Project Brief

PROJECT NAME: E-Commerce Datebase

CANDIDATE _NUMBER: 51927

GOALS:	The goal of this e-commerce database project is to create a realistic dataset that closely mirrors real-world scenarios, enabling to stimulate accurate analysis of customer behaviors, order trends, and inventory management.

FOR CONCEPETUAL MODEL:

1. Use of Specialization

- There is specialization within the PAYMENT entity. Specifically, PAYMENT has been specialized into CREDIT_DEBIT and VOUCHERS_GIFT tables.

- Total specialization is used here, as all payments are classified as either CREDIT_DEBIT or VOUCHERS_GIFT.

2. Cardinality and Relationship Assumptions

- Consumer and Basket Relationship: Each CONSUMER is linked to a BASKET through a has-a relationship. Each consumer has exactly one basket, making it a 1-to-1 relationship.

- Consumer and Payment Relationship: Each consumer can have multiple PAYMENTS, but each PAYMENT must be linked to one CONSUMER. This is a 1-to-Many (1:N) relationship.

- Basket and Product Relationship: A basket can contain multiple products, and each product can appear in multiple baskets. This make a Many-to-Many (M:N) relationship with an associative entity contains.

- Consumer and Order Relationship: A consumer can place multiple orders, and each oder only associated with one consumer. This is represented by the places relationship, making it a 1-to-Many (1:N) relationship.

- Orders and Product Relationship: Each ORDER can include multiple PRODUCTS, and each PRODUCT can appear in multiple orders. This relationship captures the quantity ordered and is a Many-to-Many (M:N) relationship with an associative entity INCLUDES.

- Order and Delivery Relationship: Each ORDER has at most one DELIVERY. This is a 1-to-1 relationship where each order can have only one corresponding delivery record.

- Order and Return Relationship: Each ORDER can have only one RETURN record associated if the order is returned. This relationship is 1-to-1.

- Order and Payment Relationship: Each ORDER must have a payment record and one payment can place multiple orders. This relationship is 1-to-Many.

- Review and Product Relationship: Each consumer can write multiple reviews for products they ordered. Each REVIEW is associated with one CONSUMER and one PRODUCT, and one product can receive many reviews from different consumers forming a Many-to-Many relationship.

3. Use of NULL Values

- For the ORDERS table, Deduction_Promotion allows NULL values, indicating that some orders might not have any promotions applied.

4. Assumptions and Adaptations from Context

- Assumptions Consumers can have more than one payment method, didn't associate consumers with one fixed payment method.

- Is_default in the CREDIT_DEBIT table could be NULL,but at this time I use [0,1] to identify default or not.

- In the RETURN table, statuses, like Completed, Cancelled, Denied, and Pending, are represented as optional, mutually exclusive columns. Each status is indicated using 0 and 1, where 0 denotes that the status is not applicable, and 1 confirms it as the active status.

- In the DELIVERY table, statuses like Cancelled, Delivered, Postponed, and Pending, are represented as optional, mutually exclusive columns. Each status is indicated using 0 and 1, where 0 denotes that the status is not applicable, and 1 confirms it as the active status.

FOR ER MODEL:

1. Adaptations from Conceptual/E-R Model

- **Multivalued Attributes Mapped as Tables:**

The ***Dimensions*** attribute is marked as multivalued because it contains multiple values that collectively describe an attribute. For example, Dimensions include separate values for height, width, and depth, each of which represents a different aspect of the product's size. Thus, Dimensions has multiple components rather than a single value.

- **Many-to-Many Relationships Mapped as Tables:**

1)Review and Product:

• Each consumer can write multiple reviews for products they ordered, and each product can receive multiple reviews from different consumers.

• This relationship is resolved by the REVIEW table, linking each Review to one Consumer and one Product.

2)Orders and Product:

• A single order can contain multiple products, and a single product can appear in multiple orders.

• This many-to-many relationship is resolved through the INCLUDES table, which links Orders and Product entries with quantity details.

3) Basket and Product:

• A single basket can contain multiple products, and a single product can be in multiple baskets.

• This relationship is managed by the CONTAINS table, connecting Basket and Product entries with the quantity of each product in the basket

- **Composite Attribute:**

1) Address (in CONSUMER):

This attribute is composed of multiple sub-attributes: Country, Street, Postcode, Building, and City. By breaking down the address into distinct parts, the model allows for more granular querying, such as filtering by city or country.

2)Dimensions (in PRODUCT):

This attribute includes Depth, Width, and Height, representing the physical dimensions of a product. Storing each dimension separately allows for specific queries, such as calculating the product's volume.

3) Status (in DELIVERY):

The Status attribute includes Delivered, Postponed, Cancelled, and Pending, representing the various stages a delivery might go through. Each status can be represented with a binary indicator (e.g., 1 for active status, 0 for inactive), enabling clear tracking of the delivery's current state.

4) Status (in RETURN):
This attribute consists of Completed, Cancelled, Denied, and Pending, representing the potential outcomes of a return request. As with delivery status, each component can be set as 1 or 0 to indicate whether it applies, allowing flexible handling of return states.

2. Views and Structure

Will demonstrate below, no views form at this stage

3.Constraints on Relationships and Participation

- **Minimum and Maximum Cardinalities:**

1)Basket and Product:

• The minimum cardinality for both Basket and Products in their relationship is optional (0), as a Basket might not contain any Products if it's inactive or empty, and a Product may not necessarily be in any basket.

2)Consumer and Basket:

• The maximum cardinality for both Consumer and Basket in their relationship is mandatory (1), as each Consumer is associated with exactly one Basket, and each Basket is directly linked to one Consumer.

- **Optional vs. Mandatory Relationships:**

1) Orders and Consumers:

Each order is mandatory for a consumer, as every order must be associated with a specific consumer via email. However, consumers are optional for orders, as some consumers may choose not to place an order.

2)Payment and Consumers:

Payments are mandatory for consumers, as each payment must be linked to a specific consumer. However, a consumer is optional for a payment, as some consumers may browse the website without making a purchase.

3) Payment and Orders:

Each Order is mandatory to be associated with a Payment, as an order cannot be completed without a payment method. However, each Payment is optional in terms of placing orders; a payment method can exist in the system, linked to a Consumer, without necessarily being used to make an order.

4)Reviews and Consumers:

Reviews are mandatory for consumers, as each review must be written by a specific consumer. However, consumers are optional to write a review, as not all consumers choose to leave feedback.

5)Reviews and Products:

Reviews are mandatory for products, as each review must be associated with a specific product. However, products are optional to receive a review, as not all products may have reviews.

6)Returns and Orders:

Returns are mandatory to have an order, as each return must be linked to an existing order. However, orders are optional to be returned, as it depends on the consumer's decision whether to return the product.

7)Deliveries and Orders:

Deliveries are mandatory to have an order, as each delivery must correspond to an order. However, orders are optional to be delivered or returned, depending on the consumer's choice.

8)Products and Orders:

Products are mandatory for orders, as an order cannot be empty. However, orders are optional for products, as some products may never be ordered by consumers.

4. Additional Adaptations

- **Null Value Usage:**
  1) Return.Start_date and Return.Due_date may be nullable if returns are initiated but not yet processed.

DATABASE CREATION:

1. For Table Creation

- Start with base tables, such as CONSUMER, PRODUCT, and PAYMENT, as these are foundational and do not rely on foreign keys to other tables.
- Next, create tables that have foreign key dependencies, like ORDERS, BASKET, REVIEW, RETURN, and DELIVERY, which rely on the primary tables.
- Finish by creating tables that handle many-to-many relationships, like INCLUDES and CONTAINS, as these will connect primary entities (ORDERS and PRODUCT or BASKET and PRODUCT).

2. For View Creation

- **CheckDuplicateEmails:**

This view checks for duplicate email entries in the PAYMENT table, ensuring data integrity and identifying cases where consumers may have multiple payments associated with the same email.

- **ConsumerBasketView:**

This view joins CONSUMER and BASKET tables, providing an easy way to see what each consumer has in their basket without complex joins.

- **ConsumerOrderView:**

This view joins CONSUMER and ORDERS tables, displaying consumer details along with their order details and grand total after deductions.

- **ProductSubtotalView:**

This view calculates the subtotal for each product in PRODUCT by applying a markup, allowing for easy access to adjusted prices.

3.Foreign Key Constraints:

- Foreign key constraints ensure referential integrity across related tables.

4.Composite Keys:

- The **INCLUDES** table uses composite keys (Order number + Product number) to handle the many-to-many relationship between ORDERS and PRODUCT, allowing multiple products within one order.

- The **CONTAINS** table uses composite keys(Basket number + Product number) to handle the many – to-many relationship between BASKETS and PRODUCT, allowing multiple products within one order.

DATA SOURCE:

For this database project, synthetic data was generated to simulate a realistic e-commerce scenario. Most of the data in the tables was created using Python's Faker library(attached as '51927_Synthetic_Data.ipynb'), which provided synthetic data that mimics real-world attributes such as names, addresses, dates, and quantities and so on.

1. Synthetic Data Generation with Faker:

- All tables except for **PRODUCT** and **REVIEW** were populated entirely with data generated by the Faker library. This approach was used to quickly produce consistent, structured data for attributes like customer details, orders, and payments.

2.Product and Review Text Generation with ChatGPT:

The PRODUCT and REVIEW tables required a higher level of realism for certain fields to accurately represent real-world products and customer feedback. Specifically:

- **Product Table:** Attributes such as Product Name, Brand, and Description were generated using ChatGPT to ensure they resemble real-world items with meaningful names, recognized brands, and descriptive content.
- **Review Table:** The Text attribute in the REVIEW table was also generated using ChatGPT. Using general prompts, ChatGPT created reviews that include realistic language and phrasing, ensuring a genuine customer feedback feel.

3.Data Consistency and Adherence:

- After generating the data, additional processing steps were taken to ensure all tables adhered to the defined relational model and foreign key constraints. This included checking referential integrity across relationships and ensuring the quantity and content of generated data were consistent with the requirements of each entity in the model.

4.Chat log:

python Copy

```python
import pandas as pd
import random

# Define product categories and corresponding brands
category_brands = {
    "Electronics": ["Apple", "Samsung", "Sony", "LG", "Dell", "HP", "Lenovo", "Asus", "Panasonic", "Phili
    "Furniture": ["IKEA", "Ashley Furniture", "Steelcase", "Herman Miller", "La-Z-Boy"],
    "Grocery": ["Whole Foods", "Trader Joe's", "Kroger", "Safeway", "Publix"],
    "Cosmetic": ["L'Oréal", "Estée Lauder", "Nivea", "Clinique", "Maybelline"],
    "Appliances": ["Whirlpool", "GE Appliances", "Bosch", "KitchenAid", "Frigidaire"],
    "Outdoor": ["The North Face", "Columbia", "Patagonia", "REI", "Black Diamond"],
    "Sports": ["Nike", "Adidas", "Under Armour", "Puma", "Reebok"],
    "Toys": ["Lego", "Mattel", "Hasbro", "Fisher-Price", "Nerf"],
    "Books": ["Penguin Random House", "HarperCollins", "Simon & Schuster", "Hachette", "Macmillan"],
    "Automotive": ["Bosch", "Michelin", "Goodyear", "Bridgestone", "Continental"]
}

# Define product names and descriptions by category
product_names = {
    "Electronics": ["Laptop", "Smartphone", "Smartwatch", "Tablet", "Headphones", "Bluetooth Speaker"],
    "Furniture": ["Desk Chair", "Standing Desk", "Bookshelf", "Dining Table", "Sofa"],
    "Grocery": ["Banana", "Organic Avocado", "Almond Milk", "Whole Grain Bread", "Free-range Eggs"],
    "Cosmetic": ["Facial Cleanser", "Moisturizing Cream", "Sunscreen SPF 50", "Lip Balm", "Foundation"],
    "Appliances": ["Washing Machine", "Refrigerator", "Microwave Oven", "Blender", "Electric Kettle"],
    "Outdoor": ["Tent", "Sleeping Bag", "Camping Stove", "Hiking Backpack", "Cycling Helmet"],
    "Sports": ["Running Shoes", "Yoga Mat", "Dumbbells", "Treadmill", "Exercise Bike"],
    "Toys": ["Building Blocks", "Action Figure", "Dollhouse", "Puzzle", "Board Game"],
    "Books": ["Mystery Novel", "Science Fiction", "Biography", "Cookbook", "Children's Book"],
    "Automotive": ["Car Battery", "Tire", "Engine Oil", "Brake Pads", "Car Vacuum"]
}

# Define descriptions
descriptions = [
    "High quality product for daily use",
    "Eco-friendly and durable",
    "Best in class performance",
    "Compact and lightweight design",
    "Stylish and affordable",
```

```
    "Perfect for outdoor activities",
    "Long-lasting battery life",
    "Water-resistant and reliable",
    "Advanced technology integration",
    "Essential for modern lifestyle",
    "Made from premium materials",
    "Provides excellent comfort and support",
    "Energy-efficient and silent",
    "Easy to clean and maintain",
    "Perfect gift for friends and family",
    "High speed and efficiency",
    "Includes warranty and customer support",
    "Adjustable settings for personalized use"
]

# Generate product data
num_records = 200
product_data = []

for i in range(num_records):
    category = random.choice(list(category_brands.keys()))
    brand = random.choice(category_brands[category])
    product_name = f"{brand} {random.choice(product_names[category])}"
    product_data.append({
        "Product_number": i + 1,
        "Product_name": product_name,
        "Colour": random.choice(["Black", "Blue", "White", "Green", "Red", "Silver", "Gray", "Pink"]),
        "Brand": brand,
        "Depth": round(random.uniform(0, 50), 2),
        "Width": round(random.uniform(0, 50), 2),
        "Height": round(random.uniform(0, 50), 2),
        "Weight": round(random.uniform(0.1, 5.0), 2),
        "Price": round(random.uniform(10, 1000), 2),
        "Warranty_period": random.randint(1, 5),
        "Category": category,
        "Description": random.choice(descriptions),
        "Availability": random.randint(0, 1),
        "Stock": random.randint(1, 500)
    })

# Convert to DataFrame and save as CSV file
product_df = pd.DataFrame(product_data)
file_path = "product_data_realistic.csv"
product_df.to_csv(file_path, index=False)
```

To add a `Review_date` column with random dates from 1 to 180 days after the `Order_date` in your `review table.csv`, I'll show you how to do it with Pandas. Here's the code:

python Copy

```python
import pandas as pd
from datetime import timedelta
import random

# Load the CSV file
review_df = pd.read_csv('review table.csv')

# Ensure 'Order_date' is in datetime format
review_df['Order_date'] = pd.to_datetime(review_df['Order_date'])

# Generate 'Review_date' as 1 to 180 days after 'Order_date'
review_df['Review_date'] = review_df['Order_date'] + pd.to_timedelta(
    [random.randint(1, 180) for _ in range(len(review_df))], unit='D'
)

# Display the updated DataFrame
print(review_df.head())

# Save the updated DataFrame back to CSV
review_df.to_csv('updated_review_table.csv', index=False)
```

Here's the full content of the REVIEW CSV file with the specified requirements. Please copy this into a text editor and save it as `updated_review_data.csv`.

Copy

```
Review_number,Order_number,Review_date,Product_name,Text,Ranking,Product_number
1,100234,2024-06-01,Product_A,"Great quality, very satisfied with the purchase."
2,100235,2024-07-10,Product_B,"Item was as described, highly recommended.",4,
3,100237,2024-07-20,Product_C,"Product quality is decent, but delivery was delay
4,100238,2024-08-18,Product_D,"Exceeded my expectations, will buy again.",5,
5,100240,2024-09-02,Product_E,"Not as good as expected, but still okay.",3,
6,100242,2024-06-01,Product_F,"Affordable price and decent quality.",4,
7,100245,2024-07-05,Product_G,"Very disappointed, item not as described.",2,
8,100246,2024-07-16,Product_H,"Packaging was damaged, but product is fine.",3,
9,100249,2024-08-10,Product_I,"Fast shipping and excellent product quality.",5,
10,100251,2024-09-01,Product_J,"Item is okay, but color was slightly off.",3,
```

Questions&Trigger

Q5.1

The goal of this query is to retrieve a comprehensive list of all customers along with their respective orders, showing details in each order, and the total amount paid after deductions.

I started by selecting key customer information from the CONSUMER table.

2. Then, I joined the ORDERS table on the Email field to retrieve orders associated with each customer.

3. Using the INCLUDES table, I linked each order to the products within it, and from the PRODUCT table, I retrieved the product names.

4. I used GROUP_CONCAT to display the list of products in each order as a single string, making the result more readable.

5. Finally, I calculated the total amount paid per order by subtracting any Deduction_Promotion from the Subtotal.

	Customer_Name	Customer_Email	Order_number	Order_date	Product_List	Total_Paid
1	Aaron	natalie24@example.net	3	2024-01-16	Hasbro Dollhouse,Estée Lauder Facial Cleanser,Columbia ...	649.972
2	Amy	joe99@example.org	60	2024-08-20	Bosch Engine Oil	564.724
3	Amy	joe99@example.org	215	2024-10-19	Bosch Engine Oil	533.424
4	Andrew	paulhardin@example.org	97	2024-10-26	Samsung Tablet,Maybelline Lip Balm	957.996
5	Andrew	paulhardin@example.org	134	2024-04-23	Columbia Hiking Backpack,Samsung Tablet	1320.802
6	Andrew	paulhardin@example.org	179	2024-08-24	Columbia Hiking Backpack,Samsung Tablet	1308.372
7	Angela	christopher46@example.net	185	2024-07-17	Publix Banana,Steelcase Standing Desk	83.126
8	Anna	harveybrooke@example.com	1	2024-02-07	Black Diamond Camping Stove	376.628
9	Anthony	meganwheeler@example.net	141	2024-08-15	Steelcase Bookshelf,IKEA Bookshelf,Clinique Facial ...	1659.17
10	Anthony	meganwheeler@example.net	158	2024-07-19	Steelcase Standing Desk,HP Headphones	289.374
11	Anthony	moranchristopher@example.net	72	2024-06-09	Fisher-Price Building Blocks,Lego Action Figure,Fisher-...	736.35
12	Anthony	moranchristopher@example.net	114	2024-03-31	Columbia Hiking Backpack,Fisher-Price Building ...	1954.528
13	Anthony	moranchristopher@example.net	190	2024-01-06	Columbia Hiking Backpack	407.826
14	Anthony	smithjulie@example.net	49	2024-01-09	HarperCollins Science Fiction,Fisher-Price Building ...	1844.514
15	Anthony	smithjulie@example.net	92	2024-07-10	HarperCollins Biography,KitchenAid Electric Kettle	70.33
16	Ashley	johnsonnancy@example.com	193	2024-08-28	Columbia Hiking Backpack,Steelcase Desk Chair,Trader ...	1466.81
17	Audrey	darren04@example.org	194	2024-07-17	Lego Building Blocks,HP Tablet	814.846
18	Audrey	darren04@example.org	210	2024-03-19	Lego Building Blocks,Under Armour Treadmill,HP Tablet	959.324
19	Brenda	dyerbrittany@example.com	44	2024-08-31	Steelcase Standing Desk	53.4
20	Brenda	dyerbrittany@example.com	54	2024-02-19	Estée Lauder Facial Cleanser	179.834
21	Brenda	dyerbrittany@example.com	132	2024-04-07	Safeway Banana,Steelcase Standing Desk,Estée Lauder ...	242.126
22	Brenda	dyerbrittany@example.com	159	2024-05-28	Safeway Banana	-27.138
23	Brenda	dyerbrittany@example.com	174	2024-01-02	Safeway Banana,Estée Lauder Facial Cleanser	164.106
24	Brian	rachelcarter@example.org	153	2024-11-01	Black Diamond Camping Stove	420.588
25	Brian	vrichardson@example.com	52	2024-08-23	Safeway Banana,Nike Dumbbells,Adidas Exercise ...	2557.032
26	Brian	vrichardson@example.com	68	2024-10-28	Safeway Banana,Nike Dumbbells,Adidas Exercise ...	2367.424
27	Carolyn	galvanchelsea@example.org	95	2024-07-04	Publix Banana	-33.694

Q5.2

For this query, we need to identify customers who have items in their baskets so that targeted promotions can be made. This involves retrieving customer identification information, birthdays, gift card balances (if any), and the products in their baskets.

I started by selecting key customer information from the CONSUMER table.
2. Then, I joined the ORDERS table on the Email field to retrieve orders associated with each customer.
3. Using the INCLUDES table, I linked each order to the products within it, and from the PRODUCT table, I retrieved the product names.
4. I used GROUP_CONCAT to display the list of products in each order as a single string, making the result more readable.
5. Finally, I calculated the total amount paid per order by subtracting any Deduction_Promotion from the Subtotal.

	Customer_Email	Customer_Name	Birthday	Gift_Card_Balance	Basket_Products
1	adam23@example.com	Amanda	1979-12-14	NULL	31,8,55,63,21,54
2	aking@example.org	Kimberly	1983-07-01	1865.61	16,3,88,17,82,30,71,63
3	albertreyes@example.net	Courtney	1995-09-14	NULL	10,55,56,14
4	allenjennifer@example.org	Eric	1951-05-11	NULL	36,77
5	amandatrevino@example.org	Joshua	1956-12-28	1358.46	7,4,5,30,65,69
6	andersonjennifer@example.org	Michael	1951-02-02	880.24	48,84,9,65,61,47,75,69
7	andersonkirk@example.com	Linda	1971-05-29	NULL	9,47,26,38,28,49,69,94,55
8	andreamatthews@example.com	Kelly	1988-07-26	NULL	85,95,56,90,63,18
9	andrewgomez@example.com	Christy	1964-02-08	1134.57	96,34,3,22,10,82,11,88,91
10	april56@example.net	Heather	1969-07-13	654.59	12,25,67
11	ashley19@example.net	Drew	1954-08-12	2258.53	51,99,83,76,18,4,36,31
12	ashley60@example.org	Emily	1948-10-28	1101.57	65,46,47,12,10
13	ashleyfrank@example.org	Wesley	1946-04-28	830.86	55
14	barrerawilliam@example.com	Jennifer	1974-12-17	NULL	54
15	bensonjoshua@example.net	Ana	1986-05-09	NULL	99

Q5.3

This query aims to find the two best-selling products in each category based on total sales. The information will help the company ensure these products are adequately stocked and promoted. To achieve this, I calculated the total sales for each product by multiplying the quantity sold by its price, then ranked these products within their categories.

1. I used a CTE (Sales) to calculate the total sales for each product in each category by joining PRODUCT, INCLUDES, and ORDERS.
2. I grouped by Category and Product_name, using SUM to accumulate the total sales of each product.
3. I applied the RANK() window function in another CTE (RankedSales) to rank products by total sales within each category.
4. Finally, I filtered the results to only include the top two products per category, ordered by category and rank.

	Category	Product_name	Total_sales
1	Appliances	KitchenAid Microwave Oven	3666.08
2	Appliances	Whirlpool Microwave Oven	3476.85
3	Automotive	Goodyear Car Vacuum	7044.6
4	Automotive	Continental Car Vacuum	6957.9
5	Books	Simon & Schuster Children's Book	352.08
6	Books	HarperCollins Biography	304.72
7	Cosmetic	Estée Lauder Facial Cleanser	1569.7
8	Cosmetic	Maybelline Facial Cleanser	530.6
9	Electronics	Samsung Tablet	6059.84
10	Electronics	HP Laptop	5915.91
11	Furniture	IKEA Desk Chair	6504.63
12	Furniture	IKEA Bookshelf	5166.88
13	Grocery	Whole Foods Almond Milk	893.48
14	Grocery	Kroger Free-range Eggs	134.16
15	Outdoor	Columbia Hiking Backpack	5291.7
16	Outdoor	Black Diamond Camping Stove	4953.76
17	Sports	Nike Yoga Mat	9417.4
18	Sports	Reebok Exercise Bike	6959.8
19	Toys	Nerf Action Figure	8524.46
20	Toys	Lego Action Figure	6322.94

Q5.4

The objective of this query is to compute the monthly total sales, adjusted for any refunds, and to calculate the month-over-month growth rate in sales. This information is valuable for assessing sales trends and understanding the company's financial performance.

1. I used a CTE (MonthlySales) to calculate total sales per month, joining ORDERS with RETURN to account for refunded orders. I used STRFTIME to extract year and month from Order_date.
2. Using SUM on adjusted totals (after deducting refunds), I calculated the monthly sales.
3. In a second CTE (SalesWithGrowth), I employed the LAG function to get the sales from the previous month, which I used to calculate the month-over-month growth rate.
4. I handled cases where previous month sales data was unavailable by returning NULL for those instances.

	Year	Month	Total_sales	Sales_growth_percentage
1	2024	01	7695.41	*NULL*
2	2024	02	19727.06	156.35
3	2024	03	23591.81	19.59
4	2024	04	16782.93	-28.86
5	2024	05	14266.76	-14.99
6	2024	06	18635.17	30.62
7	2024	07	18997.22	1.94
8	2024	08	16844.42	-11.33
9	2024	09	8795.71	-47.78
10	2024	10	21417.76	143.5
11	2024	11	2695.03	-87.42

Q6 (ii)

1. **Identify the Payment Type Condition**

The first step was to determine whether the order is paid using a gift card. This was done by checking the Payment_type in the PAYMENT table based on the Payment_ID associated with the new order. The WHEN clause is used here to filter only orders where Payment_type equals 'VOUCHERS_GIFT', ensuring that the trigger only applies to gift card payments.

2. Select the Gift Card Balance

After establishing that the order is a gift card payment, the next step is to retrieve the available balance from the VOUCHERS_GIFT table. The balance is accessed using Payment_ID, which is the link between the ORDERS and VOUCHERS_GIFT tables. This step is essential to verify if the gift card can cover the order cost.

3. Calculate the Order Total

The total amount needed for the order is calculated as Subtotal - Deduction_Promotion. This calculation allows for any deductions or promotions that might reduce the total cost, providing a precise figure for comparison against the gift card balance.

4. Add the Trigger Condition

With both the order total and gift card balance identified, a CASE statement is used to perform the comparison. If the balance is less than the order's calculated total, the trigger raises an error to abort the transaction. The RAISE(ABORT, 'Order total exceeds the gift card balance.') command is used to stop the insertion if this condition is met, thus preventing orders from being processed without sufficient funds.

5. Finalizing the Trigger Structure

The BEGIN...END structure encapsulates the entire condition and ensures the trigger is executed BEFORE INSERT on the ORDERS table. This means that the balance check occurs before any new order data is committed to the database, preserving data consistency and avoiding potential issues with insufficient funds.

```
# Function to test inserting an order with the trigger
def insert_order(order_number, payment_id, subtotal, deduction_promotion):
    try:
        cursor.execute("""
            INSERT INTO ORDERS (Order_number, Order_date, Product, Subtotal, Deduction_Promotion, Email, Payment_ID)
            VALUES (?, DATE('now'), 'Sample Product', ?, ?, 'test_email@example.com', ?)
        """, (order_number, subtotal, deduction_promotion, payment_id))

        # Commit if successful
        conn.commit()
        print(f"Order {order_number} inserted successfully.")

    except sqlite3.IntegrityError as e:
        # Trigger error message
        print(f"Trigger fired for Order {order_number}: {e}")
        conn.rollback()

    except Exception as e:
        # General error
        print(f"An error occurred: {e}")
        conn.rollback()

# Test Cases
# Case 1: Order that passes (balance is sufficient)
insert_order(order_number=500, payment_id=2, subtotal=1000, deduction_promotion=10)

# Case 2: Order that triggers the trigger (balance insufficient)
insert_order(order_number=501, payment_id=5, subtotal=1500, deduction_promotion=10)
```

✓ 0.0s

```
Order 500 inserted successfully.
Trigger fired for Order 501: Order total exceeds the gift card balance.
```

I created a query that successfully passes the trigger (when the balance is sufficient) and another query that intentionally triggers an error (when the balance is insufficient), resulting in a trigger firing and displaying an error message. The test cases for both scenarios are available in the **Test-Trigger.ipynb** file on GitHub.

Reference：

https://www.datacamp.com/tutorial/creating-synthetic-data-with-python-faker-tutorial