**Table of contents**

[Introduction 2](#_Toc165900805)

[Part 1: Database Design and Implementation 2](#_Toc165900806)

[1.1 : E-R Diagram Design 2](#_Toc165900807)

[1.2 : SQL Database Schema Creation 6](#_Toc165900808)

[Part 2: Data Generation and Management 11](#_Toc165900809)

[2.1 : Synthetic Data Generation 11](#_Toc165900810)

[2.2 : Data Import and Quality Assurance 16](#_Toc165900811)

[Part 3: Data Pipeline Generation 24](#_Toc165900812)

[3.1 : GitHub Repository and Workflow Setup 24](#_Toc165900813)

[3.2 : GitHub Actions for Continuous Integration 24](#_Toc165900814)

[Part 4: Advanced Data Analysis 27](#_Toc165900815)

[Conclusion 32](#_Toc165900816)

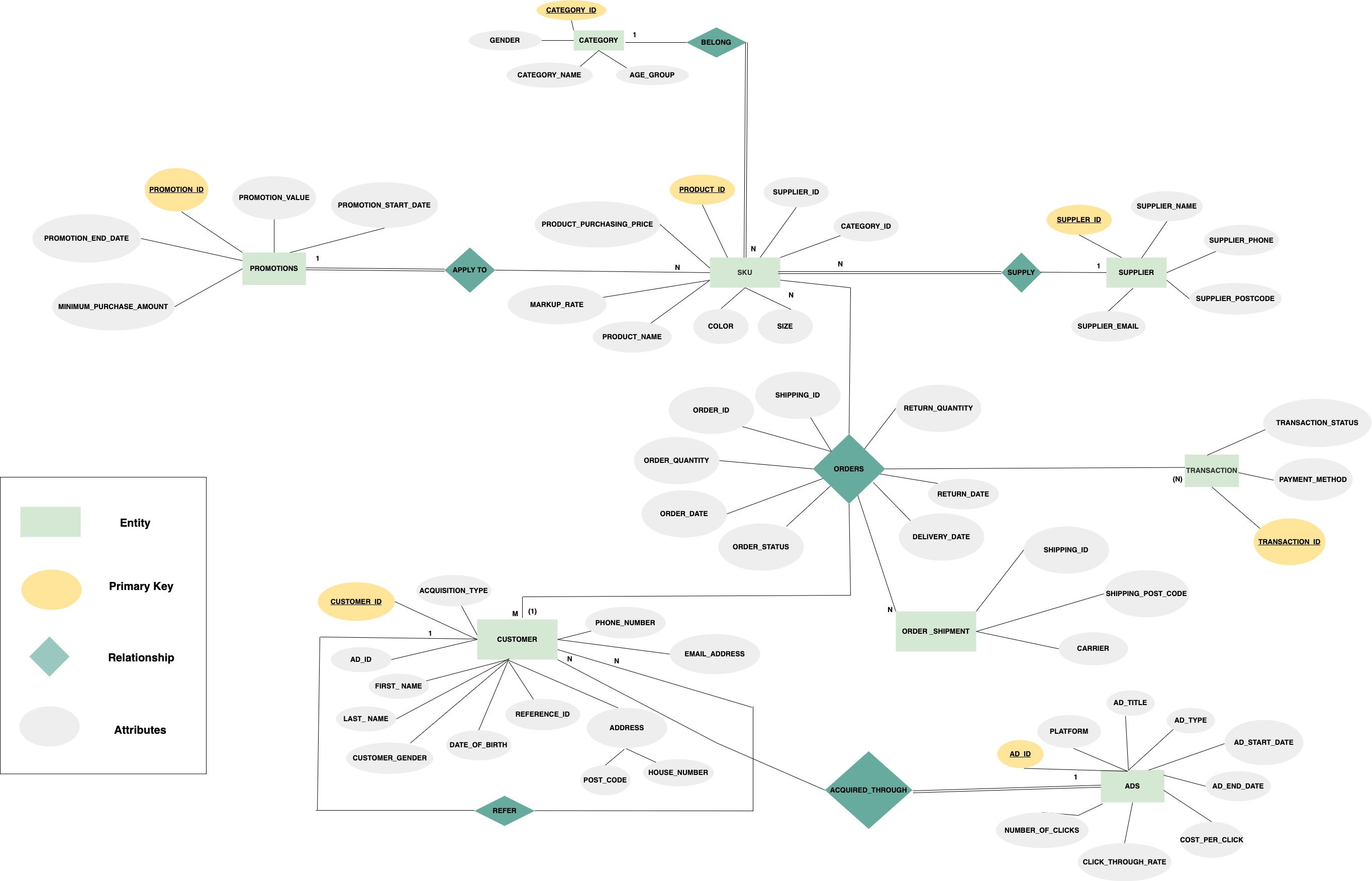
[References 33](#_Toc165900817)

# Introduction

This project undertook a comprehensive approach to data management for an apparel e- commerce retailer based in the UK, encompassing database design, data analysis, and re- porting. The database architecture was structured around eight entities including products, customers, shipments, promotions, advertisements, suppliers, categories, and transactions, depicted through an Entity-Relationship (ER) diagram. An SQL schema was then implemented to substantiate this design. R was employed to generate synthetic data aligned with our schema to simulate real-world retail transactions. Rigorous data quality assurance was con- ducted before writing the data into the database. Subsequently, Quarto with R was utilised for data analysis to offer practical insights for strategic decision-making. Automation of data validation, loading, and analysis processes was achieved through a GitHub workflow, ensuring collaborative oversight and accountability throughout the project lifecycle.

# Part 1: Database Design and Implementation

## : E-R Diagram Design



The E-R diagram contained eight entities and relationships, depicting the operational structure of an apparel e-commerce retailer based in the UK. It illustrated the journey of an order starting from a registered customer selecting a product (SKU), through to the order’s delivery, and handling of any subsequent returns. It also covered the process of customer acquisition through three main channels, organic, advertisement and referrals. Please refer to Appendix 1 for the definition of each attribute of each entity.

We made the following assumptions:

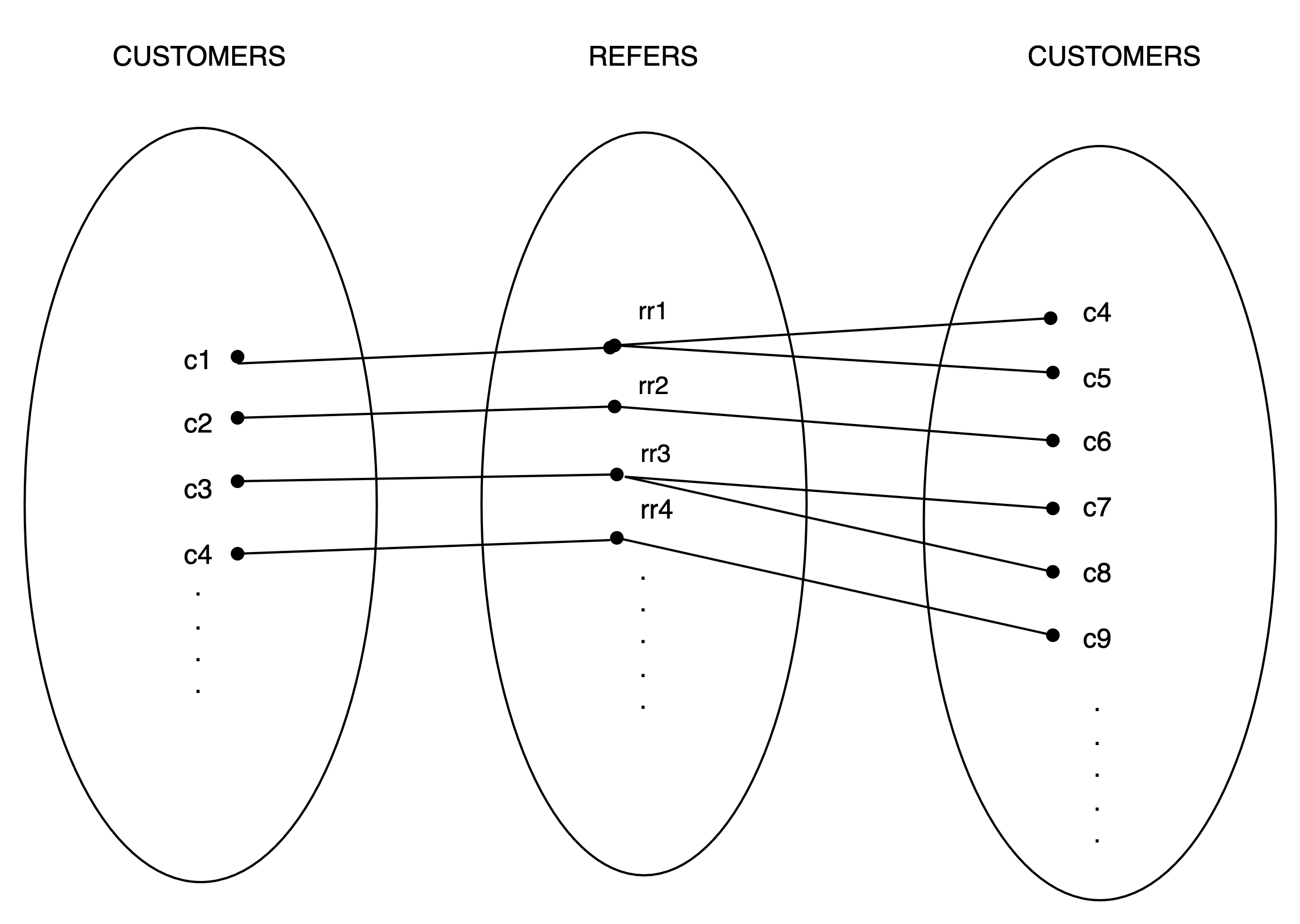
* + - Product is tracked at the SKU level.
    - Promotion codes are applied at SKU level.
    - Customers must register before placing orders.
    - Third-party analytics (e.g., Apps flyer) is used for marketing and customer attribution, thus we can track paid customers are acquired from which ads.
    - One order can be paid by only one transaction.
    - One new customer can be referred by only one existing customer.
    - Customers can only return the order within 30 days from order placement date.
    - Every order will be delivered in one shipment.

**Relationships Between Entities:**

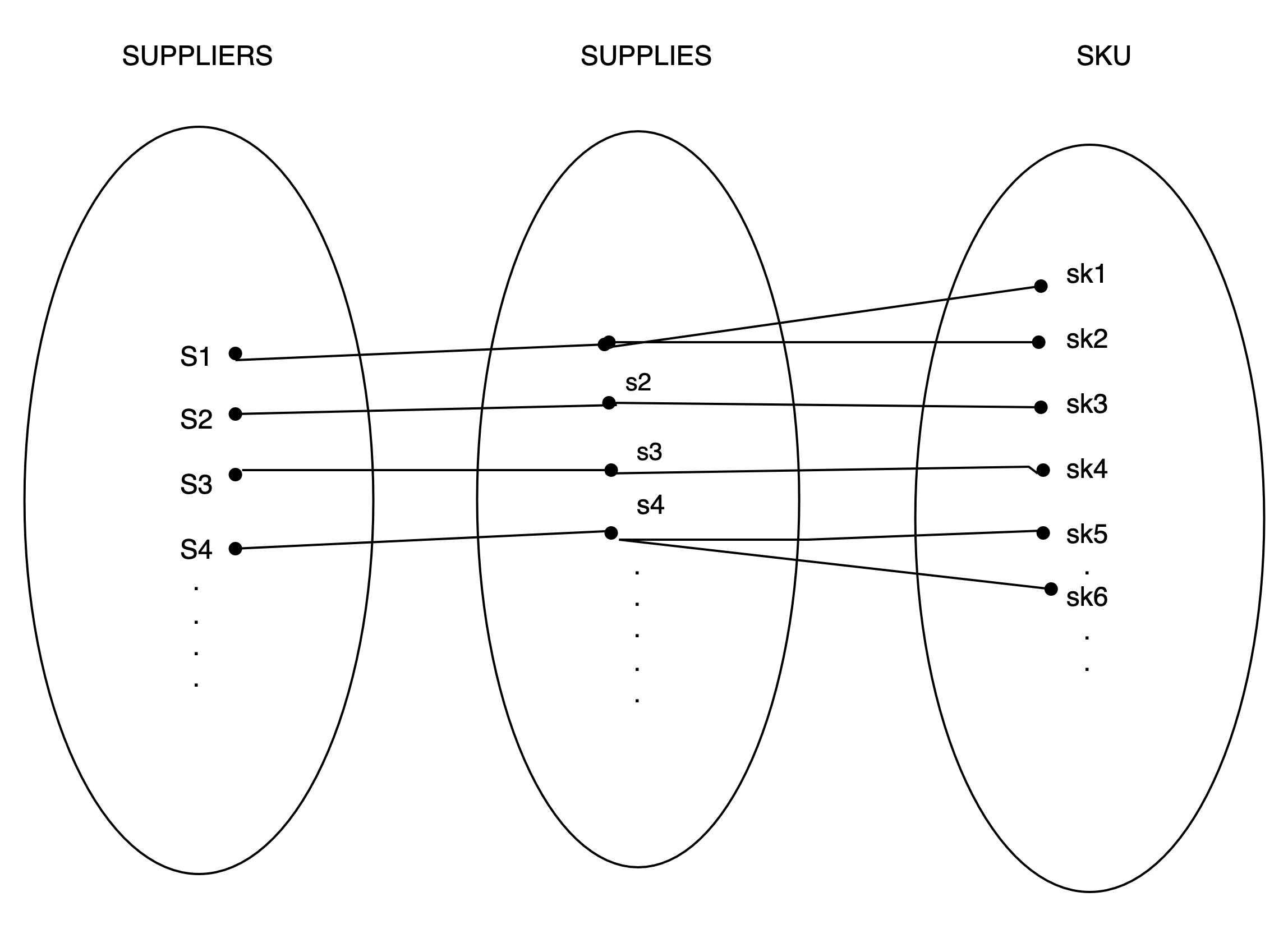
Our E-R diagram had two types of relationships between different entities, many-to-one and many-to-many.

**One-to-many:**

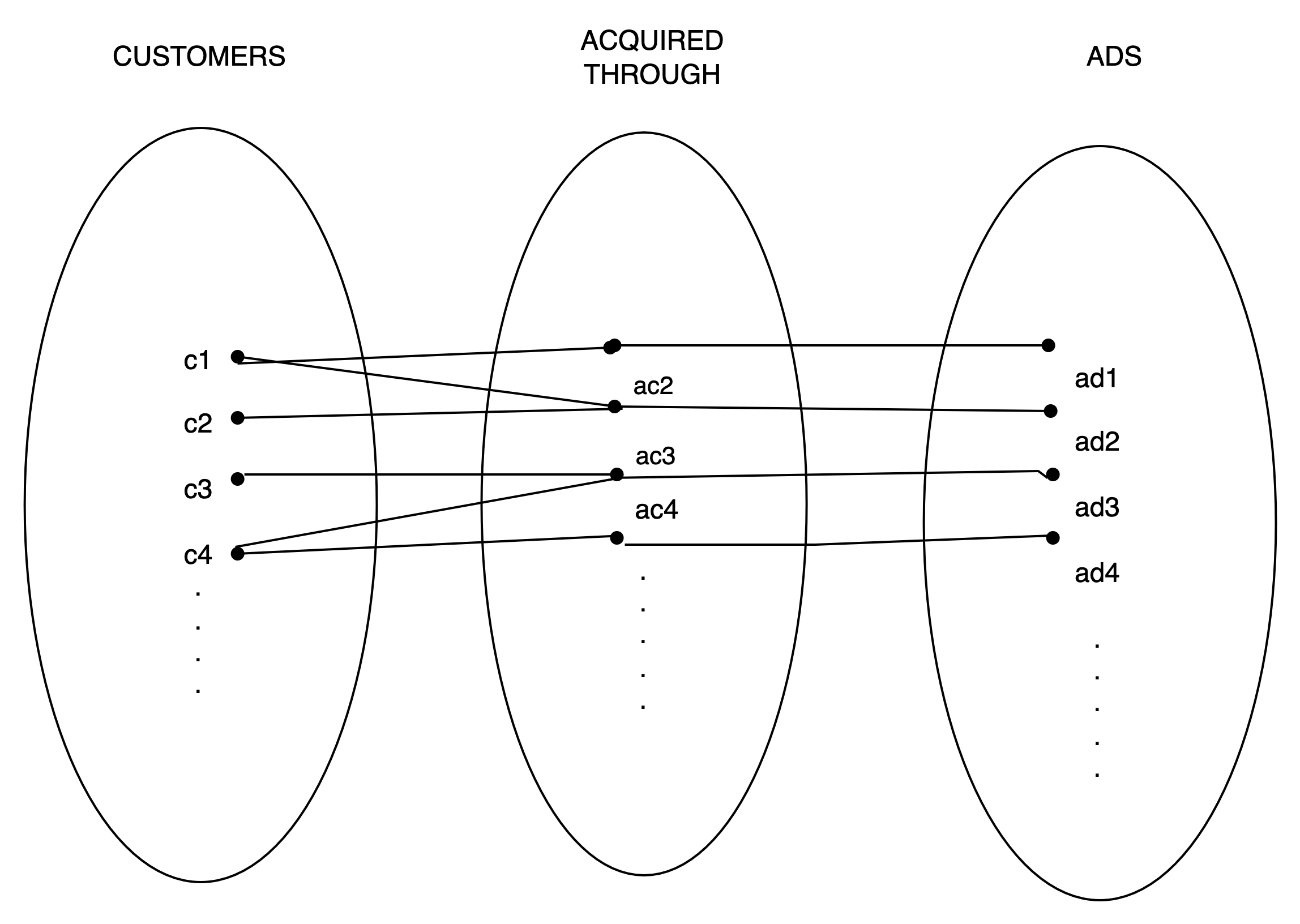
* + - CUSTOMERS refers CUSTOMERS: A self-recursive relationship, in which one existing customer can refer many new customer, but one new customer can only be referred by one existing customer.



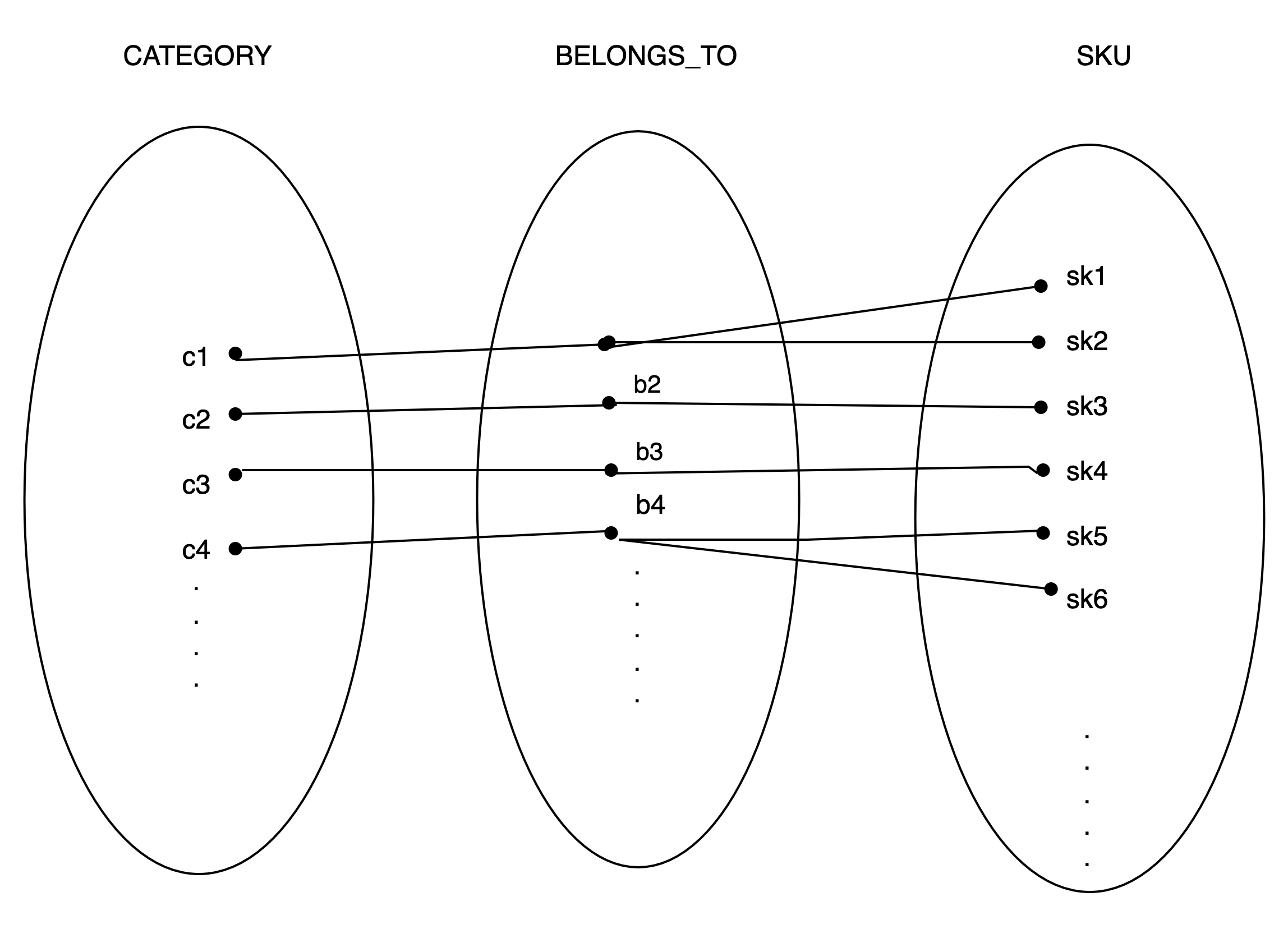
* + - SUPPLIERS supply SKU: One supplier can supply many products while one SKU is only provided by only one supplier.



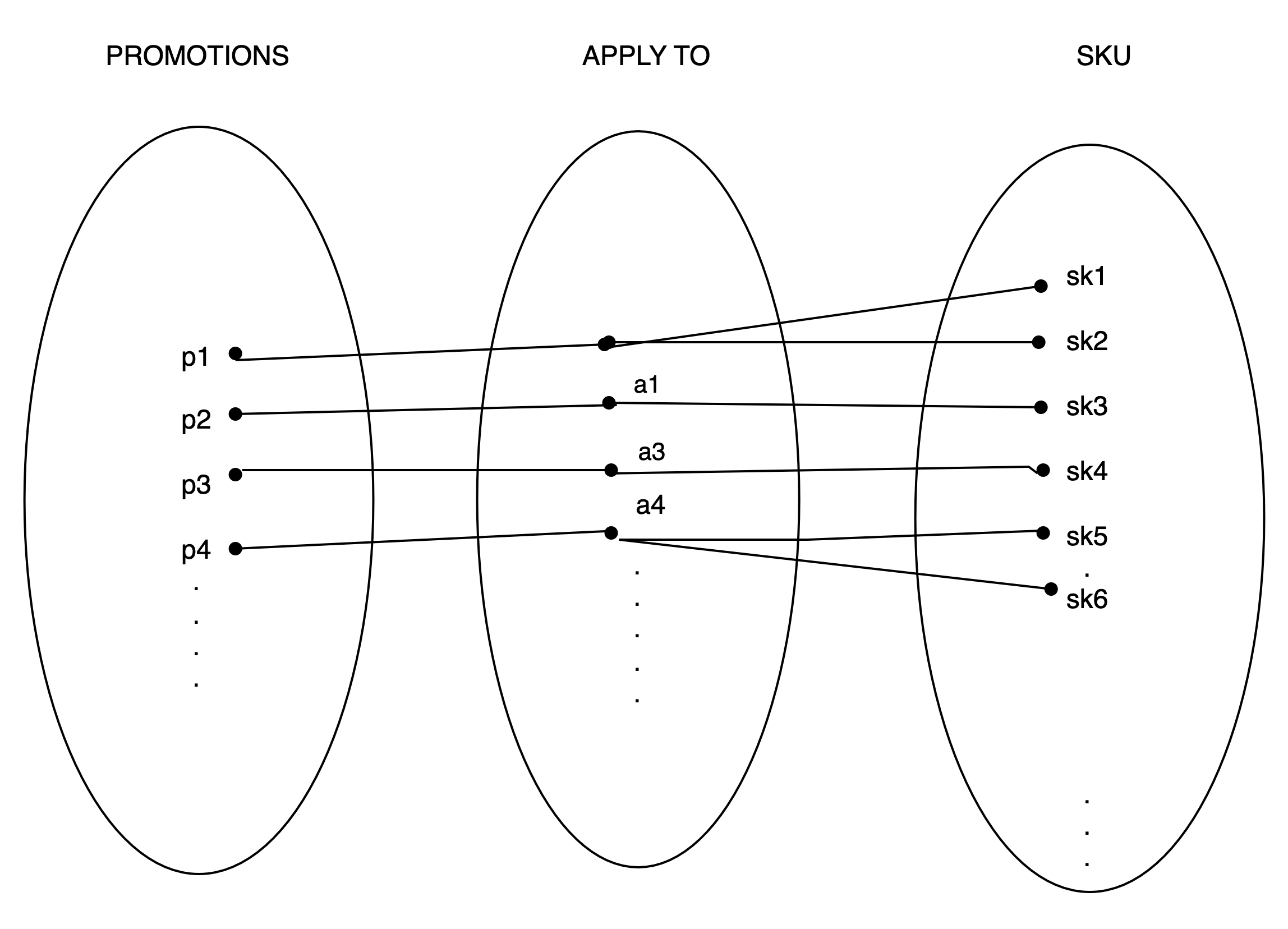
* + - CUSTOMERS acquired through ADS: One advertisement can be used to acquire many customers whilst one customer can only be acquired through one advertisement.



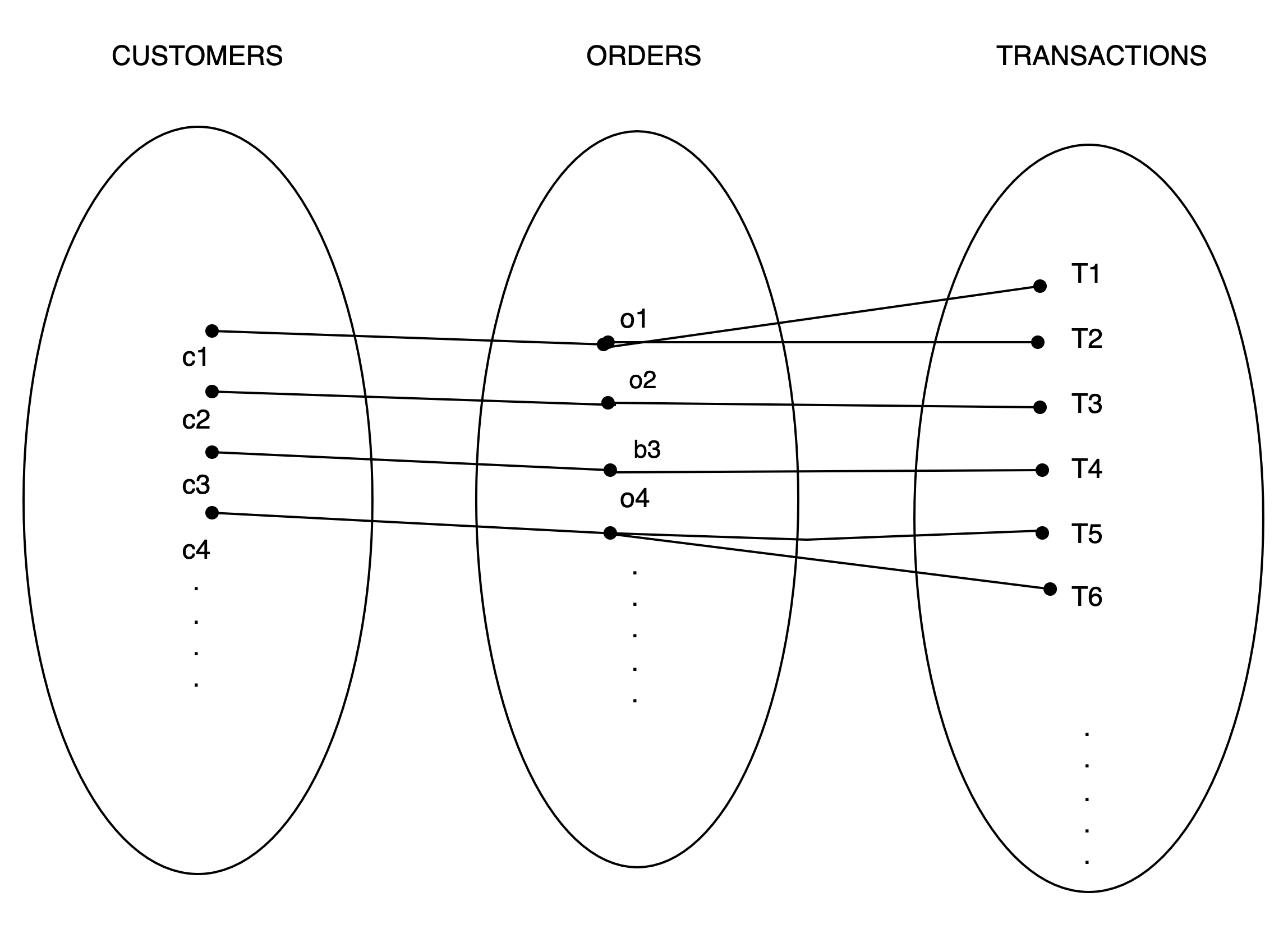
* + - SKU belongs to CATEGORY: Many products can belong to one category.



* + - PROMOTIONS apply to SKU: One promotion can be applied to many products.

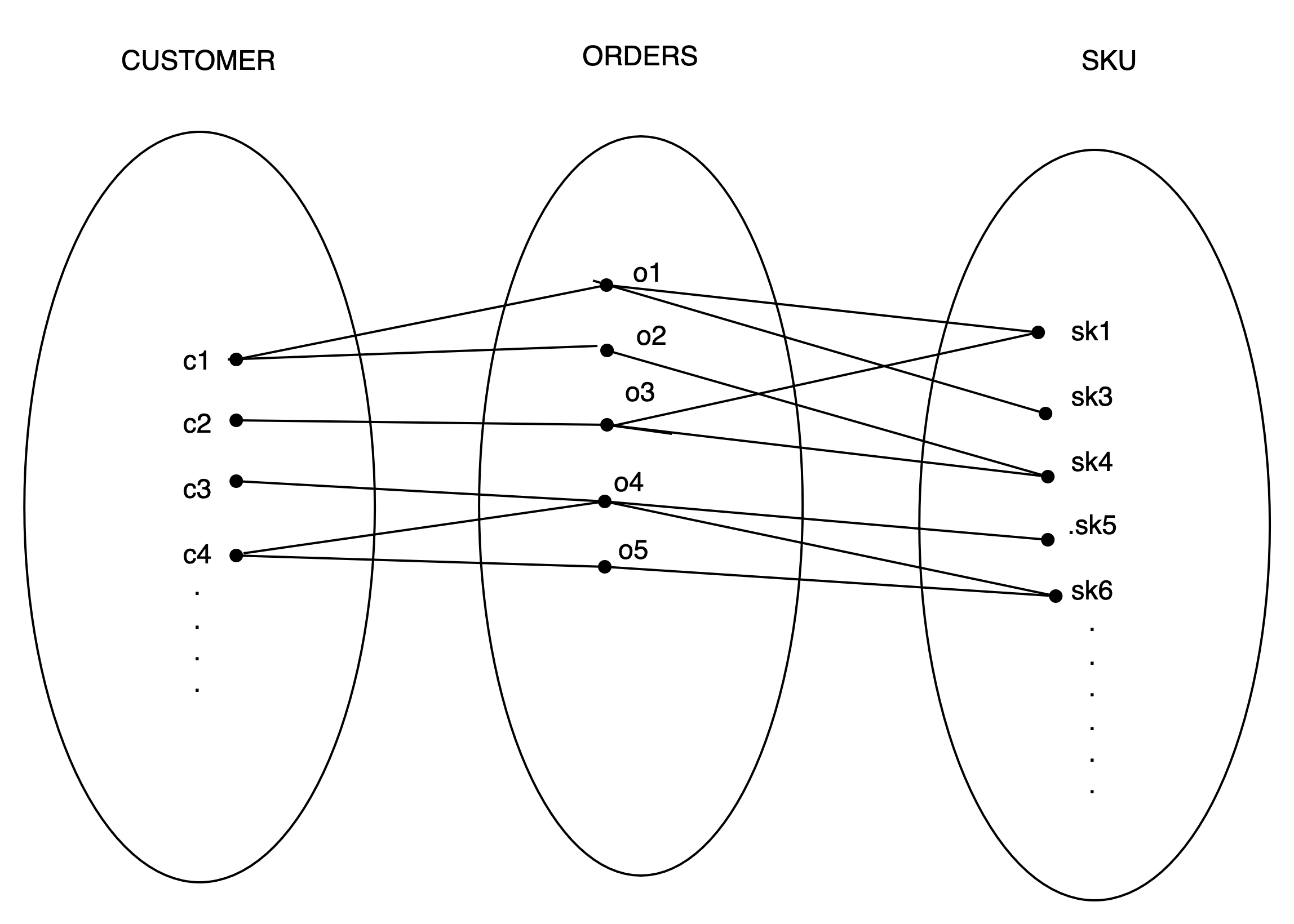


* + - CUSTOMERS pay TRANSACTIONS: one transaction can only be paid by one customer while one customer can pay multiple transactions.

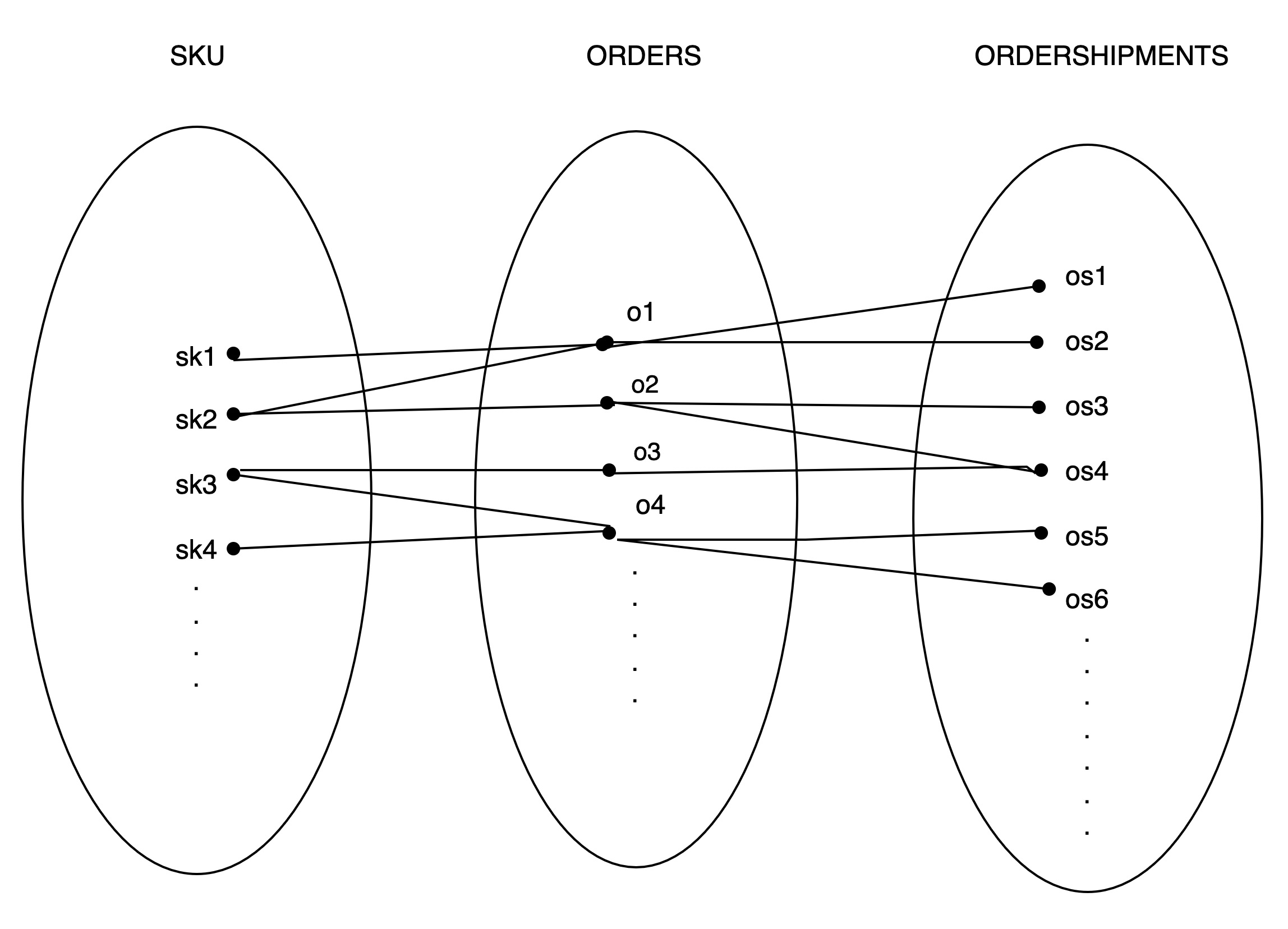


**Many-to-many:**

* + - CUSTOMERS order SKU: One customer can order many products and one product can be ordered multiple times.



* + - SKU delivered through ORDER\_SHIPMENT: Many products can have multiple order shipments.



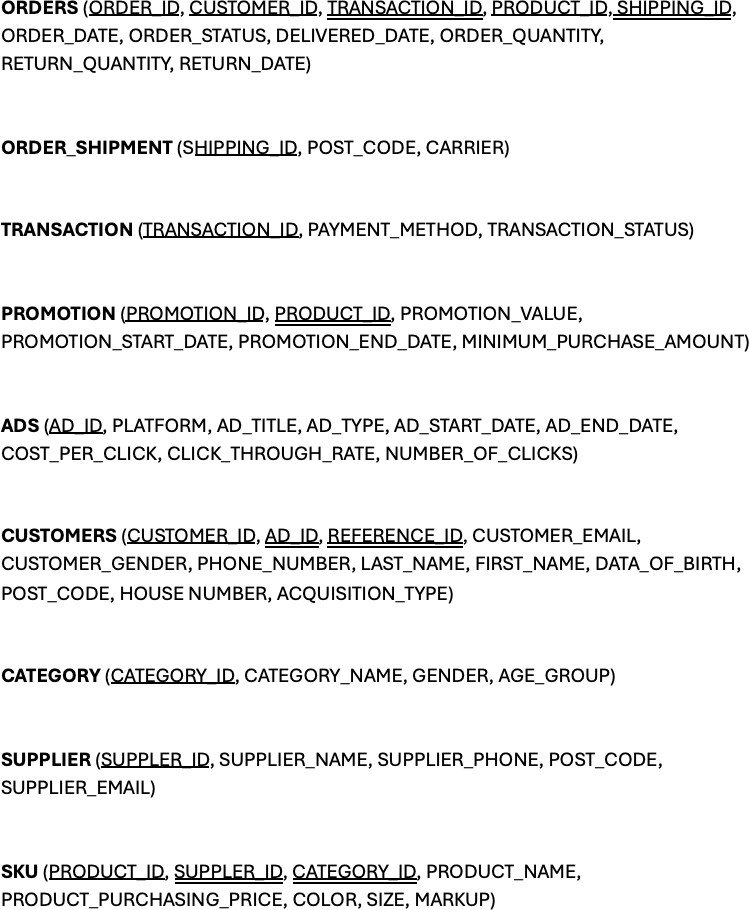
## : SQL Database Schema Creation

In creating logical and physical schema, to satisfy 3NF normalisation, we ensured that (1) each attribute was single-value, (2) each table had its unique primary key, (3) all attributes in all tables represented distinctive information, (4) no inferred data was stored (e.g. calculated field), (5) in a table, except for the primary key, other attributes were not dependent on each other. Subsequently, considerations were made for data volume and performance requirements, selecting appropriate data types and indexes to optimise query performance. Based on these designs, the physical schema was created using the SQL, including table structures, indexes,

and constraints. Notably, as ORDER is a many-to-many relationship, an ORDERS table was created in the logical and physical schema.

To derive the physical schema of the database, entity-relationship modeling and normalization were conducted initially. This involved identifying entities, attributes, and relationships, ensuring that the data model adhered to third normal form (3NF). Subsequently, considerations were made for data volume and performance requirements, selecting appropriate data types and indexes to optimize query performance. Finally, based on these designs, the physical schema was created using the SQL language, including table structures, indexes, and constraints.

**Logical Schema**



**Physical Schema**

rm(list=ls()) library(readr) library(RSQLite) library(dplyr)

#Creating a connection to a database

my\_db <- RSQLite::dbConnect(RSQLite::SQLite(),"/cloud/project/ecommerce.db")

* + - ADS

CREATE TABLE ADS(

AD\_ID VARCHAR(30) PRIMARY KEY, PLATFORM VARCHAR(255), AD\_TITLE VARCHAR(255),

AD\_TYPE VARCHAR(70), AD\_START\_DATE VARCHAR(15), AD\_END\_DATE VARCHAR(15), COST\_PER\_CLICK FLOAT, CLICK\_THROUGH\_RATE FLOAT, NUMBER\_OF\_CLICK FLOAT

);

* + - CATEGORY

CREATE TABLE CATEGORY( CATEGORY\_NAME TEXT NOT NULL, GENDER TEXT,

AGE\_GROUP VARCHAR(30),

CATEGORY\_ID VARCHAR(30) PRIMARY KEY

);

* + - SUPPLIER

CREATE TABLE SUPPLIER ( SUPPLIER\_NAME VARCHAR(50), SUPPLIER\_EMAIL VARCHAR(30), SUPPLIER\_PHONE NUMERIC(10), POST\_CODE VARCHAR(30),

SUPPLIER\_ID VARCHAR(30) PRIMARY KEY

);

* + - CUSTOMERS

CREATE TABLE IF NOT EXISTS CUSTOMERS( CUSTOMER\_ID VARCHAR(30) PRIMARY KEY, ACQUISITION\_TYPE TEXT,

REFERENCE\_ID VARCHAR(30), PHONE\_NUMBER NUMERIC(10), CUSTOMER\_GENDER TEXT, DATE\_OF\_BIRTH VARCHAR(15), FIRST\_NAME TEXT,

LAST\_NAME TEXT, CUSTOMER\_EMAIL VARCHAR(30), POST\_CODE VARCHAR(30), HOUSE\_NUMBER INT,

AD\_ID VARCHAR(30),

FOREIGN KEY (AD\_ID) REFERENCES ADS (AD\_ID)

FOREIGN KEY (REFERENCE\_ID) REFERENCES CUSTOMER(CUSTOMER\_ID)

);

* + - PRODUCT

CREATE TABLE IF NOT EXISTS SKU( COLOR TEXT,

SIZE TEXT, PRODUCT\_NAME TEXT,

PRODUCT\_ID VARCHAR(30) PRIMARY KEY, PRODUCT\_PURCHASING\_PRICE FLOAT, MARKUP FLOAT,

SUPPLIER\_ID VARCHAR(30), CATEGORY\_ID VARCHAR(30),

FOREIGN KEY (SUPPLIER\_ID) REFERENCES SUPPLIER(SUPPLIER\_ID), FOREIGN KEY (CATEGORY\_ID) REFERENCES CATEGORY(CATEGORY\_ID)

);

* + - PROMOTION

CREATE TABLE PROMOTION (

PROMOTION\_ID VARCHAR(30) PRIMARY KEY, PROMOTION\_VALUE FLOAT, PROMOTION\_START\_DATE VARCHAR(15), PROMOTION\_END\_DATE VARCHAR(15), MINIMUM\_PURCHASE\_AMOUNT FLOAT,

PRODUCT\_ID VARCHAR(30),

FOREIGN KEY (PRODUCT\_ID) REFERENCES SKU(PRODUCT\_ID)

);

* + - TRANSACTION

CREATE TABLE TRANSACTIONS ( TRANSACTION\_ID VARCHAR(30) PRIMARY KEY, PAYMENT\_METHOD VARCHAR(50), TRANSACTION\_STATUS VARCHAR(50)

);

* + - ORDER\_SHIPMENT

CREATE TABLE ORDER\_SHIPMENT( SHIPPING\_ID VARCHAR(30) PRIMARY KEY, POST\_CODE VARCHAR(30),

CARRIER TEXT

);

* + - ORDERS

CREATE TABLE ORDERS ( ORDER\_ID VARCHAR(30), CUSTOMER\_ID VARCHAR(30), ORDER\_DATE VARCHAR(15), ORDER\_STATUS TEXT, SHIPPING\_ID VARCHAR(30), DELIVERY\_DATE VARCHAR(15), TRANSACTION\_ID VARCHAR(30), PRODUCT\_ID VARCHAR(30), ORDER\_QUANTITY INTEGER, RETURN\_QUANTITY INTEGER, RETURN\_DATE VARCHAR(15),

PRIMARY KEY (ORDER\_ID, PRODUCT\_ID, CUSTOMER\_ID), FOREIGN KEY (PRODUCT\_ID) REFERENCES SKU(PRODUCT\_ID),

FOREIGN KEY (CUSTOMER\_ID) REFERENCES CUSTOMERS(CUSTOMER\_ID),

FOREIGN KEY (TRANSACTION\_ID) REFERENCES TRANSACTIONS(TRANSACTION\_ID) FOREIGN KEY (SHIPPING\_ID) REFERENCES SHIPMENT(SHIPPING\_ID)

);

# Part 2: Data Generation and Management

## : Synthetic Data Generation

All synthetic data generation was conducted solely in R and saved into csv files before pushed to GitHub, ensuring adherence to attribute conditionalities and inter-entity connections with the support of LLM. Initially, Mockaroo was explored for data generation, but its high level of randomisation proved more complex compared to R for setting precise rules to control data values (see Figure 1).

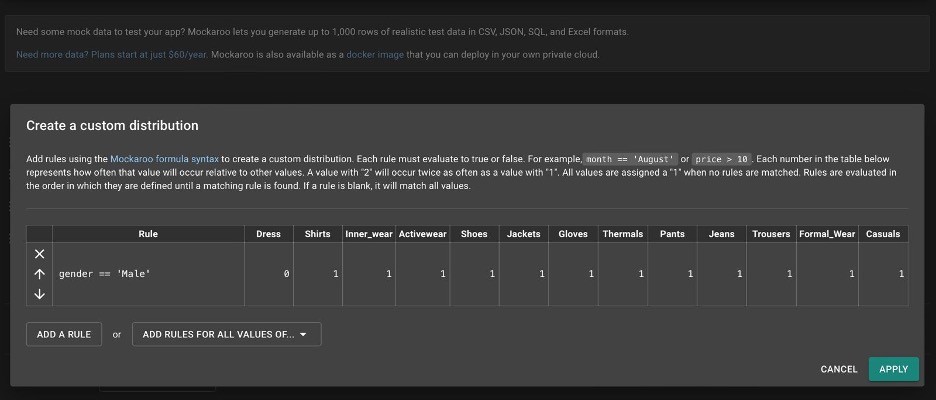


Figure 1 - Customising data values in Mockaroo

In R, independent entities such as CATEGORY, SUPPLIER, PROMOTION, ADS, SKU and CUSTOMERS were created first. We asked LLM to generate values for CATEGORY\_NAME, PRODUCT\_NAME, SUPPLIER\_NAME, AD\_TITLE, tailored to the fashion retail industry (Figure 2 and 3). The generation of SUPPLIER\_EMAIL and CUSTOMER\_EMAIL utlised a function suggested by LLM (Figure 4). Postal codes were randomly generated from UK post- codes using “PostcodesioR” package drawing on data from Oﬀice for National Statistic (Walczak, E., 2021). Customer names were randomly created employing “randomNames” package (Betebenner, D.W., 2021). All numerical fields such PRODUCT\_PURCHASING\_PRICE, MARKUP, etc. were randomised using either “runif” or “sample” functions to ensure realis- tic value distributions. Furthermore, for CUSTOMERS, REFERENCE\_ID, denoting CUS- TOMER\_ID of the referees, was conditioned such that the referred customers were only included in our database after their referees. REFERENCE\_ID was exclusively limited to customers with an ACQUISITION\_TYPE of “Referral” while AD\_ID was randomly assigned from ADS table for customers with an ACQUISITION\_TYPE of “Paid”.

**Please refer to Appendix A2 for the full prompt sequences in LLM.**

# Create name

## Filter out names with apostrophes filtered\_names <- randomNames(nrow(CUSTOMER))

filtered\_names <- filtered\_names[!grepl("'", filtered\_names)]

## If the number of filtered names is less than

## the number of rows in CUSTOMER, generate additional names if (length(filtered\_names) < nrow(CUSTOMER)) {

additional\_names <- randomNames(nrow(CUSTOMER) - length(filtered\_names)) additional\_names <- additional\_names[!grepl("'", additional\_names)] filtered\_names <- c(filtered\_names, additional\_names)

}

## Generate random full names

random\_full\_names <- sample(filtered\_names, nrow(CUSTOMER), replace = TRUE)

#W Split full names into first and last names split\_names <- strsplit(random\_full\_names, ",")

## Extract first names

CUSTOMER$FIRST\_NAME <- sapply(split\_names, "[", 2)

## Extract last names

CUSTOMER$LAST\_NAME <- sapply(split\_names, "[", 1)

# CREATE CUSTOMER\_POST\_CODE CUSTOMER$POST\_CODE <-

sapply(1:nrow(CUSTOMER), function(x) random\_postcode()$postcod

Figure 4 - LLM prompt for email generation

Subsequently, dependent tables including ORDERS, ORDER\_SHIPMENT and TRANSAC- TIONS were created based on the data from the previous tables. Specifically, in ORDERS, combinations of CUSTOMER\_ID and PRODUCT\_ID were randomly selected from SKU and CUSTOMERS tables so that a customer can purchase multiple products in an order. The generation of DELIVERY\_DATE and RETURN\_DATE adhered to constraints derived from OR- DER\_DATE, ensuring that DELIVERY\_DATE fell within a maximum of 7 days after order placement, and RETURN\_DATE occurred after DELIVERY\_DATE while remaining within 30 days from ORDER\_DATE. ORDER\_STATUS was randomised to reflect an authentic distribution encompassing “Returned”, “Delivered” and “In transit”. RETURN\_QUANTITY and RETURN\_DATE was exclusively assigned to orders marked as “Returned”, with RE- TURN\_QUANTITY restricted not to exceed ORDER\_QUANTITY. Moreover, for TRANS- ACTIONS table, we also ensured that all orders have matching successful transactions, with the proportion of failed transaction set to 6% of total transaction numbers.

.

# Create ORDER dataframe

ORDER <- data.frame(ORDER\_ID = paste0("OD", seq\_len(1000) + 10000), stringsAsFactors = TRUE

)

# Add CUSTOMER\_ID

ORDER$CUSTOMER\_ID <- sample(CUSTOMER$CUSTOMER\_ID,nrow(ORDER),replace=TRUE)

# Add ORDER\_DATE ORDER$ORDER\_DATE <-

sample(seq(start\_date, end\_date, by = "day"), nrow(ORDER), replace = TRUE)

# Add ORDER\_STATUS:

order\_status <- c("Delivered", "Returned") proportions <- c("Delivered" = 0.8, "Returned" = 0.2)

# Sample proportionally with replacement and assign to ORDER$ORDER\_STATUS ORDER$ORDER\_STATUS <-

sample(order\_status, size = nrow(ORDER), replace = TRUE, prob = proportions) ORDER$ORDER\_STATUS[ORDER$ORDER\_DATE>'2024-03-05'] = "In transit"

# Add SHIPPING\_ID

ORDER$SHIPPING\_ID <- ORDER\_SHIPMENT$SHIPPING\_ID # Add DELIVERY\_DATE

ORDER$DELIVERY\_DATE <- ORDER$ORDER\_DATE + sample(7,nrow(ORDER),replace=TRUE)

ORDER$DELIVERY\_DATE[ORDER$ORDER\_STATUS %in% c("Order placed", "In transit")] <- NULL

# Add TRANSACTION\_ID ORDER$TRANSACTION\_ID <-

TRANSACTION$TRANSACTION\_ID[TRANSACTION$TRANSACTION\_STATUS == "Successful"]

# Create an empty dataframe to store order-product mappings order\_product\_mapping <-

data.frame(ORDER\_ID = character(), PRODUCT\_ID = character(), stringsAsFactors = FALSE)

# Define the number of products per order

products\_per\_order <- round(runif(nrow(ORDER), min = 1, max = 10))

# Loop through each order ID and sample products for (i in 1:nrow(ORDER)) {

order\_id <- ORDER$ORDER\_ID[i] product\_ids <-

sample(SKU$PRODUCT\_ID, size = products\_per\_order[i], replace = FALSE) order\_product\_mapping <- rbind(order\_product\_mapping,

data.frame(ORDER\_ID = rep(order\_id, length(product\_ids)), PRODUCT\_ID = product\_ids))

}

# Merge order\_product\_mapping with ORDER dataframe to retain other order details ORDER <- merge(ORDER, order\_product\_mapping, by = "ORDER\_ID", all.x = TRUE)

# Add ORDER\_QUANTITY

ORDER$ORDER\_QUANTITY <-round(runif(nrow(ORDER),min=1,max=5),0)

# Add RETURN\_QUANTITY ORDER$RETURN\_QUANTITY <-

ifelse(ORDER$ORDER\_STATUS=="Returned",

mapply(function(x) sample(1:x, 1), ORDER$ORDER\_QUANTITY),"")

# Add RETURN\_DATE

ORDER$RETURN\_DATE <- ORDER$ORDER\_DATE + sample(9:30,nrow(ORDER),replace=TRUE) ORDER$RETURN\_DATE[ORDER$ORDER\_STATUS != "Returned"] <- ""

# Change DATE to CHARACTER

ORDER$ORDER\_DATE <- as.character(ORDER$ORDER\_DATE) ORDER$RETURN\_DATE <- as.character(ORDER$RETURN\_DATE) ORDER$DELIVERY\_DATE <- as.character(ORDER$DELIVERY\_DATE)

#Save to csv write.csv(ORDER[1:15,],

file = file.path("/cloud/project/Data","ORDERS.1.csv"), row.names = FALSE) write.csv(ORDER[16:nrow(ORDER),],

file = file.path("/cloud/project/Data","ORDERS.2.csv"), row.names = FALSE)

## : Data Import and Quality Assurance

An R script was developed to load all data files in their respective corresponding tables within our database, ensuring robust data quality and integrity. This process involved connecting to the database and iteratively validating data entries against predefined criteria.

The script employed a loop cross-checking primary key values to prevent duplications and enforcing strict validation rules on a row by row basis. Entries were scrutinised for null values, adherence to formatting standards for phone numbers and emails, and non-negativity in numeric fields. For formatting check, we defined functions to automatically validate. Data entries satisfying these validation criteria were then appended to the database whilst problematic entries were recorded in “error\_log.txt” file for further reviews.

library(readr) library(RSQLite) library(dplyr) library(DBI)

# Define a function to check if phone numbers are of length 10 # and contain only numeric characters

validate\_phone\_numbers <- function(phone) {

phone <- as.character(phone) # Convert integer to character

if (!is.na(phone) && nchar(phone) == 10 && !any(is.na(as.numeric(phone)))) { return(TRUE) # Phone number is valid

} else {

return(FALSE) # Phone number is not valid

}

}

# Function to validate email addresses

email\_pattern <- "^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$" validate\_emails <- function(email) {

grepl(email\_pattern, email)

}

# Function to validate that numeric attributes are non-negative check\_non\_negative\_numeric <- function(data) {

numeric\_cols <- sapply(data, is.numeric)

negative\_values <- sapply(data[numeric\_cols], function(col) any(col < 0)) return(!any(negative\_values))

}

# Establishing the connection to db

my\_db <- RSQLite::dbConnect(RSQLite::SQLite(), "ecommerce.db")

file\_paths <- list(

"ADS" = list.files(path = "Data", pattern = "ADS.\*\\.csv$", full.names = TRUE), "CATEGORY" = list.files(path = "Data",

pattern = "CATEGORY.\*\\.csv$", full.names = TRUE), "SUPPLIER" = list.files(path = "Data",

pattern = "SUPPLIER.\*\\.csv$", full.names = TRUE), "CUSTOMERS" = list.files(path = "Data",

pattern = "CUSTOMERS.\*\\.csv$", full.names = TRUE), "SKU" = list.files(path = "Data",

pattern = "SKU.\*\\.csv$", full.names = TRUE), "PROMOTION" = list.files(path = "Data",

pattern = "PROMOTION.\*\\.csv$", full.names = TRUE), "TRANSACTIONS" = list.files(path = "Data",

pattern = "TRANSACTIONS.\*\\.csv$", full.names = TRUE), "ORDER\_SHIPMENT" = list.files(path = "Data",

pattern = "ORDER\_SHIPMENT.\*\\.csv$", full.names = TRUE), "ORDERS" = list.files(path = "Data",

pattern = "ORDERS.\*\\.csv$", full.names = TRUE)

)

tables <- list( "ADS" = "AD\_ID",

"CATEGORY" = "CATEGORY\_ID", "SUPPLIER" = "SUPPLIER\_ID", "CUSTOMERS" = "CUSTOMER\_ID", "SKU" = "PRODUCT\_ID", "PROMOTION" = "PROMOTION\_ID",

"TRANSACTIONS" = "TRANSACTION\_ID", "ORDER\_SHIPMENT" = "SHIPPING\_ID",

"ORDERS" = c("ORDER\_ID", "PRODUCT\_ID", "CUSTOMER\_ID")

)

# Define write\_errors function with folder path argument write\_errors <- function(errors, folder\_path, file\_name) {

# Ensure the folder exists, if not, create it if (!dir.exists(folder\_path)) {

dir.create(folder\_path, recursive = TRUE)

}

file\_path <- file.path(folder\_path, file\_name)

if (length(errors) > 0) { cat("Errors:\n", file = file\_path) for (error in errors) {

cat(error, "\n", file = file\_path, append = TRUE)

}

cat("\n", file = file\_path, append = TRUE)

}

}

# List to store errors error\_list <- c()

# Function to check if data entries exist and load new entries for (table\_name in names(tables)) {

for (file\_path in file\_paths[[table\_name]]) { table\_data <- read\_csv(file\_path,n\_max = Inf)

## Apply specific rules for attributes based on the table if (table\_name == "ADS") {

# Initialize error list error\_list <- vector("list")

# Convert AD\_START\_DATE and AD\_END\_DATE to character table\_data$AD\_START\_DATE <- as.character(table\_data$AD\_START\_DATE) table\_data$AD\_END\_DATE <- as.character(table\_data$AD\_END\_DATE)

# Initialize vector to store indices of invalid rows invalid\_rows <- vector("numeric")

# Ensure numeric attributes are non-negative numeric\_attrs <- c("COST\_PER\_CLICK", "CLICK\_THROUGH\_RATE", "NUMBER\_OF\_CLICK")

for (attr in numeric\_attrs) {

if (any(table\_data[[attr]] < 0)) { error\_list <- c(error\_list, paste("Negative values found in", attr, "column of ADS table."))

invalid\_rows <- c(invalid\_rows, which(table\_data[[attr]] < 0))

}

}

# Remove invalid rows from table\_data if (length(invalid\_rows) > 0) {

table\_data <- table\_data[-invalid\_rows, ]

}

}

if (table\_name == "CUSTOMERS") { # Initialize error list error\_list <- vector("list")

# Convert DATE\_OF\_BIRTH to character

table\_data$DATE\_OF\_BIRTH <- as.character(table\_data$DATE\_OF\_BIRTH)

# Initialize vector to store indices of invalid rows invalid\_rows <- vector("numeric")

# Check phone numbers and emails for (i in 1:nrow(table\_data)) {

if (!validate\_phone\_numbers(table\_data$PHONE\_NUMBER[i])) { error\_list <-

c(error\_list,

paste("Invalid phone number in CUSTOMERS table:", table\_data$PHONE\_NUMBER[i])) invalid\_rows <- c(invalid\_rows, i)

}

if (!validate\_emails(table\_data$CUSTOMER\_EMAIL[i])) { error\_list <- c(error\_list,

paste("Invalid email in CUSTOMERS table:", table\_data$CUSTOMER\_EMAIL[i])) invalid\_rows <- c(invalid\_rows, i)

}

}

# Remove invalid rows from table\_data if (length(invalid\_rows) > 0) {

table\_data <- table\_data[-invalid\_rows, ]

}

}

if (table\_name == "SKU") { # Initialize error list

error\_list <- vector("list")

# Ensure numeric attributes are non-negative numeric\_attrs <- c("MARKUP", "PRODUCT\_PURCHASING\_PRICE")

for (attr in numeric\_attrs) {

if (any(table\_data[[attr]] < 0)) { error\_list <- c(error\_list,

paste("Negative values found in", attr, "column of SKU table.")) invalid\_rows <- c(invalid\_rows, which(table\_data[[attr]] < 0))

}

}

# Remove invalid rows from table\_data if (length(invalid\_rows) > 0) {

table\_data <- table\_data[-invalid\_rows, ]

}

}

if (table\_name == "PROMOTION") { # Initialize error list error\_list <- vector("list")

# Convert PROMOTION\_START\_DATE and PROMOTION\_END\_DATE to character table\_data$PROMOTION\_START\_DATE <- as.character(table\_data$PROMOTION\_START\_DATE) table\_data$PROMOTION\_END\_DATE <- as.character(table\_data$PROMOTION\_END\_DATE)

# Initialize vector to store indices of invalid rows invalid\_rows <- vector("numeric")

# Ensure numeric attributes are non-negative numeric\_attrs <- c("MINIMUM\_PURCHASE\_AMOUNT") for (attr in numeric\_attrs) {

if (any(table\_data[[attr]] < 0)) { error\_list <- c(error\_list,

paste("Negative values found in", attr, "column of PROMOTION table.")) invalid\_rows <- c(invalid\_rows, which(table\_data[[attr]] < 0))

}

}

# Remove invalid rows from table\_data if (length(invalid\_rows) > 0) {

table\_data <- table\_data[-invalid\_rows, ]

}

}

if (table\_name == "SUPPLIER") { # Initialize error list error\_list <- vector("list")

# Initialize vector to store indices of invalid rows invalid\_rows <- vector("numeric")

# Check phone numbers and emails for (i in 1:nrow(table\_data)) {

if (!validate\_phone\_numbers(table\_data$SUPPLIER\_PHONE[i])) { error\_list <- c(error\_list,

paste("Invalid phone number in SUPPLIER table:", table\_data$SUPPLIER\_PHONE[i])) invalid\_rows <- c(invalid\_rows, i)

}

if (!validate\_emails(table\_data$SUPPLIER\_EMAIL[i])) { error\_list <- c(error\_list,

paste("Invalid email in SUPPLIER table:", table\_data$SUPPLIER\_EMAIL[i])) invalid\_rows <- c(invalid\_rows, i)

}

}

# Remove invalid rows from table\_data if (length(invalid\_rows) > 0) {

table\_data <- table\_data[-invalid\_rows, ]

}

}

if (table\_name == "ORDERS") { # Initialize error list error\_list <- vector("list")

# Convert ORDER\_DATE, DELIVERY\_DATE, RETURN\_DATE to character table\_data$ORDER\_DATE <- as.character(table\_data$ORDER\_DATE) table\_data$DELIVERY\_DATE <- as.character(table\_data$DELIVERY\_DATE) table\_data$RETURN\_DATE <- as.character(table\_data$RETURN\_DATE)

# Initialize vector to store indices of invalid rows invalid\_rows <- vector("numeric")

# Check numeric attributes for non-negativity numeric\_attrs <- c("ORDER\_QUANTITY", "RETURN\_QUANTITY") for (attr in numeric\_attrs) {

if (any(!is.na(table\_data[[attr]]) & table\_data[[attr]] < 0)) { error\_list <- c(error\_list,

paste("Invalid or negative values found in", attr, "column of ORDERS

table."))

invalid\_rows <- c(invalid\_rows, which(!is.na(table\_data[[attr]]) & table\_data[[attr]] < 0))

}

}

# Remove invalid rows from table\_data if (length(invalid\_rows) > 0) {

table\_data <- table\_data[-invalid\_rows, ]

}

}

## Check for primary key duplication for (i in seq(nrow(table\_data))) {

new\_record <- table\_data[i, ] pk\_columns <- tables[[table\_name]] pk\_values <- new\_record[pk\_columns]

# Check if primary key values are non-null if (any(is.na(pk\_values))) {

error\_list <- c(error\_list,

paste("Null primary key value found in", table\_name, "table.")) next # Skip to the next record if primary key is null

}

conditions <- paste(pk\_columns, "=", paste0("'", pk\_values, "'"), collapse = " AND ")

key\_exists <- dbGetQuery(my\_db,

paste("SELECT COUNT(\*) FROM", table\_name, "WHERE", conditions))

if (key\_exists == 0) { tryCatch({

RSQLite::dbAppendTable(my\_db, table\_name, new\_record)

}, error = function(e) { error\_list <- c(error\_list,

paste("Error inserting record with primary key", paste(pk\_values, collapse = ", "), "into table", table\_name)) print(paste("Error inserting record with primary key", paste(pk\_values, collapse = ", "), "into table", table\_name)) print(e)

})

} else {

print(paste("Record with primary key",

paste(pk\_values, collapse = ", "), "already exists in table", table\_name))

}

}

}

}

# Save errors to a folder named "Error logs" within the current directory write\_errors(error\_list, "Error logs", "error\_log.txt")

# Part 3: Data Pipeline Generation

## : GitHub Repository and Workflow Setup

We initiated our project by creating a new Git repository, connecting Posit Cloud/ RStudio to the repository and uploading essential files, including (1) database, (2) data schema, (3) synthetic data generation, (4) data validation and database writing, (5) data query and analysis scripts. This setup allows us to eﬀiciently track changes and revert to previous versions as needed.

## : GitHub Actions for Continuous Integration

To automate our project’s operations, we implemented a GitHub Actions workflow, detailed through the following key components:

* + - Trigger: Activated by either a push event to the main branch or a scheduled run every 24 hours, ensuring real-time integration of contributions.
    - Runner: Utilise the most recent Ubuntu environment to execute the job.
    - Steps: Comprising eight sequential tasks, each step executes a specific operation within the job:
    - Repository Checkout: Clone the project’s code into the runner, providing a foundation for subsequent tasks.
    - R Environment Setup: Prepare the R environment, ensuring all R-based operations can be performed without hitches.
    - R Package Caching: Preserve installed R packages between runs, significantly reducing setup time by bypassing redundant installations.
    - Package Installation: Engage only if the cache does not contain the necessary packages, ensuring all dependencies are available for the script execution.
    - Script Execution: Run our R scripts from the repository to validate and load satisfying data entries to the database, subsequently creating analyses.
    - Add Changes: Scans the project’s database for any changes following the script’s execution and notifies with a “Changes found” message if updates are identified. Meanwhile, new analyses are automatically generated and saved to folder “figures”.
    - Commit Changes: If changes are detected, this step prepares and commits the updated files, maintaining a current state within the repository.
    - Push Updates: Conclude the workflow by uploading the latest commit to the repository, ensuring all changes are synchronised and stored.

name: ETL workflow

on:

push:

branches: [ main ] schedule:

- cron: '0 \*/24 \* \* \*' # Runs every 24 hours

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: Create figures directory run: mkdir -p figures
* name: Install packages

if: steps.cache.outputs.cache-hit != 'true' run: |

Rscript -e 'install.packages(c("ggplot2", "dplyr", "readr", "RSQLite", "DBI"))'

* name: Execute R script run: |

Rscript R\_codes/Workflow.R

* name: Execute Data Analysis run: |

Rscript R\_codes/Query\_workflow.R

* name: Add database changes and commit run: |

git config --global user.email ["Teng-Yi.Chen@warwick.ac.uk"](mailto:Teng-Yi.Chen@warwick.ac.uk) git config --global user.name "DylanCTY"

git add ecommerce.db git add --all figures/

git commit -m "Update database" || echo "No changes to commit" # Check if error logs exist

if [ -d "Error logs" ] && [ "$(ls -A Error\ logs)" ]; then echo "Error logs exist"

# Create error logs folder mkdir -p "Error logs"

# Write error\_log.txt

echo "Errors:" > "Error logs/error\_log.txt"

# Append each error to the error\_log.txt file for error in Error\ logs/\*; do

cat "$error" >> "Error logs/error\_log.txt"

echo "" >> "Error logs/error\_log.txt" # Add a newline after each error done

# Commit error logs

git add "Error logs/error\_log.txt"

git commit -m "Add error log" || echo "No error log changes to commit" else

echo "No error logs found"

fi

* name: Push changes

uses: [ad-m/github-pus](mailto:ad-m/github-push-action@v0.6.0)[h-action@v0.6.0](mailto:h-action@v0.6.0) with:

github\_token: ${{ secrets.GITHUB\_TOKEN }} branch: main

# Part 4: Advanced Data Analysis

Once the database was updated with the newly generated data via GitHub automation, the data was analysed utilising the R package packages dplyr, GGplot2, and tidyr in conjunction with the SQL DQL command. The procedure entailed retrieving the data and converting it into a format suitable for subsequent analysis. The complete procedure is outlined below.

**Data Query**

The data from the database was obtained using a SQL Data Query Language (DQL) statement. By utilising the calculated function and aggregate command, the data were converted into a format and value suitable for analysis. The following example is a query that displays the top revenue earned by product SKUs in the last 1 year, calculated by multiplying unit sales with the selling price.

Revenue\_analysis\_df <- RSQLite::dbGetQuery(my\_db, "SELECT T1.PRODUCT\_ID AS PRODUCT\_ID,

T2.PRODUCT\_NAME AS PRODUCT\_NAME, T2.SIZE AS SIZE,

T2.COLOR AS COLOR, T1.UNIT\_SOLD AS UNIT\_SOLD,

T2.SELLING\_PRICE\_PER\_UNIT AS SELLING\_PRICE,

T1.UNIT\_SOLD \* T2.SELLING\_PRICE\_PER\_UNIT AS TOTAL\_REVENUE

FROM

( SELECT ORDERS.PRODUCT\_ID AS PRODUCT\_ID, ORDERS.ORDER\_DATE AS ORDER\_DATE,

SUM(ORDERS.ORDER\_QUANTITY) AS UNIT\_SOLD FROM ORDERS

WHERE cast(julianday('now') - julianday(ORDERS.ORDER\_DATE) AS INTEGER ) <= 365 GROUP BY ORDERS.PRODUCT\_ID

) AS T1 LEFT JOIN

( SELECT SKU.PRODUCT\_ID AS PRODUCT\_ID, SKU.PRODUCT\_NAME AS PRODUCT\_NAME, SKU.SIZE AS SIZE,

SKU.COLOR AS COLOR, SKU.PRODUCT\_PURCHASING\_PRICE \* (1 + SKU.MARKUP) AS SELLING\_PRICE\_PER\_UNIT

FROM SKU

) AS T2

ON T1.PRODUCT\_ID = T2.PRODUCT\_ID"

)

**Data Manipulation**

Once the data was imported into a R data frame, data manipulation was carried out using R to prepare the data for visualisation. The following code example generates a new column containing the Product SKU description.

# Create product description name of each product\_ID for using in analysis Revenue\_analysis\_df$PRODUCT\_DESCRIPTION <- paste(Revenue\_analysis\_df$PRODUCT\_ID, Revenue\_analysis\_df$PRODUCT\_NAME, Revenue\_analysis\_df$SIZE, Revenue\_analysis\_df$COLOR, sep = " ")

**Data Visualisation**

Finally, once we retrieved and formatted the data for analysis, we used the ggplot2 package to create visual representations of the data. The code snippet shown below demonstrates a bar chart that visually represents the top SKUs with the highest sales in the past year, as obtained from the preceding phases of our query.

ggplot(data= Revenue\_analysis\_df %>% slice\_max(TOTAL\_REVENUE, n = 10) , aes(x = TOTAL\_REVENUE,

y = reorder( PRODUCT\_DESCRIPTION, TOTAL\_REVENUE ))) +

geom\_bar(stat = "identity", position = position\_dodge(width = 0.75), aes(fill=TOTAL\_REVENUE), show.legend = FALSE) +

labs(x = "Revenue", y = "Product ID and Description", title = "10 Best Selling Products in Last 1 Year ", subtitle = "Labels indicate the revenue in GBP generating from each product") +

geom\_text(aes(label = paste("£", round(TOTAL\_REVENUE), sep="")), color='white', hjust = 1) + theme(axis.text.x = element\_text(hjust = 1))

To present the data analysis, we use the R Quarto report, which can be easily changed to reflect the latest dataset. The analyses show short-term analysis for monitoring current performance in the last thirty days and long-term analysis, where we analyse changes on a yearly basis to demonstrate long-term progress.

Top 10 Most Sold SKUs in Units in Last 30 days

Labels indicate revenue generated by the SKUs

£1301

£1102

£961

£698

£469

£442

£394

£374

£371

£190

0.0 2.5 5.0 7.5 10.0 12.5

Product Names

Units Sold

10 Best Selling Products in Last 1 Year

Labels indicate revenue generated by the SKUs

£3022

£2630

£2432

£2373

£2346

£2280

£2232

£2184

£2108

£2106

0 1000 2000 3000

Product ID and Description

Revenue

10 Best Selling Categories in Last 1 Year

Labels indicate the revenue generated from category

£1210

£1104

£1094

£1081

£1062

£920

£899

£798

£737

£730

0 250 500 750 1000 1250

Category Name

Revenue

Customer Acquisition in Last 1 Year

Labels indicate the number of new customer from each platform

6

6

5

4

2

1

2

1

4

New Customers

2

0

90 to 180 days

180 to 270 days

Acquisition Channel

270 to 365 days

PLATFORM

Facebook

Google Ads

Instagram

TikTok X

Customer Acquisition by Gender in Last 1 Year

Labels indicate the number of new customer from each platform

8

4

5

4

3

2

2

1

8

6

New Customers

GENDER

4 Female

Male

2

0

Instagram X

TikTok

Facebook

Google Ads

Acquisition Channel

# Conclusion

This project implemented a thorough approach for data management for a UK-based apparel e-commerce store. The database structure, depicted conceptually through an ER diagram, featured eight principal entities: products, customers, shipments, promotions, advertisements, suppliers, categories, and transactions, implemented via a SQL schema. Synthetic data, simulating genuine retail transactions, was generated in R. Rigorous quality assurance procedures preceded data insertion into the database. Subsequent data analysis was conducted using Quarto with R to provide actionable insights for management. Automation of data validation, loading, and analysis processes via a GitHub workflow enabled multi-stakeholder oversight and accountability across all project phases.

# References

* + - Betebenner, D.W. (2021) *Generate Random Given and Surnames [R package random- Names version 1.5-0.0]*. [https://cran.r-project.org/web/packages/randomNames/index](https://cran.r-project.org/web/packages/randomNames/index.html)

[.html](https://cran.r-project.org/web/packages/randomNames/index.html). [date accessed: 1 March 2024

* + - Walczak, E. (2021) *API Wrapper Around ‘Postcodes.io’* [R package PostcodesioR version 0.3.1]. https://cran.r-project.org/web/packages/PostcodesioR/index.html.