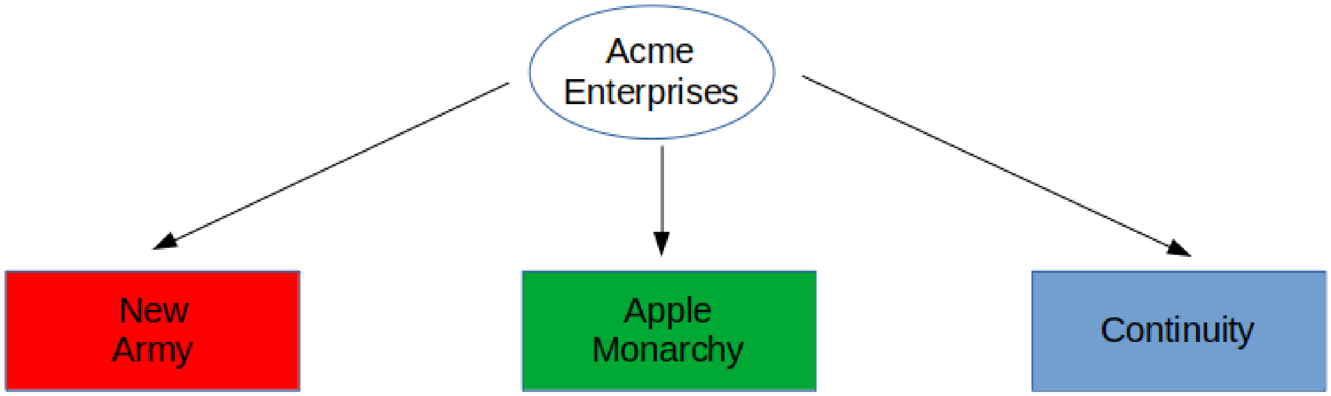
Project requirements:

1. Develop an interface that depicts class inheritance hierarchy
2. Leverage polymorphism to allow dynamic binding of object methods belonging to different classes

Problem Statement:

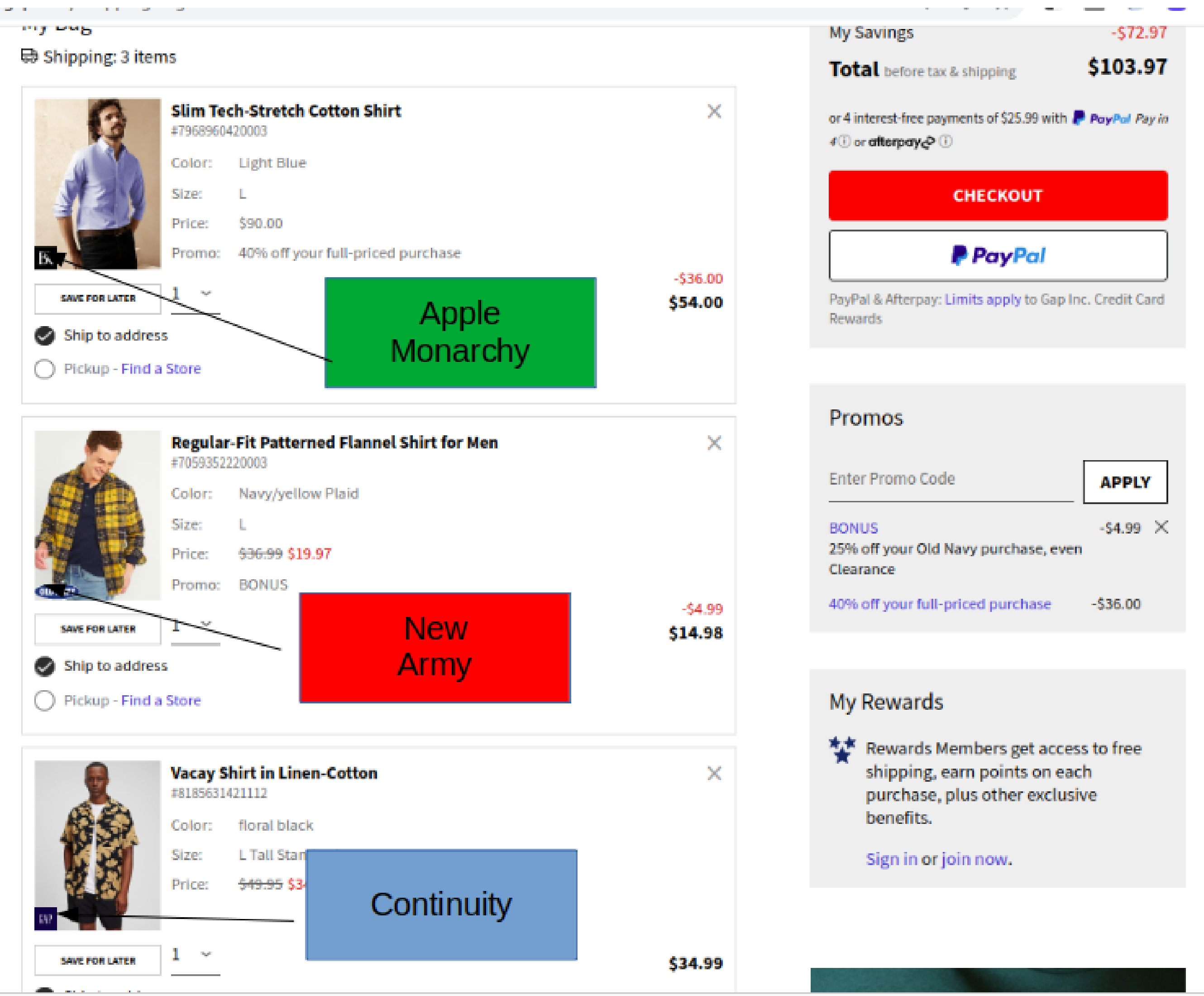
We have been tasked with Acme Enterprises to develop enterprise wide application that will capture consumer sales across all of their brands within their online retail umbrella.
This is the organization chart -



The different brands and their focus areas are -

- New Army -
  - Target demographic – middle class
  - Target consumer types – wholesome
  - Brand focus – value oriented
  - Brand loyalty – low
- Continuity -
  - Target demographic – age group between 18 and 25
  - Target consumer types – casual
  - Brand focus – classic
  - Brand loyalty – medium
- Apple Monarchy -
  - Target demographic – working professionals
  - Target consumer types – sophistication
  - Brand focus - exclusivity
  - Brand loyalty - high

Acme's enterprise team has been tasked with creating a universal shopping bag that could be plugged across these individual organizations.



My Bag

Shipping: 3 items

Slim Tech-Stretch Cotton Shirt
#7968960420003

Color: Light Blue
Size: L
Price: $90.00
Promo: 40% off your full-priced purchase

SAVE FOR LATER 1

-$36.00
$54.00

● Ship to address
○ Pickup - Find a Store

Apple Monarchy

Regular-Fit Patterned Flannel Shirt for Men
#7059352220003

Color: Navy/yellow Plaid
Size: L
Price: ~~$36.99~~ $19.97
Promo: BONUS

SAVE FOR LATER 1

-$4.99
$14.98

● Ship to address
○ Pickup - Find a Store

New Army

Vacay Shirt in Linen-Cotton
#8185631421112

Color: floral black
Size: L Tall Stan
Price: ~~$49.95~~ $3

Continuity

SAVE FOR LATER 1

$34.99

My Savings -$72.97

Total before tax & shipping **$103.97**

or 4 interest-free payments of $25.99 with PayPal Pay in 4 ⓘ or afterpay ⓘ

CHECKOUT

PayPal

PayPal & Afterpay: Limits apply to Gap Inc. Credit Card Rewards

Promos

Enter Promo Code APPLY

BONUS -$4.99 ×
25% off your Old Navy purchase, even Clearance

40% off your full-priced purchase -$36.00

My Rewards

Rewards Members get access to free shipping, earn points on each purchase, plus other exclusive benefits.

Sign in or join now.

We will be developing a class hierarchy that can be used as a universal shopping cart for all of the individual brands within Acme Enterprises.

Brand Hierarchy -
Based on the organization chart, we can derive the following about the brand hierarchy -

- Continuity -
  - This is the middle of the road brand and covers a wider audience
- New Army -
  - This is the entry level brand
  - The consumers tend to be savings oriented and are attracted by offers
  - There is limited brand loyalty
- Apple Monarchy -
  - This is the flagship brand
  - The consumers tend to be top-earning in their age ranges.
  - There is strong brand loyalty

Class hierarchy -
We will use established brand hierarchy from the previous section to create a Class hierarchy.
Let’s start with a base class called Bag.

Bag -
This class will need to capture the following information -

- Consumer details -
    - First Name
    - Last Name
    - Shipping Address details
        - Address, City, State and Zip
- Purchase details -
    - Product SKU
    - Product Quantity
    - Product Unit Price
- Brand

This class will also have the following functionality -

- Calculate total cost of the purchase
    - Total Cost = [UnitPrice * Quantity] for all SKU’s
- Create a common customer id that can be used across brands
- Store purchase data in a database -
    - We need to capture what a consumer bought within each brand separately
    - We need to capture the consumer purchase totals as revenue for global Acme brand

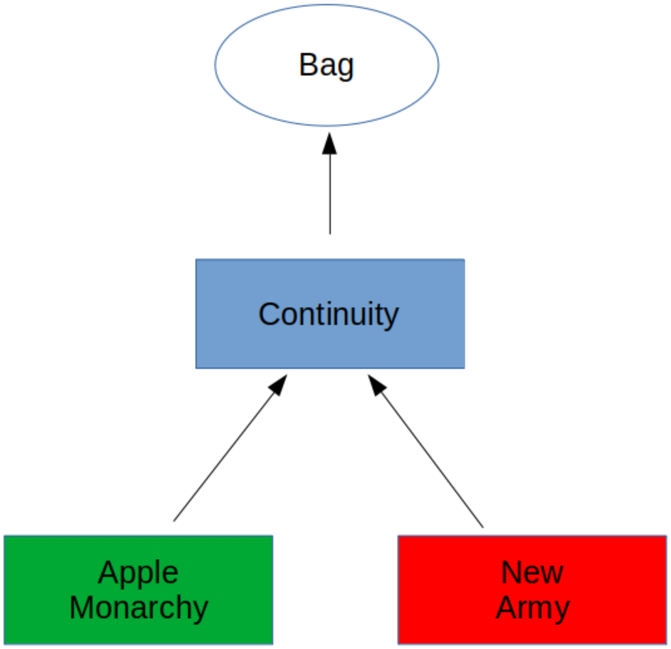
We will be keeping the Bag class abstract to setup inheritance within brands.

Continuity
Let’s move on to a class that we would use for the brand Continuity.
This class will inherit all of the features of the Bag class. We will be storing all of the purchase data for Continuity Clothing in a separate schema and associated table in the company database. We will capture the sales totals as revenue in a separate schema and table for Acme. This class would cover the entire implementation of ShoppingBag with no special features.

NewArmy
Let’s move on to a class that we would use for the brand New Army
This class will inherit all of the features of the Continuity class. We will be storing all of the purchase data for New Army in a separate schema and associated table in the company database. We will capture the sales totals as revenue in a separate schema and table for Acme. This class would cover the entire implementation of ShoppingBag but has an additional discount feature. Consumers for this brand are attracted by coupons. We need to account for coupon discounts.

AppleMonarchy
Let’s move on to a class that we would use for the brand Apple Monarchy.
This class will inherit all of the features of the Bag class. We will be storing all of the purchase data for Apple Monarchy in a separate schema and associated table in the company database. We will capture the sales totals as revenue in a separate schema and table for Acme. This class would cover the entire implementation of ShoppingBag but has an additional loyalty point feature.
Consumers tend to be loyal to this brand and are rewarded for their loyalty by using their points to get discounts.

Class relationship



Project Implementation details

JSON inputs

We will be serving JSON data as inputs to our classes. We will be using the nlohmann JSON library to accomplish this. A good number of modern web services use JSON and we felt that we should provide support for it. The nlohmann JSON library was simple to interact and can be attached to our source code as its only two header files.

Database targets

We are using PostgreSQL as our database to host the data written by our classes. The implementation is specific for Ubuntu 20.04. To support the PostgreSQL integration, we need to -

- Install PostgreSQL database server
  - Update properties so that it can start accepting connections
- Install the C-based libpq library
  - Serves as API interface for PostgreSQL. Serves as underlying engine for other programming interfaces – C++/Python/Perl – specifically libpqxx. This gets installed when PostgreSQL is installed
- Install the C++ libpqxx library
  - This is the primary C++ library to interact with PostgreSQL.

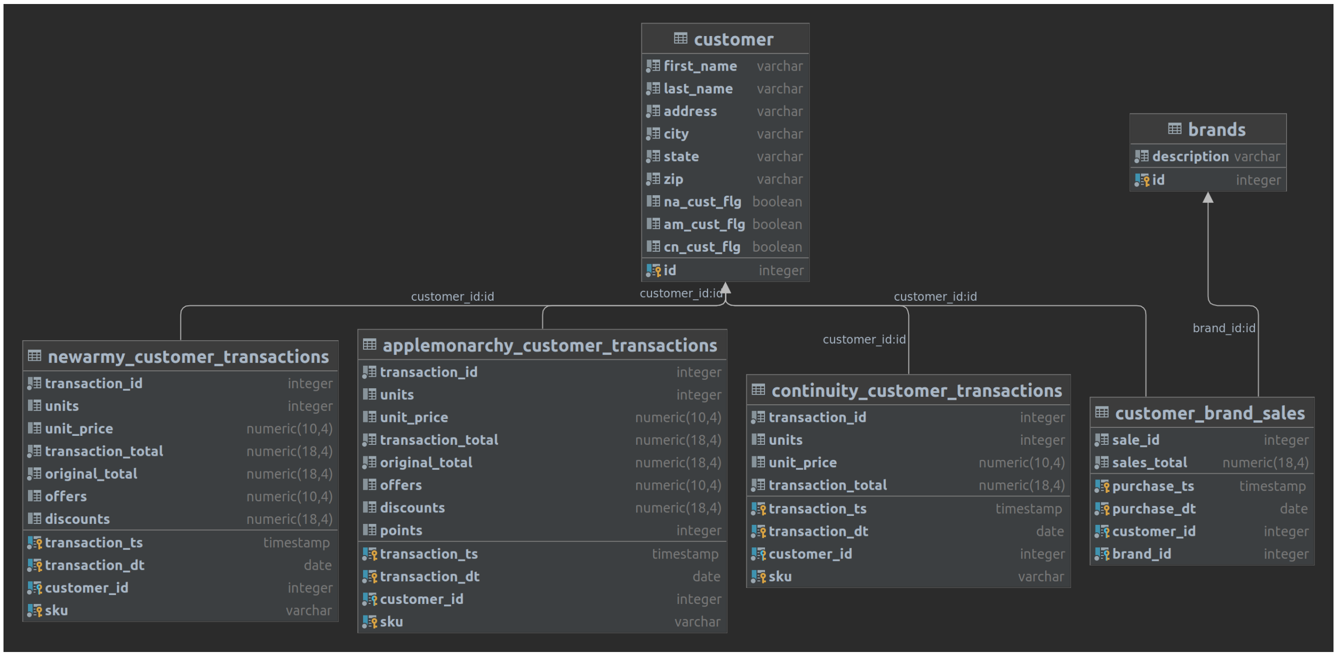
We have added the steps to install and setup the libraries in the appendix section.

After PostgreSQL has been setup, we will create the necessary database objects starting with database (AcmeDB). Within the AcmeDB, we create separate schemas – ACME, CONTINUITY, NEWARMY and APPLEMONARCHY.

The ACME schema will host the customer information table and broader Sales table that connects customers and their sales activity across the three brands.

The CONTINUITY schema will host the purchase transaction table detailing out what each customer purchased on the Continuity brand. The NEWARMY schema will host the purchase transaction table details out what each customer purchased on the NewArmy brand. Lastly, the APPLEMONARCHY schema will host the purchase transaction table that what each customer purchased on the AppleMonarchy brand.

This is the data model -



customer	
first_name	varchar
last_name	varchar
address	varchar
city	varchar
state	varchar
zip	varchar
na_cust_flg	boolean
am_cust_flg	boolean
cn_cust_flg	boolean
id	integer

brands	
description	varchar
id	integer

newarmy_customer_transactions	
transaction_id	integer
units	integer
unit_price	numeric(10,4)
transaction_total	numeric(18,4)
original_total	numeric(18,4)
offers	numeric(10,4)
discounts	numeric(18,4)
transaction_ts	timestamp
transaction_dt	date
customer_id	integer
sku	varchar

applemonarchy_customer_transactions	
transaction_id	integer
units	integer
unit_price	numeric(10,4)
transaction_total	numeric(18,4)
original_total	numeric(18,4)
offers	numeric(10,4)
discounts	numeric(18,4)
points	integer
transaction_ts	timestamp
transaction_dt	date
customer_id	integer
sku	varchar

continuity_customer_transactions	
transaction_id	integer
units	integer
unit_price	numeric(10,4)
transaction_total	numeric(18,4)
transaction_ts	timestamp
transaction_dt	date
customer_id	integer
sku	varchar

customer_brand_sales	
sale_id	integer
sales_total	numeric(18,4)
purchase_ts	timestamp
purchase_dt	date
customer_id	integer
brand_id	integer

Classes and Objects

Bag class

The starting point is the abstract Bag class. This class has the following data members, broken down by types -

1. String
    - custBrand
        - Denotes the brand
    - custPurchaseTime
        - Denotes the timestamp when the purchase was made
    - custFirstName
        - Denotes the Customer’s first name. Will be blank for existing customers.
    - custLastName
        - Denotes the Customer’s last name. Will be blank for existing customers.
    - custAddress
        - Denotes the Customer’s Address. Will be blank for existing customers.
    - custCity
        - Denotes the Customer’s Address. Will be blank for existing customers.
    - custState
        - Denotes the Customer’s State. Will be blank for existing customers.
    - custZip
        - Denotes the Customer’s ZipCode. Will be blank for existing customers.

2. JSON
    - custPurchases
        - Denotes the Customer’s purchases. We expect the following JSON keys are present -
            - SKU
                - Denotes Product code
            - UNITS
                - Denotes how many units being bought
            - UNIT_PRICE
                - Denotes unit price

3. Integer
    - custID
        - Denotes the Customer id. The customer id will not be present for new customers. These will be defaulted to -999 initially. Existing customers will have a CUSTOMER_ID key in the incoming JSON record.

4. Double
    - custTotal
        - Denotes the total of all purchases by a customer across all SKU transactions within a brand.

Constructor

The explicit constructor that we will be using is

- Bag(const nlohmann::json &purchase_json)
    - This constructor will call all the necessary setters which internally will validate the necessary data points from the JSON.

Setters

The setters used are -

- void setCustomerBrand(const nlohmann::json &purchase_json);
    - This method will set the custBrand private data member.
    - This method will throw a runtime error if the BRAND key is not present in the JSON data

- void setCustomerPurchaseTime(const nlohmann::json &purchase_json);
    - This method will set the custPurchaseTime private data member.
    - This method will throw a runtime error if the PURCHASE_TIME key is not present in the JSON data

- void setCustomerFirstName(const nlohmann::json &purchase_json);
    - This method will set the custFirstName private data member.
    - This method will throw a runtime error if the FIRST_NAME key is not present in the JSON data OR if the CUSTOMER_ID key is not present.

- void setCustomerLastName(const nlohmann::json &purchase_json);
    - This method will set the custLastName private data member.
    - This method will throw a runtime error if the LAST_NAME key is not present in the JSON data OR if the CUSTOMER_ID key is not present.

- void setCustomerAddress(const nlohmann::json &purchase_json);
    - This method will set the custAddress private data member.
    - This method will throw a runtime error if the ADDRESS key is not present in the JSON data OR if the CUSTOMER_ID key is not present.

- void setCustomerCity(const nlohmann::json &purchase_json);
    - This method will set the custCity private data member.
    - This method will throw a runtime error if the CITY key is not present in the JSON data OR if the CUSTOMER_ID key is not present.

- void setCustomerState(const nlohmann::json &purchase_json);
    - This method will set the custState private data member.
    - This method will throw a runtime error if the STATE key is not present in the JSON data OR if the CUSTOMER_ID key is not present.

- void setCustomerZip(const nlohmann::json &purchase_json);
    - This method will set the custZip private data member.
    - This method will throw a runtime error if the ZIP key is not present in the JSON data OR if the CUSTOMER_ID key is not present.

- void setCustomerPurchaseDetails(const nlohmann::json &purchase_json);
    - This method will set the custPurchases private data member.
    - This method will throw a runtime error if the CUSTOMER_PURCHASES key is not present in the JSON data
    - This method will throw a runtime error if the SKU, UNITS or UNIT_PRICE keys are not present within CUSTOMER_PURCHASES JSON value.

- void setCustomerId(const nlohmann::json &purchase_json);
    - This method will set the custId private data member for an existing customer by referencing CUSTOMER_ID key
    - If the CUSTOMER_ID is not there, it will default to -999

Sample example -

```
void Bag::setCustomerLastName(const nlohmann::json &purchase_json) {
  // if json structure already has a customer id -> returning customer!
  if(purchase_json.contains("CUSTOMER_ID")){
    // assign default
    this->custLastName = " ";
  } else {
    // now you should check if Last Name was provided or not!
    if(purchase_json.contains("LAST_NAME")){
      this->custLastName = purchase_json.at("LAST_NAME");
    } else {
      throw std::runtime_error("LAST_NAME key not found: " + std::string(purchase_json.dump()));
    }
  }
}
```

Calculating Sales

We will be using setCustomerPurchaseTotals method to calculate total purchase sales for a customer. We will be declaring this method as virtual as we have to override in derived classes later on to account for discounts or points. In the Bag base class, this is the behavior we initialize a variable and iterate over the Purchase Details container and derive total sales, by multiplying UnitPrice with UnitSales.

```
void Bag::setCustomerPurchaseTotals(){
    // get all purchase details
    nlohmann::json purchaseDetails = getCustomerPurchases();
    // declare a sum variable
    double total = 0;
    // iterate over the purchase details
    for(nlohmann::json::iterator it = purchaseDetails.begin(); it != purchaseDetails.end(); ++it){
        total += static_cast<double>(it->at("UNIT_PRICE")) * static_cast<int>(it->at("UNITS"));
    }
    // let's assign to this private data member
    this->custTotal = total;
}
```

Getters

The getters used are -

- void std::string getCustomerBrand()
  - This method will set the site brand the customer is on
- std::string getCustomerPurchaseTime()
  - This method will return the customer's purchase time
- std::string getCustomerFirstName()
  - This method will return the customer's first name
- std::string getCustomerLastName()
  - This method will return the customer's last name
- std::string getCustomerAddress()
  - This method will return the customer's address
- std::string getCustomerCity()
  - This method will return the customer's city
- std::string getCustomerState()
  - This method will return the customer's state
- std::string getCustomerZip()
  - This method will return the customer's zip
- nlohmann::json getCustomerPurchases()
  - This method will return the customer's purchases in JSON
- int getCustomerId() const

- This method will return the customer id

- virtual double getCustomerTotals() const
  - This method will calculate the total of customer's purchases
  - This method is declared virtual so that different derived classes can return dedicated private cost data members

Sample example -

```
int Bag::getCustomerId() const {
    return this->custId;
}
```

Database operations

These methods are used to create database entries-

- void Bag::insertCustomerRecord()
  - This creates customer id record for new customers into ACME.CUSTOMER table
  - This method is inherited across all derived classes

```
void Bag::insertCustomerRecord() {
    try {
        // let's use the function to get the connection string
        std::string cnxDetails = read_connection();
        // let's use the connection string to set up a connection object using PQXX library
        pqxx::connection cnx{cnxDetails};
        // Using the connection object, we will create a transaction to run our SQL instructions against
        pqxx::work txn{cnx};

        // Let's set the brand flags while we are here
        char newArmyFlg = (this->getCustomerBrand() == "NEW_ARMY") ? '1' : '0';
        char appleMonarchyFlg = (this->getCustomerBrand() == "APPLE_MONARCHY") ? '1' : '0';
        char continuityFlg = (this->getCustomerBrand() == "CONTINUITY") ? '1' : '0';

        // Now we will build the SQL to be inserted using private data members
        std::string buildSQL =
                        std::string("INSERT  INTO  ACME.CUSTOMER(FIRST_NAME,  LAST_NAME,  ADDRESS,  CITY,  STATE,  ZIP,
NA_CUST_FLG, AM_CUST_FLG, CN_CUST_FLG) VALUES(") +
                "'" + this->getCustomerFirstName() + "'" + "," +
                "'" + this->getCustomerLastName() + "'" + "," +
                "'" + this->getCustomerAddress() + "'" + "," +
                "'" + this->getCustomerCity() + "'" + "," +
                "'" + this->getCustomerState() + "'" + "," +
                "'" + this->getCustomerZip() + "'" + "," +
                "'" + newArmyFlg + "'" + "," +
                "'" + appleMonarchyFlg + "'" + ',' +
                "'" + continuityFlg + "'" + ") ON CONFLICT DO NOTHING" + ";";

        // We will supply the buildSQL and execute within the transaction
        pqxx::result insertRes{txn.exec(buildSQL)};
        // display the SQL
        std::cout << "Customer Insert Query executed: " << buildSQL << std::endl;
        txn.commit();
    }
    catch(pqxx::sql_error const &e){
        std::cout << "SQL error: " << e.what() << std::endl;
        std::cerr << "SQL error: " << e.what() << std::endl;
        std::cout << "Query was:; " << e.query() << std::endl;
        std::cerr << "Query was:; " << e.query() << std::endl;
    }
}
```

- void Bag::retrieveCustomerIdDB()
    - This retrieves the customerId from ACME.CUSTOMER table and assigns it to the customerId data member
    - This method is inherited across all derived classes

- int Bag::retrieveBrandId()
    - This retrieves the brand Id primary key from the ACME.BRANDS table associated with the Brand
    - This method is inherited across all derived classes

- void Bag::insertCustomerBrandSaleTotals()
    - This method will write into ACME.CUSTOMER_BRAND_SALES the total sales across a customer and brand on a given purchase timestamp
    - This method is inherited across all derived classes

- void Bag::insertCustomerTransactions()
    - This method is empty in the base class
    - It is declared virtual so that it be overridden by the derived class

Tying it together
The entryMethod ties all the methods together.

```
void Bag::entryMethod() {
  // This will be the primary method for Bag Class
  // Flow -
  // 1) Calculate customerPurchaseTotals
  // 2) Check the customerId -
  //  a) If -999, this is a new customer, will create an entry in ACME.CUSTOMER table
  //     i) Perform additional checks to see if data is valid or not.
  //  b) If not -999, this is existing customer, will NOT create an entry in ACME.CUSTOMER table

  // Calculate customerPurchaseTotals
  setCustomerPurchaseTotals();

  // Proceed to write info. to database
  if(getCustomerId() == -999 && getCustomerFirstName() != " " && getCustomerLastName() != " " &&
  getCustomerAddress() != " " && getCustomerState() != " " && getCustomerCity() != " " &&
  getCustomerZip() != " "){
    // Insert records into Customer table
    insertCustomerRecord();
    // Retrieve customer id associated with insert
    retrieveCustomerIdDB();
    // Insert records into Customer Brand Sales table
    insertCustomerBrandSaleTotals();
    // Insert records into Customer Transactions table
    insertCustomerTransactions();
  } else if (getCustomerId() != -999){
    // This is existing customer flow -
    // Insert records into Customer Brand Sales table
    insertCustomerBrandSaleTotals();
    // Insert records into Customer Transactions table
    insertCustomerTransactions();
  } else {
    throw std::runtime_error("Unexpected behavior path!");
  }
}
```

The entry method is declared virtual as derived classes will be able to override their respective insertCustomerTransactions. At the same time, we have locked the method with final so that all derived classes have the same orchestration behavior.

Derived Classes

There are 3 derived classes -

- Continuity
- NewArmy
- AppleMonarchy

Continuity

This class pretty much inherits all of the abstract Bag class. It does not introduce any new data members. There is only 1 method (insertCustomerTransactions) which gets overridden with functionality to load data into CONTINUITY.CUSTOMER_TRANSACTIONS table.

```
void Continuity::insertCustomerTransactions() {
    try {
        // let's use the function to get the connection string
        std::string cnxDetails = read_connection();

        // let's use the connection string to set up a connection object using PQXX library
        pqxx::connection cnx{cnxDetails};

        // Using the connection object, we will create a transaction to run our SQL instructions against
        pqxx::work txn{cnx};

        // Let's get all the purchase details
        nlohmann::json purchaseDetails = getCustomerPurchases();

        // we will need to iterate over the purchase details to understand the components
        for(nlohmann::json::iterator it = purchaseDetails.begin(); it != purchaseDetails.end(); ++it){
            std::string sku = it->at("SKU");
            int units = it->at("UNITS");
            double unitPrice = it->at("UNIT_PRICE");
            double txnTotal = unitPrice*units;

            // Now we will build the SQL to be inserted using private data members
            std::string buildSQL =
                                                                                                    std::string("INSERT        INTO
CONTINUITY.CUSTOMER_TRANSACTIONS(TRANSACTION_TS,TRANSACTION_DT,CUSTOMER_ID,SKU,UNITS,UNIT_PRI
CE,TRANSACTION_TOTAL) VALUES(") +
                    "TO_TIMESTAMP(" + "'" + this->getCustomerPurchaseTime() + "'" + "," + "'" + "YYYY-MM-DD HH24:MI:SS" + "'" + ")"
+ "," +
                     "TO_TIMESTAMP(" + "'" + this->getCustomerPurchaseTime() + "'" + "," + "'" + "YYYY-MM-DD HH24:MI:SS" + "'" +
")::DATE" + "," +
                    std::to_string(this->getCustomerId()) + "," +
                    "'" + sku + "'" + "," +
                    std::to_string(units) + "," +
                    std::to_string(unitPrice) + "," +
                    std::to_string(txnTotal)  +
                    ") ON CONFLICT DO NOTHING" + ";";

            // We will supply the buildSQL and execute within the transaction
            pqxx::result insertRes{txn.exec(buildSQL)};

            // display the SQL
            std::cout << "Query executed for Continuity Customer transactions: " << buildSQL << std::endl;
        }
        txn.commit();
    }
    catch(pqxx::sql_error const &e){
        std::cout << "SQL error: " << e.what() << std::endl;
        std::cerr << "SQL error: " << e.what() << std::endl;
        std::cout << "Query was:; " << e.query() << std::endl;
        std::cerr << "Query was:; " << e.query() << std::endl;
    }
}
```

NewArmy

This class pretty much inherits the continuity class. It introduces a new data member- custTotalWDiscount. This class overrides the following methods -

- setCustomerPurchaseTotals
  - Takes into account discounts offered for an item
- getCustomerTotals
  - Reflects new private data member - custTotalWDiscount
- insertCustomerTransactions
  - Will load new table – NEWARMY.CUSTOMER_TRANSACTIONS
  - This table reflects prices before and after discounts

AppleMonarchy

This class pretty much inherits the continuity class. It introduces new data members- custPoints and custTotalWDiscount.

This class introduces new methods -

- setCustomerPoints
  - Sets the custPoints private data member
- getCustomerPoints
  - Gets the custPoints private data member
- translatePointsToDiscounts
  - Translates points to transaction wide discounts

This class overrides the following methods -

- setCustomerPurchaseTotals
  - Takes into account points offered to a customer via translatePointsToDiscounts
- getCustomerTotals
  - Reflects new private data member - custTotalWDiscount
- insertCustomerTransactions
  - Will load new table – APPLEMONARCHY.CUSTOMER_TRANSACTIONS
  - This table reflects prices before and after discounts

Building

We are using g++ to compile all the classes and link with libpqxx and libpq.

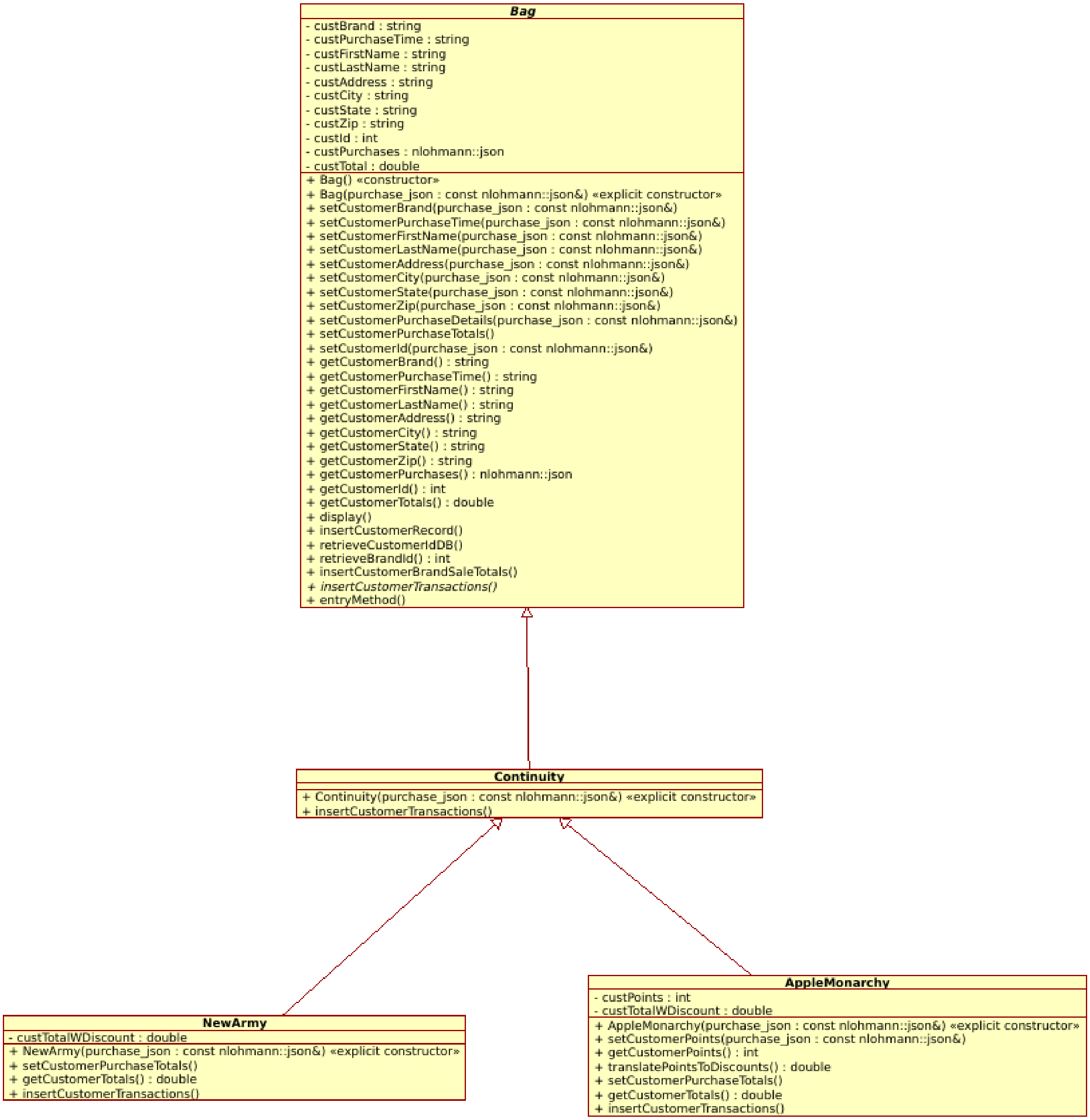
This is the command -

```
g++ --std=c++17 main.cpp Bag.h Bag.cpp Continuity.h Continuity.cpp NewArmy.h NewArmy.cpp AppleMonarchy.h AppleMonarchy.cpp -lpqxx -lpq -o runAcme
```

Running the executable -

```
./runAcme <path_to_json_file>
```

UML diagram


Bag
- custBrand : string
- custPurchaseTime : string
- custFirstName : string
- custLastName : string
- custAddress : string
- custCity : string
- custState : string
- custZip : string
- custId : int
- custPurchases : nlohmann::json
- custTotal : double
+ Bag() «constructor»
+ Bag(purchase_json : const nlohmann::json&) «explicit constructor»
+ setCustomerBrand(purchase_json : const nlohmann::json&)
+ setCustomerPurchaseTime(purchase_json : const nlohmann::json&)
+ setCustomerFirstName(purchase_json : const nlohmann::json&)
+ setCustomerLastName(purchase_json : const nlohmann::json&)
+ setCustomerAddress(purchase_json : const nlohmann::json&)
+ setCustomerCity(purchase_json : const nlohmann::json&)
+ setCustomerState(purchase_json : const nlohmann::json&)
+ setCustomerZip(purchase_json : const nlohmann::json&)
+ setCustomerPurchaseDetails(purchase_json : const nlohmann::json&)
+ setCustomerPurchaseTotals()
+ setCustomerId(purchase_json : const nlohmann::json&)
+ getCustomerBrand() : string
+ getCustomerPurchaseTime() : string
+ getCustomerFirstName() : string
+ getCustomerLastName() : string
+ getCustomerAddress() : string
+ getCustomerCity() : string
+ getCustomerState() : string
+ getCustomerZip() : string
+ getCustomerPurchases() : nlohmann::json
+ getCustomerId() : int
+ getCustomerTotals() : double
+ display()
+ insertCustomerRecord()
+ retrieveCustomerIdDB()
+ retrieveBrandId() : int
+ insertCustomerBrandSaleTotals()
+ insertCustomerTransactions()
+ entryMethod()
Continuity
+ Continuity(purchase_json : const nlohmann::json&) «explicit constructor»
+ insertCustomerTransactions()
NewArmy
- custTotalWDiscount : double
+ NewArmy(purchase_json : const nlohmann::json&) «explicit constructor»
+ setCustomerPurchaseTotals()
+ getCustomerTotals() : double
+ insertCustomerTransactions()
AppleMonarchy
- custPoints : int
- custTotalWDiscount : double
+ AppleMonarchy(purchase_json : const nlohmann::json&) «explicit constructor»
+ setCustomerPoints(purchase_json : const nlohmann::json&)
+ getCustomerPoints() : int
+ translatePointsToDiscounts() : double
+ setCustomerPurchaseTotals()
+ getCustomerTotals() : double
+ insertCustomerTransactions()

Next Steps for enhancements -

1. Implement tax calculations by address which uses tax lookups
2. Native date and timestamps handling
3. Threading
4. ORM vs. hand-coding SQL's
5. Implementing the class as a web-service

Appendix

PostgresSQL installs

```
# update local system packages
sudo apt update && sudo apt upgrade

# install necessary libs (these were already installed on mine)
sudo apt -y install gnupg2 wget vim

# the latest version in Ubuntu package directories are old - we need to be on Postgres14
# lets add the Postgres package repo so that our package directory can find the latest versions
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'

# download key from Postgres package repo, installation will check for the signing key to verify the program being installed is from Postgres
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -

# now that Postgres package repo and has been, lets update our local list
sudo apt -y update

# we are now ready to install Postgres14
sudo apt -y install postgresql-14

# the installation should kick off the systemd daemon which brings up the database service
# lets check if it did -
systemctl status postgresql

# You should see messages that PostgreSQL RDBMS is up and running

# PostgresSQL database runs as postgres system user
# You can login as postgres
sudo su - postgres

# Run the psql command
psql -V

# you should see something like this -
# psql (PostgreSQL) 14.5 (Ubuntu 14.5-1.pgdg20.04+1)

# lets back up the conf file as we are modifying
sudo cp /etc/postgresql/14/main/pg_hba.conf /etc/postgresql/14/main/pg_hba.conf.bkp

# lets modify peer identification to trust
sudo sed -i '/^local/s/peer/trust/' /etc/postgresql/14/main/pg_hba.conf

# allow password login
sudo sed -i '/^host/s/ident/md5/' /etc/postgresql/14/main/pg_hba.conf

# modify entries to allow access from everywhere
sudo vi /etc/postgresql/14/main/pg_hba.conf

###
## IPv4 local connections
# host all all 0.0.0.0/24 md5
#
## IPv6 local connections
# host all all 0.0.0.0/0 md5
###

# modify postgresql.conf to allow connections to listen to connections from everywhere
sudo vi /etc/postgresql/14/main/postgresql.conf

###
## CONNECTIONS AND AUTHENTICATION
#listen_addresses='*'
```

```
###

# restart the PostgreSQL service
sudo systemctl restart postgresql

# enable the daemon
sudo systemctl enable postgresql

# check if the service has restarted successfully
systemctl status postgresql

# lets change the password of postgres user
sudo passwd postgres

# this verifies your postgresql is ready to take connections
sudo pg_isready


## Install postgres library for C (libpq-dev)and C++ (libpqxx-dev)
sudo apt-get install libpq-dev libpqxx-dev
```

SQL IDE setup

```
## installing SQL ide

sudo apt update

sudo apt install ubuntu-make

umake --version

umake ide datagrip

# I have setup in the same paths as CLion (JetBrains)

chmod +x ~/.local/share/applications/jetbrains-datagrip.desktop
```

Database setup

```
### login to PostgreSQL

### Database schemas and user setup
CREATE ROLE admin WITH LOGIN SUPERUSER CREATEDB CREATEROLE PASSWORD 'ooPs!Hello';

### Let's create a database called acmedb
create database acmedb;

### Let's create an acme_user
create user acme_user with encrypted password 'AcM#2022';

### Let's grant full access to acme_user on acmedb
grant all privileges on database acmedb to acme_user;

### Create schemas
create schema newarmy authorization acme_user;
create schema applemonarchy authorization acme_user;
create schema continuity authorization acme_user;



#########################
```

```
#  Psql commands -
#  \l - Display database
#  \c - Connect to database
#  \dn - List schemas
#  \dt - List tables inside public schemas
#  \dt schema1. - List tables inside particular schemas. For eg: 'schema1'.
#
########################

## Run this SQL’s to create database objects

--- Brand lookup table
CREATE TABLE ACME.BRANDS
(
    id          INTEGER NOT NULL,
    description VARCHAR NOT NULL,
    CONSTRAINT PK_BRAND_ID PRIMARY KEY (id)
);

INSERT INTO ACME.BRANDS
VALUES(1,'CONTINUITY');
INSERT INTO ACME.BRANDS
VALUES(2,'NEW_ARMY');
INSERT INTO ACME.BRANDS
VALUES(3,'APPLE_MONARCHY');

--- Sequence to create customer ids
CREATE SEQUENCE ACME.CUSTOMER_ID_SEQ;
--- Customer table in ACME schema
CREATE TABLE ACME.CUSTOMER(
    id          INTEGER NOT NULL DEFAULT NEXTVAL('ACME.CUSTOMER_ID_SEQ'),
    first_name  VARCHAR NOT NULL,
    last_name   VARCHAR NOT NULL,
    address     VARCHAR NOT NULL,
    city        VARCHAR NOT NULL,
    state       VARCHAR NOT NULL,
    zip         VARCHAR NOT NULL,
    na_cust_flg BOOL DEFAULT FALSE,
    am_cust_flg BOOL DEFAULT FALSE,
    cn_cust_flg BOOL DEFAULT FALSE,
    CONSTRAINT PK_CUSTOMER_ID PRIMARY KEY (first_name, last_name, address, city, state, zip)
);

--- Sequence to create transaction ids
CREATE SEQUENCE ACME.TXN_ID_SEQ;
--- Table to host customer totals across brands and purchase times
--- it references Customer id, their purchase timestamps and the total across their purchases for a brand
CREATE TABLE ACME.CUSTOMER_BRAND_SALES
(
    sale_id         INTEGER NOT NULL DEFAULT NEXTVAL('ACME.TXN_ID_SEQ'),
    purchase_ts     TIMESTAMP NOT NULL,
    purchase_dt     DATE NOT NULL,
    customer_id     INTEGER NOT NULL,
    brand_id        INTEGER NOT NULL,
    sales_total          DECIMAL(18, 4) NOT NULL,
    CONSTRAINT FK_BRAND FOREIGN KEY(brand_id) REFERENCES ACME.BRANDS(id),
    CONSTRAINT PK_SALES_ID PRIMARY KEY (purchase_ts,purchase_dt,customer_id,brand_id)
) PARTITION BY RANGE (purchase_dt);
--- Create partitions
CREATE TABLE ACME.CUSTOMER_SALES_2021_Q1
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2021-01-01') TO ('2021-04-01');
CREATE TABLE ACME.CUSTOMER_SALES_2021_Q2
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2021-04-01') TO ('2021-07-01');
CREATE TABLE ACME.CUSTOMER_SALES_2021_Q3
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2021-07-01') TO ('2021-10-01');
```

```
CREATE TABLE ACME.CUSTOMER_SALES_2021_Q4
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2021-10-01') TO ('2022-01-01');
CREATE TABLE ACME.CUSTOMER_SALES_2022_Q1
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2022-01-01') TO ('2022-04-01');
CREATE TABLE ACME.CUSTOMER_SALES_2022_Q2
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2022-04-01') TO ('2022-07-01');
CREATE TABLE ACME.CUSTOMER_SALES_2022_Q3
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2022-07-01') TO ('2022-10-01');
CREATE TABLE ACME.CUSTOMER_SALES_2022_Q4
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2022-10-01') TO ('2023-01-01');
CREATE TABLE ACME.CUSTOMER_SALES_2023_Q1
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2023-01-01') TO ('2023-04-01');
CREATE TABLE ACME.CUSTOMER_SALES_2023_Q2
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2023-04-01') TO ('2023-07-01');
CREATE TABLE ACME.CUSTOMER_SALES_2023_Q3
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2023-07-01') TO ('2023-10-01');
CREATE TABLE ACME.CUSTOMER_SALES_2023_Q4
    PARTITION OF ACME.CUSTOMER_BRAND_SALES FOR VALUES FROM ('2023-10-01') TO ('2024-01-01');

--- insert some "EXISTING" customers
INSERT INTO ACME.CUSTOMER(
    FIRST_NAME,
    LAST_NAME,
    ADDRESS,
    CITY,
    STATE,
    ZIP,
    NA_CUST_FLG,
    AM_CUST_FLG,
    CN_CUST_FLG
)
VALUES(
    'Michael',
    'Hopkins',
    '123 Balboa St',
    'La Jolla',
    'CA',
    '98012',
    '0',
    '1',
    '0'
);

INSERT INTO ACME.CUSTOMER(
    FIRST_NAME,
    LAST_NAME,
    ADDRESS,
    CITY,
    STATE,
    ZIP,
    NA_CUST_FLG,
    AM_CUST_FLG,
    CN_CUST_FLG
)
VALUES(
    'Thomas',
    'George',
    '1890 Main St',
    'Boise',
    'ID',
    '83701',
    '1',
    '0',
    '0'
);
```

```
INSERT INTO ACME.CUSTOMER(
    FIRST_NAME,
    LAST_NAME,
    ADDRESS,
    CITY,
    STATE,
    ZIP,
    NA_CUST_FLG,
    AM_CUST_FLG,
    CN_CUST_FLG
)
VALUES(
    'Felix',
    'Mathew',
    '2908 Garfield Lane',
    'Miami',
    'FL',
    '33101',
    '0',
    '0',
    '1'
);

--- Sequence to create transaction ids
CREATE SEQUENCE CONTINUITY.TXN_ID_SEQ;
--- Table to host customer transactions for Continuity brands and purchase times
--- it references Customer id, their purchase timestamps and their purchases for a brand
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS
(
    transaction_id    INTEGER NOT NULL DEFAULT NEXTVAL('CONTINUITY.TXN_ID_SEQ'),
    transaction_ts    TIMESTAMP NOT NULL,
    transaction_dt    DATE NOT NULL,
    customer_id       INTEGER NOT NULL,
    sku               VARCHAR,
    units             INTEGER,
    unit_price        DECIMAL(10,4),
    transaction_total DECIMAL(18, 4) NOT NULL,
    CONSTRAINT PK_TXN PRIMARY KEY (transaction_ts,transaction_dt,customer_id,sku)
) PARTITION BY RANGE (transaction_dt);
--- Create partitions
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2021_Q1
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-01-01') TO ('2021-04-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2021_Q2
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-04-01') TO ('2021-07-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2021_Q3
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-07-01') TO ('2021-10-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2021_Q4
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-10-01') TO ('2022-01-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2022_Q1
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-01-01') TO ('2022-04-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_022_Q2
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-04-01') TO ('2022-07-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2022_Q3
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-07-01') TO ('2022-10-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2022_Q4
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-10-01') TO ('2023-01-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2023_Q1
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-01-01') TO ('2023-04-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2023_Q2
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-04-01') TO ('2023-07-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2023_Q3
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-07-01') TO ('2023-10-01');
CREATE TABLE CONTINUITY.CUSTOMER_TRANSACTIONS_2023_Q4
    PARTITION OF CONTINUITY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-10-01') TO ('2024-01-01');
```

```
--- Sequence to create transaction ids
CREATE SEQUENCE NEWARMY.TXN_ID_SEQ;
--- Table to host customer transactions for New Army brands and purchase times
--- it references Customer id, their purchase timestamps and their purchases for a brand
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS
(
    transaction_id    INTEGER NOT NULL DEFAULT NEXTVAL('NEWARMY.TXN_ID_SEQ'),
    transaction_ts    TIMESTAMP NOT NULL,
    transaction_dt    DATE NOT NULL,
    customer_id       INTEGER NOT NULL,
    sku               VARCHAR,
    units             INTEGER,
    unit_price        DECIMAL(10,4),
    transaction_total DECIMAL(18,4) NOT NULL,
    original_total    DECIMAL(18,4) NOT NULL,
    offers            DECIMAL(10,4),
    discounts         DECIMAL(18,4),
    CONSTRAINT PK_TXN PRIMARY KEY (transaction_ts,transaction_dt,customer_id,sku)
) PARTITION BY RANGE (transaction_dt);
--- Create partitions
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2021_Q1
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-01-01') TO ('2021-04-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2021_Q2
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-04-01') TO ('2021-07-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2021_Q3
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-07-01') TO ('2021-10-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2021_Q4
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-10-01') TO ('2022-01-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2022_Q1
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-01-01') TO ('2022-04-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_022_Q2
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-04-01') TO ('2022-07-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2022_Q3
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-07-01') TO ('2022-10-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2022_Q4
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-10-01') TO ('2023-01-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2023_Q1
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-01-01') TO ('2023-04-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2023_Q2
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-04-01') TO ('2023-07-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2023_Q3
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-07-01') TO ('2023-10-01');
CREATE TABLE NEWARMY.CUSTOMER_TRANSACTIONS_2023_Q4
    PARTITION OF NEWARMY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-10-01') TO ('2024-01-01');


--- Sequence to create transaction ids
CREATE SEQUENCE APPLEMONARCHY.TXN_ID_SEQ;
--- Table to host customer transactions for AppleMonarchy brands and purchase times
--- it references Customer id, their purchase timestamps and their purchases for a brand
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS
(
    transaction_id    INTEGER NOT NULL DEFAULT NEXTVAL('APPLEMONARCHY.TXN_ID_SEQ'),
    transaction_ts    TIMESTAMP NOT NULL,
    transaction_dt    DATE NOT NULL,
    customer_id       INTEGER NOT NULL,
    sku               VARCHAR,
    units             INTEGER,
    unit_price        DECIMAL(10,4),
    transaction_total DECIMAL(18,4) NOT NULL,
    original_total    DECIMAL(18,4) NOT NULL,
    offers            DECIMAL(10,4),
    discounts         DECIMAL(18,4),
    points            INTEGER,
    CONSTRAINT PK_TXN PRIMARY KEY (transaction_ts,transaction_dt,customer_id,sku)
) PARTITION BY RANGE (transaction_dt);
--- Create partitions
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2021_Q1
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-01-01') TO ('2021-04-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2021_Q2
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-04-01') TO ('2021-07-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2021_Q3
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-07-01') TO ('2021-10-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2021_Q4
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2021-10-01') TO ('2022-01-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2022_Q1
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-01-01') TO ('2022-04-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_022_Q2
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-04-01') TO ('2022-07-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2022_Q3
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-07-01') TO ('2022-10-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2022_Q4
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2022-10-01') TO ('2023-01-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2023_Q1
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-01-01') TO ('2023-04-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2023_Q2
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-04-01') TO ('2023-07-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2023_Q3
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-07-01') TO ('2023-10-01');
CREATE TABLE APPLEMONARCHY.CUSTOMER_TRANSACTIONS_2023_Q4
    PARTITION OF APPLEMONARCHY.CUSTOMER_TRANSACTIONS FOR VALUES FROM ('2023-10-01') TO ('2024-01-01');
```