2026

Lab 07 Report

SUBMITTED BY:

DANIYA AAMIR

Problem 1: Customer Segmentation Analysis

a) Defined Customer Segments

To incorporate business intelligence into the PVFC order processing system, customers were divided into a 2x2 matrix focusing on Frequency (Total Orders) and Monetary Value (Total Spend). The baseline thresholds were set at > 1 Order and >= $1,000 Spend.

1. **Premium Customers:** High Frequency (> 1 order) and High Spend (>= $1,000). These are the most valuable returning clients.
2. **High-Value Buyers:** Low Frequency (Exactly 1 order) but High Spend (>= $1,000). These clients make large, expensive purchases but infrequently.
3. **Frequent Shoppers:** High Frequency (> 1 order) but Low Spend (< $1,000). These clients return regularly for smaller purchases.
4. **Casual Customers:** Low Frequency (Exactly 1 order) and Low Spend (< $1,000).

b) Database & SQL Methodology

To effectively demonstrate this quadrant analysis, the database was expanded using SET IDENTITY_INSERT to safely add additional mock customers and order lines without violating Primary Key constraints.

A highly optimized master SQL query was designed to aggregate the total orders and calculate the gross spend per customer by joining the CUSTOMER_t, ORDER_t, Order_line_t, and PRODUCT_t tables. Rather than filtering the data in the application layer, a CASE WHEN statement was utilized to push the segmentation logic directly into the SQL Server engine for maximum efficiency.

Master SQL Query:

SQL

```
SELECT c.Customer_Name AS Name,

    COUNT(DISTINCT o.Order_Id) AS TotalOrders,

    SUM(ol.Ordered_Quantity * p.Standard_Price) AS TotalSpend,

    CASE
```

```
        WHEN COUNT(DISTINCT o.Order_Id) > 1 AND SUM(ol.Ordered_Quantity *
p.Standard_Price) >= 1000 THEN 'Premium'

        WHEN COUNT(DISTINCT o.Order_Id) = 1 AND SUM(ol.Ordered_Quantity *
p.Standard_Price) >= 1000 THEN 'HighValue'

        WHEN COUNT(DISTINCT o.Order_Id) > 1 AND SUM(ol.Ordered_Quantity *
p.Standard_Price) < 1000 THEN 'Frequent'

        ELSE 'Casual'

      END AS Segment

FROM CUSTOMER_t c

JOIN ORDER_t o ON c.Customer_Id = o.Customer_Id

JOIN Order_line_t ol ON o.Order_Id = ol.Order_Id

JOIN PRODUCT_t p ON ol.Product_Id = p.Product_Id

GROUP BY c.Customer_Id, c.Customer_Name
```

c) Visual Studio Implementation (Backend Handler)

The user interface (segmentationDashboard.aspx) was built using a CSS Grid layout to visually represent the 2x2 matrix. The backend handler executes the SQL query, reads the categorized rows, and dynamically populates four separate DataTable objects bound to specific UI GridViews.

VB.Net

```
Protected Sub btnAnalyze_Click(sender As Object, e As EventArgs) Handles btnAnalyze.Click

    Dim dtPremium As DataTable = CreateSegmentTable()

    Dim dtHighValue As DataTable = CreateSegmentTable()

    Dim dtFrequent As DataTable = CreateSegmentTable()

    Dim dtCasual As DataTable = CreateSegmentTable()


    Using con As New SqlConnection(connString)
```

```
    Using cmd As New SqlCommand(query, con)
        con.Open()
        Dim reader As SqlDataReader = cmd.ExecuteReader()
        Dim totalCount As Integer = 0

        While reader.Read()
            totalCount += 1
            Dim name As String = reader("Name").ToString()
            Dim orders As Integer = Convert.ToInt32(reader("TotalOrders"))
            Dim spend As Decimal = Convert.ToDecimal(reader("TotalSpend"))
            Dim segment As String = reader("Segment").ToString()

            ' Route the customer to the correct memory table based on the SQL Engine's
categorization
            Select Case segment
                Case "Premium"
                    dtPremium.Rows.Add(name, orders, spend)
                Case "HighValue"
                    dtHighValue.Rows.Add(name, orders, spend)
                Case "Frequent"
                    dtFrequent.Rows.Add(name, orders, spend)
                Case Else
                    dtCasual.Rows.Add(name, orders, spend)
            End Select
```

```
        End While

        If totalCount > 0 Then
            gvPremium.DataSource = dtPremium
            gvPremium.DataBind()
            gvHighValue.DataSource = dtHighValue
            gvHighValue.DataBind()
            gvFrequent.DataSource = dtFrequent
            gvFrequent.DataBind()
            gvCasual.DataSource = dtCasual
            gvCasual.DataBind()

            pnlQuadrants.Visible = True
            lblMessage.Text = "Segmentation completed successfully."
        End If
    End Using
  End Using
End Sub
```

Problem 2: Automating Targeted Marketing

Targeted marketing is automated by embedding actionable application controls directly within the segmented UI. A specific "Send VIP Gift" handler is wired to the Premium quadrant. When triggered, the system simulates batch processing for all customers matching the highest-tier criteria, automating the dispatch of physical gifts and thank-you communications.

VB.Net

```
' The Targeted Marketing Automation Handler

Protected Sub btnPremiumPromo_Click(sender As Object, e As EventArgs) Handles
btnPremiumPromo.Click

    ' Simulating sending an email/gift to the Premium customers

    lblMessage.ForeColor = System.Drawing.Color.Blue

    lblMessage.Text = "System Notification: VIP Courier Gifts dispatched to Premium Customers!"

End Sub
```

Problem 3: Segmentation Platform Test Cases

Test Case	Description	Expected Output
1. Empty Segment Handling	Running analysis when no users meet the criteria for "Frequent Shoppers".	The specific GridView remains completely empty without throwing a NullReferenceException, while other quadrants populate normally.
2. Overlapping Customers	Querying Customer A (1 order, exactly $1000 spend) and Customer B (2 orders, $999 spend).	Customer A strictly falls into "High-Value" (inclusive >= 1000). Customer B strictly falls into "Frequent" (exclusive < 1000). No duplicates occur.
3. Accurate Aggregate Spend	Querying a customer with three separate historical orders of $400 each.	The SQL query successfully aggregates the sum to $1,200 and routes them to the Premium grid, rather than evaluating orders individually.
4. Dormant Customer Exclusion	Querying a registered user with 0 total orders.	The customer is correctly excluded from all quadrants, as the minimum matrix frequency is Total Orders = 1.
5.	A standard user (Role 2) attempts to	The Page_Load session validation

Unauthorized Access Restriction	directly navigate to segmentationDashboard.aspx.	intercepts the request and redirects the user to login.aspx.

Notes on some of the problems encountered

Deployment Architecture & Server Constraints

1. Enterprise SQL Optimization A key architectural decision was pushing the filtering logic into the SQL engine using a CASE WHEN block. Instead of pulling the entire unorganized dataset across the network into the application layer's memory, the database pre-sorts the data. This drastically reduces the memory footprint on the web server, making the application highly scalable for large datasets.

2. The "Architectural Shield" Deployment Strategy A significant challenge occurred during live deployment to the free Somee.com hosting environment. Due to the 150MB strict storage quota, all legacy labs (Lab 4, 5, 6) and the current Lab 7 were forced to share a single root bin folder (a "Shared Brain").

This caused the legacy Lab 4 payment.aspx UI to crash via a NullReferenceException because it inadvertently triggered the Lab 7 dynamic shopping cart code running in the shared .dll.

To resolve this without exceeding server storage quotas, an "Architectural Shield" was engineered into the VB.NET backend. By utilizing a dynamic object detection check (If pnlCard Is Nothing Then Exit Sub), the application intelligently determines which HTML interface it is currently rendering. If it detects the legacy Lab 4 interface (where modern controls are missing), it safely halts backend execution. This prevents the null reference crash entirely while allowing the older UI to render flawlessly alongside the modern Lab 7 architecture.