Featherman’s Analytics Adventures©:
*Manipulating Data with Arrays – Accountants Aging Table Example*

Once you have transaction data available in webpage arrays or SQL Server Database tables, then you can manipulate (reshape) the data to condense it. Condensing and compiling transaction data into summary formats to enable analysis at a different levels is one of the major goals of analytics.

In this example we perform a seemingly simple but deceptively difficult data manipulation. We have a table of sales transactions as shown on the right, each sale has an ID, date, Total\_Sale amount, CustomerID and indicant of whether the invoice is paid or not. What the business requirement specifies however is an aging report. The long list of invoices needs to be culled and organized into different time buckets relating to common accounting practice of categorizing unpaid invoices into the time buckets 0-30 days old, 31-60 days old, 61-90 days old, 91-120 days old, 121-180 days old and 181-360 days old.

This program then shows how to loop the entire fact table of sales transactions, identify the invoices that are unpaid, calculate the number of days that the invoice is past due, and place the unpaid amount into one of the time buckets mentioned above. This of course is calculating the accounts receivable aging report. First we produce this report for all unpaid invoices (no further categorizations), then we calculate the aging report for each customer, as shown on the left.

We need to examine each and every row in the sales table to see if it is unpaid. If the row is unpaid then the sales amount needs to be added to one of the time buckets. While here we sum up the accounts receivable, this same type of data management would make it possible to produce reports of how many transactions occurred in the time buckets specified (frequency report). Another application could be to average the dollar amount of invoices in each of the time buckets. Marketers and supply chain analysts would use these reports to better understand consumer behavior and seasonality of demand.

How do you analyze every row in a datatable, and examine and possibly retrieve values from its columns? Use a loop. Once you calculate the number of days an unpaid invoice is overdue, how do you choose which time bucket to append the sales amount to? Use CASE processing. Where do you put the unpaid Sales\_amounts? In a new array with columns that you specify. How do you perform this data management for each customer, showing each customer on its own row? You need to loop the sales table once for each customer, we use a loop within a loop to provide this functionality.

Ok here is the code!

Imports System.Data

Imports System.Data.SqlClient

'This program was created to further the data manipulation capabiities of all students, and especially accountants. An aging report is generated. The sales for the last 180 days are retrieved from a databasea table and copied into an array. A second array is then created and the invoice values are poured into the columns (time buckets) 0-30 days, 31-60 days, 61-90 days, 91-145 days, and 146-180 days.

'One purpose of this program is to encourage you to realize that once data is in a database, it can be compiled into formats using code. While it may be hard to write the code (at first), then the functionality is automated and you never have to manually perform it.

Partial Class NewFor2018\_RetrievingData

 Inherits System.Web.UI.Page

 'No need to analyze this next line of code until later in the course. The next line is what a connection string looks like, it specifies what database to connect to (initital catalog), on a specified server (datasource) with a specified userID and password. Because the program read/edits/deletes data from some of Featherman’s datatables the UserID and Password are intentionally ommitted. You can however make the same datatables in your database and replicate the functionality

 Public Shared con As New SqlConnection("Data Source=cb-ot-devst05.ad.wsu.edu;Initial Catalog=Featherman\_Analytics;Persist Security Info=True;User ID=;Password=")

 'In this project we use the dataadapter to run a sql select statement and to retrieve data into a datatable (which is an array). Then we set the datasource property of a gridview to that datatable. We could have started with Select \* to retrieve all the columns from the specified table and not included a WHERE statement to retrieve all the rows. Here we retrieve the last 25 rows of data since we select the top 100 rows but sort them in DESC (decending order) by invoice #.

 Public Shared gdtAgingReport As New DataTable

 Public Shared gdtAgingReportbyCustomer As New DataTable

 Public Shared gdtSales As New DataTable

 Public Shared gdtCustomers As New DataTable

#Region "Page Init - Create the Aging Table array"

 Private Sub NewFor2018\_RetrievingData\_Init(sender As Object, e As EventArgs) Handles Me.Init

 'Notice the gdtAgingReport datatable was created at the global level. Here we add the columns to gdtAgingReport table, however if this procedure was already run, there is no need to create the columns again so we stop.

 Call GetSales()

 If gdtAgingReport.Columns.Count > 0 Then Exit Sub

 With gdtAgingReport 'add columns to the array. Each column is a time bucket to drop sales into.

 .Columns.Add("0-30", GetType(Decimal))

 .Columns.Add("31-60", GetType(Decimal))

 .Columns.Add("61-90", GetType(Decimal))

 .Columns.Add("91-120", GetType(Decimal))

 .Columns.Add("121-180", GetType(Decimal))

 .Columns.Add("181-360", GetType(Decimal))

 End With

 With gdtAgingReport

 .Columns("0-30").DefaultValue = 0

 .Columns("31-60").DefaultValue = 0

 .Columns("61-90").DefaultValue = 0

 .Columns("91-120").DefaultValue = 0

 .Columns("121-180").DefaultValue = 0

 .Columns("181-360").DefaultValue = 0

 End With

 Dim dr As DataRow = gdtAgingReport.NewRow

 'add the row to the table and display the table

 gdtAgingReport.Rows.Add(dr)

 GridView2.DataSource = gdtAgingReport

 GridView2.DataBind()

 '--------------------- Below this line is for the advanced content -------------------------------------

 'If you just stick to the content above this line and learn how to make the aging report for all the customers combined, then you have learned good informatio about looping arrays and reformatting data from tall tables of transactions, to a condensed one row table that has time buckets that are used compile the unpaid invoice amounts into the categories 0-30 days, 31-60 days, etc. This procedure then compiles all the invoices to one row of data, compiling all the invoices for all the customers. But what if you want more specific breakpout of accounts receivable? What if you want to see separate lines on the aging report for region, store, or customer? Then you would need to analyze the data by the extra dimension and create a row of data for each value of that dimension. We do that here for customers to see the Agging report by Customer.

 'Before we get into explaining the code, it is important to not that this problem cannot be solved by a simple GROUP BY or PIVOT query in SQL. Those approaches use established columns that can easily be specified by a column in the table for example month or yearmonth combination which can be used in the GROUP BY or PIVOT. Here we use columns that are not in the base data (0-30, 31-60, etc.) so each row needs to be examined and based on a # days since invoice calculation, and values used to incremented time buckets in a second table. Another interesting aspect of this data management problem is that over time the values change buckets for example an invoice moving from the 31-60 day column to the 61-90 day column.

 'So this is not a simple data management manipulation, rather custom code is needed either in your ASP.NET webpage or SQL query. The SQL version discussed elsewhere can be accomplished using a series of sub-queries, one for each time bucket (0-30 days). That approach is superior in that the code is fewer lines and less complex (which is always the goal with technology). Here we use SELECT CASE processing inside a FOR EACH NEXT loop to provide the same functionality. FYI better DBA's use more advanced SQL code, which is invariably less complex in that it has fewer line of code, but is more advanced so requires learning. A SELECT CASE inside a loop (whether in code or SQL query) is old-school. The code works but there are easier ways to do this. So the long term suggession is don't fall in love with loops to do advanced data manipulation, learn more SQL. IN general if you are beating your head to get data management to work, it just means you need to learn more advanced SQL functionality.

 '----------------------- OK here is the functionality -------------------------------------------------

 'To recap we run a loop to examine each and every sale to see which time bucket its invoice total needs to be added to. Now we need to run this loop once for each customer. So we need a customers list. The Sales in the Sales table include the CustomerID field so we just need to retrieve a list of the CustomerID's as shown next. While we could retrieve a list of CustomerID's from the Customers table we can also stick with our use of just the one Sales table by introducing the SELECT DISTINCT(fieldname) SQL code. Becuase the CustomerID's are repeated in the Sales table when customer have > 1 sale, we utilize the DISTINCT() function to retrieve just one instance of each CustomerID (get rid of the duplicates).

 Dim daGertCustomers As New SqlDataAdapter("SELECT DISTINCT(CustomerID) as CustomerID FROM featherman.Sales ORDER BY CustomerID", con)

 Try

 daGertCustomers.Fill(gdtCustomers)

 Catch ex As Exception

 Response.Write(ex.Message)

 End Try

 'We will display the aging data by customer. This more granular (detailed) data is poured into a new array So that we can see a summary aging report and a more detailed aging report with one row for each customer that has an unpaid invoice. After the data is calcualted and placed into the time buckets, it will be saved back to the SQL Server Database. Here is a look at the table schema that will hold the calculated aging report. So we are creating an array with the same data structure (column names, datatypes, default values, primary key, etc.) as the SQL database table. This is important because if the schema of the webpage’s array matches the schema of the database table, then you can use the super-easy DataAdapter to save the array data back to the SQL Serve database.



 If gdtAgingReportbyCustomer.Columns.Count > 0 Then Exit Sub

 With gdtAgingReportbyCustomer 'notice that this array has a customerID field.

 .Columns.Add("CustomerID", GetType(Decimal))

 .Columns.Add("0-30", GetType(Decimal))

 .Columns.Add("31-60", GetType(Decimal))

 .Columns.Add("61-90", GetType(Decimal))

 .Columns.Add("91-120", GetType(Decimal))

 .Columns.Add("121-180", GetType(Decimal))

 .Columns.Add("181-360", GetType(Decimal))

 End With

 With gdtAgingReportbyCustomer

 .Columns("0-30").DefaultValue = 0

 .Columns("31-60").DefaultValue = 0

 .Columns("61-90").DefaultValue = 0

 .Columns("91-120").DefaultValue = 0

 .Columns("121-180").DefaultValue = 0

 .Columns("181-360").DefaultValue = 0

 End With

 'setting these default values is important becuase we will add the rows to this array in a loop, and the se default values are critical. If you want to add values in a column, that column's initial value CANNOT by NULL, so we set the initial value to 0.

 End Sub

#End Region

#Region "Create Summary Aging Report: Pour the data from one array into another"

 Protected Sub Button1\_Click(sender As Object, e As EventArgs) Handles Button1.Click

'This procedure is the new learning that you are ready for. Here we loop the array of sales and use timespan to calculate the # days the unpaid invoice is outstanding. Then based on the categories created (0-30, 31-60, etc.) the sales\_total amount is added to the category bucket (the sales\_total value is added to the right column based on how old it it).

'From an analytics perspective we are taking a narrow and tall table (few columns, many rows) and pour the values from one column into a new array that is narrow and short (few columns one row.) When doing data management, it is common to summarize data and condense it. Often this is referred to reshaping data.

'So we use the values from one array to load up another array. We use a loop with a CASE statement inside it to perform the data management. The next step would be to provide this aging report for each customer or region, to add more rows. How would you do this?

 'make sure the invoices array is up to date. This is necessary if the program user adds a new sale with the hyperlink

 Call GetSales()

 Dim ts As TimeSpan ' used to calculate the age of each unpaid invoice

 'Clear out the data table of prior info and clear out the gridview

 GridView2.DataSource = Nothing

 With gdtAgingReport.Rows(0) 'here we reset each column back to zero. There may be a fancier way to do this.

 .Item("0-30") = 0

 .Item("31-60") = 0

 .Item("61-90") = 0

 .Item("91-120") = 0

 .Item("121-180") = 0

 .Item("181-360") = 0

 End With

'This loop creates the datarow, then loops each and every row in the sales datatable. First the number of days the invoice is outstanding is calculated. Becuase the timespan calculation is inside a loop it is performed over and over, once for each row in the sales table.

 For Each dr As DataRow In gdtSales.Rows

 ts = Now.Subtract(dr("Invoice\_Date"))

'now we increase one of the column's value based on the number of days the timespan calculated. Notice the dr refers to the arrays on the right hand side of the = sign. THe value on the right hand side of the = sign is being assigned to the column on the left hand side of the = sign. Notice we are being very careful to specify the column to update (0-30, 31-60, etc.). We could use a With End With statement to reduce the code here, but as a teaching tool, the long column specification is kept so that you realize where the value is being added to, the agingReport array.

 Select Case ts.TotalDays

 Case 0 To 30

 gdtAgingReport.Rows(0).Item("0-30") += dr("Total\_Sale")

 Case 31 To 60

 gdtAgingReport.Rows(0).Item("31-60") += dr("Total\_Sale")

 Case 61 To 90

 gdtAgingReport.Rows(0).Item("61-90") += dr("Total\_Sale")

 Case 91 To 120

 gdtAgingReport.Rows(0).Item("91-120") += dr("Total\_Sale")

 Case 121 To 180

 gdtAgingReport.Rows(0).Item("121-180") += dr("Total\_Sale")

 Case 180 To 360

 gdtAgingReport.Rows(0).Item("180-360") += dr("Total\_Sale")

 End Select

 Next

‘now that the data is compiled, show the array to the program user.

 GridView2.DataSource = gdtAgingReport

 GridView2.DataBind()

 End Sub

#End Region

#Region "Fetch the Sales Data from a database into an the array inside our webpage"

 Protected Sub GetSales()

'this is the dataAdapter object that knows how to run SQL statements against databases. It has two properties, a SQL statement in red and the name of the connection upon which to run the SQL statement. The conenction specifies which database on what server to run the SQL against. It also specifies the userID and password to get thru the firewall.

 If gdtSales.Rows.Count > 0 Then gdtSales.Rows.Clear()

 Dim AllSalesDataAdapter As New SqlDataAdapter("SELECT [Sale\_ID], [CustomerID], FORMAT ([Invoice\_Date], 'MM/dd/yyyy') as [Invoice\_Date], [Region], [Total\_Sale] FROM featherman.Sales WHERE [Paid] = 0 AND Invoice\_Date > Getdate() - 180 ORDER BY Sale\_ID DESC", con)

'the data that is returned is placed into an in-memory datatable (which is a strongly typed array, meaning each column has a name, data type and perhaps restrictions)

'this next line runs the red select statement from a prior line. Next we can get and show a row count as follows. If any errors are found they will be shown in at the top of the page. We use a try/catch block to run fragile code that might fail (here since the server is down).

 Try

 AllSalesDataAdapter.Fill(gdtSales)

'the control on the webpage needs to be databound (glued) to the data so that it can be displayed. We use a gridview control to show a table of data.

 GridView1.DataSource = gdtSales

 GridView1.DataBind()

 Catch ex As Exception

 Response.Write(ex.Message)

 End Try

 End Sub

#End Region

#Region "Build Aging Report for Each Customer"

 Protected Sub Button3\_Click(sender As Object, e As EventArgs) Handles Button3.Click

'This procedure is outside the score of this class, however as soon as you show an aging report, the account will say "I need that automated reporting by Customer account. So here is the solution for that. We create a new procedure, new datatable, new loop inside a loop metaphor and after all that data compiling (creating one row in the array per customer) the next step is to save that data into the database so here is a lot of funcitonality. Select CASE processing inside a Loops (to examine each sale) within a loop (one loop iteration per customer).

 Dim ts As TimeSpan 'used to calculate for each invoice how many days its been.

'After building the array of customers, why not save that array to a SQL database to store the information and make it available to connect to with Tableau, PowerBUI or Excel. The save to database concept is explained step by step in another module, and is included here NOT to have you focus on it too soon, but rather to see the full solution. Build an array, do amazing data processing, then save your work back to the database. Creating this dataAdapter allow a later UPDATE of the database.

 Dim DAAgingReportByCustomer As New SqlDataAdapter("SELECT \* FROM [featherman].[AgingReportByCustomer]", con)

 Dim cbAgingReport As New SqlCommandBuilder(DAAgingReportByCustomer) 'this is needed to do the update of the database.

 Dim cmdClearAgingTable As New SqlCommand("DELETE FROM [featherman].[AgingReportByCustomer] ", con)

 'Clear out the data table of prior info and clear out the gridview

 If gdtAgingReportbyCustomer.Rows.Count > 0 Then gdtAgingReportbyCustomer.Rows.Clear()

 GridView3.DataSource = Nothing

'The loop inside a loop means a) for each customer (loop#1), b) loop all of the sales in the sales array and for the current customer being processed, fill each of the time buckets to create the aging report.

'This next line sets up the outer loop to ensure each of the customers has a new row of data added to the array, and invoices analyzed and bucketed. So this is the outer loop.

 For Each dr2 As DataRow In gdtCustomers.Rows

'Here we create the datarow for the customer so we can pour the invoice amount into the column. So this line of code wil be run once for each customer.

 Dim dr3 As DataRow = gdtAgingReportbyCustomer.NewRow

'Here we provide the value to the first column of our new row in the detailed aging report (housed in the gdtAgingReportbyCustomer)

 dr3.Item("CustomerID") = dr2.Item("CustomerID")

'This is the inner loop. Just as before we examine each and every sale in the gdtSales array. We examine the invoice\_date filed and calculate how old the invoice is (number days).

 For Each dr As DataRow In gdtSales.Rows

 ts = Now.Subtract(dr("Invoice\_Date"))

'now that we have a new row of data with a customer ID, we examine each invoice.'

'IF the invoice's CustomerID number is the same as the customerID that we are examining with this loop iteration, then we increase one of the column's value based on the number of days the timespan calculated. Notice the dr refers to the arrays on the right hand side of the = sign. THe value on the right hand side of the = sign is being assigned to the column on the left hand side of the = sign. Notice we are being very careful to specify the column to update (0-30, 31-60, etc.). We could use a With End With statement to reduce the code here, but as a teaching tool, the long column specification is kept so that you realize where the value is being added to, the gdtAgingReportbyCustomer array.

 If dr3.Item("CustomerID") = dr("CustomerID") Then 'this line then is very important to ensure only the invoices for the current customer are being examined and added to the column's totals.

 Select Case ts.TotalDays

 Case 0 To 30

 dr3.Item("0-30") += dr("Total\_Sale")

 Case 31 To 60

 dr3.Item("31-60") += dr("Total\_Sale")

 Case 61 To 90

 dr3.Item("61-90") += dr("Total\_Sale")

 Case 91 To 120

 dr3.Item("91-120") += dr("Total\_Sale")

 Case 121 To 180

 dr3.Item("121-180") += dr("Total\_Sale")

 Case 181 To 360

 dr3.Item("181-360") += dr("Total\_Sale")

 End Select

 End If

 Next

'At this point of the code all of the invoices have been examined by the inner loop. The code adds to the column's as needed, so the row should have its columns filled with data IF unpaid invoices are found. So we save the completed row of data for the customer and then go to the next customer. When all the customer have been processed, then we can display the data and save it to the database.

 gdtAgingReportbyCustomer.Rows.Add(dr3)

 Next

'At this point in the code all the invoices have been scanned over and over, once for each customer. The thinng about computing is that it does the processing so fast that we don't realize all the hard work being performed. As of this writing there are 2420 sales invoices and 28 customers, so the sales table has been analyzed over and over.

'So now we display the aging report array in a gridview control which are used to display tables of data. We clear the SQL Server database table of old data with the cmdClearAgingTable.ExecuteNonQuery() then use the DAAgingReportByCustomer data adapter to save the data in the gdtAgingReportbyCustomer array back to the database table for permanent storage.

 Try

 GridView3.DataSource = gdtAgingReportbyCustomer

 GridView3.DataBind()

 If con.State = ConnectionState.Closed Then con.Open()

 cmdClearAgingTable.ExecuteNonQuery()

 DAAgingReportByCustomer.Update(gdtAgingReportbyCustomer)

 Catch ex As Exception

 Response.Write(ex.Message)

 End Try

 End Sub

#End Region

End Class