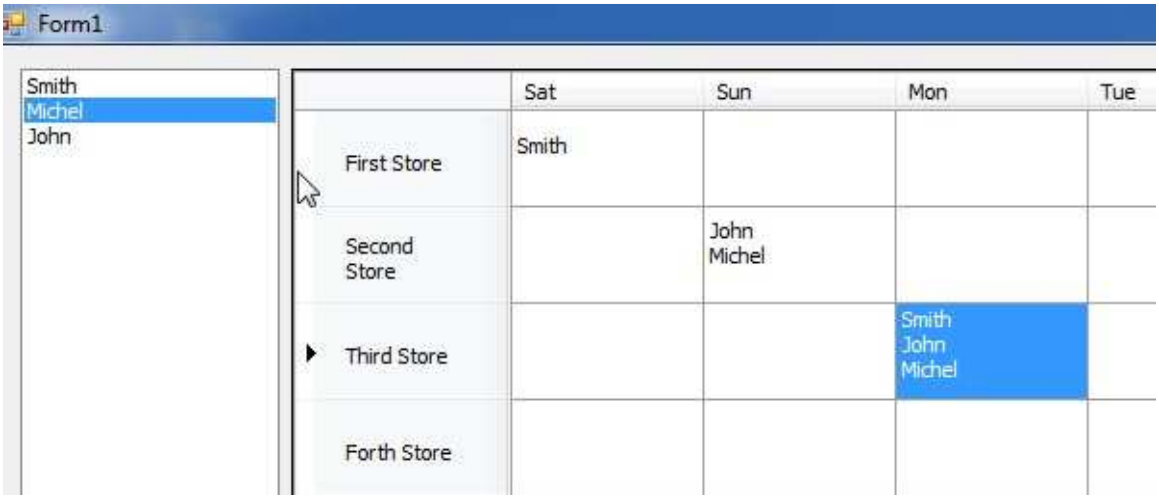
Database Tutorials

Hello everyone… welcome to database tutorials. These are going to be very basic tutorials about using the database to create simple applications, hope you enjoy it. If you have any notes about it, please send them to **notes@mka-soft.com.** Finally if you find these tutorials are useful, it would be nice from you to send a small donation via PayPal to **donation@mka-soft.com**.

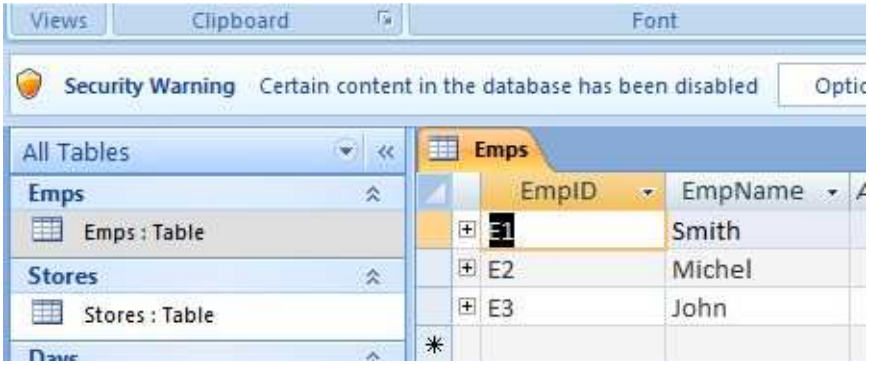
The work with this tutorial started on 2010-NOVEMBER-23.

Accessing DataTables From VB.NET Code

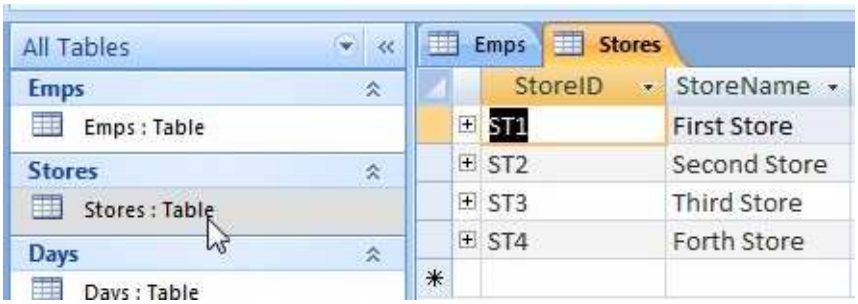
This tutorial is about accessing the tables programmatically. We are going to create a simple app what will help us assign a number of employees into stores for any given day as shown below.



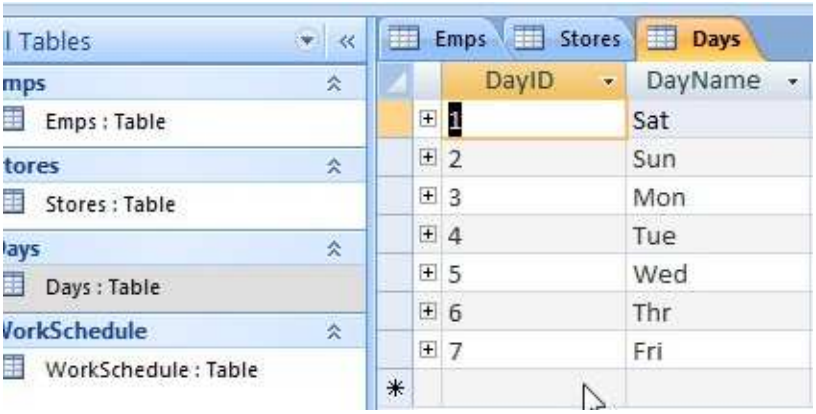
Before you start, you need to create 4 tables as follows:



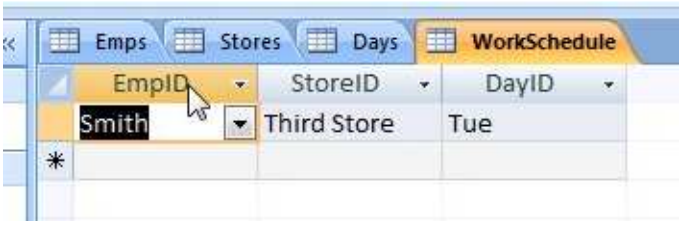
The Emps table contains the EmpID and EmpName columns. The EmpID is the primary key.



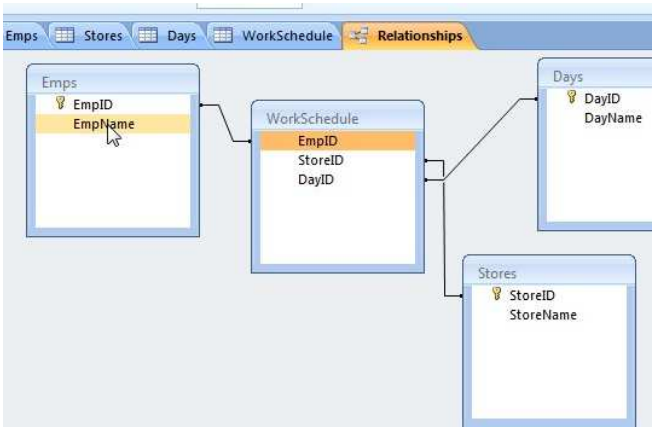
The Stores table is similar. It has StoreID as primary key, and StoreName as the name.



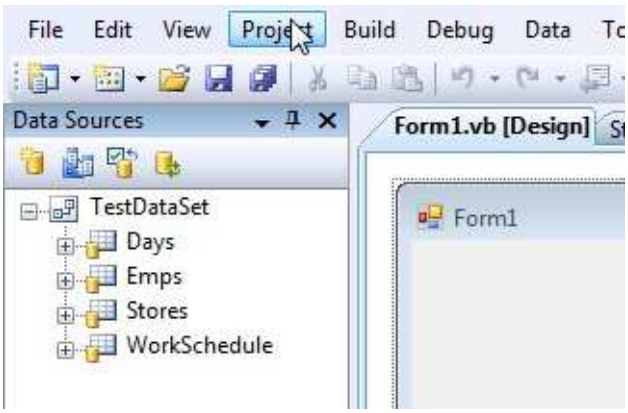
The days table is similar. You can actually make this a single column table containing the name of the day only. However this is just an example (and this might be a bad design as well).



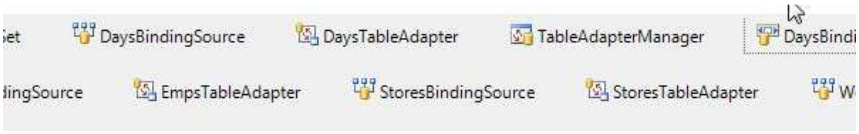
The last table contains information used to link the three tables. Here it contains the primary key values , but the lookup wizard shows the values from other tables, not the actual value stored in the table. Finally the relationship graph should be something like the following:



Now we are ready to start with our project. Start a new project, and link it to the database. Make sure you include all 4 tables in the dataset.



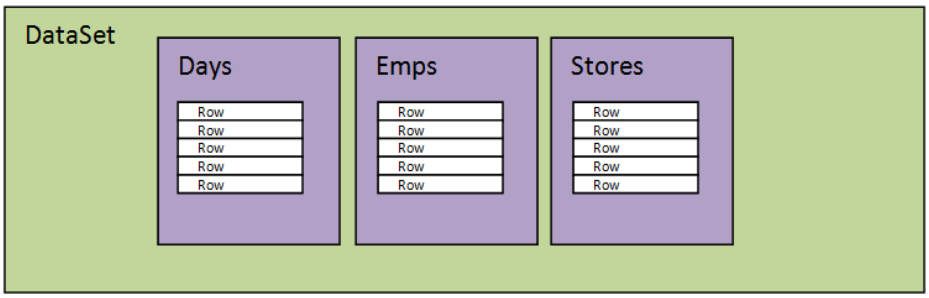
Next we need to access all 4 tables on this form, so drag each table and drop it on the form. This will make the wizard create all the needed controls and the required code.



Remove the DataGridView controls and the Navigator controls created by the wizard but keep all the controls created at the button. Double click the form to see the code created by the wizard.

```
        'TODO: This line of code loads data into the 'TestDataSet.WorkSchedule' table.
You can move, or remove it, as needed.
        Me.WorkScheduleTableAdapter.Fill(Me.TestDataSet.WorkSchedule)
        'TODO: This line of code loads data into the 'TestDataSet.Stores' table. You can
move, or remove it, as needed.
        Me.StoresTableAdapter.Fill(Me.TestDataSet.Stores)
        'TODO: This line of code loads data into the 'TestDataSet.Emps' table. You can
move, or remove it, as needed.
        Me.EmpsTableAdapter.Fill(Me.TestDataSet.Emps)
        'TODO: This line of code loads data into the 'TestDataSet.Days' table. You can
move, or remove it, as needed.
        Me.DaysTableAdapter.Fill(Me.TestDataSet.Days)
```

This code will load the information from the database into your dataset. Now let us see how does the computer store the tables into memory, and how you can access these tables.



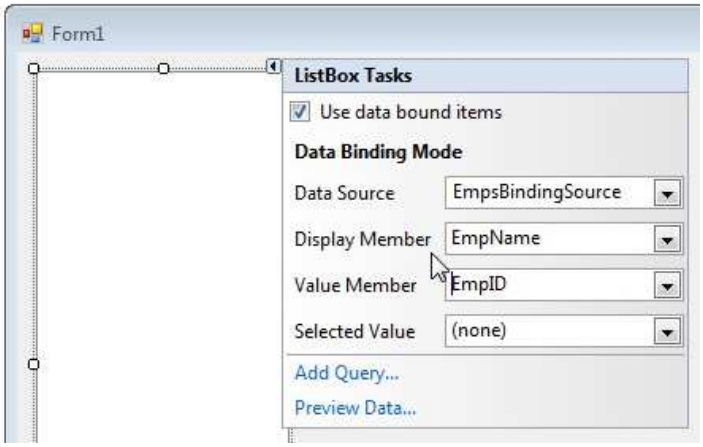
As you can see from the graph, the DataSet is the top level object. From that you can access any of the tables. And within each table you see its rows. So now the code:

```
Me.WorkScheduleTableAdapter.Fill(Me.TestDataSet.WorkSchedule)
```

means fill the WorkSchedule table which is part of the TestDataSet. In a similar manner:

```
Me.StoresTableAdapter.Fill(Me.TestDataSet.Stores)
```

means fill the Stores table which is part of the TestDataSet, and so on. Now, we want to create the interface. So go back to the form design, drop a listbox and bind it to the Emps table.



Next place a DataGridView. The DataGridView will be used to display all the information of the employee schedule. You will need to change the behavior of the DataGridView so that it work in a way that serve our problem. So change the following properties

1- Change Name to DGV

We want the name shorter to make it easier in writing the code. But you can leave it as it is.

2- Change Anchor to Top,Button,Left,Right

When you want to resize the window, the control adjusts itself as well.

3-Change AllowUserToAddRow to false

You don't want a user to add rows into the table at random because the rows represent stores, and you already have the stores table.

4-Change AllowUserToDeleteRows to false

You don't want a user to remove rows from the table because each row represent a store.

5-Change AllowUserToResizeColumns to false

You want to have a fixed width for each cell

6- Change BackgroundColor to White

Just for looks

7-Change ReadOnly to true

You don't want a user to change the content of a cell by just typing into it.

8-Change RowHeaderWidth to 200

This gives enough space to display store name later. You can change it if you like.

9-Change RowsDetaultCellStyle, WrapMode to true

This will allow you to display multiple lines into the cell, which can't be done in previously

10-Change SelectionMode to CellSelect

This will allow the user to select a cell instead of a complete row, or column

11- Change MultiSelect to False

This will force the user to select one cell at a time

Now we are ready to start displaying the schedule in the DataGridView (DGV). In the code window we will create a function that displays the information in the DGV control. First thing we want to do is to display the columns. Each column must display the name of the day. The names of the days are available in the Days table. So to access these, write the following code:

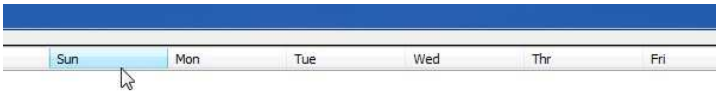
```
    Public Sub FillDGV()

        ' clear the DGV
        DGV.Rows.Clear()
        DGV.Columns.Clear()

        ' display all days as columns
        Dim I As Integer
        For I = 0 To TestDataSet.Days.Count - 1
            DGV.Columns.Add(TestDataSet.Days.Rows(I).Item(0),
TestDataSet.Days.Rows(I).Item(1))
        Next
    End Sub
```

the first two commands removes all the available rows (if they exist) then removes all available columns from the DGV. This makes the control ready for our new set of controls

Next comes the For loop. The TestDataSet.Days obviously return the Days table (as explained in the figure above). The .Count returns the total number of rows. So this loop works on every single row in the table. Inside the loop, we are adding the columns. To display a title for each column, we need to specify its header text. The TestDataSet.Days.Rows(I) returns the Ith row from the table. To get the value from a specific column, you use the Item property and specify which field you want to access. So now TestDataSet.Days.Rows(I).Item(1) return the name of the

day for the Ith row in the Days table. Item(1) represent the second column in the table. Item(0) represent the first column in the table. You can also use actual column name to get the value (as we will do so later).

In the load event, call this subroutine before the Load sub ends. Run the code, you should see something similar to the following:



Now we try adding the rows, which represent the stores. Obviously we will need to access the stores table. So update the FillDGV by adding the following:

```
' display all stores as rows
For I = 0 To Me.TestDataSet.Stores.Count - 1
    DGV.Rows.Add("")
    DGV.Rows(I).HeaderCell.Value = TestDataSet.Stores(I).Item("StoreName")
Next
```

In a similar way, the TestDataSet.Stores.Count tells how many rows available in the stores table. The DGV.Rows.Add(“”) will create an empty row. The line after that will change the text in the header of that row to contain the second column from the Stores table (which is the store name). Run the code you should see something like this:

Finally we need to fill the content of every cell. Obviously, we have the name in the Emps table, the store name in the stores table, and the name of the day in the days table. The schedule however uses only the primary keys of all these table. For example if E1 should work on day 1 at store ST01, this would mean:

- get the name of E1

- get the row of the store ST01

- get the column of the day 1

- update the content of the cell

The problem with our code is that we did not store information about the primary key for the values (in the rows and columns). The simple solution to this is to use the tag property to store these. Modify the code to be like this:

```
        ' display all days as columns
        Dim I As Integer
        For I = 0 To TestDataSet.Days.Count - 1
            DGV.Columns.Add(TestDataSet.Days.Rows(I).Item(0),
TestDataSet.Days.Rows(I).Item(1))
            DGV.Columns(I).Tag = TestDataSet.Days.Rows(I).Item(0)
        Next


        ' display all stores as rows
        For I = 0 To Me.TestDataSet.Stores.Count - 1
            DGV.Rows.Add("")
            DGV.Rows(I).HeaderCell.Value = TestDataSet.Stores(I).Item("StoreName")
            DGV.Rows(I).Tag = TestDataSet.Stores(I).Item(0)
        Next
```

The tag generally can store any kind of information that you need to attach to the control. In this case the tag for each column will represent the primary key of the column, and the tag of each row will represent the store id. Now we can search the rows and column directly to find the correct location of the cell to update.

Now we start the code, we will loop on every row, get the ID of the employee, from that we get the name (by searching the Emps table). Then we find which store (find the row no), and which day (find the column no), then we update the information.

```
        For I = 0 To TestDataSet.WorkSchedule.Rows.Count - 1

            ' get the values from the table
            Dim EmpID = TestDataSet.WorkSchedule.Rows(I).Item(0)
            Dim StoreID = TestDataSet.WorkSchedule.Rows(I).Item(1)
            Dim DayID = TestDataSet.WorkSchedule.Rows(I).Item(2)

            ' search the Emps table to get the name
            Dim R() As DataRow = TestDataSet.Emps.Select("empid='" & EmpID & "'")

            ' remember the name
            Dim NameValue As String = R(0).Item(1)


            Dim RowNo As Integer = -1
            Dim ColumnNo As Integer = -1
            Dim J As Integer

            ' identify which column represents the correct day
            For J = 0 To DGV.Columns.Count - 1
                If DGV.Columns(J).Tag = DayID Then
                    ColumnNo = J
                    Exit For
                End If
            Next

            ' identify which row represents the correct store
            For J = 0 To DGV.Rows.Count - 1
                If DGV.Rows(J).Tag = StoreID Then
                    RowNo = J
                    Exit For
                End If
            Next

            DGV.Rows(RowNo).Cells(ColumnNo).Value = DGV.Rows(RowNo).Cells(ColumnNo).Value
& vbNewLine & NameValue
```

```
        Next
```

The code above is not difficult to understand. The first part gets the name of the employee from the ID. The select method makes things easy. The second part is a search based on the column number to find which column displays the information for that specific day. The third part is used to find which row displays the information of that specific store. The last part is used to add the text info the cell. Run the code and you should be able to see the cells filled.

We will modify the code a little bit. We will add a collection with each cell which will store the assigned employees for that specific store on that specific day. This will make updating the grid easier. Update the code to be:

```
    Public Sub FillDGV()

        ' clear the DGV
        DGV.Rows.Clear()
        DGV.Columns.Clear()

        ' display all days as columns
        Dim I As Integer
        For I = 0 To TestDataSet.Days.Count - 1
            DGV.Columns.Add(TestDataSet.Days.Rows(I).Item(0),
TestDataSet.Days.Rows(I).Item(1))
            DGV.Columns(I).Tag = TestDataSet.Days.Rows(I).Item(0)
        Next


        ' display all stores as rows
        For I = 0 To Me.TestDataSet.Stores.Count - 1
            DGV.Rows.Add("")
            DGV.Rows(I).HeaderCell.Value = TestDataSet.Stores(I).Item(1)
            DGV.Rows(I).Tag = TestDataSet.Stores(I).Item(0)
        Next

        For I = 0 To DGV.Rows.Count - 1
            For j = 0 To DGV.Columns.Count - 1
                DGV.Rows(I).Cells(j).Tag = New Collection
            Next
        Next


        For I = 0 To TestDataSet.WorkSchedule.Rows.Count - 1

            ' get the values from the table
            Dim EmpID = TestDataSet.WorkSchedule.Rows(I).Item(0)
            Dim StoreID = TestDataSet.WorkSchedule.Rows(I).Item(1)
            Dim DayID = TestDataSet.WorkSchedule.Rows(I).Item(2)

            ' search the Emps table to get the name
            Dim R() As DataRow = TestDataSet.Emps.Select("empid='" & EmpID & "'")

            ' remember the name
            Dim NameValue As String = R(0).Item(1)


            Dim RowNo As Integer = -1
            Dim ColumnNo As Integer = -1
            Dim J As Integer

            ' identify which column represents the correct day
            For J = 0 To DGV.Columns.Count - 1
                If DGV.Columns(J).Tag = DayID Then
                    ColumnNo = J
                    Exit For
                End If
            Next
```

```
            ' identify which row represents the correct store
            For J = 0 To DGV.Rows.Count - 1
                If DGV.Rows(J).Tag = StoreID Then
                    RowNo = J
                    Exit For
                End If
            Next

            DGV.Rows(RowNo).Cells(ColumnNo).Value = DGV.Rows(RowNo).Cells(ColumnNo).Value
& vbNewLine & NameValue
            Dim C As Collection = DGV.Rows(RowNo).Cells(ColumnNo).Tag
            C.Add(EmpID)
            C.Add(NameValue)
        Next
    End Sub
```

The parts in bold are the code you should add. The collection in each cell is used to store the ID of the employee followed by the name. Next add the following sub:

```
    Public Sub RefreshCellValue(ByVal RowNo As Integer, ByVal ColumnNo As Integer)
        Dim C As Collection
        C = DGV.Rows(RowNo).Cells(ColumnNo).Tag

        Dim I As Integer
        Dim NMS As String = ""
        For I = 1 To C.Count Step 2
            NMS = NMS & C(I + 1) & vbNewLine
        Next

        DGV.Rows(RowNo).Cells(ColumnNo).Value = NMS
    End Sub
```

This subroutine will update the display of a specific cell. This makes the coding easier when you add a new employee. Just update the collection and then call this sub. All of this so far has no visual effect because we did not work on adding employees. So now lets go back to the GUI. Modify both the DGV and the listbox by changing the AllowDrop property to true for both. Next add the following code:

```
    Private Sub ListBox1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles ListBox1.MouseDown
        ListBox1.DoDragDrop(ListBox1.SelectedValue.ToString, DragDropEffects.All)
    End Sub
```

This sub tells the list box to start the DragDrop effect. The data that should be send to the DGV control should be the selectedvalue (i.e. the ID of the employee). As for the DragDropEffects.All, I honestly don't know what is that for ☹

Now the ListBox allows you to perform a drag, next the DGV should perform a drop. Add the following code:

```
    Private Sub DGV_DragOver(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles DGV.DragOver
        e.Effect = DragDropEffects.All
    End Sub
```

You need to add this code for the DragDrop effect to work correctly. Again I don't know much about the code here ☹. Sorry guys.

```
    Private Sub DGV_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles DGV.DragDrop
        Dim ID As String = e.Data.GetData(DataFormats.StringFormat)

        Dim C As Collection
        C = DGV.SelectedCells(0).Tag
        Dim I As Integer
        For I = 1 To C.Count - 1 Step 2
            If C(I) = ID Then
                Exit Sub
            End If
        Next

        For I = 0 To TestDataSet.Emps.Rows.Count - 1
            If TestDataSet.Emps.Rows(I).Item(0) = ID Then


                ' insert the row
                Try

                    WorkScheduleTableAdapter.Insert(ID,
DGV.Rows(DGV.SelectedCells(0).RowIndex).Tag,
DGV.Columns(DGV.SelectedCells(0).ColumnIndex).Tag)
                    ' add the item to the collection
                    C.Add(ID)
                    C.Add(TestDataSet.Emps.Rows(I).Item(1))

                    RefreshCellValue(DGV.SelectedCells(0).RowIndex,
DGV.SelectedCells(0).ColumnIndex)

                    WorkScheduleTableAdapter.Fill(TestDataSet.WorkSchedule)

                    Exit Sub
                Catch ex As Exception
                    MsgBox("Error")
                End Try
            End If
        Next
    End Sub
```

This is where the actual insertion happens. The code:

```
Dim ID As String = e.Data.GetData(DataFormats.StringFormat)
```

gets the value that is being dragged from the listbox (or other control).

```
Dim C As Collection
C = DGV.SelectedCells(0).Tag
```

This will gets the collection associated with the selected cell. This (as we said before) contains the IDs and the Names of the employees for this specific cell (this day+sotre).

```
        Dim I As Integer
        For I = 1 To C.Count - 1 Step 2
            If C(I) = ID Then
                Exit Sub
            End If
        Next
```

This code will check to see if the employee was already chosen on that day. If so then you should not add the employee again, this is why we have exit sub.

```
For I = 0 To TestDataSet.Emps.Rows.Count - 1
    If TestDataSet.Emps.Rows(I).Item(0) = ID Then
        …
    End If
Next
```

This code will search for all the employees to get the employee name.

```
' insert the row
Try

        WorkScheduleTableAdapter.Insert(ID, DGV.Rows(DGV.SelectedCells(0).RowIndex).Tag,
DGV.Columns(DGV.SelectedCells(0).ColumnIndex).Tag)
        ' add the item to the collection
        C.Add(ID)
        C.Add(TestDataSet.Emps.Rows(I).Item(1))

        RefreshCellValue(DGV.SelectedCells(0).RowIndex, DGV.SelectedCells(0).ColumnIndex)

        WorkScheduleTableAdapter.Fill(TestDataSet.WorkSchedule)

        Exit Sub
Catch ex As Exception
        MsgBox("Error")
End Try
```

The WorkScheduleTableAdapter.Insert method allows you to insert a row into the database. The DGV.Rows(DGV.SelectedCells(0).RowIndex).Tag gets the store ID, and the DGV.Columns(DGV.SelectedCells(0).ColumnIndex).Tag gets the Day ID. Next, the collection of the cell is updated, and the text of the cell is refreshed. Finally, the Fill method of the adapter is called to update the DataTable.

Run the code and try to drag a number of names and drop them on the grid you will see that the values appear in the grid.

To remove some employees from a number of cells, you need to remove the corresponding row from the database table. Use the `WorkScheduleTableAdapter.Delete` to remove the row, and then refresh the DGV control. You may try to do that as a homework ☺.

So this will be all for today. If you have questions or notes, send them to notes@mka-soft.com.

Thank you.

mkaatr