TSOHA 2024_09_08 / Thy
The final subset of user stories to be implemented for the excercise, which will be delivered in october-24 should be selected from this document next during phase.

MusicStore-system, User stories, MSus_v1

These user stories outline the key functionalities and acceptance criteria for registering and operating stores, store managers, customers, product groups and products in the MusicStore database.

The core concept of MusicStore system is the ShoppingCart-entity.

Story 1: Customer Registration

As a new user, I want to register as a customer,
so that I can create an account and start shopping.

Acceptance Criteria:

The registration form should include fields for custoName, custoPassw, custoEmail, custoPhone, and custoStatus.
The system should validate that custoName, custoPassw are unique.
- The system should save custoPssw field in a hashed format.

The system should allow optional fields for custoEmail and custoPhone.
The system should set custoStatus to FALSE by default.
Upon successful registration, custoStatus is set to TRUE the customer should receive a confirmation message.
The system should set custoBlocked to FALSE by default.

User Story 2: Store Manager Registration

As a store owner, I want to register as a store manager,
so that I can manage my store’s information and customer service operations.

Acceptance Criteria:
The registration form should include fields for storeManagerName, storeManagerPssw, storeName, storeTaxId, storePhone, storeEmail, storeAddress, and storeLogoUrl.
The system should validate that storeName, storeManagerName, storeManagerPssw, storeTaxId, storePhone, storeEmail and storeAddress are entered.
- The system should save storeManagerPssw field in a hashed format.

The system should allow optional field for storeLogoUrl.
The system should link the store manager to a Customer record (for making purchases as a customer) using storeManager_id.
Upon successful registration, the store manager should receive a confirmation message.

Here is an **example** of how a password can be hashed by using the PostgreSQL’s crypt function:

INSERT INTO Customer (custoName, custoPassw, custoEmail, custoPhone, custoStatus,

```
custoBlocked)
VALUES (
        'Matti Koskela',
        crypt('mypassword', gen_salt('bf')),
        'matti.koskela@outlook.com',
        '040545321',
        TRUE,
        FALSE
);
```

These user stories outline the key functionalities and acceptance criteria for logging in and out for both customers and store managers in the MusicStore.

User Story 3: Customer Login

As a registered customer, I want to log in to my account,
so that I can access my shopping cart, view my order history,
and manage my account details.

Acceptance Criteria:

The login form should include fields for custoName and custoPassw.
The system should validate the credentials against the stored hashed password.
If the credentials are correct, the customer should be redirected to their account dashboard.
If the credentials are incorrect, an error message should be displayed.
The system should update the last login timestamp upon successful login.

User Story 4: Customer Logout

As a logged-in customer, I want to log out of my account,
so that I can ensure my account is secure when I am not using it.

Acceptance Criteria:

The customer should be able to log out by clicking a “Logout” button.
Upon logging out, the customer should be redirected to the homepage.
The system should invalidate the current session to prevent unauthorized access.

User Story 5: Store Manager Login

As a registered store manager, I want to log in to my store management account, so that I can manage store information, inventory, and customer orders.

Acceptance Criteria:

The login form should include fields for storeManagerName and storeManagerPssw.
The system should validate the credentials against the stored hashed password.
If the credentials are correct, the store manager should be redirected to the store management dashboard.
If the credentials are incorrect, an error message should be displayed.

The system should update the last login timestamp upon successful login.

User Story 6: Store Manager Logout

As a logged-in store manager,
I want to log out of my store management account,
so that I can ensure the store’s information is secure when I am not managing it.

Acceptance Criteria:

The store manager should be able to log out by clicking a “Logout” button.
Upon logging out, the store manager should be redirected to the homepage.
The system should invalidate the current session to prevent unauthorized access.

User Story 7: Customers' shopping cart view on the Customer dashboard

As a registered customer,
I want to list my ShoppingCart headers as a table,
so that I can view my ShoppingCarts categorized by their status and sorted by the most recent dates.

Acceptance Criteria:

The table should display ShoppingCart headers categorized by cartStatus (open, closed, delivered).
Each category should list ShoppingCarts sorted by the most recent dates at the top.
The table should include columns for cartEditedTime, cartPurchasedTime, and cartDeliveryTime.
Clicking on a cart_id in the cartHeader should display the details of the selected ShoppingCart.
The user should be able to close the details view.
This table should be part of the Customer dashboard.

SQL View for the Customer ShoppingCart view on the Customer Dashboard

```
CREATE VIEW CustomerShoppingCartHeaders AS
SELECT
    sc.id AS cart_id,
    sc.cartCustomer_id AS customer_id,
    sc.cartStatus AS cart_status,
    sc.cartEditedTime AS edited_time,
    sc.cartPurchasedTime AS purchased_time,
    sc.cartDeliveryTime AS delivery_time,
    sc.cartTotal AS total_amount
FROM
    ShoppingCart sc
WHERE
    sc.cartCustomer_id = customer_id, -- the actual user ID
ORDER BY
    sc.cartStatus,
    CASE
        WHEN sc.cartStatus = FALSE THEN sc.cartEditedTime
```

```
        WHEN sc.cartStatus = TRUE AND sc.cartDeliveryTime IS NULL THEN
    sc.cartPurchasedTime
        WHEN sc.cartStatus = TRUE AND sc.cartDeliveryTime IS NOT NULL THEN
    sc.cartDeliveryTime
  END DESC;
```

Following essential sql-statements and clause are used:
SELECT Statement:
Selects relevant columns from the ShoppingCart table, including cart_id, customer_id, cart_status, current_time, edited_time, purchased_time, delivery_time, and total_amount.

WHERE Clause:
Filters the results to include only the ShoppingCarts of the current user (replace CURRENT_USER_ID with the actual user ID).

ORDER BY Clause:
Orders the results by cartStatus and the respective timestamps in descending order to ensure the most recent dates are at the top.
This view will provide a comprehensive list of ShoppingCart headers for the customer, categorized by their status and sorted by the most recent dates.

StoreManager Dashboard User Stories

User Story 8: StoreManager Dashboard Overview

As a store manager,
I want a dashboard that provides an overview of my store’ s performance,
so that I can quickly access key metrics and reports.

Acceptance Criteria:
The dashboard should include links to reports on monthly and yearly sales, profitability, campaign sales, and customer purchase behavior.
The dashboard should provide basic information listings about registered customers and product assortment with all product attributes per product group and product amounts in numbers in the store.
The dashboard should include basic listings of customers, product groups, and MusicStore attributes.
The dashboard should display key performance indicators (KPIs) such as total sales, total profit, and total discounts.

User Story 9: Access Sales Reports

As a store manager,
I want to access detailed sales reports,
so that I can analyze the store’ s performance and make informed decisions.

Acceptance Criteria:
The dashboard should include links to the following sales reports:
Customers’ shopping cart report

Total (yearly up to date and monthly) sales and profit margin per product group report
Campaign sales and profit margin report
Best selling products report
Best buying customers report

User Story 10: View Customer Information

As a store manager,
I want to view a list of registered customers,
so that I can manage customer relationships and loyalty programs.

Acceptance Criteria:
The dashboard should include a link to a detailed list of registered customers with all their attributes.

User Story 11: View Product Assortment

As a store manager,
I want to view a list of products with all their attributes,
so that I can manage inventory and product offerings.

Acceptance Criteria:
The dashboard should include a link to a detailed list of products grouped by product group, including all product attributes and amounts in numbers in the store.

User Story 12: View Basic Listings

As a store manager,
I want to view basic listings of customers, product groups, products and MusicStore attributes, so that I can have a quick reference to essential information. These are described more in details in the last part of this document.

Acceptance Criteria:
The dashboard should include links to basic listings and maintenance of customers, product groups, products and MusicStore attributes.

User Story 13: Customer Shopping Experience

As a registered customer,
I want to buy products from the MusicStore using a virtual shopping cart-entity,
so that I can manage my purchases easily and efficiently.

Acceptance Criteria:
Product Search and Selection:
The customer should be able to search for products by name, category, or other attributes like product code.
The customer should be able to view product details, including price, discount, and VAT percentage.
The customer should be able to add products to the shopping cart.

Shopping Cart Maintenance:

The customer should be able to view the contents of the shopping cart.
The customer should be able to add, update, or delete products in the shopping cart.
The customer should be able to increase or decrease the quantity of items per row.
The shopping cart should display the total amount of discounts, VAT, and the overall purchase sum.

Saving and Purchasing:
The customer should be able to save the shopping cart temporarily, keeping the status as “open”.
The customer should be able to proceed to purchase, changing the cart status to “purchased” or “closed” if does not want to purchase.
Upon delivery, the cart status should change to “delivered”.

Shopping Cart Header:
The shopping cart header should include customer_id, Ssore_id, receipt number (shoppingCart_id), and the timestamp of the purchase or save time if not closed.

SQL View Script for Customer Shopping Activities.

This view will provide an overview of the customer’s shopping cart activities, supporting the functionalities described above.

```
CREATE VIEW CustomerShoppingCartView AS
SELECT
    sc.id AS shoppingCart_id,
    sc.cartCustomer_id AS customer_id,
    sc.cartStore_id AS store_id,
    sc.cartStatus AS cart_status,
    sc.cartCurrentTime AS current_time,
    sc.cartEditedTime AS edited_time,
    sc.cartPurchasedTime AS purchased_time,
    sc.cartDeliveryTime AS delivery_time,
    sc.cartTotal AS total_amount,
    sc.cartDiscount AS total_discount,
    sc.cartVat AS total_vat,
    p.id AS product_id,
    p.productName AS product_name,
    p.productSalesPrice AS product_price,
    p.productDiscount AS product_discount,
    p.productVat AS product_vat,
    scp.quantity AS product_quantity,
    (p.productSalesPrice * scp.quantity) AS product_total_price,
    (p.productDiscount * scp.quantity) AS product_total_discount,
    (p.productVat * scp.quantity) AS product_total_vat
FROM
    ShoppingCart sc
JOIN
    ShoppingCartProduct scp ON sc.id = scp.shoppingCart_id
JOIN
    Product p ON scp.product_id = p.id
```

```
ORDER BY
    sc.cartStatus, sc.cartCurrentTime DESC;
```

Explanation:
JOIN Statements:
The ShoppingCart table is joined with the ShoppingCartProduct table to get the products in the cart. The Product table is joined to get the product details.

SELECT Statement:
Selects relevant columns from the ShoppingCart, ShoppingCartProduct, and Product tables, including product details and calculated totals for price, discount, and VAT.

ORDER BY Clause:
Orders the results by cartStatus and cartCurrentTime in descending order to show the most recent carts first.

User Story 14: Update Cumulative Sums and Inventory Control

As a StoreManager,
I want customers' shopping cart state to update daily and monthly cumulative sums and inventory when they make a purchase,
so that the system accurately reflects sales, discounts, VAT, and inventory levels.

Acceptance Criteria:
State Change to “Purchased”:
When the customer changes the shopping cart state to “purchased”, the system should update the daily and monthly
cumulative sums for each product, including discounts and VAT.
The system should decrease the amount of products in store based on the quantity sold.

Daily and Monthly Cumulative Sums:
The system should update the daily sales, discounts, and VAT for each product.
The system should update the monthly sales, discounts, and VAT for each product.

Inventory Control:
The system should decrease the productsInStore field in the Product table based on the quantity sold.

This script ensures that the system accurately updates the cumulative sums and inventory levels
when a customer’s shopping cart state is changed to “purchased”:

```
-- Update daily and monthly cumulative sums and inventory control
WITH UpdatedSales AS (
    SELECT
        sc.id AS shoppingCart_id,
        sc.cartCustomer_id AS customer_id,
        sc.cartStore_id AS store_id,
        sc.cartTotal AS total_amount,
```

```
            sc.cartDiscount AS total_discount,
            sc.cartVat AS total_vat,
            scp.product_id,
            scp.quantity AS product_quantity,
            p.productSalesPrice,
            p.productDiscount,
            p.productVat
        FROM
            ShoppingCart sc
        JOIN
            ShoppingCartProduct scp ON sc.id = scp.shoppingCart_id
        JOIN
            Product p ON scp.product_id = p.id
        WHERE
            sc.cartStatus = 'purchased'
)
-- Update daily cumulative sums
UPDATE Product
SET
        dailySales = dailySales + us.total_amount,
        dailyDiscounts = dailyDiscounts + us.total_discount,
        dailyTaxes = dailyTaxes + us.total_vat,
        productsInStore = productsInStore - us.product_quantity
FROM
        UpdatedSales us
WHERE
        Product.id = us.product_id;

-- Update monthly cumulative sums
UPDATE Product
SET
        monthlySales = monthlySales + us.total_amount,
        monthlyDiscounts = monthlyDiscounts + us.total_discount,
        monthlyTaxes = monthlyTaxes + us.total_vat
FROM
        UpdatedSales us
WHERE
        Product.id = us.product_id;

-- Update ProductGroup cumulative monthly saldo fields
UPDATE ProductGroup
SET
        monthlySales_01 = monthlySales_01 + COALESCE(us.total_amount, 0),
        monthlyDiscounts_01 = monthlyDiscounts_01 + COALESCE(us.total_discount, 0),
        monthlyTaxes_01 = monthlyTaxes_01 + COALESCE(us.total_vat, 0)
FROM
        UpdatedSales us
JOIN
        Product p ON us.product_id = p.id
WHERE
```

```
    ProductGroup.id = p.productGroup_id
    AND EXTRACT(MONTH FROM CURRENT_DATE) = 1;

-- Repeat the above block for all 12 months
```

Explanation:
Common Table Expression (CTE):
UpdatedSales gathers the necessary data from the ShoppingCart, ShoppingCartProduct, and Product tables for carts with the status “purchased”.

Update Daily Cumulative Sums:
Updates the dailySales, dailyDiscounts, and dailyTaxes fields in the Product table.
Decreases the productsInStore field based on the quantity sold.

Update Monthly Cumulative Sums:
Updates the monthlySales, monthlyDiscounts, and monthlyTaxes fields in the Product table.

Update ProductGroup Cumulative Monthly Sums:
Updates the monthlySales, monthlyDiscounts, and monthlyTaxes fields for the current month in the ProductGroup table.
This block should be repeated for all 12 months to cover the entire year.

SQL-view scripts for StoreManager Dashboard reports

Here are the SQL-view scripts for the various Dashboard reports:

Customers’ Shopping Cart view for the Store Manager

```
CREATE VIEW CustomerShoppingCartReport AS
SELECT
    sc.id AS cart_id,
    sc.cartCustomer_id AS customer_id,
    sc.cartStatus AS cart_status,
    sc.cartCurrentTime AS current_time,
    sc.cartEditedTime AS edited_time,
    sc.cartPurchasedTime AS purchased_time,
    sc.cartDeliveryTime AS delivery_time,
    sc.cartTotal AS total_amount
FROM
    ShoppingCart sc
ORDER BY
    sc.cartCustomer_id, sc.cartStatus, sc.cartCurrentTime DESC;
```

Total Sales and Profit Margin per Product and Product Group Report

Here is a SQL view script to report sales per day, month, and current year up to date, using the cumulative fields of total sales, discounts, and VAT in the Products and ProductGroup tables.

```
CREATE VIEW SalesReport AS
WITH DailySales AS (
    SELECT
        p.id AS product_id,
        p.productName AS product_name,
        p.dailySales AS daily_sales,
        p.dailyDiscounts AS daily_discounts,
        p.dailyTaxes AS daily_vat,
        CURRENT_DATE AS report_date
    FROM
        Product p
),
MonthlySales AS (
    SELECT
        p.id AS product_id,
        p.productName AS product_name,
        p.monthlySales AS monthly_sales,
        p.monthlyDiscounts AS monthly_discounts,
        p.monthlyTaxes AS monthly_vat,
        DATE_TRUNC('month', CURRENT_DATE) AS report_month
    FROM
        Product p
),
YearlySales AS (
    SELECT
        p.id AS product_id,
        p.productName AS product_name,
        SUM(p.monthlySales) AS yearly_sales,
        SUM(p.monthlyDiscounts) AS yearly_discounts,
        SUM(p.monthlyTaxes) AS yearly_vat,
        DATE_TRUNC('year', CURRENT_DATE) AS report_year
    FROM
        Product p
    GROUP BY
        p.id, p.productName
)
SELECT
    ds.product_id,
    ds.product_name,
    ds.daily_sales,
    ds.daily_discounts,
    ds.daily_vat,
    ms.monthly_sales,
    ms.monthly_discounts,
    ms.monthly_vat,
    ys.yearly_sales,
    ys.yearly_discounts,
    ys.yearly_vat,
    ds.report_date,
    ms.report_month,
```

```
    ys.report_year
FROM
    DailySales ds
JOIN
    MonthlySales ms ON ds.product_id = ms.product_id
JOIN
    YearlySales ys ON ds.product_id = ys.product_id
ORDER BY
    ds.product_name;
```

Explanation:
Common Table Expressions (CTEs):
DailySales: Selects daily cumulative sales, discounts, and VAT for each product.
MonthlySales: Selects monthly cumulative sales, discounts, and VAT for each product.
YearlySales: Aggregates monthly cumulative sales, discounts, and VAT to get yearly totals for each product.

SELECT Statement:
Combines the results from the CTEs to provide a comprehensive report of daily, monthly, and yearly sales, discounts, and VAT for each product.

JOIN Statements:
Joins the CTEs on product_id to combine the daily, monthly, and yearly data.

ORDER BY Clause:
Orders the results by product_name for easy readability.

UI Layout Proposal for StoreManager Dashboard

This layout ensures that the StoreManager has
quick access to essential reports and information.

StoreManager Dashboard Layout

```
+-------------------------------------------------------------+
|                                  StoreManager Dashboard
|
+-------------------------------------------------------------+
| Key Performance Indicators (KPIs)
|
| - Total Sales:  €  XX,XXX
|
| - Total Profit:  €  X,XXX
|
| - Total Discounts:  €  X,XXX
|
+-------------------------------------------------------------+
| Reports
|
```

| - [Customers' Shopping Cart Report]
|
| - [Total Sales and Profit Margin per Product Group Report] |
| - [Campaign Sales and Profit Margin Report] |
| - [Best Selling Products Report]
|
| - [Best Buying Customers Report]
|
+---------- ---+
| Basic Information Listings
|
| - [Registered Customers]
|
| - [Product Assortment]
|
| - [Product Groups]
|
| - [MusicStore Attributes]
|
+----------+

Explanation:
Key Performance Indicators (KPIs):
Displays essential metrics such as total sales, total profit, and total discounts.

Reports Section:
Provides links to detailed sales and customer behavior reports.

Basic Information Listings:
Includes links to lists of registered customers, product assortment, product groups, and MusicStore attributes.

Customers' shopping cart view

Here is a SQL view script that lists **Customer’s ShoppingCarts**, categorized as open, closed, and delivered carts, in descending order based on different timestamps in the ShoppingCart table:

Following essential sql-statements and clause are used:

Join: The Customer table is joined with the ShoppingCart table on the customer_id.

CASE Statement: Categorizes the carts into ‘Open’, ‘Closed’, and ‘Delivered’ based on the cartStatus and cartDeliveryTime.

Order By: Orders the results by cart_category and then by the timestamps (cartEditedTime, cartPurchasedTime, cartDeliveryTime) in descending order.

CREATE VIEW CustomerShoppingCarts AS

```
SELECT
	c.id AS customer_id,
	c.custoName AS customer_name,
	sc.id AS cart_id,
	sc.cartStatus AS cart_status,
	sc.cartEditedTime AS edited_time,
	sc.cartPurchasedTime AS purchased_time,
	sc.cartDeliveryTime AS delivery_time,
	CASE
		WHEN sc.cartStatus = FALSE THEN 'Open'
		WHEN sc.cartStatus = TRUE AND sc.cartDeliveryTime IS NULL THEN 'Closed'
		WHEN sc.cartStatus = TRUE AND sc.cartDeliveryTime IS NOT NULL THEN
'Delivered'
	END AS cart_category
FROM
	Customer c
JOIN
	ShoppingCart sc ON c.id = sc.cartCustomer_id
ORDER BY
	cart_category DESC,
	sc.cartEditedTime DESC,
	sc.cartPurchasedTime DESC,
	sc.cartDeliveryTime DESC;
```

Total sales and profit marging calculated from shopping carts
(Shopping cart analyze)

Here is a SQL view script that creates a sql-view to list the **total sum of sales, discounts, profit and VAT** of the MusicStore customers' shopping carts, categorized per ProductGroup and by the status of the shopping carts (open, closed, and delivered), ordered alphabetically by ProductGroup names:

Following essential sql-statements and clause are used:

JOIN Statements:
The ShoppingCart table is joined with the Product table using the product IDs in the shopping cart.
The Product table is joined with the ProductGroup table using the productGroup_id.

CASE Statement:
Categorizes the carts into 'Open', 'Closed', and 'Delivered' based on the cartStatus and cartDeliveryTime.

SUM Function:
Calculates the total sales, discounts, and VAT for each product group and cart status.

GROUP BY:
Groups the results by ProductGroup name and cart status.

ORDER BY:
Orders the results alphabetically by ProductGroup names.

CREATE VIEW ProductGroupSales AS

```
SELECT
    pg.prodGroupName AS product_group,
    CASE
        WHEN sc.cartStatus = FALSE THEN 'Open'
        WHEN sc.cartStatus = TRUE AND sc.cartDeliveryTime IS NULL THEN 'Closed'
        WHEN sc.cartStatus = TRUE AND sc.cartDeliveryTime IS NOT NULL THEN
'Delivered'
    END AS cart_status,
    SUM(sc.cartTotal) AS total_sales,
    SUM(sc.cartDiscount) AS total_discounts,
    SUM(sc.cartVat) AS total_vat
FROM
    ShoppingCart sc
JOIN
    Product p ON p.id IN (sc.cartProd_1_id, sc.cartProd_2_id, sc.cartProd_3_id,
sc.cartProd_4_id, sc.cartProd_5_id, sc.cartProd_6_id, sc.cartProd_7_id, sc.cartProd_8_id,
sc.cartProd_9_id, sc.cartProd_10_id, sc.cartProd_11_id, sc.cartProd_12_id)
JOIN
    ProductGroup pg ON p.productGroup_id = pg.id
GROUP BY
    pg.prodGroupName, cart_status
ORDER BY
    pg.prodGroupName ASC;
```

Campaign sales view based on shopping carts

Here is a SQL view script that **lists all campaign sales products,** calculates their sales amount in euros, pieces sold, profit margin percentage, and discount percentage compared to the normal sales price. The list includes basic product information and is ordered in descending order of the amount sold in euros.

CREATE VIEW CampaignSales AS

```
SELECT
    p.id AS product_id,
    p.productName AS product_name,
    p.productDetails AS product_details,
    p.productSalesPrice AS normal_price,
    p.productCampaignPrice AS campaign_price,
    SUM(sc.cartTotal) AS total_sales_euros,
```

```
    COUNT(p.id) AS pieces_sold,
    ((p.productCampaignPrice - p.producPurchPrice) / p.productCampaignPrice) * 100
AS profit_margin_percent,
    ((p.productSalesPrice - p.productCampaignPrice) / p.productSalesPrice) * 100 AS
discount_percent
FROM
    ShoppingCart sc
JOIN
    Product p ON p.id IN (sc.cartProd_1_id, sc.cartProd_2_id, sc.cartProd_3_id,
sc.cartProd_4_id, sc.cartProd_5_id, sc.cartProd_6_id, sc.cartProd_7_id, sc.cartProd_8_id,
sc.cartProd_9_id, sc.cartProd_10_id, sc.cartProd_11_id, sc.cartProd_12_id)
WHERE
    p.productCampaignFlag = TRUE
GROUP BY
    p.id, p.productName, p.productDetails, p.productSalesPrice, p.productCampaignPrice,
p.producPurchPrice
ORDER BY
    total_sales_euros DESC;
```

Following essential sql-statements and clause are used:
JOIN Statements:
The ShoppingCart table is joined with the Product table using the product IDs in the shopping cart.

WHERE Clause:
Filters the products to include only those that are part of a campaign (productCampaignFlag = TRUE).

SUM and COUNT Functions:
Calculates the total sales amount in euros (SUM(sc.cartTotal)) and the number of pieces sold (COUNT(p.id)).

Profit Margin and Discount Percent Calculations:
Calculates the profit margin percentage as ((campaign_price - purchase_price) / campaign_price) * 100.
Calculates the discount percentage as ((normal_price - campaign_price) / normal_price) * 100.

GROUP BY:
Groups the results by product details to aggregate the sales data.

ORDER BY:
Orders the results in descending order of the total sales amount in euros.

Best selling Products view (shopping cart analyze)

Here is a SQL view script that **lists all products with their attributes,** including basic information and pricing details, along with campaign information. The view is grouped by ProductGroup and sorted by the best-selling products within each ProductGroup, which

are in alphabetical order.

CREATE VIEW ProductDetails AS

```
SELECT
    pg.prodGroupName AS product_group,
    p.id AS product_id,
    p.productName AS product_name,
    p.productDetails AS product_details,
    p.productSalesPrice AS normal_price,
    p.productCampaignPrice AS campaign_price,
    p.productDiscount AS discount,
    p.productCampaignFlag AS campaign_flag,
    p.productCampaignStart AS campaign_start,
    p.productCampaignEnd AS campaign_end,
    p.productsInStore AS in_store,
    p.productsReserved AS reserved,
    p.producPurchPrice AS purchase_price,
    p.productCreated AS created_date,
    p.productEdited AS edited_date,
    SUM(sc.cartTotal) AS total_sales_euros,
    COUNT(p.id) AS pieces_sold
FROM
    Product p
JOIN
    ProductGroup pg ON p.productGroup_id = pg.id
LEFT JOIN
    ShoppingCart sc ON p.id IN (sc.cartProd_1_id, sc.cartProd_2_id, sc.cartProd_3_id,
sc.cartProd_4_id, sc.cartProd_5_id, sc.cartProd_6_id, sc.cartProd_7_id, sc.cartProd_8_id,
sc.cartProd_9_id, sc.cartProd_10_id, sc.cartProd_11_id, sc.cartProd_12_id)
GROUP BY
    pg.prodGroupName, p.id, p.productName, p.productDetails, p.productSalesPrice,
p.productCampaignPrice, p.productDiscount, p.productCampaignFlag,
p.productCampaignStart, p.productCampaignEnd, p.productsInStore, p.productsReserved,
p.producPurchPrice, p.productCreated, p.productEdited
ORDER BY
    pg.prodGroupName ASC, total_sales_euros DESC;
```

Following essential sql-statements are used:

JOIN Statements:
The Product table is joined with the ProductGroup table using the productGroup_id.
The ShoppingCart table is left joined with the Product table using the product IDs in the shopping cart.

SUM and COUNT Functions:
Calculates the total sales amount in euros (SUM(sc.cartTotal)) and the number of pieces sold (COUNT(p.id)).

GROUP BY:
Groups the results by ProductGroup name and product details to aggregate the sales data.

ORDER BY:
Orders the results alphabetically by ProductGroup names and then by the total sales amount in euros in descending order.

Best buying customers view: (shopping cart analyze)

Here is a SQL view script that **lists customers with all their attributes, ordered by the best buyers** from the beginning of the current year to the current date. The view also includes the amount of purchases in euros per month up to the month just before the current month. Script orders the results by the total purchases for the year in descending order.

```
CREATE VIEW BestBuyers AS
WITH MonthlyPurchases AS (
    SELECT
        c.id AS customer_id,
        c.custoName AS customer_name,
        c.custoEmail AS customer_email,
        c.custoPhone AS customer_phone,
        c.custoStatus AS customer_status,
        c.custoBlocked AS customer_blocked,
        EXTRACT(YEAR FROM sc.cartPurchasedTime) AS purchase_year,
        EXTRACT(MONTH FROM sc.cartPurchasedTime) AS purchase_month,
        SUM(sc.cartTotal) AS total_purchases
    FROM
        Customer c
    JOIN
        ShoppingCart sc ON c.id = sc.cartCustomer_id
    WHERE
        sc.cartPurchasedTime >= DATE_TRUNC('year', CURRENT_DATE)
    GROUP BY
        c.id, c.custoName, c.custoEmail, c.custoPhone, c.custoStatus, c.custoBlocked,
purchase_year, purchase_month
)
SELECT
    customer_id,
    customer_name,
    customer_email,
    customer_phone,
    customer_status,
    customer_blocked,
    SUM(total_purchases) AS total_purchases_year,
    MAX(CASE WHEN purchase_month = 1 THEN total_purchases ELSE 0 END) AS
january_purchases,
    MAX(CASE WHEN purchase_month = 2 THEN total_purchases ELSE 0 END) AS
february_purchases,
    MAX(CASE WHEN purchase_month = 3 THEN total_purchases ELSE 0 END) AS
march_purchases,
    MAX(CASE WHEN purchase_month = 4 THEN total_purchases ELSE 0 END) AS
```

```
april_purchases,
    MAX(CASE WHEN purchase_month = 5 THEN total_purchases ELSE 0 END) AS
may_purchases,
    MAX(CASE WHEN purchase_month = 6 THEN total_purchases ELSE 0 END) AS
june_purchases,
    MAX(CASE WHEN purchase_month = 7 THEN total_purchases ELSE 0 END) AS
july_purchases,
    MAX(CASE WHEN purchase_month = 8 THEN total_purchases ELSE 0 END) AS
august_purchases
FROM
    MonthlyPurchases
GROUP BY
    customer_id, customer_name, customer_email, customer_phone, customer_status,
customer_blocked
ORDER BY
    total_purchases_year DESC;
```

Following essential sql-statements are used:

Common Table Expression (CTE):
MonthlyPurchases calculates the total purchases per customer per month from the beginning of the current year.
EXTRACT Function:
Extracts the year and month from the cartPurchasedTime to group purchases by month.
SUM Function:
Calculates the total purchases for each customer.
MAX Function with CASE Statements:
Calculates the total purchases for each month up to the current month.
GROUP BY:
Groups the results by customer attributes to aggregate the purchase data.
ORDER BY:
Orders the results alphabetically by ProductGroup names.

User Stories for maintenance of Product, ProductGroup, Customer, and Store Tables.

These user stories cover the create, delete, update, and listing operations for the Product, ProductGroup, Customer, and Store table-management.

Product Table

User Story 15: **Create Product**
As a store manager,
I want to create a new product,
so that I can add it to the store’s inventory.

Acceptance Criteria:
The form should include fields for productName, productPrice, productDiscount, productVAT, and productGroup_id.

The system should validate that productName is unique.
Upon successful creation, the product should be added to the Product table.

User Story 16: **Delete Product**
As a store manager,
I want to delete a product,
so that I can remove it from the store’ s inventory.

Acceptance Criteria:
The system should allow the store manager to select a product to delete.
The system should confirm the deletion action.
Upon confirmation, the product should be removed from the Product table.

User Story 17: **Update Product**
As a store manager,
I want to update product details,
so that I can keep the product information current.

Acceptance Criteria:
The system should allow the store manager to edit fields such as productName, productPrice, productDiscount, productVAT, productGroup_id and amount of product.
The system should validate the updated information.
Upon successful update, the changes should be saved to the Product table.

User Story 18: **List Products**
As a store manager,
I want to list all products,
so that I can view and manage the store’ s inventory.

Acceptance Criteria:
The system should display a list of all products with their attributes.
The list should be sortable by attributes such as productName, productPrice, and productGroup_id.

ProductGroup Table

User Story 19: **Create ProductGroup**
As a store manager,
I want to create a new product group,
so that I can categorize products effectively.

Acceptance Criteria:
The form should include fields for prodGroupName, prodGrDiscount, and prodGrVatPercent.
The system should validate that prodGroupName is unique.
Upon successful creation, the product group should be added to the ProductGroup table.

User Story 20: **Delete ProductGroup**
As a store manager,
I want to delete a product group,

so that I can remove unused categories.

Acceptance Criteria:
The system should allow the store manager to select a product group to delete.
The system should confirm the deletion action (i.e. includes no products).
Upon confirmation, the product group should be removed from the ProductGroup table.

User Story 21: **Update ProductGroup**
As a store manager,
I want to update product group details,
so that I can keep the category information current.

Acceptance Criteria:
The system should allow the store manager to edit fields such as prodGroupName, prodGrDiscount, and prodGrVatPercent.
The system should validate the updated information.
Upon successful update, the changes should be saved to the ProductGroup table.

User Story 22: **List ProductGroups**
As a store manager,
I want to list all product groups,
so that I can view and manage product categories.

Acceptance Criteria:
The system should display a list of all product groups with their attributes.
The list should be sortable by attributes such as prodGroupName.

Customer Table

User Story 23: **Create Customer**
As a new user,
I want to register as a customer,
so that I can create an account and start shopping.

Acceptance Criteria:
The registration form should include fields for custoName, custoPassw, custoEmail, custoPhone, and custoStatus.
The system should validate that custoName and custoPassw are unique.
Upon successful registration, the customer should be added to the Customer table.

User Story 24 **Delete Customer**
As a store manager,
I want to delete a customer,
so that I can remove inactive or fraudulent accounts.

Acceptance Criteria:
The system should allow the store manager to select a customer to delete.
The system should confirm the deletion action.
Upon confirmation, the customer should be removed from the Customer table.

User Story 25: **Update Customer**

As a customer,
I want to update my account details,
so that I can keep my information current.

Acceptance Criteria:
The system should allow the customer to edit fields such as custoName, custoEmail, custoPhone, and custoStatus.
The system should validate the updated information.
Upon successful update, the changes should be saved to the Customer table.

User Story 26: **List Customers**
As a store manager,
I want to list all customers,
so that I can manage customer relationships.

Acceptance Criteria:
The system should display a list of all customers with their attributes.
The list should be sortable by attributes such as custoName and custoStatus.

Store Table

User Story 27: **Create Store**
As a store owner,
I want to register my store,
so that I can manage store operations.

Acceptance Criteria:
The registration form should include fields for storeManagerName, storeManagerPssw, storeName, storeTaxId, storePhone, storeEmail, storeAddress, and storeLogoUrl.
The system should validate that storeManagerName, storeTaxId, storePhone, storeEmail, and storeAddress are unique.
Upon successful registration, the store should be added to the Store table.

User Story 28: **Delete Store**
As a store owner,
I want to delete my store,
so that I can remove it from the system.

Acceptance Criteria:
The system should allow the store owner to select a store to delete.
The system should confirm the deletion action.
Upon confirmation, the store should be removed from the Store table.

User Story 29: **Update Store**
As a store owner,
I want to update my store details,
so that I can keep the store information current.

Acceptance Criteria:
The system should allow the store owner to edit fields such as storeManagerName, storeName, storeTaxId, storePhone, storeEmail, storeAddress, and storeLogoUrl.

The system should validate the updated information.
Upon successful update, the changes should be saved to the Store table.

User Story 30: **List Stores**
As a store owner,
I want to list all stores,
so that I can view and manage store information.

Acceptance Criteria:
The system should display a list of all stores with their attributes.