suppliers", "XYZ Enterprises", "PQR Inc.", "LMN Corporation",
  "DEF Industries","GHI Corporation", "JKL Enterprises", "MNO Suppliers",
  "QRS Inc.", "TUV Industries","VWX Suppliers", "YZA Enterprises",
  "BCD Corporation", "EFG Inc.", "HIJ Industries","KLM Suppliers",
  "NOP Enterprises", "RST Corporation", "UVW Inc.", "WXY Industries",
  "YZA Suppliers", "BCD Enterprises", "EFG Corporation", "HIJ Inc.",
  "KLM Industries","NOP Suppliers", "RST Enterprises", "UVW Corporation",
  "WXY Inc.", "YZA Industries","BCD Suppliers", "EFG Enterprises",
  "HIJ Corporation", "KLM Inc.", "NOP Industries","RST Suppliers",
  "UVW Enterprises", "WXY Corporation", "YZA Inc.", "BCD Industries",
  "EFG Suppliers", "HIJ Enterprises", "KLM Corporation", "NOP Inc.",
  "RST Industries","UVW Suppliers", "WXY Enterprises", "YZA Corporation",
  "BCD Inc.", "EFG Industries","HIJ Suppliers", "KLM Enterprises",
  "NOP Corporation", "RST Inc.", "UVW Industries","WXY Suppliers",
  "YZA Enterprises", "BCD Corporation", "EFG Inc.", "HIJ Industries",
```

```
  "KLM Suppliers", "NOP Enterprises", "RST Corporation", "UVW Inc.",
  "WXY Industries","YZA Suppliers", "BCD Enterprises", "EFG Corporation",
  "HIJ Inc.", "KLM Industries","NOP Suppliers", "RST Enterprises",
  "UVW Corporation", "WXY Inc.", "YZA Industries","BCD Suppliers",
  "EFG Enterprises", "HIJ Corporation", "KLM Inc.", "NOP Industries",
  "RST Suppliers", "UVW Enterprises", "WXY Corporation", "YZA Inc.",
  "BCD Industries","EFG Suppliers", "HIJ Enterprises", "KLM Corporation",
  "NOP Inc.", "RST Industries","UVW Suppliers", "WXY Enterprises",
  "YZA Corporation", "BCD Inc.", "EFG Industries","HIJ Suppliers",
  "KLM Enterprises", "NOP Corporation", "RST Inc.", "UVW Industries"
)

supplier_first_names <- sub("\\s+.*", "", supplier_names)

# Generate unique supplier IDs
supplier_id <- seq(1, 100)

contact_number <- replicate(100, {
  numbers <- sample(7000000000:7999999999, 1)
  numbers
})

street_number <- sample(1:100, 100, replace = TRUE)
street_number <- as.numeric(street_number)
street_name <- sample(c("Main_St", "Elm_St", "Oak_St", "Maple_St",
                        "Pine_St","Park_Ave", "Hilltop_Dr",
                        "Lakeview_Blvd", "Sunrise_Ave", "Sunset Dr"),
                      100, replace = TRUE)

generate_email <- function(supplier_first_names, supplier_id, domain) {
  paste(tolower(supplier_first_names),".", supplier_id, "@",
        domain, sep = "")
}

domains <- c("gmail.com", "outlook.com", "yahoo.com", "hotmail.com",
             "icloud.com")

supplier_data <- data.frame(
  supplier_id = supplier_id,
  supplier_name = supplier_names,
  contact_number = contact_number,
  street_number = street_number,
```

```
  street_name = street_name
)

supplier_data$email <- sapply(1:nrow(supplier_data), function(i) {
  generate_email(supplier_first_names[i],supplier_id[i],
                 sample(domains, 1))
})
```

Order Data Set

```
# Assuming logistics_data exists with columns 'customer_id',
#'product_id', 'date_and_time'

# Generate quantity (random for demonstration)
quantity <- sample(1:10, 1000, replace = TRUE)

# Generate date_and_time (random for demonstration)
date_and_time <- as.POSIXct("2023-03-17 12:00:00") +
  sample(1:1000, 1000) * 21340

# Extract date and time as separate attributes
date <- as.Date(date_and_time)
time <- format(date_and_time, "%H:%M:%S")


# Create order dataset
order_data <- data.frame(
  customer_id = logistics_data$customer_id,
  product_id = logistics_data$product_id,
  quantity,
  date,
  time
)
```

Payment Data Set

```
# Set seed for reproducible results
set.seed(123)

# Generate random data for payment dataset
payment_data <- data.frame(
  payment_id =   sample(100000:500000, 1000, replace = FALSE),
```

```
  # Allocate space for 1000 IDs
  payment_amount = round(runif(1000, min = 10, max = 1000), 2),
  # Random amount between 10 and 1000
  date = order_data$date,
  time = order_data$time,
  stringsAsFactors = FALSE
)
```

```
# Save customer_data as CSV
write.csv(customer_data, "customer_data.csv", row.names = FALSE)

# Save logistics_data as CSV
write.csv(logistics_data, "logistics_data.csv", row.names = FALSE)

# Save payment_data as CSV
write.csv(payment_data, "payment_data.csv", row.names = FALSE)

# Save product_data as CSV
write.csv(product_data, "product_data.csv", row.names = FALSE)

# Save product_category_data as CSV
write.csv(product_category_data, "product_category_data.csv",
          row.names = FALSE)

# Save reviews_data as CSV
write.csv(reviews_data, "reviews_data.csv", row.names = FALSE)

# Save supplier_data as CSV
write.csv(supplier_data, "supplier_data.csv", row.names = FALSE)

# Save order-data as CSV
write.csv(order_data, "order_data.csv", row.names = FALSE)
```

2.2 Data Import and Quality Assurance

R scripts was created for validation, datasets have imported as dataframe and ran the validation check for each columns. Based on the validation result, dataframe was updated and the validated data was inserted into the database.

```
con <- dbConnect(RSQLite::SQLite(), dbname = "mydb.db")
```

```
customer <- readr::read_csv("customer_data.csv")
logistics <- readr::read_csv("logistics_data.csv")
orders <- readr::read_csv("order_data.csv")
product_category <- readr::read_csv("product_category_data.csv")
product <- readr::read_csv("product_data.csv")
reviews <- readr::read_csv("reviews_data.csv")
supplier <- readr::read_csv("supplier_data.csv")
payment <- readr::read_csv("payment_data.csv")
```

Create Customer Table

```
customer <- readr::read_csv("data_sample/customer_data.csv")
customer <- na.omit(customer)

dbExecute(con, "DROP TABLE IF EXISTS customer")

dbExecute(con, "CREATE TABLE customer (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_name VARCHAR(100),
    street_number INT,
    email VARCHAR(100),
    password VARCHAR(500),
    phone_numbers INT(10) NOT NULL,
    zip_code VARCHAR(6)
);")
```

```
RSQLite::dbWriteTable(con,"customer",customer,append=TRUE)
```

Validation Checks Customer Data

```
# Remove the duplicate entries, keeping the first occurrence
customer <- customer[!duplicated(customer$customer_id), ]

# First Name - Characters and Max Length
validate_firstname <- function(firstname) {
  !is.na(firstname) && all(grepl("^[[:alpha:]]+$", firstname)) &&
    nchar(firstname) <= 25
}
```

```r
# Apply the validation function to the CUSTOMER_FIRSTNAME column
valid_firstname <- sapply(customer$first_name, validate_firstname)

# Keep only the rows with valid first names
customer <- customer[valid_firstname, ]

# Last Name

validate_lastname <- function(lastname) {
  !is.na(lastname) &&
    all(grepl("^[-'[:alpha:][:space:]]+$", lastname)) &&
    nchar(lastname) <= 25
}

# Apply the validation function to the CUSTOMER_FIRSTNAME column
valid_lastname <- sapply(customer$last_name, validate_lastname)

# Keep only the rows with valid first and last names
customer <- customer[valid_lastname, ]

# Check if each entry in phone_numbers is numeric
valid_numeric <- function(phone) {
  !is.na(phone) && !is.na(as.numeric(phone)) && nchar(phone) ==10
}
num <- sapply(customer$phone_numbers, valid_numeric)

customer <- customer[num,]

# Check for duplicates in phone_numbers
customer <- customer[!duplicated(customer$phone_numbers), ]


# Email - Contains "@" and Valid Domain, should be unique

# Define the valid domain names
valid_domains <- c("gmail.com", "outlook.com", "yahoo.com",
                   "hotmail.com", "icloud.com")

# Function to check if the domain is valid
is_valid_domain <- function(domain) {
  domain %in% valid_domains
}
```

```r
# Function to extract domain from email and check if it's valid
valid_email <- function(email) {
  parts <- strsplit(email, "@")[[1]]
  if (length(parts) == 2) {
    is_valid_domain(parts[2])
  } else {
    FALSE
  }
}

# Filter out rows with invalid email domains
customer <- customer[sapply(customer$email, valid_email), , drop = FALSE]


# Check for duplicate rows based on email
customer <- customer[!duplicated(customer$email), ]

# Check if zip code is numeric and has length 6

valid_zip <- function(zipcode) {
  !is.na(zipcode) && !is.na(as.numeric(zipcode)) && nchar(zipcode) ==6
}
zip <- sapply(customer$zip_code, valid_zip)

customer <- customer[zip,]

# Check for numeric in street number
valid_street_num <- function(street_num) {
  !is.na(street_num) && !is.na(as.numeric(street_num)) &&
    as.numeric(street_num) > 0
}
street <- sapply(customer$street_number, valid_street_num)

customer <- customer[street,]
# Check for character in street name

valid_street_name <- function(street_name) {
  !is.na(street_name) && all(grepl("^[-_'[:alpha:] ]+$",
                                   street_name, ignore.case = TRUE))
}

valid_street <- sapply(customer$street_name, valid_street_name)
```

```
customer <- customer[valid_street, ]
```

Validation for Customer Data Set

Overall, the validation aims to clean and ensure the integrity of the customer data by:

1. Removes duplicates: Keeps only the first occurrence of each customer (based on ID). This ensures each customer record exists only once.
2. Checks names: Ensures names aren't empty, only contain letters, and have a maximum length.
3. Validates phone numbers: Ensures phone numbers aren't empty, contain only digits, and have a specific length (10 in this case). It also removes duplicates.
4. Cleans email addresses: Keeps only emails with valid domains (e.g., gmail.com) and removes duplicates.
5. Validates zip code, street number, and street name: Ensures these fields aren't empty, have the correct format (numeric for zip code and street number, characters for street name), and meet additional criteria (e.g., positive street number).

Create Supplier Table

```
supplier <- readr::read_csv("data_sample/supplier_data.csv")
supplier <- na.omit(supplier)


dbExecute(con, "DROP TABLE IF EXISTS supplier")

dbExecute(con, "CREATE TABLE supplier (
    supplier_id INT PRIMARY KEY,
    supplier_name VARCHAR(255),
    contact_number VARCHAR(20),
    street_number INT,
    street_name VARCHAR(50),
    email VARCHAR(500)
);")
```

```
RSQLite::dbWriteTable(con,"supplier",supplier,append=TRUE)
```

Validation Checks Supplier Table

```r
# Supplier ID - Uniqueness
supplier <- supplier[!duplicated(supplier$supplier_id), ]


# Supplier Name - Not Empty and Max Length
validate_name <- function(name) {
  !is.na(name) && all(grepl("^[[:alpha:]]+$", name)) && nchar(name) <= 25
}


# Street Number - Positive Integer

valid_street_num <- function(street_num) {
  !is.na(street_num) && !is.na(as.numeric(street_num)) &&
    as.numeric(street_num) > 0
}
sup_street <- sapply(supplier$street_number, valid_street_num)

supplier <- supplier[sup_street,]

# Street Name - Not Empty and Max Length

sup_street_name <- sapply(supplier$street_name, valid_street_name)

supplier <- supplier[sup_street_name, ]

# Contact Number - Numeric, Length and Uniqueness

contact <- sapply(supplier$contact_number, valid_numeric)

supplier <- supplier[contact,]

# Check for duplicates in phone_numbers
supplier <- supplier[!duplicated(supplier$contact_number), ]

# Email - Contains "@" and Valid Domain

# Filter out rows with invalid email domains
supplier <- supplier[sapply(supplier$email, valid_email), ,
                     drop = FALSE]

# Check for duplicate rows based on email
```

```
supplier <- supplier[!duplicated(supplier$email), ]
```

Validation for suppliers Data Set

1. Unique Suppliers: Only keeps the first instance of each supplier (based on ID).
2. Valid Names: Ensures names aren't empty, only have letters, and have a maximum length of 25 character.
3. Street & Contact: Checks Street numbers are positive integers and street/contact numbers aren't empty, are numeric, and have a specific length (likely 10). It removes duplicates for contact numbers.
4. Valid Emails: Keeps only emails with valid domains and removes duplicates.

Similar type of validation has been done to all the data sets as follows:

Create Product Table

```
product <- readr::read_csv("data_sample/product_data.csv")

dbExecute(con, "DROP TABLE IF EXISTS product")

dbExecute(con, "CREATE TABLE product (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(255),
    supplier_id INT,
    quantity INT,
    price DECIMAL(10, 2),
    FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id)
    );")
```

```
RSQLite::dbWriteTable(con,"product",product,append=TRUE)
```

Validation Checks Product Table

```
# Remove duplicate entries, keeping the first occurrence
product <- product[!duplicated(product$product_id), ]

# Product Name - Characters and Max Length
validate_productname <- function(productname) {
  !is.na(productname) && nchar(productname) <= 200
}

# Apply the validation function to the PRODUCT_NAME column
```

```r
valid_productname <- sapply(product$product_name, validate_productname)

# Keep only the rows with valid product names
product <- product[valid_productname, ]

# Supplier ID - Numeric
valid_supplier_id <- function(supplier_id) {
  !is.na(supplier_id) && !is.na(as.numeric(supplier_id))
}
supplier_id <- sapply(product$supplier_id, valid_supplier_id)

product <- product[supplier_id, ]

# Quantity - Numeric
valid_quantity <- function(quantity) {
  !is.na(quantity) && !is.na(as.numeric(quantity)) &&
    as.numeric(quantity) >= 0
}
quantity <- sapply(product$quantity, valid_quantity)

product <- product[quantity, ]

# Price - Numeric
valid_price <- function(price) {
  !is.na(price) && !is.na(as.numeric(price)) && as.numeric(price) >= 0
}
price <- sapply(product$price, valid_price)

product <- product[price, ]

# Create Reviews
reviews <- readr::read_csv("data.upload/reviews_data.csv")
reviews <- na.omit(reviews)

dbExecute(con, "CREATE TABLE IF NOT EXISTS reviews (
    review_id INT PRIMARY KEY,
    product_id INT,
    customer_id INT,
    score INT,
    FOREIGN KEY (product_id) REFERENCES orders(order_id),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
);")
```

```r
# Remove duplicate entries, keeping the first occurrence
reviews <- reviews[!duplicated(reviews$review_id), ]

# Product ID - Numeric
valid_product_id <- function(product_id) {
  !is.na(product_id) && !is.na(as.numeric(product_id))
}
product_id <- sapply(reviews$product_id, valid_product_id)

reviews <- reviews[product_id, ]

# Customer ID - Numeric
valid_customer_id <- function(customer_id) {
  !is.na(customer_id) && !is.na(as.numeric(customer_id))
}
customer_id <- sapply(reviews$customer_id, valid_customer_id)

reviews <- reviews[customer_id, ]

# Score - Numeric and within range (0-10)
valid_score <- function(score) {
  !is.na(score) && !is.na(as.numeric(score)) &&
    as.numeric(score) >= 0 && as.numeric(score) <= 10
}
score <- sapply(reviews$score, valid_score)

reviews <- reviews[score, ]
```

Create Orders Table

```r
dbExecute(con, "DROP TABLE IF EXISTS orders")

dbExecute(con, "CREATE TABLE orders (
    customer_id INT,
    product_id INT,
    quantity INT,
    date DATE,
    time TIME,
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);")
```

Validation Check for Order

```
# Remove duplicate entries, keeping the first occurrence
orders <- orders[!duplicated(orders[, c("customer_id", "date", "time")]), ]

# Customer ID - Numeric
valid_customer_id <- function(customer_id) {
  !is.na(customer_id) && !is.na(as.numeric(customer_id))
}
customer_id <- sapply(orders$customer_id, valid_customer_id)

orders <- orders[customer_id, ]

# Product ID - Numeric
valid_product_id <- function(product_id) {
  !is.na(product_id) && !is.na(as.numeric(product_id))
}
product_id <- sapply(orders$product_id, valid_product_id)

orders <- orders[product_id, ]

# Quantity - Numeric
valid_quantity <- function(quantity) {
  !is.na(quantity) && !is.na(as.numeric(quantity)) &&
    as.numeric(quantity) >= 0
}
quantity <- sapply(orders$quantity, valid_quantity)

orders <- orders[quantity, ]

# Check referential integrity for product table
invalid_supplier_ids <- !product$supplier_id %in% supplier$supplier_id
if (any(invalid_supplier_ids)) {
  cat("Foreign key violation: supplier_id in product table
      does not exist in supplier table. Removing invalid rows.\n")
  product <- product[!invalid_supplier_ids, ]
}

# Check referential integrity for reviews table
invalid_product_ids <- !reviews$product_id %in% product$product_id
if (any(invalid_product_ids)) {
  cat("Foreign key violation: product_id in reviews table
      does not exist in product table. Removing invalid rows.\n")
```

```
  reviews <- reviews[!invalid_product_ids, ]
}
invalid_customer_ids <- !reviews$customer_id %in% customer$customer_id
if (any(invalid_customer_ids)) {
  cat("Foreign key violation: customer_id in reviews table
      does not exist in customer table. Removing invalid rows.\n")
  reviews <- reviews[!invalid_customer_ids, ]
}

# Check referential integrity for product_category table
invalid_product_ids <- !product_category$product_id %in% product$product_id
if (any(invalid_product_ids)) {
  cat("Foreign key violation: product_id in product_category table
      does not exist in product table. Removing invalid rows.\n")
  product_category <- product_category[!invalid_product_ids, ]
}

# Check referential integrity for logistics table
invalid_supplier_ids <- !logistics$supplier_id %in% supplier$supplier_id
if (any(invalid_supplier_ids)) {
  cat("Foreign key violation: supplier_id in logistics table
      does not exist in supplier table. Removing invalid rows.\n")
  logistics <- logistics[!invalid_supplier_ids, ]
}
invalid_customer_ids <- !logistics$customer_id %in% customer$customer_id
if (any(invalid_customer_ids)) {
  cat("Foreign key violation: customer_id in logistics table
      does not exist in customer table. Removing invalid rows.\n")
  logistics <- logistics[!invalid_customer_ids, ]
}
invalid_product_ids <- !logistics$product_id %in% product$product_id
if (any(invalid_product_ids)) {
  cat("Foreign key violation: product_id in logistics table
      does not exist in product table. Removing invalid rows.\n")
  logistics <- logistics[!invalid_product_ids, ]
}

# Check referential integrity for orders table
invalid_customer_ids <- !orders$customer_id %in% customer$customer_id
if (any(invalid_customer_ids)) {
  cat("Foreign key violation: customer_id in orders table does not
      exist in customer table. Removing invalid rows.\n")
```

```
  orders <- orders[!invalid_customer_ids, ]
}
invalid_product_ids <- !orders$product_id %in% product$product_id
if (any(invalid_product_ids)) {
  cat("Foreign key violation: product_id in orders table does not
      exist in product table. Removing invalid rows.\n")
  orders <- orders[!invalid_product_ids, ]
}
```

```
RSQLite::dbWriteTable(con,"orders",orders,append=TRUE)
```

Create Reviews Table

```
dbExecute(con, "DROP TABLE IF EXISTS reviews")

dbExecute(con, "CREATE TABLE reviews (
    review_id INT PRIMARY KEY,
    product_id INT,
    customer_id INT,
    score INT,
    FOREIGN KEY (product_id) REFERENCES orders(order_id),
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
);")
```

Validation for review data

```
# Remove duplicate entries, keeping the first occurrence
reviews <- reviews[!duplicated(reviews$review_id), ]

# Product ID - Numeric
valid_product_id <- function(product_id) {
  !is.na(product_id) && !is.na(as.numeric(product_id))
}
product_id <- sapply(reviews$product_id, valid_product_id)

reviews <- reviews[product_id, ]

# Customer ID - Numeric
valid_customer_id <- function(customer_id) {
  !is.na(customer_id) && !is.na(as.numeric(customer_id))
}
```

```r
customer_id <- sapply(reviews$customer_id, valid_customer_id)

reviews <- reviews[customer_id, ]

# Score - Numeric and within range (0-10)
valid_score <- function(score) {
  !is.na(score) && !is.na(as.numeric(score)) && as.numeric(score) >=
    0 && as.numeric(score) <= 10
}
score <- sapply(reviews$score, valid_score)

reviews <- reviews[score, ]
```

```r
RSQLite::dbWriteTable(con,"reviews",reviews,append=TRUE)
```

Create Category Table

```r
dbExecute(con, "DROP TABLE IF EXISTS product_category")


dbExecute(con, "CREATE TABLE product_category (
    category_id INT PRIMARY KEY,
    product_id INT,
    category_name VARCHAR(255),
    FOREIGN KEY (product_id) REFERENCES product(product_id)
);")
```

Validate Category Table

```r
# Remove duplicate entries, keeping the first occurrence
product_category <-
  product_category[!duplicated(product_category$category_id), ]

# Product ID - Numeric
valid_product_id <- function(product_id) {
  !is.na(product_id) && !is.na(as.numeric(product_id))
}
product_id <- sapply(product_category$product_id, valid_product_id)

product_category <- product_category[product_id, ]
```

```r
# Category Name - Characters and Max Length
validate_category_name <- function(category_name) {
  !is.na(category_name) && nchar(category_name) <= 255
}

valid_category_name <- sapply(product_category$category_name,
                              validate_category_name)

product_category <- product_category[valid_category_name, ]
```

```r
RSQLite::dbWriteTable(con,"product_category",product_category,append=TRUE)
```

Create Payment Table

```r
dbExecute(con, "DROP TABLE IF EXISTS payment")

dbExecute(con, "CREATE TABLE payment (
    payment_id INT PRIMARY KEY,
    payment_amount DECIMAL(10, 2),
    date DATE,
    time TIME
);")
```

Validation for Payment

```r
# Remove duplicate entries, keeping the first occurrence
payment <- payment[!duplicated(payment$payment_id), ]

# Payment ID - Numeric
valid_payment_id <- function(payment_id) {
  !is.na(payment_id) && !is.na(as.numeric(payment_id))
}
payment_id <- sapply(payment$payment_id, valid_payment_id)

payment <- payment[payment_id, ]

# Payment Amount - Numeric and non-negative
valid_payment_amount <- function(payment_amount) {
  !is.na(payment_amount) && !is.na(as.numeric(payment_amount)) &&
    as.numeric(payment_amount) >= 0
}
```

```
payment_amount <- sapply(payment$payment_amount, valid_payment_amount)

payment <- payment[payment_amount, ]
```

```
RSQLite::dbWriteTable(con,"payment",payment,append=TRUE)
```

Create Supplies Table

```
dbExecute(con, "DROP TABLE IF EXISTS supplies")

dbExecute(con, "CREATE TABLE supplies (
    supplier_id INT,
    product_id INT,
    PRIMARY KEY (supplier_id, product_id),
    FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id),
    FOREIGN KEY (product_id) REFERENCES product(product_id)
);")
```

Create Logistics Table

```
dbExecute(con, "DROP TABLE if exists logistics")

dbExecute(con, "CREATE TABLE logistics (
    tracking_id INT PRIMARY KEY,
    status VARCHAR(50),
    product_id INT,
    supplier_id INT,
    customer_id INT,
    FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id),
    FOREIGN KEY (customer_id) REFERENCES supplier(customer_id),
    FOREIGN KEY (product_id) REFERENCES product(product_id)
);")
```

Validation for Logistics

```
# Remove duplicate entries, keeping the first occurrence
logistics <- logistics[!duplicated(logistics$tracking_id), ]

# Tracking ID - Numeric
valid_tracking_id <- function(tracking_id) {
  !is.na(tracking_id) && !is.na(as.numeric(tracking_id))
```

```
}
tracking_id <- sapply(logistics$tracking_id, valid_tracking_id)

logistics <- logistics[tracking_id, ]

# Status - Non-empty
valid_status <- function(status) {
  !is.na(status) && nchar(status) > 0
}
status <- sapply(logistics$status, valid_status)

logistics <- logistics[status, ]

# Product ID - Numeric
valid_product_id <- function(product_id) {
  !is.na(product_id) && !is.na(as.numeric(product_id))
}
product_id <- sapply(logistics$product_id, valid_product_id)

logistics <- logistics[product_id, ]

# Supplier ID - Numeric
valid_supplier_id <- function(supplier_id) {
  !is.na(supplier_id) && !is.na(as.numeric(supplier_id))
}
supplier_id <- sapply(logistics$supplier_id, valid_supplier_id)

logistics <- logistics[supplier_id, ]

# Customer ID - Numeric
valid_customer_id <- function(customer_id) {
  !is.na(customer_id) && !is.na(as.numeric(customer_id))
}
customer_id <- sapply(logistics$customer_id, valid_customer_id)

logistics <- logistics[customer_id, ]
```

```
#RSQLite::dbWriteTable(con,"logistics",logistics,append=TRUE)
```

Part 3: Data Pipeline Generation

3.1 Github Repository and Workflow Setup

We created the repository "Group7_IB9HP0", and each team member was granted access as a collaborator. To enable synchronization between the R project and the repository, a new token (classic) has been generated with the following scopes: repo, workflow, write:packages, and project.

A project called "New Project from Git Repository" has been configured in Posit Cloud, which comprises four directories:

- Data.upload: This folder includes CSV files with information for each table.
- R: This folder contains the R code that reads each CSV file and creates the database.
- Database: This folder contains the database created from various data files.
- Figures: this folder contains the plots that are generated from the analysis.

The R project has been committed and pushed from Posit Cloud to the GitHub repository, creating the same folders as mentioned above.

An action titled "workflow.yaml" has been established in GitHub, which runs every time there is a push event to the repository and every 3 hours. Initially, there were some errors when running the workflow, but after modifying the Workflow permissions to "Read and write permissions," it ran without any interruptions.

GitHub: https://github.com/TomasPDD/Group7_IB9HP0

3.2 Github Actions for Continuous Integration

```yaml
name: Workflow Group 7

on:
  schedule:
    - cron: '0 */3 * * *' # Run every 3 hours
  push:
    branches: [ main ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Setup R environment
        uses: r-lib/actions/setup-r@v2
        with:
          r-version: '4.2.0'
      - name: Cache R packages
        uses: actions/cache@v2
        with:
          path: ${{ env.R_LIBS_USER }}
          key: ${{ runner.os }}-r-${{ hashFiles('**/lockfile') }}
          restore-keys: |
            ${{ runner.os }}-r-
      - name: Install packages
        if: steps.cache.outputs.cache-hit != 'true'
        run: |
          Rscript -e 'install.packages(c("ggplot2","dplyr","readr","RSQLite","dbplyr"))'
      - name: Load initial data script
        run: |
          Rscript R/load_data.R
      - name: Execute data analysis
        run: |
          Rscript R/analysis.R

      - name: Add files
        run: |
          git config --global user.email "tomasp.dieguez@gmail.com"
          git config --global user.name "TomasPDD"
          git add --all figures/
      - name: Commit files
        run: |
          git commit -m "Updated database"
      - name: Push changes
        uses: ad-m/github-push-action@v0.6.0
        with:
            github_token: ${{ secrets.GITHUB_TOKEN }}
            branch: main
```

Our Action workflow.yaml will automatically start whenever there's a push (code commit) to the main branch:

1. Set up job: allocating resources and setting up the environment for the job to run.
2. Checkout code: retrieves the code from the GitHub repository and makes it available within the job's working directory.
3. Setup R environment: configures and installs the necessary R environment.
4. Cache R packages: attempts to reuse previously installed R packages to save time on repeated executions.
5. Install packages: installs required R packages if they are not already cached.
6. Load initial data script: executes R script load_data.R
7. Execute data analysis executes R script analysis.R
8. Add files: adds modified files to the Git staging area
9. Commit files: creates a Git commit
10. Push Changes: pushes the committed changes to GitHub repository.
11. Post Cache R packages: clean up the cached R package directory to manage storage space.
12. Post Checkout code: clean up temporary files or resources used during the code checkout process.
13. Complete job: marks the job as complete.

    load_data.R: the general function of the script is to load data from different CSV files into a new SQLite database and manage specific tables separately.

    1. Connects to the database: Establishes a connection to the SQLite database.
    2. Defines data tables: Creates a list that maps table names to their corresponding CSV file paths.
    3. Creates tables (if needed): Loops through the list and attempts to create tables in the database with names matching the list, handling potential cases where tables already exist.
    4. Reads CSV files: Iterates through the list again, reading data from each CSV file using readr::read_csv.

5. Writes data to tables: Overwrites existing data in the corresponding tables (created earlier) with the data read from the CSV files. (Some variations might choose to insert new data instead of overwriting.)

6. Prints data (optional): Depending on the code structure, it might print the contents of each table after loading the data.

7. Handles specific tables (optional): Some variations might include additional logic to handle specific tables differently, such as defining a custom schema or handling missing values before writing data.

8. Closes the connection: Ensures proper database connection closure.

9. Reports success (optional): Prints a message indicating successful data loading.

analysis.R: this code analyzes sales data stored in the SQLite database and creates various visualizations to understand customer behavior and product performance. It connects to the database, executes SQL queries to retrieve specific information, and then uses the ggplot2 library to create informative charts.

1. Customer Analysis: It calculates the number of returning and new customers based on their purchase history.

2. Product Category Analysis: It finds the average review score for each product category and visualizes the distribution.

3. Sales Analysis: It identifies the best-selling product category for each month and visualizes the total sales by month.

4. Payment Analysis: It explores the distribution of individual payment amounts and the total payment trend over time.

5. Order Analysis: It counts the number of orders placed in each month and presents this information in a bar chart.

6. Finally, the code saves each visualization as a separate PNG image file.

Part 4: Data Analysis

4.1 Advanced Data Analysis

```
RSQLite::dbListTables(con)
```

Data Visualization

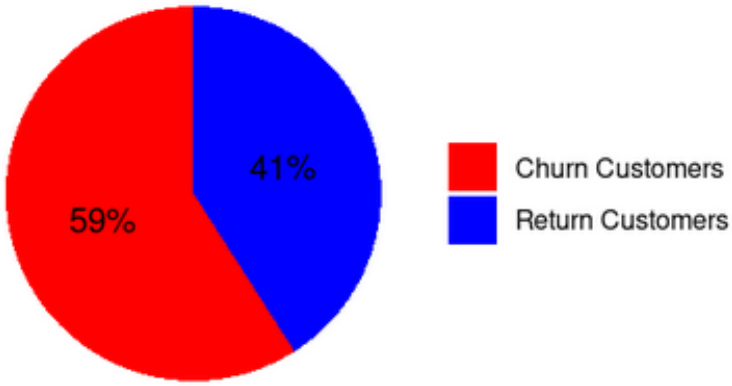
Retention Rate

```
# SQL query to count old customers
#(assuming 'old customers' have ordered more than once)
query_old_customers <- "SELECT COUNT(*) as count
FROM (SELECT customer_id FROM orders
GROUP BY customer_id HAVING COUNT(product_id) > 1)"
old_customers <- dbGetQuery(con, query_old_customers)$count

# SQL query to count new customers
# (assuming 'new customers' have ordered only once)
query_new_customers <- "SELECT COUNT(*) as count
FROM (SELECT customer_id FROM orders
GROUP BY customer_id HAVING COUNT(product_id) = 1)"
new_customers <- dbGetQuery(con, query_new_customers)$count

# Create a data frame for the pie chart
pie_data <- data.frame(
  Category = c("Return Customers", "Churn Customers"),
  Count = c(old_customers, new_customers)
)

# Create the pie chart
ggplot(pie_data, aes(x = "", y = Count, fill = Category)) +
  geom_bar(width = 1, stat = "identity") +
  coord_polar("y", start = 0) +
  theme_void() +
  theme(legend.title = element_blank()) +
  labs(fill = "Customer Type") +
  scale_fill_manual(values = c("Return Customers" = "blue",
                               "Churn Customers" = "red")) +
  geom_text(aes(label = scales::percent(Count / sum(Count))),
            position = position_stack(vjust = 0.5))
```



The e-commerce platform boasts a high customer retention rate, with 41% returning additional orders (defined as "returned customers"). Prioritizing retention fosters loyalty, boosts customer lifetime value, and fuels sustainable revenue growth. Retained customers provide repeat business, valuable insights into preferences, and purchase history. This data empowers personalized marketing, product recommendations, and tailored shopping experiences, further strengthening retention and satisfaction.

Distribution of Payment Amounts

```
payment_sql <- "SELECT * FROM payment"

# Execute the query and fetch the results
payment <- dbGetQuery(con, payment_sql)

ggplot(payment, aes(x = payment_amount)) +
  geom_histogram(bins = 30, fill = "blue", color = "black") +
  labs(title = "Distribution of Payment Amounts",
       x = "Payment Amount",
       y = "Frequency") +
  theme_minimal()
```

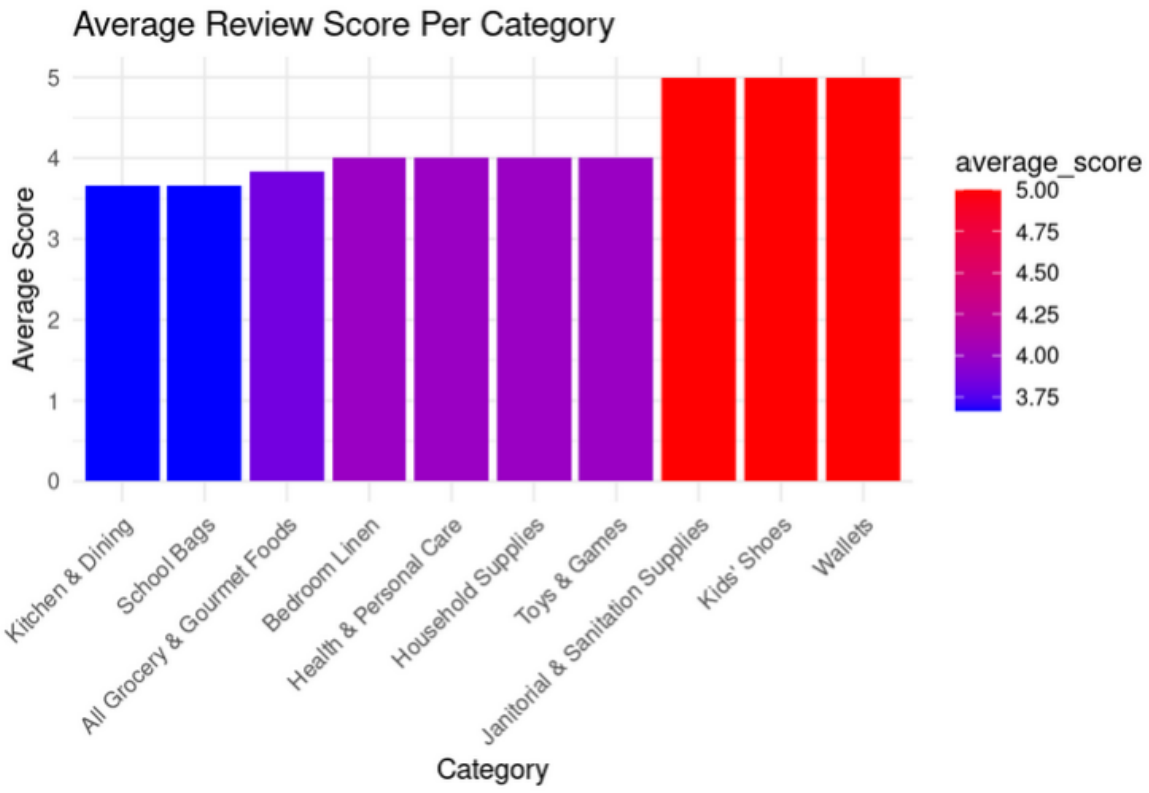
By analyzing how much customers spend, businesses can group them based on spending habits and identify times or products associated with peak spending. This allows for targeted marketing campaigns, improved inventory management, and the development of effective pricing strategies based on customer response to different price points. Interestingly, the data shows a relatively even distribution of order prices across various price levels.

Average Review Score Per Category

```
# Write your SQL query as a string
average_scores_per_category_sql <- "
SELECT
  cp.category_name,
  AVG(r.score) AS average_score
FROM
  reviews AS r
LEFT JOIN
  product_category AS cp
ON
  r.product_id = cp.product_id
GROUP BY
  cp.category_name
ORDER BY
  average_score DESC
LIMIT 10
"
# Execute the query and fetch the results
average_scores_per_category <-
  dbGetQuery(con, average_scores_per_category_sql)

ggplot(average_scores_per_category,
       aes(x = reorder(category_name, average_score),
           y = average_score, fill = average_score)) +
  geom_col() +
```

```
  scale_fill_gradient(low = "blue", high = "red") +
  labs(title = "Average Review Score Per Category",
       x = "Category",
       y = "Average Score") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Wallets, Kid's Shoes, and Janitorial Supplies top the charts in average review scores, reflecting high customer satisfaction in these categories. By tracking review trends over time, businesses can assess overall product quality, identify areas needing improvement, and implement corrective measures.

Best-Selling Product Categories by Month

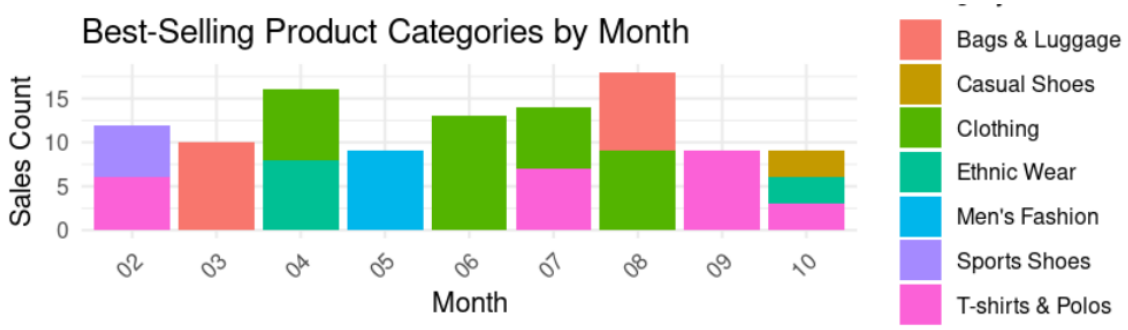
```
# SQL query to aggregate sales by category and month
sales_by_category_month_sql <- "
  SELECT
    strftime('%Y-%m', o.date) AS month,
    pc.category_name,
    COUNT(*) AS sales_count
```

```
  FROM
    orders o
    LEFT JOIN product p ON o.product_id = p.product_id
    LEFT JOIN product_category pc ON p.product_id = pc.product_id
  GROUP BY
    month,
    pc.category_name
"
# Execute the query and store the results in an R data frame
sales_by_category_month <- dbGetQuery(con, sales_by_category_month_sql)

# SQL query to identify the best-selling category for each month
best_sellers_by_month_sql <- "
  WITH MonthlySales AS (
    SELECT
      strftime('%m', o.date) AS month,
      pc.category_name,
      COUNT(*) AS sales_count
    FROM
      orders o
      LEFT JOIN product p ON o.product_id = p.product_id
      LEFT JOIN product_category pc ON p.product_id = pc.product_id
    GROUP BY
      month,
      pc.category_name
  ),
  RankedSales AS (
    SELECT
      month,
      category_name,
      sales_count,
      RANK() OVER (PARTITION BY month ORDER BY sales_count DESC) as rank
    FROM
      MonthlySales
  )
  SELECT
    month,
    category_name,
    sales_count
  FROM
    RankedSales
  WHERE
```

```
    rank = 1
"
# Execute the query and store the results in an R data frame
best_sellers_by_month <- dbGetQuery(con, best_sellers_by_month_sql)

# Use the formatted_month directly for plotting
# as it is already in 'YYYY-MM' format
ggplot(best_sellers_by_month, aes(x = month,
                                  y = sales_count,
                                  fill = category_name)) +
  geom_col() +
  labs(title = "Best-Selling Product Categories by Month",
       x = "Month",
       y = "Sales Count") +
  scale_fill_discrete(name = "Category") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Clothing (including T-shirts & Polos) led sales for 4 months, followed by bags & luggage and ethnic wear. This insight empowers e-commerce businesses to target marketing and promotions based on customer preferences for these top categories.

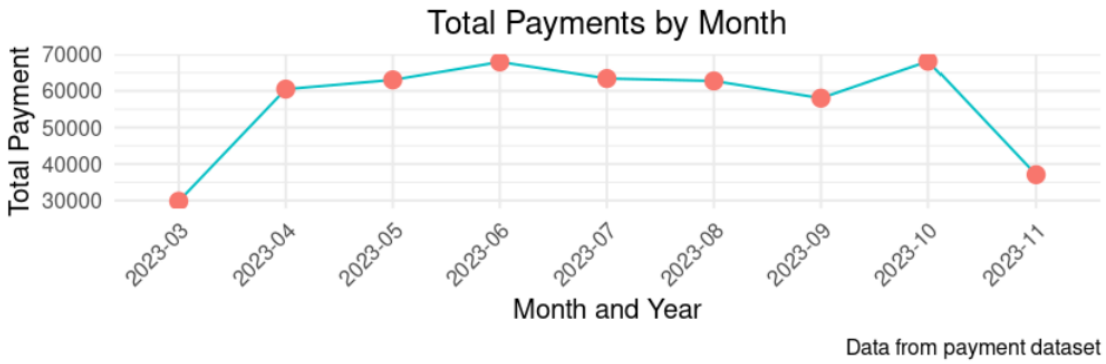
Total Payments by Month

```
# Convert date column to Date type if it's not already
payment$date <- as.Date(payment$date)

query1 <- "
SELECT
  strftime('%m', date) AS month_year,
  SUM(payment_amount) AS total_payment
```

```
FROM payment
GROUP BY month_year
"
# Execute the query and fetch results
monthly_payments <- dbGetQuery(con, query1)

# Plotting the graph
ggplot(monthly_payments, aes(x = month_year,
                             y = total_payment, group = 1)) +
  geom_line(color = "#00BFC4") +
  # Line with a specific color
  geom_point(color = "#F8766D", size = 3) +
  # Points with a specific color and size
  theme_minimal() +
  labs(title = "Total Payments by Month",
       x = "Month",
       y = "Total Payment",
       caption = "Data from payment dataset") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        # Rotate x-axis labels
        plot.title = element_text(hjust = 0.5))
        # Center the title
```



Number of Orders per Month

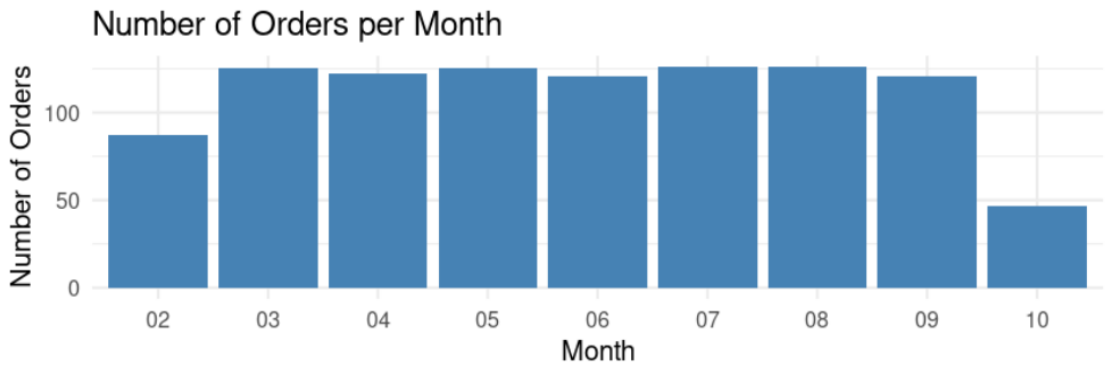
```
# Convert 'date' to Date format if it's not already
orders$date <- as.Date(orders$date)

# SQL query to extract month from date and count orders
query <- "
```

```
SELECT
    strftime('%m', date) AS month,
    COUNT(*) AS number_of_orders
FROM
    orders
GROUP BY
    month
ORDER BY
    month
"

# Execute the query and fetch results
monthly_orders <- dbGetQuery(con, query)

# Create the plot
ggplot(monthly_orders, aes(x = month, y = number_of_orders)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  theme_minimal() +
  labs(title = "Number of Orders per Month",
       x = "Month",
       y = "Number of Orders")
```



The graph indicates that the total revenue does not change significantly over the years. The higher the number of orders is, the higher the revenue is. It also means the value of each order has no big variance.

Tracking order volume and payments helps businesses assess sales trends, identify patterns, and monitor progress towards revenue goals. This allows us to optimize stock levels, prevent stockouts, and reduce costs from excess inventory.

Implications

The report provides several implications related to the stated objectives. Firstly, it offers insights into relational database design and management through the example of an e-commerce database, covering entity-relationship diagram design and SQL schema creation. Secondly, it demonstrates the use of GitHub Actions to automate data-related tasks, including data loading, validation, and version control. This allowed us to gain proficiency in automating workflows and leveraging GitHub's capabilities. Lastly, the report showcases data analysis and report generation using R and Quarto, with examples of synthetic data generation, data import, and quality assurance. By examining the provided R scripts and data pipeline generation, we can enhance our skills in data analysis, quality assurance, and report generation. Overall, the implications highlight the acquisition of knowledge and skills in relational database design, automation with GitHub Actions, and data analysis using R and Quarto.