أبوبكر شرف الدين سويدان

برمجة قواعد البيانات Access

في

Visual Basic.net 2010

2013

نسخة إلكترونية منقحة ومزيدة

بسم اللّه الرحمن الرحيم

ولسوف يعطيك ربك فترضى

الإهداء

إلى أرواح شهداء بلادي..

جدول المحتويات

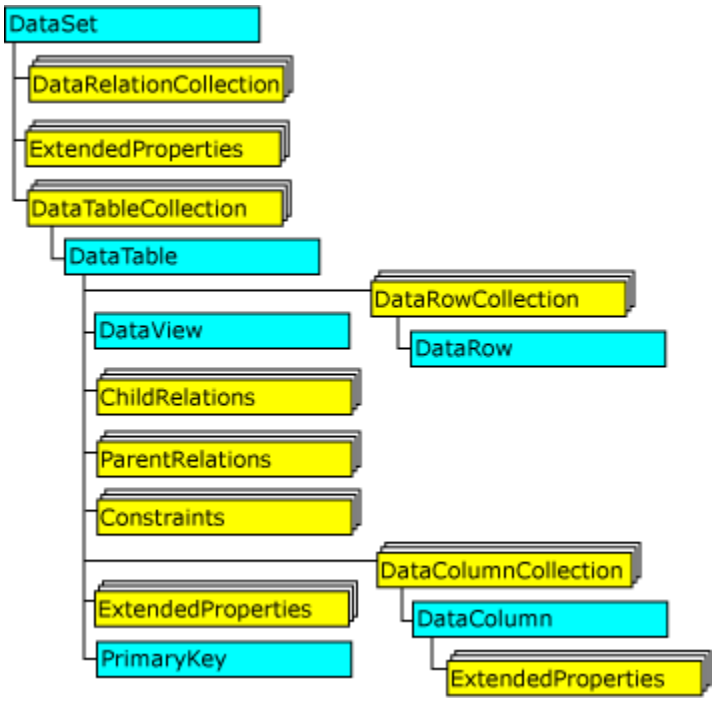
الباب الأول

DataSet

الكلاس DataSet

هي أماكن لتخزين جداول البيانات في الذاكرة، وهي مؤقتة، تفقد البيانات التي تحملها بانتهاء العملية. كل DataSet تحتوي على جدول بيانات DataTable أو أكثر وكذلك على علاقات DataRelation يمكن إنشاؤها بين الجداول، وعلى DataView. كل DataTable يحتوي على سجلات أو صفوف DataRows، وكل DataRow يحتوي على مجموعة من الأعمدة أو الحقول DataColumns. وهذا الكلاس يتبع مباشرة لفضاء الأسماء System.Data.

والشكل التالي يوضح تركيبة الـDataSet:



مصدر مخطط الـDataSet هو: http://msdn.microsoft.com/en-us/library/zb0sdh0b%28v=vs.100%29.aspx

ملاحظات:

إنشاء DataSet

نستطيع إنشاء DataSet جديدة من خلال استدعاء "منشيء الـDataSets"، أو ما يسمى بالإنجليزية DataSet Constructor، وبشكل اختياري؛ يمكننا إعطاؤها وصفاً ، وفي حال ترك هذا الخيار يتم إعطاؤها اسماً افتراضياً وهو: "NewDataSet".

```
Dim customerOrders As DataSet = New DataSet("CustomerOrders")
```

أو:

```
Dim customerOrders As New DataSet("CustomerOrders")
MsgBox(customerOrders.DataSetName, MsgBoxStyle.Information, "DataSet Name is:")
```

في الكود السابق تم إنشاء DataSet بالاسم CustomerOrders، وفي الكود التالي يتم إنشاء DataSet بالاسم الافتراضي:

```
Dim customerOrders As New DataSet
```

ولنجرب هذا الكود أيضاً:

```
Dim customerOrders As New DataSet
MsgBox(customerOrders.DataSetName, MsgBoxStyle.Information, "DataSet Name is:")
```

أهم خصائص الـDataSet:

الخاصية DataSetName: ومن خلالها يمكن ضبط / استرجاع اسم الـDataSet الحالية.

```
Dim dataSet As DataSet
dataSet = New DataSet("SuppliersProducts")
MsgBox(dataSet.DataSetName, MsgBoxStyle.Information, "DataSet Name is:")
```

الخاصية IsInitialized: وترجع قيمة منطقية تحدد ما إذا تم تهيئة الـDataSet أم لا.

```
Dim dataSet As DataSet
dataSet = New DataSet("SuppliersProducts")
MsgBox(dataSet.IsInitialized, MsgBoxStyle.Information, "IsInitialized")
```

الخاصية Tables: ومن خلالها يمكن استرجاع تجمع الـDataTables التابعة لها. أو اختيار جدول محدد عن طريق اسمه أو رقم تسلسله.

سأتطرق لها في أمثلة الفقرات التالية، ولاحقاً في باب DataTable إن شاء اللّه.

ملاحظات:

.....

.....

.....

أهم وظائف الـDataSet:

الوظيفة AcceptChanges: وتقوم بتطبيق كافة التغييرات التي طرأت على الـDataSet ومحتوياتها، منذ آخر مرة تم استدعاء هذه الوظيفة.

```
Private Sub AcceptChanges()
   Dim myDataSet As DataSet
   myDataSet = new DataSet()

   ' Not shown: methods to fill the DataSet with data.
   Dim t As DataTable

   t = myDataSet.Tables("Suppliers")

   ' Add a DataRow to a table.
   Dim myRow As DataRow
   myRow = t.NewRow()
   myRow("CompanyID") = "NWTRADECO"
   myRow("CompanyName") = "NortWest Trade Company"

   ' Add the row.
   t.Rows.Add( myRow )

   ' Calling AcceptChanges on the DataSet causes AcceptChanges to be
   ' called on all subordinate objects.
   myDataSet.AcceptChanges()
End Sub
```

الوظيفة Clear: وتقوم بمسح كافة بيانات الـDataSet وذلك من خلال مسح كافة السجلات في كافة الجداول التي تتبع هذه الـDataSet.

```
dataSet.Clear()
```

الوظيفة Clone: وتقوم بنسخ تركيبة الـDataSet بما في ذلك تركيبات الجداول والعلاقات، دون نسخ البيانات التي تحملها.

```
Dim dataSet As DataSet
dataSet = New DataSet("SuppliersProducts")
MsgBox(dataSet.DataSetName, MsgBoxStyle.Information, "DataSet Name is:")

Dim ClonedDataSet As DataSet
ClonedDataSet = dataSet.Clone()
MsgBox(ClonedDataSet.DataSetName, MsgBoxStyle.Information, "ClonedDataSet Name is:")
```

ولنجرب الكود بدون إعطاء اسم للـDataSet:

```
Dim dataSet As DataSet
dataSet = New DataSet
MsgBox(dataSet.DataSetName, MsgBoxStyle.Information, "DataSet Name is:")

Dim ClonedDataSet As DataSet
ClonedDataSet = dataSet.Clone()
MsgBox(ClonedDataSet.DataSetName, MsgBoxStyle.Information, "ClonedDataSet Name is:")
```

ملاحظات:

.....

.....

.....

الوظيفة Copy:

وتقوم بنسخ تركيبة الـDataSet بما في ذلك تركيبات الجداول والعلاقات مع كافة البيانات التي تحملها.

```
Dim dataSet As DataSet
dataSet = New DataSet("SuppliersProducts")
MsgBox(dataSet.DataSetName, MsgBoxStyle.Information, "DataSet Name is:")

Dim CopiedDataSet As DataSet
ClonedDataSet = dataSet.Copy()
MsgBox(CopiedDataSet.DataSetName, MsgBoxStyle.Information, "CopiedDataSet Name is:")
```

ولنجرب الكود بدون إعطاء اسم للـDataSet:

```
Dim dataSet As DataSet
dataSet = New DataSet
MsgBox(dataSet.DataSetName, MsgBoxStyle.Information, "DataSet Name is:")

Dim CopiedDataSet As DataSet
ClonedDataSet = dataSet.Copy()
MsgBox(CopiedDataSet.DataSetName, MsgBoxStyle.Information, "CopiedDataSet Name is:")
```

الوظيفة GetChanges: تأخذ نسخة من الـDataSet والتي تم إجراء عمليات تعديل وتغيير عليها منذ أن تم تعبئتها بالبيانات أو منذ آخر مرة تم تطبيق الوظيفة AcceptChanges، وتخزنها في DataSet جديدة. وسأتطرق للمزيد في موضوع: معالجة الحزم، لاحقاً إن شاء اللّه.

الوظيفة HasChanges: وترجع قيمة منطقية (True/False) تحدد ما إذا تم تغيير في محتويات الـDataSet، والتغييرات تشمل إضافة صفوف جديدة، أو تعديل صفوف موجودة، أو حذف لبعض أو كل الصفوف.

```
If dataSet.HasChanges = True Then
     MsgBox("There are some changes")
Else
     MsgBox("The DataSet was not changed")
End If
```

الوظيفة Merge(DataSet): وتقوم بدمج الـDataSet وتركيبتها كاملةً مع تركيبة الـDataSet الحالية. ولكي نفهمها نكتب:

```
Dim FirstDataSet As New DataSet("FirstDataSet")
Dim FirstTable As New DataTable("FirstTable")
```

وهنا قمنا بإنشاء DataSet وDataTable، الخطوة التالية هي ضم الجدول إلى FirstDataSet:

ملاحظات:

.....

.....

.....

```
FirstDataSet.Tables.Add(FirstTable)
```

والآن: FirstDataSet تحتوي على جدول واحد هو FirstTable، ويمكننا التحقق من ذلك عن طريق كتابة:

```
MsgBox(FirstDataSet.Tables(0).TableName)
```

والآن: نقوم بإنشاء DataSet جديدة باسم SecondDataSet فارغة ولا تحتوي على أي جدول:

```
Dim SecondDataSet As New DataSet("SecondDataSet")
```

ونقوم بدمج محتويات الـFirstDataSet مع محتويات الـSecondDataSet:

```
SecondDataSet.Merge(FirstDataSet)
```

وهنا تمت عملية الدمج، فعند كتابة:

```
MsgBox(SecondDataSet.Tables(0).TableName)
```

يعطينا نفس اسم الجدول الموجود في FirstDataSet، وهذا دليل على صحة العملية.

الوظيفة Merge(DataTable): سأشرحها لاحقاً عند الحديث عن الـDataTable إن شاء اللّه.

الوظيفة Merge(DataSet, Boolean): وتقوم بدمج الـDataSet وتركيبتها كاملةً مع تركيبة الـDataSet الحالية، مع قيمة منطقية تحدد ما إذا كانت عملية الدمج تشمل التغييرات أم لا.

```
SecondDataSet.Merge(FirstDataSet, False)
```

هنا تتم عملية الدمج بحيث لا تشمل التغييرات.

```
SecondDataSet.Merge(FirstDataSet, True)
```

هنا تتم عملية الدمج بحيث تشمل التغييرات.

الوظيفة RejectChanges: وتقوم بإلغاء كافة التغييرات التي طرأت على الـDataSet منذ تعبئتها بالبيانات لأول مرة، أو منذ آخر مرة تم استدعاء الطريقة AcceptChanges فيها.

سأتكلم بمزيد من التفصيل عن الوظيفتين (AcceptChanges, RejectChanges) لاحقاً في موضوع: معالجة الحزم إن شاء اللّه.

ملاحظات:

الوظيفة Reset: وتقوم بمسح محتويات الـDataSet وإرجاعها إلى حالتها السابقة.

```
Dim FirstDataSet As New DataSet("FirstDataSet")
Dim FirstTable As New DataTable("FirstTable")
FirstDataSet.Tables.Add(FirstTable)
MsgBox(FirstDataSet.Tables(0).TableName)

Dim SecondDataSet As New DataSet("SecondDataSet")
SecondDataSet.Merge(FirstDataSet)
MsgBox(SecondDataSet.Tables(0).TableName)

SecondDataSet.Reset()
MsgBox(SecondDataSet.Tables(0).TableName, , "after reset is called")
```

عند تنفيذ الكود السابق، وبعد استدعاء الوظيفة Reset يظهر خطأ يقول: Cannot find table 0 ، ويعني أن الـSecondDataset أرجعت إلى سابق عهدها فارغة.

ملاحظات:

الباب الثاني

DataTable

الكلاس DataTable

ويمثل جدولاً واحداً ضمن الـDataSet. وكل جدول لا بد أن تتوفر فيه الكلاسات: DataColumn وDataRow، فللجدول حقول (أعمدة Columns) وسجلات (صفوف Rows). وهذا الكلاس يتبع مباشرة لفضاء الأسماء System.Data.

إنشاء جدول Creating a DataTable

هذا الكلاس يتبع مباشرة لفضاء الأسماء System.Data، ولإنشاء جدول نقوم بتعريف كائن DataTable، وبشكل اختياري يمكننا منحه اسماً:

```
Dim UnNamedTable As New DataTable
```

في السطر السابق تم تعريف كائن يمثل جدولاً دون أن نسميه، وفي السطر التالي، نقوم بتعريف كائن يمثل جدولاً باسم Customers:

```
Dim Customers As New DataTable("Customers")
```

- إذا لم نحدد اسماً للجدول، يقوم ADO.NET بتسميته Table1، والجدول الذي يليه Table2 وهكذا.
- بعد إنشاء كائن الجدول، يمكننا تعديل قيمة الخاصية TableName لإعطائه اسماً أو تعديله، كما يمكننا التعامل مع بقية الخصائص والوظائف والأحداث.

إضافة DataTable إلى DatatSet

يضاف الجدول DataTable المنشأ حديثاً إلى الـDataSet بالصورة التالية:

```
Dim EmployeesDataSet As New DataSet("Employees DataSet")
Dim PersonalInfoTable As New DataTable

EmployeesDataSet.Tables.Add(PersonalInfoTable)
```

وذلك من خلال الوظيفة Add التابعة للخاصية Tables التابعة للـDataSet.

ملاحظات:

أهم خصائص DataTable:

الخاصية TableName: ومن خلالها يمكن ضبط / استرجاع اسم الـDataTable.

```
Dim FirstTable As New DataTable
MsgBox(FirstTable.TableName)
FirstTable.TableName = "Nationalities"
MsgBox(FirstTable.TableName)
```

الخاصية Columns: ومن خلالها يمكن استرجاع تجمع الأعمدة أو الحقول التابعة للـDataTable.

```
Private Sub PrintValues(ByVal table As DataTable)
    Dim row As DataRow
    Dim column As DataColumn
    For Each row in table.Rows
       For Each column In table.Columns
          Msgbox(row(column))
       Next
    Next
End Sub
```

وستتضح الصورة أكثر بعد الحديث عن كل من : DataColumn و DataRow.

الخاصية DataSet: ومن خلالها يمكن استرجاع الـDataSet التي يتبعها هذا الـDataTable.

```
Dim EmployeesDataSet As New DataSet("Employees DataSet")
Dim PersonalInfoTable As New DataTable("Personal Info")

EmployeesDataSet.Tables.Add(PersonalInfoTable)
MsgBox(PersonalInfoTable.DataSet.DataSetName)
```

الخاصية PrimaryKey: ومن خلالها يمكن ضبط / استرجاع مصفوفة من الأعمدة أو الحقول التي تعمل كمفاتيح أساسية للـDataTable.

وسأشرح عنها عند الحديث عن DataColumn.

الخاصية Rows: ومن خلالها يمكن استرجاع تجمع الصفوف التي تتبع هذا الـDataTable. وسأشرح عنها عند الحديث عن DataRow.

ملاحظات:

أهم وظائف DataTable:

الوظيفة Clear: وتقوم بمسح محتويات الجدول من البيانات فقط، ولا تحذف تركيبته.

```
PersonalInfoTable.Clear()
```

الوظيفة Clone: وتقوم بنسخ تركيبة الجدول دون البيانات.

```
Dim PersonalInfoTable As New DataTable
PersonalInfoTable.TableName = "Original Table"

Dim ClonedTable As New DataTable
ClonedTable = PersonalInfoTable.Clone()
MsgBox(ClonedTable.TableName)
```

الوظيفة Copy: وتقوم بنسخ تركيبة ومحتويات الجدول إلى جدول آخر .

```
Dim PersonalInfoTable As New DataTable
PersonalInfoTable.TableName = "Original Table"

Dim CopiedTable As New DataTable
CopiedTable = PersonalInfoTable.Copy()
MsgBox(CopiedTable.TableName)
```

الوظيفة ImportRow: وتقوم بنسخ DataRow إلى جدول DataTable مع الاحتفاظ بإعداداته السابقة والقيم التي يحملها. وسأشرح عنها عند الحديث عن DataRow.

```
Dim tblItems As New DataTable("Items")

Dim column As New DataColumn("id", GetType(System.Int32))
column.AutoIncrement = True
tblItems.Columns.Add(column)

column = New DataColumn("item", GetType(System.String))
tblItems.Columns.Add(column)

tblItems.PrimaryKey = New DataColumn() {tblItems.Columns(0)}

Dim NewRow As DataRow
NewRow = tblItems.NewRow
With NewRow
        .Item("id") = 1
        .Item("item") = "Milk"
End With

With tblItems
        .Rows.Add(NewRow)
        .AcceptChanges()
        .Rows(tblItems.Rows.Count - 1).SetAdded()
End With

Dim Newtable As New DataTable
Newtable = tblItems.Clone
With Newtable
        .ImportRow(tblItems.Rows(0))
```

ملاحظات:

```
        .AcceptChanges()
        .Rows(Newtable.Rows.Count - 1).SetAdded()
        .AcceptChanges()
End With
```

الوظيفة Merge(DataTable): وتقوم بدمج الجدول المعطى مع الجدول الحالي. سأشرحها لاحقاً في استعراض العمليات على الجداول.

```
Dim tblItems As New DataTable("Items")

Dim column As New DataColumn("id", GetType(System.Int32))
column.AutoIncrement = True
tblItems.Columns.Add(column)

column = New DataColumn("item", GetType(System.String))
tblItems.Columns.Add(column)

tblItems.PrimaryKey = New DataColumn() {tblItems.Columns(0)}

Dim NewRow As DataRow
NewRow = tblItems.NewRow
With NewRow
        .Item("id") = 1
        .Item("item") = "Milk"
End With
tblItems.Rows.Add(NewRow)

Dim Newtable As New DataTable
Newtable.Merge(tblItems)
MsgBox(Newtable.Rows(0).Item("id"))
```

الوظيفة Merge(DataTable, Boolean): وتقوم بدمج الجدول المعطى مع الجدول الحالي مع إعطاء الإمكانية لحفظ التغييرات من عدمه.

```
' Create a new DataTable.
Dim tblItems As New DataTable("Items")

' Add two columns to the table:
Dim column As New DataColumn("id", GetType(System.Int32))
column.AutoIncrement = True
tblItems.Columns.Add(column)

column = New DataColumn("item", GetType(System.String))
tblItems.Columns.Add(column)

        ' Set primary key column.
tblItems.PrimaryKey = New DataColumn() {tblItems.Columns(0)}


Dim Newtable As New DataTable("MoreFields")
column = New DataColumn("Quantity", GetType(System.Int32))
Newtable.Columns.Add(column)

column = New DataColumn("Description", GetType(System.String))
Newtable.Columns.Add(column)

tblItems.Merge(Newtable, True)
MsgBox(tblItems.Columns(2).ColumnName)
```

ملاحظات:

.....

.....

.....

الوظيفة NewRow: وتقوم بإنشاء DataRow بنفس تركيبة الجدول. بحيث تراعى أنواع البيانات وأحجامها وإعداداتها لكل عمود.

```
Dim TheNewRow As DataRow
TheNewRow = tblItems.NewRow
With TheNewRow
        .Item("id") = 1
        .Item("item") = "Milk"
End With
```

الوظيفة Reset: وتقوم بإعادة ضبط الجدول إلى حالته القديمة. وحذف كافة البيانات المخزنة فيه، بالإضافة إلى الفهارس والعلاقات والأعمدة.

```
tblItems.Reset()
```

الوظيفة Select: وتقوم باسترجاع مصفوفة تمثل كافة الـDataRows بالجدول.

```
Dim CompanyDataSet As New DataSet("Company")
Dim TheSupplierTable As DataTable = CompanyDataSet.Tables("Suppliers")
Dim SelectedRows() As DataRow = TheSupplierTable.Select()
```

ويمكن الوصول إلى كل صف في هذه المصفوفة من خلال الـIndex:

```
MsgBox(SelectedRows(0).Item("SupplierName"))
```

الوظيفة Select(String): وتقوم باسترجاع مصفوفة تمثل الـDataRows التي تخضع للمعايير المحددة.

```
Dim CompanyDataSet As New DataSet("Company")
Dim TheSupplierTable As DataTable = CompanyDataSet.Tables("Suppliers")

Dim expression As String
expression = "JoinData > #01-01-2013#"

Dim SelectedRows() As DataRow
SelectedRows = TheSupplierTable.Select(expression)

MsgBox(SelectedRows(0).Item("SupplierName"))
```

الوظيفة Select(String, String): وتقوم باسترجاع مصفوفة تمثل الـDataRows التي تخضع للمعايير المحددة بترتيب محدد.

```
Dim CompanyDataSet As New DataSet("Company")
Dim TheSupplierTable As DataTable = CompanyDataSet.Tables("Suppliers")
Dim expression As String = "JoinDate > 01/01/2013"
Dim sortOrder As String = "SupplierName DESC"
Dim SelectedRows() As DataRow

SelectedRows = TheSupplierTable.Select(expression, sortOrder)
MsgBox(SelectedRows(0).Item("SupplierName"))
```

ملاحظات:

.....

.....

.....

إضافة أعمدة (حقول) للجدول Adding Columns to DataTable

الحمد للّه، وصلنا إلى الجزء المثير والممتع! إذ أنه بعد تعريف الكائن الذي يمثل DataTable، سيكون ذلك الكائن فارغاً، وبالتالي لا نستفيد منه شيئاً! والخطوة القادمة هي بالتأكيد إنشاء الحقول أو Data Columns.

يحتوي فضاء الأسماء System.Data أيضاً على كلاس يسمى DataColumn والذي يمثل حقلاً واحداً في تركيبة الجدول Table Schema.

ولإضافة حقل إلى الجدول، نقوم بتعريف كائن يمثل الجدول:

```
Dim Customers As New DataTable("Customers")
```

ثم نعرف كائناً آخر يمثل الحقل، وأنبه هنا إلى أن الحد الأدنى لتعريف الحقل هو توفير اسم الحقل ونوع البيانات التي يحملها:

```
Dim CustomerID As New DataColumn("ID", GetType(Long))
```

ففي السطر السابق، تم تعريف الكائن CustomerID على أنه كائن يمثل حقلاً اسمه ID ونوع بياناته Long.

بقيت خطوة أخيرة، وهي ضم الكائن الجديد للجدول:

```
Customers.Columns.Add(CustomerID)
```

وفي السطر السابق، تم استخدام الوظيفة Add التابعة لتجمع الحقول Columns لإضافة الحقل الجديد إلى الجدول. ونفعل ذلك لكل عمود جديد. أو بطريقة أخرى، وهي باستخدام الوظيفة Add التابعة لتجمع الحقول Columns لإضافة حقول جديدة دون تعريف كائنات تمثل كل الحقول:

```
Dim Customers As New DataTable("Customers")
Customers.Columns.Add("ID", GetType(Long))
Customers.Columns.Add("CustomerName", GetType(String))
Customers.Columns.Add("BirthDay", GetType(Date))
```

ملاحظات:

.....

.....

.....

خصائص الكلاس DataColumn

الكود السابق يوضح طريقة إنشاء الحقول بأبسط الطرق، وبالحد الأدنى من المعلومات المتعلقة بالحقول. ولكن هناك الكثير من الخصائص الأخرى التي تجعل من إنشاء الحقول وضمها للجدول أكثر احترافية وإتقاناً، أذكر منها على سبيل المثال:

الخاصية AllowDBNull: وتحمل قيمة منطقية Boolean، وتحدد ما إذا كان من المسموح أن لا يحمل هذا الحقل أية قيمة، أي قيمته = Null.

```
Dim column As DataColumn
column = New DataColumn("ID", System.Type.GetType("System.Int32"))
column.AllowDBNull = True
```

أو

```
Dim column As DataColumn
column = New DataColumn("ID", System.Type.GetType("System.Int32"))
column.AllowDBNull = False
```

ويمكن استعمال الكلاس System.DBNull لاختبار القيمة قبل تخزينها لاحقاً في هذا الحقل (تلقائياً: كل الحقول تسمح بالقيمة Null):

```
If (DBNull.Value.Equals(FieldValue)) Then
    Do something
End If
```

وكذلك توجد دالة في فجول بيسك وظيفتها فحص قيم المتغيرات:

```
If (IsDBNull(FieldValue)) Then
    Do something
End If
```

الخصائص AutoIncrement، وAutoIncrementSeed وAutoIncrementStep: تدير هذه الخصائص عملية الزيادة التلقائية لقيم الحقول التي تحمل هذه الميزة، فعند تطبيقها، يتم زيادة القيم تلقائياً عند إضافة سجلات جديدة بمقادير تتحدد من خلال الخاصيتين AutoIncrementSeed وAutoIncrementStep. وبشكل تلقائي فإن الخاصية AutoIncrement مضبوطة على الوضع False، وبالتالي فإن هذه الآلية لا تعمل.

```
Dim column As DataColumn = New DataColumn
column.DataType = System.Type.GetType("System.Int32")
With column
     .AutoIncrement = True
     .AutoIncrementSeed = 1000
     .AutoIncrementStep = 10
End With
```

ملاحظات:

.....

.....

.....

الخاصية ColumnName: وهي خاصية نصية تمثل اسم الحقل. ويتم تحديدها وضبطها وقت إنشاء الحقل.

```
Dim column As New DataColumn
With column
        .ColumnName = "CustomerName"
        .AllowDBNull = True
End With
MsgBox(column.ColumnName)
```

الخاصية DataType: وتمثل نوع البيانات التي سيحملها الحقل. ويتم تحديدها وضبطها وقت إنشاء الحقل، وإذا ما تم إضافة البيانات، لا يمكن تعديل نوعها.

```
Dim column As New DataColumn
With column
        .ColumnName = "FirstName"
        .DataType = System.Type.GetType("System.String")
End With
MsgBox(column.DataType.ToString)
```

الخاصية DefaultValue: وتمثل القيمة الافتراضية للحقل، ويمكن لأي حقل أن يحمل قيمة افتراضية، ويتم تخزين هذه القيمة كلما تم إنشاء سجل جديد، ويمكن تعديلها لاحقاً دون أية مشكلة.

```
Dim column As New DataColumn
With column
        .ColumnName = "FirstName"
        .DataType = System.Type.GetType("System.String")
        .DefaultValue = "None"
End With
MsgBox(column.DefaultValue)
```

الخاصية MaxLength: وهي خاصة بالحقول التي تحمل قيماً نصية، وتحدد طول النص الذي سيحمله هذا الحقل، وتلقائياً: قيمته هي 1-، بمعنى أنه لا حد لطول النص.

```
Dim column As New DataColumn
With column
        .ColumnName = "FirstName"
        .DataType = System.Type.GetType("System.String")
        .MaxLength = 20
        .DefaultValue = "None"
End With
```

الخاصية ReadOnly: حقول القراءة فقط لا يمكن تعديل قيمها في أي سجل قد تمت إضافته فعلاً للجدول، وتلقائياً: جميع قيم الحقول قابلة للتعديل.

```
Dim column As New DataColumn
With column
        .ColumnName = "FirstName"
        .DataType = System.Type.GetType("System.String")
```

ملاحظات:

```
        .ReadOnly = True
        .DefaultValue = "None"
End With
```

الخاصية Unique: وهي خاصية منطقية، إذا ضبطت بالقيمة True، فإن الحقل لا يسمح بحمل قيمتين متساويتين، وكذلك في هذا النوع من الحقول لا يمكن تخزين قيمة Null.

```
Dim column As New DataColumn
With column
        .ColumnName = "ID"
        .DataType = System.Type.GetType("System.Int32")
        .Unique = True
End With
```

وهناك خاصية تتبع الكلاس DataTable وهي PrimaryKey، ومن خلالها – كما أسلفت - يمكن ضبط / استرجاع مصفوفة من الأعمدة أو الحقول والتي تعمل كمفاتيح أساسية للـDataTable:

```
Dim Customers As New DataTable("Customers")
Customers.Columns.Add("ID", GetType(Long))
Customers.Columns.Add("CustomerName", GetType(String))
Customers.Columns.Add("BirthDay", GetType(Date))

Customers.PrimaryKey = {Customer.Columns("ID")}
```

عند إضافة DataColumn للجدول DataTable، يتم إضافته إلى ما يعرف بـColumns Collection أو تجمع الأعمدة (الحقول). هذه الحقول لا تحمل أية قيم إلى حد الآن، وإنما يتم إضافة البيانات عن طريق ما يعرف بـRows Collection أو تجمع الصفوف والذي يتبع الكائن DataTable كما سنرى لاحقاً إن شاء اللّه.

إنشاء صفوف جديدة Creating New Rows

إن عملية إنشاء الجداول Creating DataTables وأعمدتها DataColumns هي عملية أساسية قبل المضي قدماً للخطوة اللاحقة وهي: عملية إضافة البيانات للجدول أو "إنشاء سجلات (صفوف Rows) جديدة".

وكما تعلمنا سابقاً، فإننا إذ ننشئ الأعمدة (الحقول DataColumns) إنما ننشئ قوالب لا نهائية من حيث العدد، يمكننا حفظ البيانات فيها بالصورة التي نريد. والآن، أود أن أحدثكم عن الكلاس DataRow..

ملاحظات:

الكلاس DataRow

ويمثل صفاً كاملاً من الـDataColumns التي يتكون منها الـDataTables. آخذاً في الاعتبار نوع وحجم كل عمود DataColumn في الجدول.

أهم خصائص الكلاس DataRow

الخاصية Item(DataColumn): ومن خلالها يتم ضبط / استرجاع القيمة المخزنة بحقل محدد.

```
Dim Customers As New DataTable("Customers")
Customers.Columns.Add("ID", GetType(Long))
Customers.Columns.Add("CustomerName", GetType(String))
Customers.Columns.Add("RegistrationDate", GetType(Date))
Customers.PrimaryKey = {Customers.Columns("ID")}

Dim NewRow As DataRow
NewRow = Customers.NewRow
With NewRow
        .Item("ID") = 1
        .Item("CustomerName") = "Abubaker Swedan"
        .Item("RegistrationDate") = Now.Date
End With

Customers.Rows.Add(NewRow)
MsgBox(Customers.Rows(0).Item("CustomerName"))
MsgBox(Customers.Rows(0).Item("RegistrationDate"))
```

الخاصية Item(Int32): ومن خلالها يتم ضبط / استرجاع القيمة المخزنة بحقل محدد بدلالة رقم فهرسه.

```
Dim Customers As New DataTable("Customers")
Customers.Columns.Add("ID", GetType(Long))
Customers.Columns.Add("CustomerName", GetType(String))
Customers.Columns.Add("RegistrationDate", GetType(Date))
Customers.PrimaryKey = {Customers.Columns("ID")}

Dim NewRow As DataRow
NewRow = Customers.NewRow
With NewRow
        .Item(0) = 1
        .Item(1) = "Abubaker Swedan"
        .Item(2) = Now.Date
End With

Customers.Rows.Add(NewRow)
MsgBox(Customers.Rows(0).Item(1))
MsgBox(Customers.Rows(0).Item(2))
```

ملاحظات:

الخاصية Item(String): ومن خلالها يتم ضبط / استرجاع القيمة المخزنة بحقل محدد بدلالة اسمه.

```
Dim Customers As New DataTable("Customers")
Customers.Columns.Add("ID", GetType(Long))
Customers.Columns.Add("CustomerName", GetType(String))
Customers.Columns.Add("RegistrationDate", GetType(Date))
Customers.PrimaryKey = {Customers.Columns("ID")}

Dim NewRow As DataRow
NewRow = Customers.NewRow
With NewRow
        .Item("ID") = 1
        .Item("CustomerName") = "Abubaker Swedan"
        .Item("RegistrationDate") = Now.Date
End With

Customers.Rows.Add(NewRow)
MsgBox(Customers.Rows(0).Item("CustomerName"))
MsgBox(Customers.Rows(0).Item("RegistrationDate"))
```

الخاصية Table: ومن خلالها يتم استرجاع اسم الجدول DataTable الذي ينتمي إليه الصف.

```
Dim Customers As New DataTable("Customers")
Customers.Columns.Add("ID", GetType(Long))
Customers.Columns.Add("CustomerName", GetType(String))
Customers.Columns.Add("RegistrationDate", GetType(Date))
Customers.PrimaryKey = {Customers.Columns("ID")}

Dim NewRow As DataRow
NewRow = Customers.NewRow
With NewRow
        .Item("ID") = 1
        .Item("CustomerName") = "Abubaker Swedan"
        .Item("RegistrationDate") = Now.Date
End With

Customers.Rows.Add(NewRow)
MsgBox(NewRow.Table.TableName)
```

ملاحظات:

أهم وأشهر وظائف DataRow:

الوظيفة Delete: وتقوم بحذف الـDataRow.

```
Dim tblItems As New DataTable("Items")

Dim column As New DataColumn("id", GetType(System.Int32))
column.AutoIncrement = True
tblItems.Columns.Add(column)
column = New DataColumn("item", GetType(System.String))
tblItems.Columns.Add(column)

tblItems.PrimaryKey = New DataColumn() {tblItems.Columns(0)}

Dim NewRow As DataRow
NewRow = tblItems.NewRow
With NewRow
        .Item("id") = 1
        .Item("item") = "Milk"
End With

With tblItems
        .Rows.Add(NewRow)
        .AcceptChanges()
        .Rows(tblItems.Rows.Count - 1).SetAdded()
End With
NewRow = tblItems.NewRow
With NewRow
        .Item("id") = 2
        .Item("item") = "Cheese"
End With

With tblItems
.Rows.Add(NewRow)
        .AcceptChanges()
        .Rows(tblItems.Rows.Count - 1).SetAdded()
End With

MsgBox(tblItems.Rows.Count)

tblItems.Rows(0).Delete()
tblItems.AcceptChanges()
MsgBox(tblItems.Rows.Count)
```

وكما تلاحظون في الكود، فإنني استعملت الوظيفة AcceptChanges لتأكيد الحذف. وسنتطرق لها في معالجة الحزم لاحقاً إن شاء اللّه.

الوظيفة IsNull(DataColumn): وتحدد ما إذا كانت القيمة التي يحملها الـDataColumn هي Null.

```
If NewRow.IsNull(ColumnObject) Then
        MsgBox("Empty Column")
Else
        MsgBox("it is not null value")
End If
```

ملاحظات:

.....

.....

.....

الوظيفة IsNull(Int32): وتحدد ما إذا كانت القيمة التي يحملها الـDataColumn (بدلالة فهرسه) هي Null.

```
If NewRow.IsNull(1) Then
        MsgBox("Empty Column")
Else
        MsgBox("it is not null value")
End If
```

الوظيفة IsNull(String): وتحدد ما إذا كانت القيمة التي يحملها الـDataColumn (بدلالة اسمه) هي Null.

```
If NewRow.IsNull("CustomerName") Then
        MsgBox("Empty Column")
Else
        MsgBox("it is not null value")
End If
```

يدعم الكلاس DataTable الطريقة NewRow لتوليد صف جديد يخضع لإعدادات الـDataColumns، من حيث نوع الحقل وطوله وخصائصه الأخرى وذلك لكل حقل من الحقول التي يحتوي عليها DataTable.

ولفتح صف (سجل DataRow) جديد في الجدول:

```
Dim TheNewRow As DataRow = TableName.NewRow()
```

بعد تنفيذ السطر البرمجي السابق، يتم وبشكل تلقائي إنشاء صف بيانات جديد بحيث يراعى فيه خصائص كل حقل، ويتم وضع القيمة Null في كافة الحقول، عدا تلك التي لها قيم افتراضية Default Values، أو التي خاصية AutoIncrement لها = True.

هناك المزيد من هذه الوظائف، سأتطرق لها لاحقاً مثل الوظيفة SetAdded وSetModified.

ملاحظات:

ضبط قيم الحقول في الـDataRow الجديد

يتضمن الكلاس DataRow خاصية مهمة وهي Item، والتي تعطي إمكانية الوصول إلى كل DataColumn معرّف في الـDataTable اعتماداً على اسم الحقل أو رقم ترتيبه (ابتداء من الصفر) أو متغير يمثل الـDataColumn، و السنّة المتبعة هي الاعتماد على اسم الحقل تجنباً لأخطاء غير مقصودة، وتوضيحاً للكود، مع الأخذ في الاعتبار خصائص كل حقل على حدة.

```
Dim TheNewRow As DataRow = TableName.NewRow()
TheNewRow.Item("ID")= 100
TheNewRow.Item(0)= 100
Dim SpecificRow As DataColumn = TableName.Columns(0)
TheNewRow.Item(SpecificRow)= 100
```

في الكود السابق تم ضبط قيمة أول حقل في الجدول (ID) من خلال اسم الحقل مرة، ورقم ترتيبه مرة ، ومن خلال مرجع يمثل الحقل المطلوب مرة ثالثة.

الخاصية Item هي الخاصية التلقائية للكلاس DataRow، ولذلك يمكن الاستغناء عن ذكرها وكتابة:

```
Dim TheNewRow As DataRow = TableName.NewRow()
TheNewRow("ID")= 100
```

كما يمكن استعمال علامة التعجب (!) لتشير إلى اسم الحقل (لا تدعم رقم فهرس الحقل) في الجدول:

```
Dim Customers As New DataTable("Customers")
Customers.Columns.Add("ID", GetType(Long))
Customers.Columns.Add("CustomerName", GetType(String))
Customers.Columns.Add("RegistrationDate", GetType(Date))
Customers.PrimaryKey = {Customers.Columns("ID")}
Dim NewRow As DataRow
NewRow = Customers.NewRow
NewRow!ID = 1
NewRow!CustomerName = "Abubaker Swedan"
NewRow!RegistrationDate = Now.Date
Customers.Rows.Add(NewRow)
```

ملاحظات:

حفظ البيانات في الجدول DataTable

بعد ضبط وتعيين القيم في السجل الجديد، تكون الخطوة اللاحقة هي ضم هذا السجل إلى الجدول DataTable، مستعملين الطريقة Rows.Add. التابعة للـDataTable:

```
Dim TheNewRow As DataRow = TableName.NewRow()

TheNewRow.Item("ID")= 100
TableName.Rows.Add(TheNewRow)
```

كما يمكن استعمال طريقة أخرى، وهي بتنفيذ الطريقة Rows.Add وإرسال قيم الحقول كبارامترات:

```
Dim TheNewRow As DataRow = TableName.NewRow()
TableName.Rows.Add(100, “Abubaker”, “Tripoli”, …)
```

وأي طريقة اخترنا، فإن الوظيفة Add تختبر القيم المرسلة من حيث توافقها مع تركيبة الجدول وحقوله، وترفض القيم التي لا تتماشى مع القواعد.

الوصول إلى الصف المطلوب في جدول DataTable

أو بصيغة أخرى، طريقة تحديد صف معين أو مجموعة من الصفوف في جدول لإجراء عمليات عليها. وتوجد عدة طرق لاسترجاع الصفوف المطلوبة منها:

أ) استرجاع الصفوف بدلالة قيم الأعمدة

وذلك باستخدام الوظيفة Select Method، التابعة للـDataTable

```
Dim foundRows() As Data.DataRow
foundRows = DataSet1.Tables("Customers").Select("LastName = 'Swedan'")
```

ففي هذا الكود تم تعريف مصفوفة لتجمع بداخلها الصفوف المسترجعة من عملية البحث باستخدام الطريقة Select، والتي مررنا لها قيمة الحقل LastName. فإن تم العثور على السجلات المطلوبة فإنها تخزن في تلك المصفوفة.

ملاحظات:

.....

.....

.....

ويمكن معرفة عدد الصفوف المسترجعة من خلال الكود التالي:

```
Dim RowsCount As Integer
RowsCount = foundRows.Count
```

ويمكن الوصول إلى حقول كل صف بالكيفية التالية:

```
Dim FirstColValue As Integer
FirstColValue = foundRows(0).Item("OrderID")
```

ب) استرجاع الصفوف بدلالة قيمة الحقل المفتاحي Primary Key Value

وفي هذه الحالة نستخدم الطريقة Find التابعة للتجمع DataRowCollection ونرسل لها القيمة المفتاحية كبارامتر:

```
Dim s As String = "primaryKeyValue"
Dim foundRow As DataRow = dataset1.Tables("Customers").Rows.Find(s)

If foundRow IsNot Nothing Then
    MsgBox(foundRow(1).ToString())
Else
    MsgBox("A row with the primary key of " & s & " could not be found")
End If
```

ملاحظات:

تعديل البيانات في الجدول DataTable

لتعديل بيانات أي صف في جدول DataTable، يلزمنا كبداية تحديد الصف المطلوب تعديل بياناته، ومن ثم إسناد القيم المناسبة لكل حقل من حقوله.

ولمعرفة الصف المطلوب تعديله، نستخدم الطريقة Select Method التابعة للـDataTable بالصورة التالية:

```
Dim customerRow() As Data.DataRow
customerRow = DataSet1.Tables("Customers").Select("CustomerID = 107")

customerRow(0)("CompanyName") = "New Company Name"
customerRow(0)("City") = "Tripoli"
```

ففي الكود السابق، الحقل CustomerID هو الحقل المفتاحي للجدول Primary Key، وبالتالي عند استخدام الطريقة Select وتمرير رقم الزبون فإن النتيجة هي إرجاع مصفوفة customerRow() تحتوي على صف واحد فقط، إذ أن رقم الزبون لا يتكرر. ثم استعملنا الرقم (0) للتعبير عن أول صف مسترجع في المصفوفة، وعدلنا قيم الحقول المقصودة.

بالإمكان تعديل الكود السابق والخاص بضبط القيم إلى:

```
customerRow(0).Item("CompanyName") = "New Company Name"
customerRow(0).Item("City") = "Tripoli"
```

ولكن كما قلتُ سابقاً أن الخاصية Item هي الخاصية التلقائية للكلاس DataRow، ولذلك يمكن الاستغناء عن ذكرها.

وفي حالة أننا نعلم رقم ترتيب الصف المطلوب تعديله، بدلالة رقم الجدول في الـDataSet نكتب:

```
DataSet1.Tables(0).Rows(4).Item(1) = "New Company Name"
DataSet1.Tables(0).Rows(4).Item(2) = "Tripoli"
```

أو بدلالة اسم الجدول:

```
DataSet1.Tables("Customers").Rows(4).Item(1) = "New Company Name"
DataSet1.Tables("Customers").Rows(4).Item(2) = "Tripoli"
```

ملاحظات:

عرض بيانات الجدول Viewing Data in a DataTable

يمكننا الوصول لبيانات الجدول من خلال استعمال التجمعات Rows وColumns الخاصة بالـDataTable. وكذلك استعمال الطريقة Select Method لاسترجاع مجموعة من السجلات التي تخضع لشروط معينة وترتيب معين وحالة صف معينة.

بالإضافة إلى استعمال الطريقة Find Method التابعة للتجمع DataRowCollection للبحث عن سجل معين عن طريق مفتاحه الأساسي.

استعمال الطريقة Select Method يرجع مجموعة من كائنات الـDataRow ، تنطبق عليها شروطنا، وتقبل بارامترات تمثل:

- صيغة الفرز (الشروط) Filter expression.
- طريقة الترتيب Sort expression.
- حالة كل صف DataViewRowState.

صيغة الفرز تحدد الصفوف المسترجعة بناء على قيم الحقول DataColumn مثلاً "Swedan" = LastName.

وطريقة الترتيب تحدد من خلال جملة Sql مثلاً ORDER BY LastName DESC.

أما حالة كل صف فسنتعرف عليها لاحقاً إن شاء اللّه في موضوع: حالة الصف Row State.

ملاحظات:

حذف البيانات من جدول Removing Data from DataTable

يمكن حذف الصفوف من الجدول DataTable باستخدام:

- الطريقة Remove التابعة للتجمع DataTable.Rows.
- الطريقة RemoveAt التابعة للتجمع DataTable.Rows.

والطريقة Remove Method تأخذ متغيراً يمثل صفاً في الجدول:

```
Dim SelectedRow As DataRow = SelectedTable.Rows(0)
SelectedTable.Rows.Remove(SelectedRow)
```

أما الطريقة RemoveAt Method فتأخذ رقم فهرس الصف مباشرة:

```
SelectedTable.Rows.RemoveAt(0)
```

وكل الطرق تؤدي إلى مكة.

ويمكن حذف جميع الصفوف بأمر واحد، وذلك باستخدام الطريقة Clear Method التابعة للكائن DataTable.Rows:

```
SelectedTable.Rows.Clear()
```

ويجب ملاحظة أنه عند حذف الصفوف بهذه الطريقة، فإنه لا يمكننا التراجع عن عملية الحذف إطلاقاً، وهذا ما سنفهمه من الموضوع التالي.

ملاحظات:

معالجة الحزم / الدفعات Batch Processing

جميع العمليات التي قمنا بها سابقاً هي عمليات مباشرة تتم على سجلات الجدول وحقوله، الأمر للوهلة الأولى جيد ورائع، ولكن لهذا النظام عيوب منها عدم القدرة على استرجاع قيم سابقة للحقول، ولا يمكن التراجع عن أي عملية والرجوع إلى الحالة السابقة.

ADO.NET يوفر مزايا جديدة بحيث يمكننا عمل تغييرات على عدة سجلات، وبعدها نقرر هل نطبق هذه التغييرات أم نتجاهلها! وهذه الطريقة تسمى: معالجة الحزم أو الدفعات.

وللاستفادة من هذه الطريقة في المعالجة، ببساطة نقوم بإحداث التغييرات التي نريد، وعندما نكون جاهزين لتطبيق هذه التغيرات:

- نستخدم الطريقة AcceptChanges Method التابعة لـDataTable ليتم تطبيق التغييرات وتحديث بيانات الجدول.
- أو نستخدم الطريقة RejectChanges Method لرفض التغييرات غير المحفوظة والحفاظ على بيانات الجدول كما هي دون تغيير.

يمكن تطبيق هاتين الطريقتين حتى على الـDataRows. فكل DataRow يدعم هاتين الطريقتين، ولكن استخدام AcceptChanges أو RejectChanges على الجدول بأكمله أفضل من استعمالهما مع كل صف على حدة، لأنه سيتم الدوران على كافة الصفوف التي حدث بها التغيير وتطبيقها (في حال استعمال AcceptChanges) أو رفض التغييرات (في حال استعمال RejectChanges).

```
SelectedTable.AcceptChanges()
SelectedTable.RejectChanges()
```

ملاحظات:

حالة الصف Row State

في أثناء إجراء التغييرات على الصف، يقوم ADO.NET بحفظ النسخة الأصلية والنسخة المعدلة لكافة الحقول التي يحتويها هذا الصف، وكذلك بمراقبة وتحديد الصفوف المضافة و/أو المحذوفة من الجدول، بحيث يمكن أن نعود للنسخة الأصلية حين الحاجة. يفعل ADO.NET كل ذلك من خلال حفظ حالة كل حقل لكل الصفوف ويتم تحديث قيمة الخاصية DataRow.RowState لكل صف على حدة، والتي تحتمل إحدى القيم التالية:

- DataRowState.Detached: وهي الحالة الافتراضية لأي صف لم يتم إضافته بعد إلى DataTable.
- DataRowState.Added: وهي حالة كل صف تم إضافته إلى DataTable ولكن لم تأكيد الإضافة. بحيث لو استعملنا RejectChanges يتم حذفها فوراً.
- DataRowState.UnChanged: وهي الحالة الافتراضية لأي صف موجود مسبقاً بالجدول ولم تتم عليه أي عملية تعديل منذ آخر مرة تم فيها استدعاء الطريقة AcceptChanges Method. وهي الحالة الافتراضية أيضاً لكافة السجلات التي تم إنشاؤها من خلال الطريقة NewRow.
- DataRowState.Deleted: الصفوف المحذوفة لا يتم إزالتها فعلياً من الجدول إلى أن يتم استدعاء الطريقة AcceptChanges، بل يتم تأشيرها على أنها ستحذف من خلال هذه الخاصية.
- DataRowState.Modified: أي صف تم تعديل حقوله بأي طريقة يؤشر على أنه Modified.

ففي كل مرة نقوم بإضافة أو تعديل صف، يتم تحديث حالته فوراً، على عكس عملية الحذف من خلال Rows.Remove و Rows.RemoveAt، فإنه يحدث التفاف حول نظام تتبع حالات السجلات هذا، ولا يتم تحديث الحالة على نفس المنوال، لأنه يتم حذف الصفوف مباشرة، ولا يجعلها تخضع لنظام معالجة الحزم.

ولحل هذه المشكلة، نستعمل الطريقة Delete Method التابعة لـDataRow عوضاً عن Rows.Remove و/أو Rows.RemoveAt، فهي لا تقوم بحذف الصفوف، بل بتغيير حالتها إلى Deleted، وبالتالي نستطيع تأكيد الحذف من خلال الطريقة AcceptChanges، أو التراجع عن الحذف من خلال RejectChanges.

```
TheRow.Delete()
```

ملاحظات:

عند استدعاء الطريقة AcceptChanges سواء من خلال DataSet أو DataTable أو DataRow فإن يتم حذف كافة السجلات التي حالتها = Deleted، وتطبيق التعديلات التي حصلت على السجلات الأخرى بحيث تحل القيم الجديدة Current Values محل القيم الأصلية Original Values.

أما عند استدعاء الطريقة RejectChanges، فإنه يتم حذف كافة السجلات التي حالتها = Added، وتعطى باقي السجلات الحالة Unchanged، وتلغى التعديلات التي حصلت على السجلات الأخرى بحيث تحل القيم الأصلية Original Values محل القيم الجديدة Current Values.

ملاحظات:

وظائف أخرى للـDataRow

هناك بعض الوظائف الإضافية التي لم أذكرها عند حديثي عن الـDataRow ، مهمة تلك الوظائف هي ضبط حالة الصف مثل:

الوظيفة SetAdded: تقوم بتغيير الـDataState للـDataRow إلى Added.

```
Dim tblItems As New DataTable("Items")

Dim column As New DataColumn("id", GetType(System.Int32))
column.AutoIncrement = True
tblItems.Columns.Add(column)
column = New DataColumn("item", GetType(System.String))
tblItems.Columns.Add(column)

tblItems.PrimaryKey = New DataColumn() {tblItems.Columns(0)}

Dim NewRow As DataRow
NewRow = tblItems.NewRow
With NewRow
        .Item("id") = 1
        .Item("item") = "Milk"
End With

With tblItems
        .Rows.Add(NewRow)
        .AcceptChanges()
        .Rows(tblItems.Rows.Count - 1).SetAdded()
End With
```

الوظيفة SetModified: تقوم بتغيير الـDataState للـDataRow إلى Modified.

```
Dim tblItems As New DataTable("Items")
Dim column As New DataColumn("id", GetType(System.Int32))
column.AutoIncrement = True
tblItems.Columns.Add(column)
column = New DataColumn("item", GetType(System.String))
tblItems.Columns.Add(column)
tblItems.PrimaryKey = New DataColumn() {tblItems.Columns(0)}

Dim NewRow As DataRow
NewRow = tblItems.NewRow
With NewRow
        .Item("id") = 1
        .Item("item") = "Milk"
End With
With tblItems
        .Rows.Add(NewRow)
        .AcceptChanges()
        .Rows(tblItems.Rows.Count - 1).SetAdded()
End With
With tblItems
        .Rows(0).Item("item") = "Cheese"
        .AcceptChanges()
        .Rows(0).SetModified()
End With
```

ملاحظات:

نُسخ الصف Row Versions

عند إجراء التعديلات والتغييرات سواء إضافة أو حذف أو تعديل في بيانات الصفوف، فإن ADO.NET يحتفظ بنسخ متعددة لكل صف حتى ولو تم تغيير قيمة حقل واحد فقط.

DataRowVersion

ويحتمل إحدى القيم التالية:

- DataRowVersion.Original: يعني الاحتفاظ بالقيم الأصلية للحقول منذ آخر استدعاء للطريقة AcceptChanges، ولا يشمل الصفوف الجديدة.
- DataRowVersion.Porposed: يعني أن قيمة حقل ما تغيرت، ولكن لم يتم تأكيد التغيير. هذه النسخة لا تتوفر حتى يتم البدء في عملية تعديل قيمة الحقل. بعد تأكيد التغييرات، تتحول القيمة إلى Original.
- DataRowVersion.Current: يعني أن التعديل جارٍ الآن، وعند تأكيده، تتحول القيمة إلى Original.
- DataRowVersion.Default: يعني النسخة التلقائية للصف. فالنسخة التلقائية لـ Added وModified وUnchanged هي Current، والنسخة التلقائية لـDeleted هي Original، والنسخة التلقائية لـDetached هي Proposed.

حيث Added وModified وUnchanged وDeleted و Detached هي حالات الصفوف.

ملاحظات:

فحص التغييرات الطارئة على محتويات جدول

قلتُ سابقاً أنه عند استعمال نظام معالجة الحزم، وفي حال حدث تغيير في الجدول، لا يتم تطبيق التعديلات إلا عند استدعاء الطريقة AcceptChanges Method. وبالتالي فالتغييرات تبقى معلقة pending حتى تطبيقها أو تجاهلها وإرجاع الجدول إلى سابق عهده (إلى حالته عند استدعاء AcceptChanges آخر مرة). وقلتُ أن عملية تتبع الصفوف تتم عن طريق:

- كل صف يحتوي على "RowState" وهي إحدى الحالات: Detached وAdded وModified وDeleted وUnchanged.
- كل صف تم تغيير محتوياته يتضمن نُسخاً مختلفة للصفوف: Original وProposed وCurrent وDefault.

معرفة ما إذا كان هناك تغيير في الصفوف

هناك طريقة Method تتبع الـDataSet هي: HasChanges، ذات قيمة منطقية، إذا ساوت True فإن هناك تغيير حدث على بعض الصفوف أو كلها. وفي تلك الحالة يمكن استدعاء الطريقة GetChanges التابعة للـDataSet و/أو للـDataTable لاسترجاع مصفوفة تحتوي على الصفوف المتغيرة، وهذه المصفوفة هي بمثابة DataSet أو DataTable مملوء بالصفوف المتغيرة فقط. ومن نافلة القول أنه يجب استدعاء الطريقة GetChanges قبل تطبيق التغييرات باستخدام الطريقة AcceptChanges، وإلا فلن نتحصل على أي صف!.

أ) استرجاع الصفوف المتغيرة من DataSet

وهنا نقوم بإنشاء DataSet وملئها بالصفوف المتغيرة:

```
Dim changedRecords As DataSet = dataset1.GetChanges()
```

ب) استرجاع الصفوف المتغيرة من DataTable

وهنا نقوم بإنشاء DataTable وملئه بالصفوف المتغيرة:

```
Dim changedRecordsTable As DataTable = Customers.GetChanges()
```

ج) استرجاع الصفوف المتغيرة بحسب نوعية التغيير (النسخة)

وهنا نرسل نوع الإصدارة إلى الطريقة GetChanges كبارامتر:

```
Dim addedRecords As DataSet = dataset1.GetChanges(DataRowState.Added)
```

ملاحظات:

العلاقات بين الجداول DataTable Relations

في الـDataSet متعددة الجداول، يمكن الربط بين جدول وأكثر، من خلال بعض الحقول المشتركة، ويمكن بعد ذلك استعراض العلاقة ومعرفة السجلات المرتبطة ببعضها البعض.

لكي نربط جدولين، نحتاج لمصفوفة من الأعمدة المشتركة بين الجدولين، واسم لهذه العلاقة.

```
Dim Customers As New DataSet("Customers")
Dim tblItems As New DataTable("Items")

' Add two columns to the table:
Dim column As New DataColumn("id", GetType(System.Int32))
column.AutoIncrement = True
tblItems.Columns.Add(column)

column = New DataColumn("item", GetType(System.String))
tblItems.Columns.Add(column)

' Set primary key column.
tblItems.PrimaryKey = New DataColumn() {tblItems.Columns(0)}

Dim NewRow As DataRow
NewRow = tblItems.NewRow

With NewRow
        .Item("id") = 1
        .Item("item") = "Milk"
End With

With tblItems
        .Rows.Add(NewRow)
        .AcceptChanges()
End With
Customers.Tables.Add(tblItems)

' -----------------
Dim tblOrders As New DataTable("Orders")
Dim Orderscolumn As New DataColumn("id", GetType(System.Int32))
tblOrders.Columns.Add(Orderscolumn)

Orderscolumn = New DataColumn("RequestedQuantity", GetType(System.Double))
tblOrders.Columns.Add(Orderscolumn)

NewRow = tblOrders.NewRow
With NewRow
        .Item("id") = 1
        .Item("RequestedQuantity") = 200
End With

With tblOrders
        .Rows.Add(NewRow)
        .AcceptChanges()
End With
Customers.Tables.Add(tblOrders)
Dim CustOrders As DataRelation = Customers.Relations.Add("CustOrders", _
Customers.Tables("Items").Columns("id"), Customers.Tables("Orders").Columns("id"))
```

ملاحظات:

.....

.....

.....

ففي الكود السابق تم تعريف DataSet، وتم تعريف جدول باسم tblItems وتم إضافة بعض السجلات له، وتم إضافة الجدول tblOrders وإضافة سجل واحد.

وتم إنشاء العلاقة بينهما من خلال الربط بين الحقلين id في الجدولين من خلال الكود:

```
Customers.Relations.Add("CustOrders", Customers.Tables("Items").Columns("id"), _
Customers.Tables("Orders").Columns("id"))
```

وبالتالي نتج عن تنفيذ الكود علاقة باسم CustOrders، تحتفظ بداخلها كافة السجلات المرتبطة ببعضها. وبصيغة أخرى، فإن كل سجل في الجدول tblItems له ما يقابله من (أبناء) مرتبطين به في جدول tblOrders.

والممتع في الأمر، أنه لو حذفنا السجل المشترك من الجدول الأول، سيتم حذفه أيضاً من الجدول المرتبط به!

```
Customers.Tables("Items").Rows(0).Delete()
Customers. AcceptChanges()
MsgBox(tblOrders.Rows.Count)
```

ويمكن التحقق من ذلك بوضع DataGridView على الفورم (باسم dgvCustomers) وتجربة تنفيذ السطر التالي:

```
dgvCustomers.DataSource = tblOrders
```

ولكي نفهم المزيد، نقوم بإنشاء مشروع جديد في فجول بيسك 2010، نضع على الـForm عدد 2 من DataGridView، ونسميهما: dgvItems وdgvOrders .

نستدعي فضاء الأسماء System.Data:

```
Imports System.Data
```

في قسم التعريفات نكتب:

```
Dim Customers As New DataSet("Customers")
Dim tblItems As New DataTable("Items")
Dim tblOrders As New DataTable("Orders")
Dim CustOrders As DataRelation
```

ونقوم بكتابة روتين لإنشاء أعمدة الجدول tblItems كما يلي:

ملاحظات:

.....

.....

.....

```
Private Sub Create_tblItems()
Dim column As New DataColumn("id", GetType(System.Int32))
column.AutoIncrement = True
tblItems.Columns.Add(column)

column = New DataColumn("item", GetType(System.String))
tblItems.Columns.Add(column)

tblItems.PrimaryKey = New DataColumn() {tblItems.Columns(0)}
```

في السطور السابقة قمنا بتعريف الأعمدة، والآن نقوم بإنشاء صفوف جديدة وضبط قيم أعمدتها:

```
Dim NewRow As DataRow
NewRow = tblItems.NewRow
With NewRow
        .Item("id") = 1
        .Item("item") = "Milk"
End With
With tblItems
        .Rows.Add(NewRow)
        .AcceptChanges()
End With

NewRow = tblItems.NewRow
With NewRow
        .Item("id") = 2
        .Item("item") = "Cheese"
End With
With tblItems
        .Rows.Add(NewRow)
        .AcceptChanges()
End With

NewRow = tblItems.NewRow
With NewRow
        .Item("id") = 3
        .Item("item") = "Butter"
End With

With tblItems
        .Rows.Add(NewRow)
        .AcceptChanges()
End With
```

والآن نضيف الجدول tblItems إلى الـDataSet المسماة Customers:

```
Customers.Tables.Add(tblItems)
End Sub
```

وبنفس الطريقة نكتب روتين لإنشاء أعمدة الجدول tblOrders وصفوفه:

```
Private Sub Create_tblOrders()
Dim Orderscolumn As New DataColumn("id", GetType(System.Int32))
tblOrders.Columns.Add(Orderscolumn)

Orderscolumn = New DataColumn("RequestedQuantity", GetType(System.Double))
tblOrders.Columns.Add(Orderscolumn)

Dim NewRow As DataRow
```

ملاحظات:

```
NewRow = tblOrders.NewRow
With NewRow
        .Item("id") = 1
        .Item("RequestedQuantity") = 200
End With
With tblOrders
        .Rows.Add(NewRow)
        .AcceptChanges()
End With

NewRow = tblOrders.NewRow
With NewRow
        .Item("id") = 2
        .Item("RequestedQuantity") = 150
End With
With tblOrders
        .Rows.Add(NewRow)
        .AcceptChanges()
End With

NewRow = tblOrders.NewRow
With NewRow
        .Item("id") = 2
        .Item("RequestedQuantity") = 75
End With

With tblOrders
        .Rows.Add(NewRow)
        .AcceptChanges()
End With
Customers.Tables.Add(tblOrders)
End Sub
```

وفي حدث التحمل لـForm نكتب:

```
Create_tblItems()
Create_tblOrders()

CustOrders = Customers.Relations.Add("CustOrders", Customers.Tables("Items").Columns("id"),
Customers.Tables("Orders").Columns("id"))

dgvItems.DataSource = tblItems
```

وكما نلاحظ، في السطرين الأول والثاني تم استدعاء روتينات إنشاء أعمدة الجداول وصفوفها، ثم إنشاء علاقة تربط بين الجدولين وإضافتها (أي العلاقة) إلى الـDataSet.

وفي السطر الأخير تم اعتماد الجدول tblItems كمصدر لبيانات الـDataGridView المسماة dgvItems.

ملاحظات:

.....

.....

.....

والآن، ننقر الـDataGridView المسماة dgvItems نقرتين ونختار الحدث DoubleClick ونكتب فيه:

```
Dim selectedItemOrders As New DataTable
Dim Orderscolumn As New DataColumn("id", GetType(System.Int32))
selectedItemOrders.Columns.Add(Orderscolumn)

Orderscolumn = New DataColumn("RequestedQuantity", GetType(System.Double))
selectedItemOrders.Columns.Add(Orderscolumn)

' ------------------------------------------------------
Dim ReturnedRows() As Data.DataRow
Dim Search As Integer = CInt(dgvItems.SelectedRows(0).Cells(0).Value)
Dim ReturnedRowsCount As Integer
ReturnedRows = Customers.Relations("CustOrders").ChildTable.Select("id=" & Search & "")

ReturnedRowsCount = ReturnedRows.Length
If ReturnedRowsCount > 0 Then
        For i As Integer = 0 To ReturnedRows.Length - 1
                Dim NewRowInResult As DataRow
                NewRowInResult = selectedItemOrders.NewRow
                With NewRowInResult
                        .Item(0) = ReturnedRows(i).Item(0)
                        .Item(1) = ReturnedRows(i).Item(1)
                End With
                selectedItemOrders.Rows.Add(NewRowInResult)
                selectedItemOrders.AcceptChanges()
        Next
End If
dgvOrders.DataSource = selectedItemOrders
```

فكرة التطبيق أنه يتم عرض الأصناف في الـDataGridView الأولى، وعند النقر المزدوج على أحدها، يتم عرض الطلبيات المقابلة له في الـDataGridView الثانية.

في الكود السابق، نقوم بتعريف جدول جديد ليحمل كل السجلات المرتبطة بالسجل المختار، فنقوم بإنشاء الأعمدة التي يتكون منها الجدول وهي: id و RequestedQuantit .

```
Dim selectedItemOrders As New DataTable
Dim Orderscolumn As New DataColumn("id", GetType(System.Int32))
selectedItemOrders.Columns.Add(Orderscolumn)

Orderscolumn = New DataColumn("RequestedQuantity", GetType(System.Double))
selectedItemOrders.Columns.Add(Orderscolumn)
```

عند اختيار صنف، نتوقع أن تقابله مجموعة من الصفوف في جدول الطلبيات، ولذلك عرّفنا مصفوفة باسم ReturnedRows تقوم بحفظ هذه الصفوف المسترجعة.

```
Dim ReturnedRows() As Data.DataRow
```

ملاحظات:

.....

.....

.....

وعملية اختيار الصفوف تتم عن طريق رقم الـid للصنف المختار، ولذلك كتبنا السطر:

```
Dim Search As Integer = CInt(dgvItems.SelectedRows(0).Cells(0).Value)
```

ولكي نضيف الصفوف المسترجعة للجدول selectedItemOrders نحتاج إلى معرفة عددها:

```
Dim ReturnedRowsCount As Integer
```

وللبدء في العمل، نقوم بفلترة الصفوف المطلوبة:

```
ReturnedRows = Customers.Relations("CustOrders").ChildTable.Select("id=" & Search & "")
```

وإرجاع عدد السجلات المسترجعة:

```
ReturnedRowsCount = ReturnedRows.Length
```

فإذا تم استرجاع سجلات، يعني عددها أكبر من صفر:

```
If ReturnedRowsCount > 0 Then
        For i As Integer = 0 To ReturnedRows.Length - 1
                Dim NewRowInResult As DataRow
                NewRowInResult = selectedItemOrders.NewRow
                With NewRowInResult
                        .Item(0) = ReturnedRows(i).Item(0)
                        .Item(1) = ReturnedRows(i).Item(1)
                End With
                selectedItemOrders.Rows.Add(NewRowInResult)
                selectedItemOrders.AcceptChanges()
        Next
End If
```

نقوم بالدوران في مصفوفة السجلات المسترجعة وإضافة كل سجل إلى الجدول selectedItemOrders.

وأخيراً، بعد انتهاء عملية تخزين الصفوف، نجعل الـselectedItemOrders مصدراً لبيانات الـDataGridView المسماة dgvOrders.

ملاحظات:

.....

.....

.....

الباب الثالث

العمليات على قواعد البيانات

العمليات على قواعد البيانات

في البابين السابقين، تعلمنا بعض الأمور المتعلقة بالـDataSet وجداولها، وفي هذا الباب، سنتقدم بعض الخطوات، وسنتعلم كيفية التعامل مع قواعد البيانات من حيث العمليات الأساسية:

- الإضافة.
- البحث.
- العرض.
- التعديل.
- الحذف.

سنتحدث عن بعض ما يتعلق بتقنية ADO.NET، ثم نعرض مثالاً متكاملاً للتعامل مع قواعد البيانات أكسس من مايكروسوفت. وعند إتقانك للتعامل مع قواعد بيانات أكسس من خلال تطبيق فجول بيسك، فإنك ستكون قادراً بإذن اللّه على التعامل مع الأنواع الأخرى من قواعد البيانات بيسر وسهولة، فالفكرة واحدة، والكود لا يكاد يختلف إلا في تسميات الكلاسات فقط.

ADO.NET

هي عائلة من التقنيات التي تسمح لنا بالتعامل والتفاعل مع البيانات بطريقة معيارية ومهيكلة في وضع منفصل، بحيث نستطيع التعامل مع البيانات الداخلية (بيانات أنشئت وخزنت في الذاكرة المؤقتة كما في البابين السابقين)، والخارجية (المخزنة في وسائط تخزين خارج التطبيقات) كقواعد البيانات (وهو موضوع هذا الباب) والملفات النصية وغيرها.

يمكننا من خلال ADO.NET الوصول إلى البيانات العلائقية مثل قواعد البيانات Microsoft Access أو قواعد بيانات SQL SERVER بالإضافة إلى قواعد بيانات أخرى. و يمكننا استخدام ADO.NET للوصول إلى مصادر البيانات غير العلائقية أيضاً. تتواجد هذه الكلاسات في ملف system.data.dll الذي يتضمن أيضاً فضاء الأسماء system.data وغيره.

ملاحظات:

مكونات ADO.NET

يتكون ADO.NET من مكونين أساسيين هما:

Data providers	DataSet
Connection	DataTable
Command	DataRelation
DataAdapter	
DataReader	

مزوّدات البيانات Data Providers

وظيفة مزوّدات البيانات هي الاتصال بقاعدة البيانات وربطها بالتطبيق، ثم إجراء العمليات المختلفة عليها، واسترجاع وعرض البيانات، وغير ذلك من العمليات. النتائج المترتبة عن عمليات البحث يتم تخزينها مؤقتاً في DataSet.

توجد عدة أنواع من المزوّدات منها:

المزود	الوظيفة
.NET Framework Data Provider for SQL Server	وتوفر اتصالاً بقواعد بيانات سيكول سيرفر من مايكروسوفت بدايةً من الإصدار 7 وما يليه، ويستعمل الفضاء System.Data.SqlClient.
.NET Framework Data Provider for OLE DB	وتوفر اتصالاً بقواعد البيانات مثل أكسس من مايكروسوفت، ويستعمل الفضاء System.Data.OleDb.
.NET Framework Data Provider for ODBC	وتوفر اتصالاً بقواعد البيانات عن طريق ODBC، ويستعمل الفضاء System.Data.Odbc

وغير ذلك من مزوّدات البيانات.

ومما سبق، فإن كل مزوّد من مزوّدات البيانات يحتوي على مجموعة من الكائنات هي التي تسهل الوصول إلى مصادر البيانات، وإجراء العمليات عليها، وغير ذلك.

ومن هذه الكائنات:

ملاحظات:

.....

.....

.....

الكائن	الوظيفة
Connection	ينشئ اتصالاً بمصدر بيانات محدد.
Command	ينفذ أوامر على مصادر البيانات.
DataReader	يستخدم لقراءة البيانات من مصادر البيانات فقط، ولا يستخدم للكتابة.
DataAdapter	يستخدم لتعبئة الـDataSet بالبيانات، وتحديث مصادر البيانات.
CommandBuilder	هو أداة مساعدة للكائن DataAdapter، ويستخدم في عملية تحديث مصدر البيانات.

.NET Framework Data Provider for OLE DB

نستخدمه عند الاتصال بقواعد بيانات أكسس من مايكروسوفت، سواء ذوات اللاحقة MDB.، أو اللاحقة ACCDB.، ويمكننا التفريق بينهما من خلال مزوّد البيانات.

- فالمزوّد: Microsoft.Jet.OLEDB.4.0 خاص بقواعد البيانات أكسس ذات اللاحقة MDB.
- والمزوّد: Microsoft.Jet.OLEDB.12.0 خاص بقواعد البيانات أكسس ذات اللاحقة ACCDB.

وقبل البدء في استعمال ما سبق، يجب استدعاء فضاء الأسماء المناسب لمزوّد البيانات لديك، وفي حالتنا هذه، فإن فضاء الأسماء الذي سنستدعيه هو:

System.Data.OleDb

وطريقة استدعائه كالتالي: في أول سطر في شاشة الكود نكتب:

```
Imports System.Data.OleDb
```

ومن الآن وصاعداً، يمكننا التعامل مع مزوّد البيانات Microsoft.Jet.OLEDB.

دعونا الآن نتطرق لبعض كائنات المزوّد Microsoft.Jet.OLEDB.

ملاحظات:

أولاً) كائن الاتصال Connection

وظيفة هذا الكائن هي الاتصال بمصدر البيانات. ولكي تتم هذه العملية، يجب تعريف جملة الاتصال Connection string.

جملة الاتصال Connection String

هناك العديد من الخصائص Properties، والوظائف Methods التابعة لكائن الاتصال، وأحدها هو خاصية جملة الاتصال Connection String، والتي تأخذ العديد من البارامترات Parameters أهمها: المزود Provider، ومصدر البيانات Data Source. المزوّد يمكن أن يكون أحد اثنين (في حالتنا) كما عرفنا.

أما مصدر البيانات Data Source، فهو مسار واسم ملف قاعدة البيانات، لنفرض أن اسم ملف قاعدة البيانات هو students.mdb، وموجود داخل المجلد students على السواقة C، فيكون مصدر البيانات بالشكل التالي:

```
Data Source=C:/students/students.mdb
```

في حالة كان ملف قاعدة البيانات هو students.accdb:

```
Data Source=C:/students/students.accdb
```

وبجمعهما معاً، ينتج لدينا جملة الاتصال، والتي ستكون كما يلي في حالة المزوّد Microsoft.Jet.OLEDB.4.0.:

```
"Provider=Microsoft.Jet.OLEDB.4.0; Data Source= C:/students/students.mdb"
```

وفي حالة المزوّد Microsoft.Jet.OLEDB.12.0:

```
"Provider=Microsoft.Jet.OLEDB.12.0; Data Source= C:/students/students.accdb"
```

وللمزيد عن جمل الاتصال، يمكنكم زيارة الموقع التالي: connectionstrings.com

ملاحظات:

تعريف كائن الاتصال OleDbConnection

عرفنا في ما سبق جملة الاتصال، والآن نأتي لتعريف كائن الاتصال في التطبيق.

في المكان المناسب نكتب في حالة المزوّد Microsoft.Jet.OLEDB.4.0:

```
Dim Con As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;  Data Source= C:/students/students.mdb")
```

و في حالة المزوّد Microsoft.Jet.OLEDB.12.0:

```
Dim Con As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.12.0; Data Source=
C:/students/students.accdb")
```

ويمكن التعديل على التعريف السابق، في حال أن ملف قاعدة البيانات موجود في نفس مجلد التطبيق، كما يلي:

```
Dim Con As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" & Application.StartupPath
& "\ students.mdb")
```

في حالة المزوّد Microsoft.Jet.OLEDB.4.0.

أو

```
Dim Con As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.12.0; Data Source=" & Application.StartupPath
& "\ students.accdb")
```

في حالة المزوّد Microsoft.Jet.OLEDB.12.0.

فتح الاتصال بقاعدة البيانات

الآن، قمنا بتعريف كائن الاتصال، وضبط جملة الاتصال، فبأمر واحد يمكننا فتح الاتصال بقاعدة البيانات باستخدام الوظيفة Open التابعة لكائن الاتصال بالشكل التالي:

```
Con.Open()
```

إغلاق الاتصال بقاعدة البيانات

بعد نجاح الاتصال بقاعدة البيانات، نستطيع استرجاع ما نريده منها، وبعد القيام بالعمليات المطلوبة على قاعدة البيانات، يجب إغلاق الاتصال بالشكل التالي:

```
Con.Close()
```

ملاحظات:

.....

.....

.....

حالة الاتصال Connection State

وفي بعض الأحيان، وخاصة عندما يكون التطبيق كبيراً، وبه الكثير من العمليات على قواعد البيانات، يفترض بنا التحقق من حالة الاتصال، هل الاتصال مفتوح أم مغلق؟ حتى لا نقع في مشاكل نحن في غنىً عنها.

وبالتالي قبل فتح الاتصال نكتب جملة التحقق من حالة الاتصال كما يلي:

```
If Con.State = ConnectionState.Closed Then
        Con.Open()
        Do something
        Con.Close()
Else
        Do something
        Con.Close()
End if
```

يعني، سنقوم بعمل ما على قواعد البيانات، فنفحص الاتصال هل هو مغلق؟ في حالة نعم يتم فتح الاتصال والقيام بالعمل وغلق الاتصال. وإلا، أي الاتصال مفتوح، فنقوم بنفس العمل ثم نغلق الاتصال.

ملاحظات:

ثانياً) الكائن OleDbDataAdapter

وهو الوسيط بين مصدر البيانات والـDataSet. ولكي يعمل الـDataAdapter يجب توفير اتصال بمصدر البيانات Connection، وجملة Sql.

```
Dim MyDataAdapter As New OleDb.OleDbDataAdapter(SQLstatemant, Connection)
```

وعندئذ يمكن تعبئة جداول الـDataSet من خلال الوظيفة Fill التابعة له.

أهم وظائف الـDataAdapter

الوظيفة Fill: ويمكن من خلالها تعبئة جدول في الـDataSet بالبيانات المسترجعة من الـDataAdapter.

```
MyDataAdapter.Fill(MyDataSet, "TableName")
```

الوظيفة Update: ويمكن من خلالها تحديث الـDataSet بعد إجراء العمليات عليها مثل Insert وUpdate وDelete.

```
MyDataAdapter.Update(MyDataSet)
```

ملاحظات:

ثالثاً) الكائن OleDbCommand

وظيفته تنفيذ الأوامر على مصدر البيانات. ويتم تعريفه في التطبيق كالتالي:

```
Dim command As New OleDbCommand()
```

أهم خصائص الـCommand

الخاصية Connection: ويمكن من خلالها ضبط أو استرجاع المتغير الذي يمثل كائن الاتصال بمصدر البيانات.

```
Dim command As New OleDbCommand()
command.Connection = Con
```

حيث Con متغير يمثل كائن الاتصال.

الخاصية CommandText: وهي خاصية نصية يمكن من خلالها ضبط أو استرجاع جملة الـSQL التي سينفذها الكائن Command.

```
Dim command As New OleDbCommand()
command.CommandText = "SELECT * FROM Categories ORDER BY CategoryID"
```

الخاصية CommandType: ويمكن من خلالها ضبط أو استرجاع قيمة تحدد كيفية تفسير الخاصية CommandText. وتحتمل عدة صيغ:

- إذا تم اختيار الصيغة CommandType.StoredProcedure: فيجب أن تكون قيمة الخاصية CommandType مساوية لاسم الـStoredProcedure.
- وإذا تم اختيار الصيغة CommandType.TableDirect: فيجب أن تكون قيمة الخاصية CommandType مساوية لاسم الجدول.
- وإذا تم اختيار الصيغة CommandType.Text: فيجب أن تكون قيمة الخاصية CommandType عبارة عن جملة SQL.

ملاحظات:

أهم وظائف الـCommand

الوظيفة ExecuteNonQuery: وتقوم بتنفيذ استعلام الـSQL المعطى على مصدر البيانات، وإرجاع عدد الصفوف المتأثرة بالعملية. ومن مهامها على سبيل المثال: إنشاء قواعد البيانات والجداول ، أو إضافة وتعديل وحذف البيانات دون الحاجة للتعامل مع الـDataSet عن طريق استعمال الجمل Insert into وUpdate وDelete وغيرها.

```
Dim command As New OleDbCommand()
command.CommandText = "SELECT * FROM Categories ORDER BY CategoryID"
command.ExecuteNonQuery()
```

الوظيفة Dispose: وتستخدم عند الانتهاء من العمليات المجراة بواسطة الـCommand، وتقوم بتحرير المصادر المستعملة من قبل هذا الكائن.

```
Dim command As New OleDbCommand()
command.Dispose()
```

ملاحظات:

رابعاً) الكائن OleDbDataReader

وظيفة هذا الكائن هي القراءة فقط من مصدر البيانات. يعتمد الكائن DataReader في عمله على الكائن Command، ويكون تعريفه في التطبيق بالصورة التالية:

```
Dim command As New OleDbCommand()
With command
        .Connection = Con
        .CommandType = CommandType.Text
        .CommandText = "SELECT * FROM Categories ORDER BY CategoryID"
End With
Dim TheDataReader As OleDbDataReader = command.ExecuteReader
```

أهم خصائص الـDataReader

الخاصية FieldCount: ويمكن من خلالها استرجاع عدد الأعمدة في الصف الحالي.

```
MsgBox(TheDataReader.FieldCount)
```

الخاصية HasRows: وترجع قيمة تتحقق من احتواء الـDataReader على صفوف أم لا.

```
MsgBox(TheDataReader.HasRows)
```

الخاصية IsClosed: وتحدد هل تم إغلاق الـDataReader أم لا.

```
MsgBox(TheDataReader.IsClosed)
```

الخاصية RecordsAffected: وتعطي عدد السجلات المضافة أو المعدلة أو المحذوفة.

```
MsgBox(TheDataReader.RecordsAffected)
```

ملاحظات:

أهم وظائف الـDataReader:

الوظيفة Read: وتقوم بحث الـDataReader على قراءة السجل التالي.

```
TheDataReader.Read()
```

الوظيفة Close: وتقوم بإغلاق الـDataReader.

```
TheDataReader.Close()
```

الوظيفة Dispose: وتستخدم لتحرير المصادر المستعملة من قبل الـDataReader.

```
TheDataReader.Dispose()
```

تنبيه

- لإنشاء DataReader يجب استدعاء الوظيفة ExecuteReader التابعة للكائن Command.
- قبل إغلاق الاتصال يجب إغلاق الـDataReader أولاً.
- يجب إغلاق الـDataReader إذا كنت تخطط لإعادة استعمال الـCommand.
- يمكن استدعاء الخاصيتين IsClosed وRecordsAffected بعد إغلاق الـDataReader.
- بالرغم من أن الخاصية RecordsAffected تعمل وقت عمل الـDataReader، إلا أنه ينصح بإغلاق الـDataReader قبل استعمالها للحصول على نتيجة صحيحة.

ملاحظات:

استرجاع البيانات باستخدام DataReader

نستطيع استرجاع بيانات للقراءة فقط من خلال DataReader ، البيانات المسترجعة ستكون مخزنة في ذاكرة احتياطية إلى حين طلب عرضها من خلال الوظيفة Read التابعة لهذا الكائن.

ميزة استعمال الـDataReader هي التخفيف من استعمال المصادر لأنه يتم قراءة صف واحد من البيانات في كل مرة يتم فيها استدعاء الوظيفة Read. وبالتالي جعْل التطبيق أكثر سرعة.

قلت سابقاً أنه نستطيع استخدام الوظيفة Read لقراءة صف من نتيجة استعلام. وبالتالي نستطيع الوصول إلى قيمة كل حقل من خلال تمرير اسمه أو رقم فهرسه.

DataSet و DataAdapter

عرفنا فيما سبق كيفية تعريف كائن الاتصال بقاعدة البيانات، وكيفية ضبط وتحديد جملة الاتصال، وتمكنا من فتح الاتصال وإغلاقه، ولكن هذا ليس نهاية المطاف، فقط هو مجرد البداية!.. فالبيانات المخزنة في قاعدة البيانات تحتاج إلى أن نأخذها من قاعدة البيانات، ثم نضعها في مكان ما، ومن ثم نتعامل معها.

تقنية ADO.NET تستخدم DataSet كذاكرة مؤقتة تحمل نسخاً من البيانات، وهي ليست مرئية نضعها على الـForm ونراها، بل هي أشياء مخفية تؤدي وظيفتها والسلام. وقد قلت في بداية الكتاب أنها أماكن لتخزين جداول البيانات في الذاكرة، وهي مؤقتة، تفقد البيانات التي تحملها بانتهاء العملية. كل DataSet تحتوي على جدول بيانات DataTable أو أكثر، وكل DataTable يحتوي على سجلات أو صفوف Rows، وكل Row يحتوي على مجموعة من الأعمدة أو الحقول Columns. هذه هي الـDataSet، ولكن.. ماذا عن الـDataAdapter؟ أقول لكم: كائن الاتصال Connection، والكائن DataSet لا يمكنهما رؤية بعضهما البعض، ولا التواصل مع بعضهما البعض إلا في وجود وسيط، هذا الوسيط هو DataAdapter. فيقوم الـDataAdapter بالاتصال بكائن الاتصال Connection، وينفذ استعلاماً نحن نحدده SQL Statement، ونتيجة الاستعلام تُخزّن في جدول أو أكثر في الـDataSet.

وبذا، فإنه لكي يعمل الـDataAdapter يجب أن يتوفر اتصال بقاعدة البيانات Connection، وجملة استعلام Sql Statement.

ملاحظات:

Structured Query Language

وتدعى اختصاراً SQL وتُنطق: سيكويل.

ومهمتها الأساسية هي الاستفسار عن وجود سجلات تطابق ما نبحث عنه، وكذلك القيام بعمليات أخرى كإضافة البيانات وتعديلها وحذفها وغير ذلك. ليس على قواعد بيانات ACCESS فقط، بل تعمل على أكثر أنواع قواعد البيانات المعروفة الآن. هذه اللغة سهلة جداً، وممتعة في آن! نستطيع أن نشكّلها ونطوّعها إلى ما نريد.

أعطيكم مثالاً: لو أردنا جلب كافة محتويات جدول معين نكتب:

```
SELECT * FROM TableName
```

هذه اللغة ليست حساسة لحالة الأحرف، فيمكننا كتابة أوامرها بالأحرف الصغيرة والكبيرة، لا مشكلة على الإطلاق. ولكن المبرمجين تعودوا على كتابة الكلمات المفتاحية Key words بالحروف الكبيرة لتسهيل عملية قراءة الكود لا أكثر.

النجمة (*) في الجملة السابقة = جميع السجلات، وبذلك فإن الترجمة العربية للسطر البرمجي السابق هي: من فضلك: أريد جلب جميع السجلات المخزنة في الجدول المسمى TableName. سهلة.. أليس كذلك؟

ولكننا لا نريد استجلاب جميع الحقول في كل وقت، فربما نريد استجلاب بعض الحقول وترك البعض الآخر، فمثلاً نريد استجلاب الاسم، وتاريخ الميلاد، والعنوان من الجدول people، وترك بقية البيانات، فيلزمنا تعديل الجملة السابقة لتكون بالشكل التالي:

```
SELECT PName, PBirthDate, PAddress FROM people
```

فعند تنفيذ الاستعلام، نتحصل على قائمة تحمل الأسماء وتواريخ الميلاد والعناوين لكل المسجلين في هذا الجدول دون استثناء. ولكن ماذا لو أردنا نفس هذه المعلومات، ولكن للأشخاص الذين يقيمون في مدينة طرابلس فقط؟!؟ نعدل الجملة السابقة لتكون بالشكل التالي:

```
SELECT PName, PBirthDate, PAddress FROM people WHERE PCity = “Tripoli”
```

فاستخدمنا الكلمة المفتاحية where والتي تأخذ الشرط في الحسبان، وبإمكاننا وضع ما نريد من الشروط ونفصل بينها بكلمات مفتاحية مثل AND وOr على حسب حاجتنا من الاستعلام.

مازال الكثير من أوامر لغة SQL، ولكنني سأكتفي بهذا القدر، فما يهمنا سأتكلم عنه بعد قليل.

ملاحظات:

تعريف الـDataSet و DataAdapter في التطبيق

والآن، لتعريف الـ DataSet نكتب:

```
Dim MyDataSet as New DataSet
```

ولتعريف الـDataAdapter، نعرف أولاً جملة الاستعلام التي يحتاجها:

```
Dim SQLstatemant As String = "SELECT * FROM TableName"
```

ويأتي دور الـDataAdapter الآن، فنكتب:

```
Dim MyDataAdapter As New OleDb.OleDbDataAdapter(SQLstatemant, Con)
```

وكما نرى في الكود الأخير: تم تعريف الـDataAdapter وتم تمرير البارامترات التي تمثل كائن الاتصال وجملة الاستعلام إليه.

تعبئة الـDataSet بالبيانات

الآن، الـDataAdapter يحمل نتيجة تنفيذ الاستعلام، ولكننا لا نراها، وبالتالي يأتي دور الـDataSet لاستلامها من الـDataAdapter من خلال الوظيفة Fill التابعة للـDataAdapter والتي تحتاج إلى بارامترين اثنين وهما: اسم الـDataSet، واسم الجدول التابع للـDataSet، كما بالكود التالي:

```
MyDataAdapter.Fill(MyDataSet, "GettingRows")
```

يعني تم تعبئة الجدول GettingRows بالبيانات المسترجعة من MyDataAdapter.

ملاحظات:

ما عدد السجلات المسترجعة ؟

يمكن معرفة عدد السجلات التي أرجعها الـDataAdapter من خلال الكود التالي:

```
Dim RowsCount As Integer
RowsCount = MyDataSet.Tables("GettingRows").Rows.Count
```

لو كان RowsCount = صفراً، فإن الـDataAdapter لم يرجع أي شيء!.

كيف نصل إلى قيم الحقول في الـDataSet ؟

بعد الاتصال وتعبئة الـDataSet بالبيانات المسترجعة، يمكن الوصول إلى قيمة كل حقل بدلالة كل من: رقم السجل، واسم الحقل أو رقم ترتيبه كما هو مخزن في الجدول في قاعدة البيانات.

Item

Row	0 ID	1 PName	2 PBirthDate	3 PCountry	4 PCity	5 PAddress	6 PMobile
0	1	Ahmed	19-04-1960	Libya	Tripoli	Gargaresh	091-8889900
1	2	Ali	21-12-1980	Saudi Arabia	Mecca	Mecca	78628768760
2	3	Hamed	10-09-1977	Egypt	Cairo	Nasr City	87484764387

لو فرضنا أن الجدول في الـDataSet تحتوي على سجل واحد (صف واحد)، فيمكن الوصول إلى قيمة الحقل Pname مثلاً كالتالي:

```
txtPatientName.Text = MyDataSet.Tables("GettingRows").Rows(0).Item("Pname")
```

أو بدلالة رقم الحقل

```
txtPatientName.Text = MyDataSet.Tables("GettingRows").Rows(0).Item(1)
```

وفي حالة وجود أكثر من سجل، نقوم بعمل حلقة تكرار، بحيث يبدأ العد من الصفر إلى (عدد الصفوف ناقص واحد). فالمثال التالي يعرض

ملاحظات:

طريقة عرض محتويات الـDataSet على DataGridView:

```
Public Sub LoadPatientsInfo()
Dim LSQL As String = "select * from Patients order by ID"
Dim LAdapter As New OleDbDataAdapter(LSQL, Con)
Dim LDataSet As New DataSet
Dim RowsCount As Integer
Dim i As Integer

dgvPatientList.Rows.Clear()

If Con.State = ConnectionState.Open Then Con.Close()
      Con.Open()
      LAdapter.Fill(LDataSet, "Pats")
      RowsCount = LDataSet.Tables("Pats").Rows.Count
      If RowsCount = 0 Then
        dgvPatientList.Rows.Clear()
        LDataSet.Reset()
        Con.Close()
        Exit Sub
      Else
        dgvPatientList.Rows.Add(RowsCount)

        For i = 0 To RowsCount - 1
          With dgvPatientList
          .Rows(i).Cells(0).Value = LDataSet.Tables("Pats").Rows(i).Item("FileNumber")
          .Rows(i).Cells(1).Value = LDataSet.Tables("Pats").Rows(i).Item("PatName")
          .Rows(i).Cells(2).Value = LDataSet.Tables("Pats").Rows(i).Item("Nationality")
          .Rows(i).Cells(3).Value = LDataSet.Tables("Pats").Rows(i).Item("Address")
          .Rows(i).Cells(4).Value = LDataSet.Tables("Pats").Rows(i).Item("EnteranceDate")
          End With
       Next
       LDataSet.Reset()
       Con.Close()
      End If
End If 'main
End Sub
```

ملاحظات:

.....

.....

.....

تفريغ الـDataSet

عند اكتمال العمل، والرغبة في تفريغ الـDataSet من محتوياتها نكتب:

```
MyDataSet.Reset()
```

وهذا الأمر يفرغ الـDataSet من محتوياتها، ويجعلها في وضع استعداد لتنفيذ عملية لاحقة، وبالتالي أي إشارة إليها بعد هذا الكود يسبب ظهور أخطاء. لأنه لا بيانات فيها بعد الآن. والمثال السابق يوضح متى نقوم بتفريغ الـDataSet.

إضافة وتعديل وحذف البيانات

بعد أن عرفنا كيفية البحث والاستعلام في قاعدة البيانات، والتجول بين السجلات المستجلبة، حان الوقت للحديث عن العمليات الأساسية على قواعد البيانات وهي الإضافة والتعديل والحذف.

وقبل المضي قدماً في هذا الحديث، أود أن أُلفت انتباهكم إلى أن البيانات المخزنة في الـDataSet هي بيانات منفصلة عن قاعدة البيانات، وأنه لا يوجد اتصال مباشر بين قاعدة البيانات والـDataSet، ولذلك فإن أي عملية نُجريها على السجلات الموجودة في الـDataSet تنطبق فقط عليها، ولا يتم تحديث بيانات قاعدة البيانات إلا بعد تجديد الاتصال بها وتطبيق التغييرات عليها كما سنرى بعد قليل.

وهناك طريقتان لتحديث البيانات:

- عن طريق استخدام الـCommandBuilder مع الـDataAdapter.
- عن طريق استخدام الـCommand.

ملاحظات:

تحديث البيانات باستخدام الـCommandBuilder مع الـDataAdapter

هذه الطريقة تعتمد على الـDataAdapter كوسيط بين مصدر البيانات والـDataSet التي يتم تحديثها، ولا يستطيع الـDataAdapter تحديث مصدر البيانات إلا في وجود عنصر مساعد وهو الكائن CommandBuilder.

فلندرس الكود التالي معاً:

```
Private Sub SaveMed()
Dim SavSQl As String = "select * from medicine where MedCode='" & txtMedCode.Text & "'"
Dim SavAdapter As New OleDbDataAdapter(SavSQl, Con)
Dim SavDataSet As New DataSet
Dim RowsCount As Integer
Dim dsNewRow As DataRow

If Con.State = ConnectionState.Open Then Con.Close()
     Con.Open()
     SavAdapter.Fill(SavDataSet, "SavingMeds")
     RowsCount = SavDataSet.Tables("SavingMeds").Rows.Count
     If RowsCount > 0 Then
        MsgBox("تم تسجيل هذا الدواء مسبقاً", MsgBoxStyle.Critical, "عذراً")
        SavDataSet.Reset()
        Con.Close()
        ClearControls()
     Else
        dsNewRow = SavDataSet.Tables("SavingMeds").NewRow()
        dsNewRow.Item("MedCodeNumber") = txtMedCode.Text
        dsNewRow.Item("MedPublicName") = txtPublicName.Text
        dsNewRow.Item("MedSientificName") = txtScientificName.Text
        dsNewRow.Item("MedCapacity") = txtCapacity.Text
        dsNewRow.Item("MedForm") = cmbForm.Text

        Dim cb As New OleDb.OleDbCommandBuilder(SavAdapter)
        SavDataSet.Tables("SavingMeds").Rows.Add(dsNewRow)
        SavAdapter.Update(SavDataSet, "SavingMeds")
        SavDataSet.Reset()
        Con.Close()
        MsgBox("تتم تسجيل الدواء بنجاح", MsgBoxStyle.Information, "شكراً لك")
    End If
End If
End Sub
```

ففي البداية قمنا بتعريف جملة الاستعلام التي سنستخدمها مع الـDataAdapter، وتقوم بالاستعلام عن وجود سجل يحمل نفس الرقم التسلسلي للدواء والمكتوب في txtMedCode.Text، فإن وجد السجل، فلا نقوم بعملية الإضافة، ونظهر رسالة تقول: تم تسجيل هذا الدواء مسبقاً. أما في حالة عدم العثور على هذا السجل، ومعناه أن عدد الصفوف المسترجعة = صفراً، وبالتالي فالـDataSet المسماة SavDataSet فارغة، فنقوم بفتح سجل جديد مؤقت، نضع فيه قيم الحقول التي نريدها، ثم نستدعي الوظيفة Update التابعة للـDataAdapter في وجود الـ OleDbCommandBuilder، فيتم تحديث الـDataSet بالتغييرات الجديدة، وبالتالي تحديث مصدر البيانات.

ملاحظات:

.....

.....

.....

تحديث البيانات باستخدام الكائن Command

هذه الطريقة هي الأسهل، فلا داع للإضافة (إلى) أو التعديل (على) أو الحذف (من) الـDataSet، ثم تطبيق ذلك التحديث على مصدر البيانات من خلال الـDataAdapter الذي لن يعمل إلا في وجود الـCommandBuilder، بل يمكن عمل كل ذلك مباشرة من خلال استعمال الكائن Command كما سنرى.

إضافة سجل جديد

```
Private Sub SaveCustomer()
Dim SSql As String = "INSERT INTO Customers (CustomerName, CustomerAddress, _ CustomerPhones) VALUES
(@CustomerName,@CustomerAddress,@CustomerPhones)"
 Dim SCMD As New OleDbCommand

 If Con.State = ConnectionState.Open Then Con.Close()
     Con.Open()
     With SCMD
          .Connection = Con
          .CommandType = CommandType.Text
          .CommandText = SSql

          .Parameters.AddWithValue("CustomerName", txtCustomerName.Text)
          .Parameters.AddWithValue("CustomerAddress", txtCustomerAddress.Text)
          .Parameters.AddWithValue("CustomerPhones", txtCustomerPhones.Text)

          .ExecuteNonQuery()
          .Dispose()
     End With
     Con.Close()
 End If
End Sub
```

وكما نلاحظ من الكود السابق، نقوم بتعريف جملة الاستعلام الخاصة بعملية الإضافة، وفيها تم تحديد الحقول التي سنقوم بتخصيص قيمها، ثم تعريف الـCommand نفسه. الـCommand يحتاج إلى تحديد كائن الاتصال، ونوع الأمر، وجملة الاستعلام، وتم تحديدها وضبطها في الكود:

```
.Connection = Con
.CommandType = CommandType.Text
.CommandText = SSql
```

بعد ذلك نستدعي الطريقة AddWithValue والتي تقبل بارامترين هما:

- اسم الحقل
- قيمة الحقل

وبعد ذلك نستدعي الطريقة ExecuteNonQuery() لتنفيذ الأمر وإضافة السجل الجديد إلى قاعدة البيانات. هذا كل ما في الأمر. مثال آخر:

ملاحظات:

.....

.....

.....

تعديل سجل

```
Private Sub UpdateCustomer()
Dim SSql As String = "UPDATE Customers SET CustomerName=@CustomerName, CustomerAddress=@CustomerAddress,
CustomerPhones=@CustomerPhones WHERE ID=" & CustID & ""
Dim SCMD As New OleDbCommand
If Con.State = ConnectionState.Open Then Con.Close()
Con.Open()
With SCMD
        .Connection = Con
        .CommandType = CommandType.Text
        .CommandText = SSql

        .Parameters.AddWithValue("CustomerName", txtCustomerName.Text)
        .Parameters.AddWithValue("CustomerAddress", txtCustomerAddress.Text)
        .Parameters.AddWithValue("CustomerPhones", txtCustomerPhones.Text)
        .ExecuteNonQuery()
        .Dispose()
End With
Con.Close()
Me.Close()
End Sub
```

وكما نلاحظ، فإن الكود هو نفسه المستعمل في عملية الإضافة، والاختلاف فقط في جملة الاستعلام!.

حذف البيانات باستخدام الكائن Command

وبنفس الطريقة، باستثناء تعديلات بسيطة على جملة SQL ، نستطيع حذف البيانات من قاعدة البيانات:

```
Private Sub DeleteCustomer()
Dim SSql As String = "DELETE FROM Customers WHERE ID=" & CustID & ""
Dim SCMD As New OleDbCommand
If Con.State = ConnectionState.Open Then Con.Close()
        Con.Open()
        With SCMD
                .Connection = Con
                .CommandType = CommandType.Text
                .CommandText = SSql

                .ExecuteNonQuery()
                .Dispose()
        End With
Con.Close()
Me.Close()
End Sub
```

ملاحظات:

.....

.....

.....

التداخل بين DataAdapter وDataCommand في العمليات على قواعد البيانات

نستطيع إجراء عمليات الإضافة والتعديل والحذف من خلال التعاون بين كل من الـDataAdapter والـDataCommand.

أولاً: DataAdapter InsertCommand

وهو خاص بعملية الإضافة. لندرس الكود التالي:

```
Dim Con As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Library.mdb")
Dim InsertSQL As String = "INSERT INTO books VALUES('Databases','Abubaker Swedan')"
Dim InsertAdapter As New OleDbDataAdapter(InsertSQL, Con)
Con.Open()
With InsertAdapter
        .InsertCommand = New OleDbCommand(InsertSQL, Con)
        .InsertCommand.ExecuteNonQuery()
End With
Con.Close()
MsgBox("Rows Inserted")
```

ثانياً: DataAdapter UpdateCommand

وهو خاص بعملية التعديل. لندرس الكود التالي:

```
Dim Con As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Library.mdb")
Dim UpdateSQL As String = "UPDATE books SET BookTitle='Databases', BookAuther='Aziz Omran' WHERE ID=1"
Dim UpdateAdapter As New OleDbDataAdapter(UpdateSQL, Con)
Con.Open()
With UpdateAdapter
        .UpdateCommand = New OleDbCommand(UpdateSQL, Con)
        .UpdateCommand.ExecuteNonQuery()
End With
Con.Close()
MsgBox("Rows Updated")
```

ثالثاً: DataAdapter DeleteCommand

وهو خاص بعملية الحذف. لندرس الكود التالي:

```
Dim Con As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\Library.mdb")
Dim DeleteSQL As String = "DELETE FROM books WHERE ID=15"
Dim DeleteAdapter As New OleDbDataAdapter(DeleteSQL, Con)
Con.Open()
With DeleteAdapter
        .DeleteCommand = New OleDbCommand(DeleteSQL, Con)
        .DeleteCommand.ExecuteNonQuery()
End With
Con.Close()
MsgBox("Row Deleted")
```

ملاحظات:

.....

.....

.....