

الفصل الثامن عشر استخدام مجموعات البيانات DataSets

بدلاً من تعديل البيانات واسترجاعها من قاعدة البيانات عن طريق الاتصال المستمر، يمكنك أخذ نسخة من جزء معين داخل قاعدة البيانات أو من قاعدة البيانات كلها إن أحببت ثم معالجة هذا الجزء في نمط الانفصال وتحديث قاعدة البيانات مرةً أخرى. يمكنك أداء كل هذا من خلال كائن مجموعة البيانات **DataSet** وهو أهم سمات **ADO.NET**، وهو موضوع هذا الفصل.

بانتهاء هذا الفصل ستتعرف على:

- تركيب واستخدام مجموعة البيانات.
- استخدام الكائن **DataAdapter**.
- الوصول لقيم الحقول.
- تحميل أكثر من جدول.
- تخطيط الجداول والأعمدة.
- إضافة العلاقات والقيود.
- استخدام شبكة البيانات والمرشحات.
- التحقق من السجلات المعدلة.
- تحديث قاعدة البيانات.
- استخدام الإجراءات **Transactions**.
- تقليل الاتصال بالشبكة.

تعتبر مجموعة البيانات **DataSet** أداة لتخزين وتحديث البيانات في نمط عدم الاتصال **Disconnected Mode**، لذا فهي أهم أدوات **ADO.NET** على الإطلاق. وبيانات هذه المجموعة تكون دائماً آمنة وغير قابلة للضياع، كما أنها تحتوي على وظائف مشابهة لتلك الموجودة داخل قواعد البيانات المخزنة إلا أنها تقع داخل الذاكرة فقط. وقد تحدثنا في الفصل السابق عن أهمية الكائن **DataSet** والفرق بينه وبين الكائن **Recordset** الموجود في الإصدارات السابقة من **ADO**. سنقوم في هذا الفصل بالتعرف على تركيب الكائن **DataSet** أو مجموعة البيانات وكيفية استخدامها لإجراء العمليات المختلفة على قاعدة البيانات، فكن معنا.

تركيب مجموعة البيانات

يحتوي تصنيف مجموعة البيانات **DataSet Class** على مجموعة من التصنيفات الأخرى التي يمكنك من التعامل مع البيانات الموجودة داخل مجموعة البيانات. ويتكون نموذج مجموعة البيانات أساساً من الجداول **Tables** والأعمدة **Columns** والعلاقات **Relations** والقيود **Constraints** وأخيراً الصفوف **Rows** التي تمثل البيانات الحقيقية، حيث تقوم العناصر الأخرى بتحقيق مخطط العلاقات. ويمكن بيان التركيب الهرمي لمجموعة البيانات كما يلي:

- يحتوي كائن **DataSet** على مجموعة من الجداول المشتقة من التصنيف **DataTable**.
- يحتوي كل تصنيف من التصنيفات **DataTable** على بيانات أحد الجداول الموجودة داخل قاعدة البيانات.
- يحتوي كائن **DataTable** بدوره على مجموعة من الأعمدة المشتقة من التصنيف **DataColumn**. فمثلاً يحتوي الكائن **DataColumn** المرتبط بالعمود **St_ID** داخل الجدول **Students** على بيانات هذا العمود بما في ذلك اسم العمود ونوع بياناته بالإضافة إلى القيم والصفات والخصائص المرتبطة بهذا العمود (الحقل).

- يتم اشتقاق كائن العلاقات **Relations Object** من التصنيف **DataRelation** ويقوم بتخزين بيانات عن العلاقات بين الأعمدة الموجودة داخل جدول معين **.DataTable**.
 - تحتوي الصفوف **Rows** الموجودة داخل الجدول على نسخة من البيانات الحقيقية المخزنة داخل مجموعة البيانات **DataSet**.
 - يتم اشتقاق الصفوف **Rows** من التصنيف **DataRow**. ويمكنك الوصول إلى الصفوف المختلفة الموجودة داخل الجدول من خلال الخاصية **Rows** المصاحبة للتصنيف **.DataTable**.
- تتشابه عملية الوصول إلى مجموعة البيانات **DataSet** كثيراً مع عملية الوصول إلى قاعدة البيانات نفسها وذلك لأن مجموعة البيانات تحتوي على جميع بيانات الجداول وخصائص الكائنات التي تنتمي لكل جدول. كما تحتوي أيضاً على معلومات عن العلاقات بين الجداول، لذا تعمل مجموعة البيانات في حالة عدم الاتصال كما لو كانت في حالة اتصال بالفعل.

استخدام مجموعة البيانات

الخطوة الأولى في استخدام مجموعة بيانات **ADO.NET** هي تضمين المسمى **System.Data** وتضمين مزود بيانات **OLE DB** (أو **SQL**) كما في الكود التالي:

```
Imports System.Data
```

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

وبعد ذلك يتم إنشاء مجموعة البيانات عن طريق إنشاء حالة من التصنيف **DataSet** ثم ربط مجموعة البيانات بقاعدة بيانات معينة بإنشاء سلسلة الاتصال المناسبة والتي تقوم بإجراء عملية الاتصال والتأكد من صحتها وذلك من خلال الكائن **OleDbConnection** (أو **SqlConnection**) الذي يقوم بتأسيس عملية الاتصال. ولا يمكنك استدعاء كائن الاتصال مباشرة عن طريق مجموعة البيانات وإنما يتم بدلاً من ذلك ربطها من خلال كائنات

DataAdapter كما سنشرح لاحقاً. لإنشاء كائن لمجموعة بيانات خالية لاستخدامها في تخزين بيانات من مجموعة جداول داخل قاعدة البيانات، يمكنك استخدام الصيغة التالية:
DataSet ds = New DataSet()

استخدام الكائن **DataAdapter**

بعد الانتهاء من إنشاء كائن مجموعة البيانات، يجب نسخ القيم الموجودة داخل قاعدة البيانات إلى مجموعة البيانات **DataSet** وذلك من خلال الكائن **DataAdapter** الذي يعمل كموصل أو كوبرى بين قاعدة البيانات ومجموعة البيانات الممثلة في الكائن **DataSet** وهذا يختلف بالطبع عن الكائن **Command** الذي يعمل كما لو كان دالة تقوم بإرجاع مجموعة من السجلات. يحتوى التصنيف **DataAdapter** على الوظائف (**Fill()**) و (**Update()**) التي تستخدم في تبادل البيانات مع قاعدة البيانات المستخدمة، حيث تستخدم الوظيفة (**Fill()**) لملاء مجموعة البيانات بالقيم الموجودة داخل جداول قاعدة البيانات بينما تستخدم الوظيفة (**Update()**) لتحديث قاعدة البيانات بالبيانات الموجودة بكائن مجموعة البيانات. كما يحتوى الكائن **DataAdapter** على مجموعة خصائص من أهمها **SelectCommand** و **InsertCommand** و **UpdateCommand** و **DeleteCommand** حيث يتم تبادل البيانات باستدعاء أى منها تبعاً لنوع الاستعلام المستخدم.

لتوضيح ذلك، دعنا نرى الكود التالي:

الإعلان عن الكائنات،

1. **Dim con As OleDbConnection**
2. **Dim cmd As OleDbCommand**
3. **Dim strSQL As String**
4. **Dim adStudents As OleDbDataAdapter**
5. **Dim ds As DataSet**

تعيين القيم الابتدائية للكائنات،

6. **con = New OleDbConnection("Provider= Microsoft.Jet.OLEDB.4.0; Data Source = d:\Students.mdb")**

```
7. adStudents = New OleDbDataAdapter()  
8. ds = New DataSet()
```

إعداد الأمر،

```
9. strSQL = "SELECT * From Students"  
10. cmd = New OleDbCommand(strSQL,con)  
11. adStudents.SelectCommand = cmd
```

تعبئة مجموعة البيانات،

```
12. con.Open()  
13. adStudents.Fill(ds)  
14. con.Close()
```

وعن هذا الكود، نوضح ما يلي:

- يتم في السطور من ١ إلى ٥ الإعلان عن الكائنات الأساسية وهي كائن الاتصال **OleDbConnection** وكائن الأمر **OleDbCommand** وكائن منظم البيانات **OleDbDataAdapter** وكائن مجموعة البيانات **DataSet** وكذلك سلسلة البيانات **strSQL** التي ستحتوي على عبارة **SQL**.
- يتم في السطور من ٦ إلى ٨ تعيين القيم الابتدائية للكائنات السابقة وذلك بتعريف كائن الاتصال **con** وكائن منظم البيانات **adStudents** وكائن مجموعة البيانات **ds**.
- يتم في السطور من ٩ إلى ١١ تعيين عبارة **SQL** المستخدمة ثم تعريف كائن الأمر وذلك بتمرير عبارة **SQL** وكائن الاتصال وأخيراً تخصيص كائن الأمر للخاصية **SelectedCommand** المصاحبة لمنظم البيانات.
- في السطور من ١٢ إلى ١٤ يتم فتح عملية الاتصال من خلال الوظيفة **Open()** ثم ملء مجموعة البيانات بنتيجة عبارة **SQL** المحددة من قبل وذلك من خلال الوظيفة **Fill()** وأخيراً يتم قطع عملية الاتصال من خلال الوظيفة **Close()**.

الوصول لقيمة الحقول

تختلف عملية الوصول للسجلات داخل مجموعة البيانات عن تلك المستخدمة مع قارئ البيانات والتي شرحناها في الفصل السابق وذلك لعدم وجود مفهوم السجل الحالى أو وظيفة الحركة المصاحبة. وبدلاً من ذلك يتم استخدام التجمعات في ترتيب الجداول والصفوف وقيم الحقول. فالسطر التالى على سبيل المثال، يقوم بعرض محتويات الحقل الأول داخل السجل الأول بالجدول الأول:

MessageBox.Show(ds.Tables(0).Rows(0).Item(0).ToString)

كما يمكنك بهذه الطريقة التنقل داخل المجموعة بسهولة باستخدام المعرفات المسماة والدورات وعبارة **With** كما في الكود التالى الذى يقوم بطباعة الاسم الأول من كل صف على نافذة الإخراج.

With ds.Tables(0)

For i = 0 To .Rows.Count-1

Debug.WriteLine(.Rows(i).Item("St_fname"))

Next

End With

تحميل أكثر من جدول

كما رأيت فإن عملية ملء مجموعة البيانات بجدول واحد أمر غاية في السهولة. ولكن كى تحصل على الميزات العديدة لمجموعة البيانات، ربما تحتاج إلى تحميل أكثر من جدول إلى نفس المجموعة وإنشاء العلاقات بين هذه الجداول سواءً كانت هذه الجداول من قاعدة بيانات واحدة أو أكثر. ولأداء ذلك، يمكنك استخدام إحدى الطريقتين الآتيتين:

- **الطريقة الأولى:** استدعاء الوظيفة **Fill()** على أكثر من كائن **DataAdapter** باستخدام نفس مجموعة البيانات كمعامل لجميع الكائنات.
- **الطريقة الثانية:** تنفيذ إجراء مخزن يحتوى على أكثر من عبارة **SELECT** وفى هذه الحالة يتم استخدام كائن **DataAdapter** واحد فقط.

وفيما يلى نضرب مثلاً على كلتا الطريقتين.

استخدام أكثر من أمر

يوضح الكود التالي كيفية إضافة السجلات من الجدولين Students و Subjects داخل نفس مجموعة البيانات:

الإعلان عن الكائنات،

```
Dim con As OleDbConnection
Dim cmdGetStudents As OleDbCommand
Dim cmdGetSubjects As OleDbCommand
Dim strSQL As String
Dim adStudents As OleDbDataAdapter
Dim adSubjects As OleDbDataAdapter
Dim ds As DataSet
```

تعيين القيم الابتدائية للكائنات،

```
con = New OleDbConnection("Provider= Microsoft. _
Jet.OLEDB.4.0;Data Source = d:\Students.mdb")
adStudents = New OleDbDataAdapter()
adSubjects = New OleDbDataAdapter()
ds = New DataSet()
```

إعداد الأمر،

```
strSQL = "SELECT * From Students"
cmdGetStudents = New OleDbCommand(strSQL, con)
strSQL = "SELECT * From Subjects"
cmdGetSubjects = New OleDbCommand(strSQL, con)
adStudents.SelectCommand = cmdGetStudents
adStudents.TableMappings.Add("Table", "Students")
adSubjects.SelectCommand = cmdGetSubjects
adSubjects.TableMappings.Add("Table", "Subjects")
```

تعبئة مجموعة البيانات،

```
con.Open()
adStudents.Fill(ds)
adSubjects.Fill(ds)
con.Close()
```

قمنا في هذا الكود باستخدام كائن DataAdapter مع كل جدول. ولا جديد في هذا الكود عن الكود السابق غير استخدام مفهوم تخطيط الجداول Table Mappings الذي

يجبر منظم البيانات بحقول وجداول مجموعة البيانات التي يتم استخدامها عند ملء مجموعة البيانات، أو بمعنى آخر يمكنك استخدام تخطيط الجدول في تسمية الحقول والجدول بأسماء غير الموجودة في قاعدة البيانات نفسها، وسوف نتعرف على هذا المفهوم بشئٍ من التفصيل بعد قليل.

استخدام إجراء مخزن

يتيح لك استخدام أكثر من منظم بيانات **DataAdapter**؛ كما في الطريقة الأولى، ملء مجموعة البيانات من قواعد بيانات مختلفة. لكن إذا كانت الجدول داخل نفس قاعدة البيانات، يمكنك استخدام إجراء مخزن للحصول على بيانات الجدول في نفس الوقت. لنقوم أولاً بإنشاء الإجراء المخزن كما يلي:

```
CREATE PROCEDURE spGetAllStudentInfo
```

```
AS
```

```
SELECT * FROM Students
```

```
SELECT * FROM Subjects
```

وكما ترى، يحتوي الإجراء المخزن على عبارتي **SELECT**، الأولى لاسترجاع جميع بيانات الجدول **Students** والأخرى لاسترجاع بيانات الجدول **Subjects**. والآن نقوم باستخدام هذا الإجراء بتعديل الكود المستخدم في الطريقة الأولى كما يلي:

الإعلان عن الكائنات،

```
Dim con As SqlConnection
Dim cmdGetAll As SqlCommand
Dim adAllInfo As SqlDataAdapter
Dim ds As DataSet
```

تعيين القيم الابتدائية للكائنات،

```
con = New
SqlConnection("server=localhost;uid=sa;pwd=;database=Students")
adAllInfo = New SqlDataAdapter()
ds = New DataSet()
```

إعداد الأمر،

```
cmdGetAll = New
SQLCommand("spGetAllStudentInfo",con)
cmdGetAll.CommandType =
CommandType.StoredProcedure
adAllInfo.SelectCommand = cmdGetAll
'Setup Mappings
adAllInfo.TableMappings.Add("Table", "Students")
adAllInfo.TableMappings.Add("Table1", "Subjects")
```

تعبئة مجموعة البيانات،

```
con.Open()
adAllInfo.Fill(ds)
con.Close()
```

يتم تخصيص الأسماء Table و Table1 لمجموعتي السجلات الناتجة عن الإجراء المخزن، لذا تم استخدام التخطيط Mapping لتحويلها إلى أسماء معبرة. وعند العمل مع الجداول داخل منظم البيانات (الكائن DataAdapter)، تذكر دائماً أن عدد الجداول الموجود بالكائن يعتمد على عدد المجموعات الناتجة وليس عدد الجداول المحدد في عبارة الاستعلام المستخدمة. فعلى سبيل المثال، تظهر العبارة التالية كجدول واحد فقط بالنسبة للكائن DataAdapter على الرغم من احتوائها على جدولين وذلك لأن الناتج عبارة عن مجموعة واحدة فقط:

```
SELECT * FROM Students St, Subjects Sb WHERE St.St_ID =
Sb.St_ID
```

في حالة رجوع أكثر من حقل بنفس الاسم وفي نفس الجدول (Table أو Table1 وهكذا) إلى الكائن DataAdapter (مثل الحقل St_ID في العبارة السابقة)، يقوم الكائن DataAdapter بمعاملتها نفس معاملة الجداول (أى تصيح St_ID و St_ID1 وهكذا). والآن لتتعرف على مفهوم تخطيط الجداول والأعمدة بشيءٍ من التفصيل.

تخطيط الجداول والأعمدة

يقصد بتخطيط الجداول والأعمدة (الحقول) تغيير أسماء الجدول أو العمود (الحقل) إلى اسم محبب إلى المستخدم غير الاسم المستخدم داخل الكائن **DataAdapter** والذي يكون **Table** و **Table1** وهكذا، حيث يشير الجدول إلى مجموعة البيانات الناتجة برمتها كما ذكرنا منذ قليل. ويكون ذلك بإنشاء رابط منطقي بين الجدول الموجود بمجموعة البيانات والجدول المماثل الموجود بقاعدة البيانات. وقد رأينا منذ قليل كيفية تخطيط الجداول، لنرى الآن في الكود التالي كيفية إنشاء تخطيط الأعمدة الموجودة داخل الجدول **Students** وذلك كما يلي:

```
OleDbAdapter.TableMappings[0].ColumnMappings.Add("St_ID",
"Student ID")
OleDbAdapter.TableMappings[0].ColumnMappings.Add("St_fname",
"Student First Name")
OleDbAdapter.TableMappings[0].ColumnMappings.Add("St_Lname",
"Student Last Name")
```

إذا قمت باستدعاء الوظيفة (**Fill()** أو الوظيفة (**Update()** (كما سنرى بعد قليل) دون أن تقوم بتحديد الجدول المصدر كما في العبارة التالية:

```
OleDbAdapter.Fill(ds)
```

فإن كائن **DataAdapter** يقوم بالبحث عن جدول باسم **Table**. فإذا لم يجد هذا الجدول، يقوم بالبحث داخل مجموعة البيانات عن الجدول البديل الذي قمت بتعيينه أثناء عملية التخطيط.

إضافة العلاقات والقيود

بالإضافة إلى تخزين الجداول، فإن الكائن **DataSet** لديه المقدرة على تخزين العلاقات **Relationships** بين هذه الجداول وكذلك القيود **Constraints** تماماً كما في قاعدة البيانات العادية. ومجموعة البيانات قابلة للتعديل أثناء الانفصال عن قاعدة البيانات، وبالتالي وجود لنظام إدارة قواعد البيانات لتطبيق القيود على بيانات الجداول مثل القيم الفريدة أو المفاتيح الأساسية. مع العلم أنه عند إضافة أكثر من جدول لمجموعة البيانات، لا يتم إضافة العلاقات أو القيود الموجودة على هذه الجداول تلقائياً وإنما يمكنك إضافتها من

خلال الكود كما يلي:

إعداد كائنات الأعمدة لسهولة الوصول إليها'

```
Dim colIDSt, colIDSb As DataColumn
colIDSt = ds.Tables("Students").Columns("St_ID")
colIDSb = ds.Tables("Subjects").Columns("St_ID")
```

إنشاء المفاتيح الأساسية'

```
Dim arStudKey(1) As DataColumn
Dim arSubjKey(2) As DataColumn
arStudKey(0) = colIDSt
arSubjKey(0) = colIDSb
arSubjKey(1) = ds.Tables("Subject").Columns("Sub_ID")
ds.Tables("Students").PrimaryKey = arStudKey
ds.Tables("Subject").PrimaryKey = arSubjKey
```

إنشاء المفتاح الإجنبي لعملية حذف السجلات التابعة تلقائياً'

```
Dim fkID As ForeignKeyConstraint
fkID = New ForeignKeyConstraint("IDForKey", colIDSt,
colIDSb)
fkID.DeleteRule = Rule.Cascade
ds.Tables("Subject").Constraints.Add(fkID)
```

وقد قمنا في هذا الكود باستخدام الكائن DataColumn الذي يمثل حقلاً داخل جدول البيانات. يمكنك تعيين الخاصية PrimaryKey لكل جدول داخل مجموعة البيانات DataSet بمصفوفة من كائنات DataColumn لتعريف المفتاح الأساسي. ويعتبر الحقل St_ID هو المفتاح الأساسي داخل الجدول Students بينما يمثل الحقل St_ID والحقل Sub_ID معاً المفتاح الأساسي داخل الجدول Subjects. ولأن St_ID عبارة عن مفتاح أجنبي أيضاً داخل الجدول Subjects وللحفاظ على تكامل قاعدة البيانات، يجب حذف السجل من الجدول Subject إذا تم حذف سجل الطالب المصاحب من الجدول Students. ويتم ذلك تلقائياً بإنشاء الكائن ForeignKeyConstraint وتخصيصه بقاعدة الحذف من خلال الخاصية DeleteRule.

استخدام شبكة البيانات

يمكنك استخدام أداة تحكم "شبكة البيانات" DataGrid لإظهار محتويات مجموعة بيانات. لتوضيح ذلك، سنقوم بإنشاء تطبيق صغير يحتوى على شبكة البيانات التي تقوم بعرض بيانات الجدول Students والجدول Subjects الموجودين بقاعدة البيانات Students.mdb. تابع معنا الخطوات الآتية:

١. من داخل بيئة تطوير Visual Studio 2012، قم بإنشاء تطبيق نوافذى جديد باسم مناسب وليكن DataGrid.
٢. أعد تسمية النموذج Form1 إلى اسم مناسب وليكن frmMain مع تغيير العنوان إلى "عرض البيانات من خلال شبكة البيانات".
٣. قم بإضافة زر أمر إلى النموذج وسمه إلى btnLoad وعنوانه إلى "تحميل البيانات".
٤. قم بإضافة أدواتى شبكة بيانات  إلى النموذج وسمه بتغيير اسميهما إلى grdSubjects و grdStudents.
٥. قم بإضافة أدواتى عنوان إلى رأس كل شبكة بيانات مع تغيير عنوان كل منهما إلى عنوان مناسب (انظر شكل ١٢-١).
٦. قم بفتح نافذة الكود الخاصة بالنموذج ثم قم بكتابة السطرين التاليين في بداية الكود وقبل كود التصنيف، وهما أساسيان كما ذكرنا لاستخدام ADO.NET:

```
Imports System.Data
```

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

٧. انقر الزر btnLoad نقرأ مزدوجاً ثم قم بتعديل كود تصنيف النموذج frmMain كما يلي:

```
Dim myds As DataSet
```

```
Private Sub btnLoad_Click(sender As Object, e As EventArgs)
```

```
Handles btnLoad.Click
```

```
Dim con As OleDbConnection
```

```
Dim cmdGetStud As OleDbCommand
```

```
Dim cmdGetSubj As OleDbCommand
```

```
Dim adStudInfo As OleDbDataAdapter  
Dim adSubjInfo As OleDbDataAdapter
```

تعيين القيم الابتدائية للكائنات'

```
con = New OleDbConnection("Provider= Microsoft.Jet.  
OLEDB.4.0; Data Source = d:\Students.mdb")  
adStudInfo = New OleDbDataAdapter()  
adSubjInfo = New OleDbDataAdapter()  
myds = New DataSet()
```

إعداد الأمر'

```
cmdGetStud = New OleDbCommand("Select * From  
Students", con)  
cmdGetSubj = New OleDbCommand("Select * From  
Subjects", con)  
adStudInfo.SelectCommand = cmdGetStud  
adSubjInfo.SelectCommand = cmdGetSubj
```

إعداد عملية تخطيط الجداول'

```
adStudInfo.TableMappings.Add("Table", "Students")  
adSubjInfo.TableMappings.Add("Table", "Subjects")
```

تعبئة مجموعة البيانات'

```
con.Open()  
adStudInfo.Fill(myds)  
adSubjInfo.Fill(myds)  
con.Close()
```

تعيين كائنات الأعمدة لسهولة الوصول إليها'

```
Dim colIDSt, colIDSb As DataColumn  
colIDSt = myds.Tables("Students").Columns("St_Id")  
colIDSb = myds.Tables("Subjects").Columns("St_Id")
```

إنشاء المفاتيح الأساسية'

```
Dim arStudKey(1) As DataColumn  
Dim arSubjKey(2) As DataColumn  
arStudKey(0) = colIDSt
```

```
arSubjKey(0) = colIDSb
arSubjKey(1) = myds.Tables("Subjects").
Columns("Sub_ID")
myds.Tables("Students").PrimaryKey = arStudKey
myds.Tables("Subjects").PrimaryKey = arSubjKey
```

إنشاء المفتاح الأجنبي للحذف التلقائي للسجلات التابعة

```
Dim fkSt_ID As ForeignKeyConstraint
fkSt_ID = New ForeignKeyConstraint("St_IDForKey",
colIDSt, colIDSb)
fkSt_ID.DeleteRule = Rule.Cascade
myds.Tables("Subjects").Constraints.Add(fkSt_ID)
```

ربط الجدولين بشبكتي البيانات

```
grdStudents.DataSource = myds
grdStudents.DataMember = "Students"
grdSubjects.DataSource = myds
grdSubjects.DataMember = "Subjects"
```

End Sub

ويحتوى هذا الكود كما ترى على توليفة من سطور الكود المختلفة التي ذكرناها من قبل، ما عدا السطور الأخيرة التي نقوم فيها بربط شبكة البيانات بالجدول المناسب وذلك بتخصيص مجموعة البيانات ككل إلى الخاصية **DataSource** المصاحبة لشبكة البيانات ثم تحديد الجدول الذي نرغب في إظهاره من خلال تخصيصه إلى الخاصية **DataMember**.
قم الآن بتشغيل التطبيق ثم انقر زر **Load Data**، تلاحظ امتلاء كل شبكة بيانات بالجدول المصاحب (انظر شكل ١٢-١).



شكل ١٢-١ عرض مجموعة البيانات من خلال شبكة البيانات

استخدام المرشحات

قمنا في المثال السابق بتخصيص كائن DataTable عبارة عن جدول من جداول مجموعة البيانات للخاصية DataMember المصاحبة لشبكة البيانات وبالتالي يتم عرض جميع حقول الجدول مع إعطاء الصلاحيات الكاملة لعملية التعديل والحذف والإضافة. لكن ماذا لو رغبت في عرض صفوف معينة بدلاً من الجدول بالكامل؟ أو أردت تقييد المستخدم في عملية التعديل؟. يمكنك أداء كل ذلك من خلال المرشحات Filters، حيث يعمل المرشح بنفس طريقة عمل العبارة WHERE دخل الاستعلام، فهو يقوم بإرجاع صفوف معينة بناءً على الشرط المحدد. سنقوم في المثال الذي بين أيدينا بعرض صفوف معينة داخل الجدول Students بناءً على الاسم الأول للطالب. لأداء ذلك، تابع معنا الخطوات الآتية:

١. قم بإضافة زر أمر جديد إلى النموذج مع تغيير اسمه إلى btnFilter وعنوانه إلى "تصفية البيانات".

٢. قم بإضافة مربع نص أسفل الزر السابق مع تغيير اسمه إلى txtFilter وتأكد من عدم وجود نص بداخله وإلا قم بحذفه.

٣. انقر الزر الجديد نقرأ مزدوجاً ثم قم بإدخال الكود التالي إلى إجراء حدث نقر الزر:

```
Private Sub btnFilter_Click(sender As Object, e As EventArgs) Handles btnFilter.Click
```

إلغاء عمليات ربط البيانات الحالية'

```
grdStudents.DataMember = ""
grdStudents.DataSource = Nothing
```

تعديل وربط عرض الجداول الافتراضى'

```
With myds.Tables("Students")
    .DefaultView.RowFilter = "St_fname Like " &
        txtFilter.Text & "%"
    .DefaultView.AllowDelete = False
    .DefaultView.AllowEdit = False
    .DefaultView.AllowNew = False
grdStudents.DataSource = .DefaultView
End With
End Sub
```

٤. قم بتشغيل التطبيق ثم انقر زر "تحميل البيانات"، تلاحظ عرض الجدولين بجميع بياناتهما.

٥. انقر الحروف الأولى من الاسم الأول للطلاب الذى ترغب فى عرضه داخل مربع النص ثم انقر زر "تصفية البيانات"، تلاحظ تغير شبكة البيانات وعرض السجل المحدد فقط (انظر شكل ١٢-٢).



شكل ١٢-٢ ترشيح البيانات لعرض بيانات معينة

والآن انظر معى سريعاً إلى الكود السابق، تجد أننا قمنا أولاً بحذف الارتباط بين شبكة البيانات ومجموعة البيانات Myds وكذلك الجدول Students الموجود بهذه المجموعة. بعد

ذلك قمنا باستخدام عرض افتراضي **DefaultView** داخل الجدول **Students** مع تعيين الشرط المطلوب أن يبدأ الاسم الأول للطالب بالحروف المحددة داخل مربع النص ثم قمنا بعد ذلك بمنع المستخدم من حذف وتعديل السجلات وإضافة سجلات جديدة وذلك بتخصيص القيمة **False** للخصائص **AllowDelete** و **AllowEdit** و **AllowNew** على الترتيب. وأخيراً يتم تخصيص هذا العرض كمصدر بيانات للشبكة من خلال الخاصية **DataSource**.

التحقق من السجلات المعدلة

يحتوي الكائن **DataTable**؛ الذي يمثل جدول من الجداول الموجودة بمجموعة البيانات، على مجموعة من الكائنات **DataRow** التي يمثل كل منها سجلاً داخل الجدول. يمكنك حقيقةً تعديل محتويات مجموعة البيانات سواءً بتغيير قيم كائن **DataRow** معين أو من خلال أدوات ربط البيانات كأداة شبكة البيانات **DataGridView**. ومن إحدى السمات الهامة لمجموعة البيانات، القدرة على إخبارك بالصفوف أو السجلات التي تم تعديلها، وهذا الأمر في غاية الأهمية لأن مجموعة البيانات تكون منفصلة عن مصدر البيانات. ومن هنا يجب أن تعرف هل تم إضافة سجلات جديدة أو تحديث السجلات الموجودة أو حذفها كي تتمكن بعد ذلك من عكس هذه التغييرات على مصدر البيانات الأصلي. فعلى فرض أن أحد جداول قاعدة البيانات يحتوي على ألف سجل، وقام المستخدم بتعديل سجل واحد فقط من هذه السجلات، فمن العبث إرسال السجلات كلها مرةً أخرى إلى قاعدة البيانات، والأولى في هذه الحالة هو إرسال السجل المعدل فقط. وعامةً يمكنك التعرف على التغييرات التي تمت على مجموعة البيانات من خلال الخصائص والوظائف الموضحة بجدول ١٢-١ التالي.

جدول ١٢-١ الخصائص والوظائف المستخدمة للتحكم في تغييرات مجموعة البيانات

| الاستخدام | الخاصية/الوظيفة |
|--|-----------------------------------|
| تحدد التغييرات التي تمت على الصف إن وجدت | الخاصية RowState |
| تقوم بإرجاع مجموعة بيانات جديدة (كائن DataSet) تحتوي على الصفوف المعدلة فقط | الوظيفة GetChanges() |
| تقوم بإرسال جميع التغييرات إلى مجموعة البيانات الحالية وتعديل الخاصية RowState | الوظيفة AcceptChanges() |
| تقوم بإرجاع تغييرات مجموعة البيانات إلى حالتها الأولى أو الحالة التي تم قبلها استدعاء الوظيفة AcceptChanges | الوظيفة RejectChanges() |

لتوضيح استخدام هذه الوظائف من خلال المثال الذي بين أيدينا، تابع معنا الخطوات الآتية:

١. قم بإضافة ثلاثة أزرار من مربع الأدوات إلى النموذج مع تغيير أسمائها إلى **btnAccept** و **btnReject** و **btnView** وعناوينها إلى "قبول التغييرات" و "رفض التغييرات" و "عرض التغييرات" على الترتيب.

٢. قم بإضافة الكود التالي الخاص بإجراءات أحداث نقر الأزرار الثلاثة:

```
Private Sub btnAccept_Click(sender As Object, e As EventArgs)
Handles btnAccept.Click
myds.AcceptChanges()
grdStudents.DataSource = myds
grdSubjects.DataSource = myds
End Sub
```

```
Private Sub btnReject_Click(sender As Object, e As EventArgs)
Handles btnReject.Click
myds.RejectChanges()
grdStudents.DataSource = myds
grdSubjects.DataSource = myds
End Sub
```

```
Private Sub btnView_Click(sender As Object, e As EventArgs)
Handles btnView.Click
Dim dsTemp As DataSet
```

```
dsTemp = myds.GetChanges
dsTemp.Tables("students").DefaultView.RowStateFilter =
DataViewRowState.ModifiedCurrentOr
DataViewRowState.Added
dsTemp.Tables("subjects").DefaultView.RowStateFilter =
DataViewRowState.ModifiedCurrentOr
DataViewRowState.Added
grdStudents.DataSource = dsTemp
grdSubjects.DataSource = dsTemp
End Sub
```

٣. قم بتشغيل التطبيق ثم انقر زر "عرض البيانات"، تظهر بيانات الجدولين داخل شبكتي البيانات.
٤. قم بتعديل الاسم في أحد الصفوف وقم بحذف صف آخر ثم قم بإضافة صف جديد إلى نهاية الشبكة.
٥. انقر زر "عرض التغييرات"، تقوم شبكة البيانات بإظهار الصفوف التي قمت بتعديلها أو إضافتها بينما لا تقوم بإظهار الصفوف التي تم حذفها وإنما يمكنك الوصول إليها من خلال الكود.
٦. انقر زر "رفض التغييرات" ، ترجع محتويات الشبكتين إلى الحالة الأولى قبل إجراء التغييرات.
٧. قم بتكرار الخطوة رقم ٤ ثم انقر زر "قبول التغييرات" لقبول التعديلات التي قمت بها.
٨. انقر زر "عرض التغييرات"، تلاحظ عدم ظهور أى بيانات داخل شبكتي البيانات وذلك لأنك قبلت التعديلات في الخطوة السابقة ولم تقم بأى تعديلات أخرى.

لاحظ أثناء الخطوات السابقة عملية الارتباط بين الجدولين الموجودين بشبكتي البيانات والتي قمنا بتعيينها من قبل من خلال الكود. فإذا قمت بحذف أحد الصفوف الموجودة بالجدول Students، يتم حذف الصفوف المصاحبة الموجودة بالجدول Subjects. وكذلك عدم القدرة على حذف أحد الصفوف من الجدول



Subjects دون أن تقوم أولاً بحذف الصف المصاحب بالجدول **Students**. وهذا يرجع إلى مفهوم تكامل البيانات داخل قواعد البيانات.

تدبير قاعدة البيانات

لتحديث قواعد البيانات في تطبيقات **ADO**، يمكنك تنفيذ عبارات **SQL** البسيطة أو استدعاء الإجراءات المخزنة. أما في **ADO.NET** فهناك صعوبة في استخدام هذه الطريقة داخل مجموعات البيانات **Datasets** لأنها تحتوي دائماً على بيانات في حالة عدم الاتصال وبالتالي لا يمكن تنفيذ عبارات التحديث مباشرةً.

يمكنك في **ADO** استخدام الوظيفة (**UpdateBatch**) في عملية التحديث التي تتم تلقائياً من خلال تعريف الصفوف ومقارنة القيم الموجودة ثم تطبيق التحديثات، كما أن عملية التحديث تتم من خلال العبارات **Update** و **Insert** و **Delete**.

أما في **ADO.NET** فيمكنك التحكم في الوظيفة (**Update**) من خلال الخصائص **SelectCommand** و **UpdateCommand** و **InsertCommand** و **DeleteCommand**. حيث تقوم بتخصيص الأوامر المناسبة لهذه الخصائص ثم يتم استدعاء الوظيفة (**Update**) التي تختار أي منها تبعاً لنوع عملية التعديل.

وتحتوي الوظيفة (**Update**) على معاملين، الأول عبارة عن مجموعة البيانات التي تحتوي على التغييرات التي أحدثها المستخدم بينما يحتوى المعامل الثاني على اسم الجدول الموجود بمجموعة البيانات والذي يحتوى على هذه التغييرات كما يلي:

```
adStudentInfo.Update(myds, "Students");
```

استخدام الإجراءات Transactions

يمكنك من خلال الإجراءات Transactions إرسال مجموعة من التحديثات مرة واحدة، بحيث إذا حدث خطأ في أحد هذه التحديثات يتم إلغاء العملية بالكامل. لتوضيح كيفية إنشاء الإجراء Transaction، قم بإضافة زر أمر جديد إلى النموذج باسم btnTransaction وعنوان "إجراء" ثم قم بإدخال الكود التالي بإجراء حدث نقر الزر الجديد:

1. Private Sub btnTransaction_Click(sender As Object, e As EventArgs) Handles btnTransaction.Click
2. Dim path As String = "d:\students.mdb"
3. Dim strcon As String = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" + path
4. Dim con As OleDbConnection = New OleDbConnection(strcon)
5. Dim Acmd As OleDbCommand = New OleDbCommand("SELECT * FROM Students", con)
6. Dim Adocmd As OleDbDataAdapter = New OleDbDataAdapter("SELECT * FROM Students", con)
7. Dim ds As DataSet = New DataSet()
8. Try
9. con.Open()
10. Adocmd.Fill(ds, "Students")
11. grdStudents.DataSource = ds.Tables("Students").DefaultView
12. Dim t As OleDbTransaction = con.BeginTransaction(IsolationLevel.ReadCommitted)
13. Acmd.Transaction = t
14. Acmd.CommandText = "Insert into Students (St_ID,St_fname) + "VALUES(10, 'Ali)'"
15. Acmd.ExecuteNonQuery()
16. Acmd.CommandText = "Insert into Students (St_ID,St_fname) + "VALUES(11, 'Sameh)'"
17. Acmd.ExecuteNonQuery()
18. t.Commit()
19. MessageBox.Show("Both the records were added.")
20. Adocmd.Fill(ds, "Students")
21. grdStudents.DataSource = Nothing
22. grdStudents.DataSource = ds.Tables("Students").DefaultView
23. ds.Tables("Students").DefaultView

24. **Catch ept As Exception**
25. **MessageBox.Show(ept.ToString())**
26. **MessageBox.Show("Neither records were written and**
27. **the transaction was rolledback.")**

28. **Finally**
29. **If (con.State = ConnectionState.Open) Then**
30. **con.Close()**
31. **End If**
32. **End Try**
33. **End Sub**

وعن هذا الكود، نوضح ما يلي:

١. في السطر رقم ١٢ يتم بدء الإجراء من خلال الوظيفة **BeginTransaction()**.
 ٢. في السطور من ١٣ إلى ١٧ يتم تعيين محتويات الإجراء وهي إضافة سجلين إلى الجدول **Students**.
 ٣. في السطر رقم ١٨ يتم إرسال الإجراء من خلال الوظيفة **Commit** وفي السطور التالية يتم إظهار رسالة بذلك مع إعادة ملء شبكة البيانات لإظهار عملية التحديث على الشاشة.
- قم الآن بتشغيل التطبيق ثم انقر زر "إجراء"، تظهر رسالة الاستثناء التي تخبرك باستحالة إضافة هذه السجلات لأنها تتعارض مع المفتاح الأساسي للجدول **Students** (انظر شكل ١٢-٣) وهذا يرجع بالطبع إلى العبارة **Try ... Catch ... Finally** المستخدمة في معالجة الاستثناءات.

سنتعرف على معالجة الاستثناءات من خلال العبارة **Try ... Catch ... Finally** في فصل مستقل من فصول هذا الكتاب.




```
Dim TempTable As DataTable
TempTable = New DataTable()
TempTable.Columns.Add(New DataColumn("Sub_ID",
                                     GetType(Integer)))
TempTable.Columns.Add(New DataColumn("Sub_name",
                                     GetType(String)))
```

```
Dim TempDsAsDataSet
TempDs = New DataSet()
TempDs.Tables.Add(TempTable)
```

وقد قمنا بإنشاء جدول واحد يحتوي على عمودين (حقليين) فقط، لذا ظهر الكود بسيطاً للغاية. لكن إذا أردت إنشاء أكثر من جدول يحتوي كل منها على العديد من الأعمدة وبينها علاقات معقدة، فإن كتابة الكود في هذه الحالة سيكون شاقاً للغاية. فإذا كان لديك شبكة اتصال سريعة، وأردت الاتصال بقاعدة البيانات لسببٍ آخر، يمكنك استرجاع مخطط الجداول من خلال الكائن **DataAdapter** وباستخدام الوظيفة **FillSchema()** كما في الكود التالي:

```
adAllInfo.FillSchema(myds, SchemaType.Mapped)
```

يتسبب هذا الكود في ملء كائن مجموعة البيانات بمخطط جميع الجداول الموجودة بقاعدة البيانات بما في ذلك العلاقات والقيود الموجودة بهذه الجداول، إلا أنها لا تقوم بإرجاع أية بيانات موجودة بقاعدة البيانات. ويمكنك في هذه الحالة إضافة الصفوف من خلال التطبيق كما في الكود التالي الذي يقوم بإضافة صف جديد للجدول **Students**:

```
Dim rowTemp As DataRow
Dim tblStudents As DataTable
tblStudents = myds.Tables("Students")
```

إنشاء الصف الجديد'

```
rowTemp = tblStudents.NewRow
rowTemp.Item("St_Id") = "20"
rowTemp.Item("St_fname") = "Sara"
rowTemp.Item("St_lname") = "Elsayed"
```

إضافة الصف'

tblStudents.Rows.Add(rowTemp)

وقد استخدمنا الوظيفة **NewRow()** لنقل المخطط المناسب إلى الكائن **rowTemp**.

من الجدير بالذكر هنا أنك تستطيع إنشاء جدول جديد من خلال الكائن **DataTable**

بمعزلٍ عن كائن مجموعة البيانات **DataSet** كما يمكنك استخدامه مع كلٍ من الوظيفة

Fill() والوظيفة **FillSchema()** المصاحبتين للكائن **DataAdapter** كما في الكود

التالي:

```
tblTemp = New DataTable()
```

```
adStudents.Fill(tblTemp)
```

```
grdStudents.DataSource = tblTemp
```

فإذا كانت متطلبات بياناتك بسيطة، يمكنك في هذه الحالة استخدام الكائن **DataTable**

بدلاً من الكائن **DataSet** الأكثر تعقيداً بالطبع.



الواجب الرابع العمل مع الملفات

١٣. العمل مع الملفات.

١٤. المزيد عن الملفات.