الفصل الثالث عشر
استخدام كائنات ADO.NET

يتم التعامل مع قواعد البيانات داخل نطاق **.NET** من خلال كائنات **ADO.NET** التى تستخدم تقنية مجموعات البيانات **DataSets** فى الوصول إلى البيانات ومعالجتها دون الحاجة إلى الاتصال الدائم بقاعدة البيانات. سنقوم فى هذا الفصل بالتعرف على السمات المختلفة لكائنات **ADO.NET** وكذلك التعرف على كيفية تنفيذ أوامر قاعدة البيانات على أن نستكمل الحديث عن مجموعات البيانات **DataSets** فى الفصل القادم إن شاء الله.

بانتهاء هذا الفصل ستتعرف على:

- تركيب **ADO.NET**.
- الوصول للبيانات فى حالة الانفصال.
- الفرق بين **ADO.NET** و **ADO**.
- مميزات **ADO.NET**.
- تصنيفات **ADO.NET**.
- تنفيذ أوامر قاعدة البيانات

عندما تقوم بتطوير أحد التطبيقات، ربما أردت الاتصال بقاعدة بيانات أو أى مصدر آخر للبيانات لتخزين واسترجاع البيانات الموجودة داخل تطبيقك. والميزة الأساسية لقواعد البيانات عن مصادر البيانات الأخرى هى احتواؤها على مقومات الأمان Security التى تحتوى على مستويات وصول للبيانات بحيث لا يستطيع أى شخص غير مخول له الدخول إلى قاعدة البيانات التعرف على محتوياتها. وإضافةً إلى ذلك، فإن قواعد البيانات تحتوى على العديد من الوظائف المختلفة مقارنةً بمصادر البيانات العادية.

وعند عملك مع التطبيقات، يكون هناك مقدمة للتطبيق تسمى Front End عبارة عن واجهة المستخدم والتى تتدرج من نموذج بسيط إلى صفحة ويب.كما أن هناك مؤخرة للتطبيق Back End عبارة عن قاعدة بيانات أو أى مصدر بيانات آخر. وفى هذه الحالة يلزمك مجموعة من الوظائف التى تحقق عملية الاتصال بين مقدمة التطبيق ونهايته وذلك من خلال تقنية ADO.NET.

وكائنات بيانات ActiveX أو ActiveX Data Objects.NET (ADO.NET) والتى يطلق عليها أيضاً ADO+ عبارة عن مكتبة تحتوى على مجموعة من خدمات الوصول إلى البيانات المستخدمة داخل نطاق NET. وهى تحسين لتقنية ADO المستخدمة مع الإصدارات القديمة من Visual Studio. وحقيقة الأمر أن استخدام ADO.NET لا يقتصر على عمليات قواعد البيانات وإنما يمتد ليشمل التطبيقات المتوافقة مع تقنية OLE DB مثل برنامج Microsoft Excel.

تركيب ADO.NET

يُبنى تركيب ADO.NET أساساً على مبدأ OLE DB وهو عبارة عن إحدى صور واجهة برمجة التطبيقات Application Programming Interface (API) التى من خلالها يتم الوصول إلى مدى واسع من مصادر البيانات. ومصادر البيانات Data Sources عبارة عن بيانات مجردة مصحوبة بأحد أنظمة إدارة قواعد البيانات Database Management System (DBMS).

تقوم OLE DB بتعريف مجموعة من الواجهات حيث تم تصميم تركيب OLE DB لتنظيم الأجزاء المختلفة من قاعدة البيانات بتقسيمها إلى مجموعة من الأجزاء المختلفة التى يمكنك استخدام أى منها فيما بعد، حيث تقوم OLE DB بتقسيم قاعدة البيانات إلى قسمين أساسيين هما مزودات الخدمة OLE DB Service Providers ومزودات البيانات OLE DB Data Providers، حيث تقوم مزودات الخدمة بربط التطبيق بمزودات البيانات، بينما تعمل مزودات البيانات على الجانب الآخر كواجهة للكود الأصلى.

يتم تقسيم وظائف ADO.NET إلى مستوين أساسيين، الأول هو مستوى الاتصال Connected Level والثانى هو مستوى عدم الاتصال أو مستوى الانفصال Disconnected Level، حيث يحتوى المستوى الأول على مجموعة من التصنيفات التى يتم التحكم فيها من خلال NET. ويطلق عليها NET Data Providers. وسوف نتعرض لها بالتفصيل فيما بعد من خلال هذا الفصل. أما المستوى الثانى فيختص بالعمليات التى تتم على مجموعات البيانات DataSets والتى تتكون بدورها من مجموعة من السجلات المصحوبة ببعض الوظائف المستخدمة لإجراء العمليات المختلفة على هذه السجلات مثل الفرز والترتيب وما إلى ذلك من العمليات الأخرى.

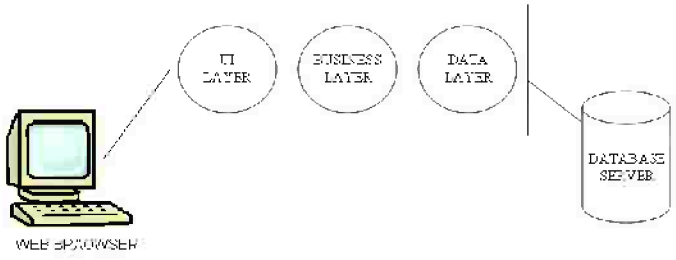
يوجد داخل ADO.NET ثلاث طرق مختلفة للاتصال بمصدر البيانات واسترجاع المعلومات منه وذلك من خلال الأوامر المباشرة Direct Commands أو من خلال مجموعات البيانات DataSet Objects أو من خلال كائنات قارئ البيانات DataReader Objects. وأشهر هذه الطرق طريقة مجموعات البيانات لأنها تمكّن من الوصول إلى البيانات فى حالة عدم الاتصال بقاعدة البيانات.

تحتوى مجموعة البيانات DataSet على نسخة من جزء داخل قاعدة البيانات، حيث تعتمد محتويات هذه النسخة على الأمر أو الاستعلام المستخدم. وبالتالى قد تحتوى المجموعة على قاعدة البيانات بالكامل بما فى ذلك مجموعة الجداول والعلاقات الموجودة بينها، لذا

يوحى استخدام مجموعات البيانات بالاتصال المباشر بقاعدة البيانات. ومن ثمَّ يمكنك التعامل مع بيانات مجموعة البيانات ثم إرجاعها مرةً أخرى إلى قاعدة البيانات. وعلى الرغم من ذلك يعاب على مجموعات البيانات **DataSets** استخدام جزء من الذاكرة، حيث يزداد هذا الجزء بزيادة كمية البيانات المأخوذة من قاعدة البيانات. لذا قامت **ADO.NET** بتوفير طريقة بديلة تتمثل فى قارئ البيانات **DataReader** الذى يقوم باسترجاع البيانات من خلال الاتصال المباشر بقاعدة البيانات وهو مفيد جداً إذا أردت مشاهدة محتويات قاعدة البيانات فقط دون الحاجة إلى إجراء العمليات المختلفة عليها.

الوصول للبيانات فى حالة الانفصال

يعتبر الوصول للبيانات فى حالة الانفصال **Disconnected Data Access** أحد السمات الهامة داخل **ADO.NET**. ولفهم هذه الأهمية، دعنا نتعرف أولاً على نموذج الاتصال التقليدى بين أجهزة الخادم والعميل والذى يتم فيه اتصال كل عميل بخادم قاعدة البيانات طالما كانت هناك حاجة للحصول على البيانات من هذه القاعدة، وبالتالى تتم العمليات المختلفة على قاعدة البيانات مباشرةً من التطبيق إلى خادم قاعدة البيانات. ولكن مع ظهور **WWW** تم استحداث نموذج جديد للاتصال بقاعدة البيانات وهو ما يطلق عليه التطبيق متعدد الصفوف أو **Multi-tier(n-tier) application** والذى يتم فيه تقسيم التطبيق إلى ثلاث طبقات منفصلة هى طبقة واجهة المستخدم **UI Layer** وطبقة الأعمال **Business Layer** وأخيراً طبقة البيانات **Data Layer** (انظر شكل ١٣-١). وفى هذه الحالة لن يكون الاحتفاظ باتصال كل مستخدم طوال الوقت عملياً ولكن بدلاً من ذلك يتم تمرير البيانات بين الطبقات المختلفة أثناء الانفصال عن قاعدة البيانات، ويتم الاتصال بين طبقة البيانات **Data Layer** وخادم قاعدة البيانات للحظات بسيطة حينما نرغب فى استرجاع البيانات أو تحديثها.



شكل ١٣-١ التطبيق متعدد الصفوف

وعلى هذا يحتوى التركيب الجديد على العديد من الميزات والتى أهمها ما يلى:

- *المرونة* ***Flexibility***: لأن تقسيم التطبيق إلى عدد من الطبقات يتيح لنا تغيير أى من منها بأقل مجهود. فمثلاً إذا تم تصميم طبقة البيانات للاتصال بقاعدة بيانات **Sybase**، يمكنك فى أى وقت تغييرها بطبقة بيانات للاتصال بقاعدة بيانات **Microsoft SQL Server** دون إجراء أى تغيير فى الطبقات الأخرى.
- *القدرة على إعادة الاستخدام* ***Reusability***: فمن الممكن أن تستخدم واجهات المستخدم المتعددة، مثل نظام التليفونات أو موقع الويب، نفس طبقة الأعمال **Business Layer** وذلك لتمثيل الطبقات عدةً فى صورة تصنيفات، وبالتالى يمكن استخدامها فى تطبيقات أخرى أو مشاركتها كخدمات ويب.
- *القدرة على الوصول إلى الخادم* ***Scalability***: وهذه أهم الميزات على الإطلاق ويمكن القول بأنها السبب الرئيسى فى تطوير التطبيق متعدد الصفوف **n-tier application**، لأن هذا أفسح المجال لاتصال أكبر عدد من المستخدمين بقاعدة البيانات والاستفادة منها فى نفس الوقت دون أن يقلل هذا من كفاءة خادم قاعدة البيانات.

الفرق بين ADO.NET و ADO

على الرغم من التشابه الكبير بين كلٍ من **ADO.NET** و**ADO**، فكلاهما يمكّن المستخدم من الوصول إلى مؤخرة التطبيق **Back End** الممثلة فى قاعدة البيانات من مقدمة التطبيق

Front End الممثلة فى واجهة المستخدم، إلا أن هناك بعض الاختلافات البسيطة، حيث لا تعد ADO.NET تحسيناً من ADO وحسب وإنما عبارة عن نموذج اتصال للبيانات تم تصميمه خصيصاً من أجل الوصول إلى البيانات واستخدامها من خلال حالة عدم الاتصال Disconnected State وهذا ما لم يكن موجوداً بشكل كامل داخل ADO. ولذلك تستخدم ADO.NET كائن مجموعة البيانات DataSet لتحقيق هذه العملية بدلاً من كائن مجموعة السجلات RecordSet المستخدمة فى ADO.

يمكنك التعرف على الاختلافات الجوهرية بين كل من ADO.NET وADO عن طريق النقاط الآتية:

- الوصول إلى البيانات Data Access
- العلاقات متعددة الجداول Multi-table Relationships
- الوصول للبيانات فى حالة عدم الاتصال Disconnected Access
- تدعيم لغة الترميز الممتدة XML

الوصول إلى البيانات

فى ADO، تحتوى مجموعة السجلات Recordset على نسخة من البيانات فى صورة جدول واحد داخل الذاكرة. أما فى ADO.NET فتحتوى مجموعة البيانات Dataset على نسخة من البيانات داخل الذاكرة ولكن ربما تكون فى أكثر من جدول.

يتم التحكم فى الوصول إلى البيانات داخل ADO.NET من خلال مزودى البيانات .NET Data Providers. كما تقوم كل من ADO.NET وADO بالوصول إلى قاعدة البيانات من خلال كائن اتصال. ويتم إنشاء الاتصال فى ADO.NET من خلال كائن التصنيف SQLConnection أو التصنيف OLEDBConnection تبعاً لنوع مصدر البيانات كما سنوضح بعد قليل. والفرق بين كائنات اتصال كل من ADO.NET وADO أن الأول لا يدعم خاصية مكان المؤشر CursorLocation حيث يتم فى ADO استرجاع البيانات من خلال هذه الخاصية (راجع الفصل السابق)، أما فى ADO.NET

فيتم تجريد البيانات عن مصدرها حيث يحتوى **ADO.NET** على واجهة برمجة منفصلة للوصول إلى البيانات. وعلى الرغم من أن افتقاد تدعيم خاصية مكان المؤشر قد يبدو أحد العيوب الموجودة فى **ADO.NET** إلا أنها تحتوى على العديد من الوظائف التى تقوم بها هذه الخاصية فى **ADO**.

العلاقات متعددة الجداول

فى **ADO**، يتم استخدام الاستعلام **JOIN** لتجميع البيانات من أكثر من جدول، أما فى **ADO.NET** فيتم استخدام الكائن **DataRelation** الذى يتيح ربط الصفوف المتوافقة من كائنات **DataTable**، حيث يمثل كائن **DataTable** مصدر بيانات يتكون من صفوف وأعمدة. كما يتم تجريد كائنات **DataTable** من مصدر البيانات لإجراء العمليات المختلفة عليها كالفرز والترتيب والتحرير ... الخ.

يمكنك من خلال كائن **DataRelation** تمثيل العلاقات بين الجداول من خلال الأعمدة التى تحتوى على نفس نوع البيانات. والعلاقات فى الواقع عبارة عن ارتباطات ديناميكية يتم فيها التأكد من عدم تغيير شروط هذه العلاقات وإلا يتم إظهار رسائل الخطأ المناسبة أثناء التشغيل. فعلى فرض أن هناك علاقة ارتباط بين الجدول **Books** الذى يحتوى على بيانات الكتب والجدول **Authors** الذى يحتوى على بيانات المؤلفين وذلك من خلال الحقل **BookID**. فإذا حاولت تعيين **BookID** داخل جدول **Authors** مع عدم وجود هذه القيمة داخل جدول **Books**، يتم على الفور ظهور رسالة الخطأ المناسبة. وهذا الأمر مشابه تماماً لعمل المفتاح الأساسى والمفتاح الأجنبى فى قواعد البيانات **ODBC** مثل قاعدة البيانات **Access** على سبيل المثال.

ويعمل هذا النوع من العلاقات على تسريع عملية البحث عن السجلات. فعملية البحث عن السجلات داخل **ADO** تتم فى شكل تسلسلى بدءاً من السجل الأول وحتى الوصول إلى السجل المطلوب. أما فى **ADO.NET** فإن هذه العلاقات تساعدك على الوصول المباشر إلى السجل مما يقلل من الزمن اللازم للوصول إلى السجل المطلوب.

يمكنك تمثيل العلاقات بين الجداول المتعددة بسهولة شديدة. تخيل أن لدينا قاعدة بيانات باسم Books تحتوى على عمود (حقل) باسم BookID داخل الجدولين Books وAuthors كما ذكرنا منذ قليل، يمكنك إنشاء العلاقة بين الجدولين داخل Visual Basic باستخدام هذا الحقل كما يلى:

```
1.  Dim ds As DataSet = new DataSet("Books")
2.  ds.Relations.Add("BkID", ds.Tables["Books"].Columns
    ["BookID"] , ds.Tables["Authors"].Columns["BookID"])
```

وعن هذا الكود، نوضح ما يلى:

- فى السطر رقم ١ تم تعريف مجموعة بيانات جديدة تمثل قاعدة البيانات Books.
- فى السطر رقم ٢ تم إضافة العلاقة BKID إلى قائمة العلاقات الموجودة داخل قاعدة البيانات Books المشار إليها بواسطة مجموعة البيانات ds وذلك من خلال الوظيفة ()Add التى تحتوى على ثلاثة معاملات. يعبر الأول عن اسم العلاقة والثانى عن اسم حقل الارتباط فى الجدول الأول بينما يعبر المعامل الثالث والأخير عن اسم حقل الارتباط فى الجدول الثانى.

الوصول للبيانات فى حالة عدم الاتصال

يعتبر الوصول للبيانات فى حالة عدم الاتصال أحد المميزات الأساسية فى ADO.NET. وعلى الرغم من أنك تستطيع توفيق مجموعات السجلات RecordSets فى ADO كى يتم الوصول إليها فى حالة عدم الاتصال، إلا أنها لا تحتوى على جميع الوظائف الموجودة فى مجموعات البيانات DataSets فى ADO.NET وذلك لأن RecordSets تم تصميمها أساساً للوصول للبيانات فى حالة الاتصال وليس فى حالة عدم الاتصال.

تدعيم لغة الترميز الممتدة XML

تستخدم ADO.NET تنسيق XML فى نقل البيانات حتى يتم تمرير البيانات بين الأجزاء المختلفة دون الحاجة إلى إجراء عمليات التحويل، حيث أن هذا التنسيق معرَّف من قِبَل جميع المكونات.

تدعم ADO استخدام XML كما رأينا فى المثال التطبيقى الذى شرحناه فى نهاية الفصل السابق، إلا أن هذا التدعيم يقتصر على عمليات الإدخال والإخراج فقط. أما فى ADO.NET فيمكنك استخدام XML لمعالجة وترتيب ومشاركة البيانات.

مميزات ADO.NET

تحتوى ADO.NET على العديد من الميزات والتى أهمها على الإطلاق ما يلى:

- استخدام مجموعات البيانات DataSets فى الوصول إلى البيانات ومن ثمّ تقليل زمن الاتصال بين مجموعة البيانات وقاعدة البيانات مما يساعد قاعدة البيانات على تدعيم عدد أكبر من الاتصالات.
- تقوم ADO.NET بفصل مصدر البيانات (قاعدة البيانات) عن مجموعة البيانات DataSet مما يمنح المبرمج مرونة أكبر فى التعامل مع هذه البيانات. ففى ADO ترتبط إمكانيات مجموعة السجلات RecordSet بإمكانيات مصدر البيانات نفسه، أما فى ADO.NET فلكلٍ من مصدر البيانات ومجموعة البيانات إمكانياتها المستقلة وسلوكها المستقل.
- يستخدم ADO نوع البيانات Variant فى معالجة البيانات مما يؤدى إلى بطئ عمليات المعالجة، أما ADO.NET فيستخدم كائنات الأنواع القوية الأخرى.

تصنيفات ADO.NET

تستخدم مزودات البيانات Data Providers فى إنشاء الاتصال بقاعدة البيانات والحفاظ على هذا الاتصال حتى يتم تنفيذ العمليات المطلوبة. يحتوى ADO.NET على نوعين من مزودات البيانات، الأول هو OLE DB.NET Data Provider والثانى هو SQL Server .NET Data Provider (وهو ما لم يكن موجوداً فى ADO). وكل من هذه المزودات يحتوى على مجموعة من التصنيفات التى تمثل مع بعضها العمود الفقرى لـ ADO.NET،حيث يتم تجميع هذه التصنيفات داخل المسمى System.Data.OLEDB

للوصول إلى بيانات OLE DB، والمسمى System.Data.SQLClient للوصـــول إلى بيانات SQL Server.

وحقيقةً هناك تشابهاً كبيراً بين وظائف التصنيفات الموجودة بالمسميين، إلا أن الاختلاف يكمن أساساً فى تمثيل كل مسمى لتصنيفاته. كما أن تصنيفات المسمى System.Data.OLEDB تستطيع التعامل مع جميع مصادر البيانات بينما تستطيع تصنيفات المسمى Syatem.Data.SQLClient التعامل مع قواعد البيانات SQL Server فقط. فالتصنيف DataSet على سبيل المثال أحد التصنيفات المشتركة بين المسميين وتحتوى كائناته دائماً على نسخة مؤقتة من قاعدة البيانات كما ذكرنا منذ قليل، حيث يقوم ADO.NET بتطبيق أى تغييرات على مجموعة البيانات DataSet ثم يقوم مزود البيانات بنقل هذه التغييرات إلى مصدر البيانات الحقيقى الذى غالباً ما يكون قاعدة بيانات.

يوضح جدول ١٣-١ التالى التصنيفات الموجودة داخل ADO.NET ووظيفة كل منها، حيث يحتوى الجدول على تصنيفات المسمى System.Data.OLEDB وهـــى نفـــس التصنيفات الموجودة داخل المسمى System.Data.SQLClient.

جدول ١٣-١ التصنيفات الموجودة بالمسمى System.Data.OLEDB

التصنيف	الوصف
OLEDBConnection	يمثل الاتصال الحى بمصدر البيانات ويتكافأ مع التصنيف SQLConnection داخل المسمى System.Data.SQLClient
OLEDBDataReader	يمثل طريقة لقراءة مجموعة من السجلات الموجودة داخل مصدر البيانات حيث يتم إنشاء كائن جديد ينتمى لهذا التصنيف بمجرد استدعاء الوظيفة ExecuteReader الموجودة داخل التصنيف OLEDBCommand ويتكافأ مع التصنيف

التصنيف	الوصف
	SQLDataReader داخل المسمى System.Data.SQLClient
DataSet	تستخدم كمخزن للبيانات فى حالة عدم الاتصال بقاعدة البيانات، ومن الممكن أن تحتوى على جداول وعلاقات متعددة. ويستخدم بنفس الاسم داخل المسمى System.Data.SQLClient
OLEDBCommand	عبارة عن أمر استعلام من مصدر البيانات فى صورة إجراء مخزَّن أو عبارة SQL ويتكافأ مع التصنيف SQLCommand داخل المسمى System.Data.SQLClient
OLEDBDataAdapter	يمثل هذا التصنيف عملية الاتصال بمصدر البيانات ويحتوى على مجموعة من الأوامر المستخدمة فى ملء مجموعة البيانات كما يقوم هذا التصنيف أيضاً بتحديث قاعدة البيانات أى يستخدم لتحقيق التزامن بين سجلات مصدر البيانات وتلك الموجودة داخل الكائن DataSet، ويتكافأ مع التصنيف SQLDataAdapter داخل المسمى System.Data.SQLClient
OLEDBErrorCollection	يمثل هذا التصنيف مجموعة الأخطاء التى تحدث أثناء عملية الاتصال ويتكافأ مع التصنيف SQLErrorCollection داخل المسمى System.Data.SQLClient
OLEDBException	يمثل الاستثناء الذى يصدر فى حالة حدوث خطأ أو تحذير ويتكافأ مع التصنيف SQLException

التصنيف	الوصف
	داخل المسمى System.Data.SQLClient
OLEDBParameter	يمثل أحد المعاملات المرتبطة بأحد أوامر قاعدة البيانات الممثل فى التصنيف OLEDBCommand ويتكافأ مع التصنيف SQLParameter داخل المسمى System.Data.SQLClient
DataTable	ويمثل جدول من البيانات يحتوى على مجموعة من الصفوف والأعمدة، ويستخدم بنفس الاسم داخل المسمى System.Data.SQLClient

إذا أردت الوصول إلى قاعدة بيانات Microsoft SQL Server، استخدم تصنيفات ADO.NET ذات البادئة SQL. أما إذا أردت الوصول إلى قواعد البيانات الأخرى، فاستخدم التصنيفات ذات البادئة OLEDB.

تنفيذ أوامر قاعدة البيانات

يتم تنفيذ أوامر قاعدة البيانات كما تعرف باستخدام عبارات SQL أو الإجراءات المخزَّنة Stored Procedures وذلك من خلال التصنيفات Connection وCommand كما سنعرف فى الأجزاء التالية.

الاتصال بقاعدة البيانات

للحصول على البيانات من قاعدة البيانات أو تحديث هذه البيانات، يجب أن تقوم أولاً بالاتصال بقاعدة البيانات تماماً كما تقوم بالدخول إلى موقع الويب المحبب إليك من خلال مستعرض الويب المثبت على حاسبك. ففى ADO.NET يتم إنشاء الاتصال باستخدام تصنيف الاتصال المناسب تبعاً لنوع البيانات التى ترغب فى الاتصال بها كما أوضحنا من قبل سواءً كانت SQL Server أو OLEDB، وذلك من خلال الوظيفة ()Open

لإجراء الاتصال والوظيفة Close() لإنهاء هذا الاتصال وما بينهما يتم إجراء العمليات المختلفة على قاعدة البيانات. يوضح الكود التالى كيفية الاتصال بقاعدة بيانات SQL Server من خلال التصنيف SQLConnection وذلك كما يلى:

```
Dim strInfo As String
Dim con As SqlConnection
strInfo = "Server=localhost;uid=ledo;pwd=;database=Students"
con = New SqlConnection()
con.ConnectionString = strInfo
con.Open()

إجراء العمليات المختلفة على قاعدة البيانات,

con.Close()
```

والأهم فى هذا الكود من الوظيفة Open() أو الوظيفة Close() هو إنشاء واستخدام سلسلة الاتصال التى تحتوى على المعاملات اللازمة لإجراء عملية الاتصال بقاعدة البيانات بشكل صحيح، حيث تختلف محتويات هذه السلسلة باختلاف نوع قاعدة البيانات المستخدمة كما أوضحنا بالتفصيل فى الفصل السابق وضربنا الأمثلة الكافية على ذلك. فعلى سبيل المثال إذا أردنا الاتصال بقاعدة بيانات OLE DB بدلاً من قاعدة البيانات SQL Server، يصبح الكود السابق كما يلى:

```
Dim strInfo As String
Dim con As OleDbConnection
strInfo = "Provider= Microsoft.Jet.OLEDB.4.0; Data Source = d:\Students.mdb"
con = New OleDbConnection()
con.ConnectionString = strInfo
con.Open()
إجراء العمليات المختلفة على قاعدة البيانات,
con.Close()
```

سنقوم فى الأمثلة القادمة بذكر إحدى الصيغتين فقط وعليك إتماماً للفائدة تحويل المثال إلى الصيغة الأخرى.

وكما ترى من الكود السابق فإن عملية إنشاء الاتصال بقاعدة البيانات أو قطع هذا الاتصال غاية فى السهولة، لكن المهم هنا ألا تنسى إغلاق عملية الاتصال بمجرد انتهاء عملك مع قاعدة البيانات، حيث يستهلك كل اتصال من الاتصالات التى تجريها على قاعدة البيانات قدراً لا بأس به من موارد النظام والشبكة فى نفس الوقت. ولكن يظهر هنا سؤالٌ هام وهو ماذا يحدث لو حدث خطأ ما (يسمى استثناء **Exception**) فى المنطقة بين استدعاء الوظيفة **()Open** واستدعاء الوظيفة **()Close**؟ فى هذه الحالة يجب مراعاة ذلك من خلال استخدام التركيب **Try ... Catch ... Finally** كما يلى:

```
Dim strInfo As String
Dim con As OleDbConnection
strInfo = "Provider= Microsoft.Jet.OLEDB.4.0; Data Source = d:\Students.mdb"
Try
        con = New OleDbConnection()
        con.ConnectionString = strInfo
        con.Open()

                إجراء العمليات المختلفة على قاعدة البيانات,
Catch exc As OleDbException
                يتم هنا احتواء الاستثناء'
Finally
        If con.State = ConnectionState.Open Then
                con.Close()
        End If
End Try
```

من الممكن حدوث أخطاء (استثناءات) فى الكود السابق فى حالتين، إما أثناء إنشاء عملية الاتصال أو أثناء إجراء بعض العمليات على قاعدة البيانات، لذا يتم وضع كل هذه العمليات داخل الجزء **Try**. وإذا ظهر أحد الاستثناءات، فربما يكون الاتصال مفتوحاً أو مغلقاً تبعاً لمكان وقوع الاستثناء، لذا يتم إغلاق قاعدة البيانات ذلك الجزء **Finally** الذى يتم تنفيذه سواءً وقع الاستثناء أم لم يقع. ولكن يتم أولاً اختبار حالة الاتصال

ConnectionState لأن إغلاق عملية اتصال مغلقة بالفعل يتسبب فى ظهور استثناء آخر.

إعداد كائن الأمر

بعد إجراء عملية الاتصال مع قاعدة البيانات، يمكنك تنفيذ الأوامر المختلفة على هذه القاعدة وذلك من خلال الوظائف المختلفة الموجودة بالتصنيفين SQLCommand وOleDbCommand تبعاً لنوع قاعدة البيانات المستخدمة، حيث يتم أولاً تعيين خصائص الكائن بما فى ذلك عبارة SQL المستخدمة، وبعد ذلك يتم مصاحبة هذا الكائن بكائن الاتصال واستدعاء الوظيفة المناسبة لتنفيذ الأمر.

تحديد نص الأمر

يتم تخزين عبارة SQL داخل الخاصية CommandText كما فى الكود التالى:

```
Dim cmd AS OleDbCommand
cmd = New OleDbCommand()
cmd.CommandText = "SELECT * FROM Students"
```

وكما تعلم، يمكنك استخدام الإجراءات المخزَّنة بدلاً من عبارات SQL للحصول على المزيد من الكفاءة. وفى هذه الحالة لا يلزمك فقط تعيين اسم الإجراء المخزَّن داخل الخاصية CommandText وإنما يجب أيضاً تعيين نوع الأمر داخل الخاصية CommandType كما فى الكود التالى:

```
Dim cmd AS OleDbCommand
cmd = New OleDbCommand()
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = "spGetStudentByID"
```

يوضح جدول ١٣-٢ التالى القيم المختلفة للخاصية CommandType واستخدام كلٍ منها.

جدول ١٣-٢ قيم الخاصية CommandType

القيمة	الاستخدام
Text	هى القيمة الافتراضية المستخدمة مع جميع أنواع عبارات

القيمة	الاستخدام
	SQL غير الإجراء المخزَّن مثل العبارات SELECT وINSERT
StoredProcedure	إذا اخترت هذه القيمة، فإن ASP.NET يتوقع وجود اسم الإجراء المخزَّن الذى ترغب فى تنفيذه داخل الخاصية CommandText
TableDirect	تكافئ هذه القيمة عبارة SQL التالية: SELECT * FROM *tablename* حيث تحتوى الخاصية CommandText على اسم هذا الجدول، ويستخدم هذا الخيار مع الكائنات OleDbCommand فقط، ومن الأفضل تجنبها قدر الإمكان

تعيين خاصية الاتصال

قبل أن تقوم بتنفيذ الأمر، تحتاج إلى تعيين الخاصية Connection المصاحبة لكائن الاتصال النشط كما فى الكود التالى:

```
Dim cmd AS New OleDbCommand("Select * FROM Students")
Dim con As New OleDbConnection ("Provider= Microsoft. Jet.
OLEDB.4.0; Data Source = d:\Students.mdb")
con.Open()
cmd.Connection = con
```

لاحظ أننا قمنا بتخصيص عبارة الأمر وسلسلة الاتصال مباشرةً إلى الكائنات cmd وcon على الترتيب أثناء عملية التعريف وهذه طريقة أخرى غير التى ذكرناها منذ قليل. كما أننا قمنا بتعيين الخاصية Connection بعد فتح عملية الاتصال، ولا بأس فى ذلك فالمهم هو تعيين عملية الاتصال قبل تنفيذ الأمر نفسه.

إنشاء معاملات الأمر

تحتوى معظم الإجراءات المخزَّنة وبعض عبارات SQL على معامل Parameter أو أكثر. والمعامل عبارة عن قيمة يتم وضعها بدلاً من رمز معين داخل عبارة SQL قبل تنفيذها. وتستخدم هذه المعاملات خاصةً داخل الإجراءات المخزَّنة لأنها تعمل بطريقة مشابهة لعمل الدوال داخل Visual Basic. فكل ما يحتاجه البرنامج لعملية الاستدعاء هو اسم الإجراء المخزَّن والمعامل أو المعاملات المستخدمة حيث يتم تنفيذ عبارة SQL الحقيقة داخل قاعدة البيانات لذا يمكنك تعديلها فى أى وقت دون الحاجة إلى إعادة ترجمة البرنامج.

يتم تمثيل معامل الإجراء المخزَّن داخل ADO.NET فى التجمع Parameters المصاحب للكائن Command. يوضح جدول ١٣-٣ التالى أهم خصائص الكائن Parameter.

جدول ١٣-٣ خصائص الكائن Parameter

الخاصية	الاستخدام	مثال
Name	اسم المعامل	@strID
Type	نوع بيانات المعامل	String
Length	حجم نوع بيانات المعامل	11
Direction	اتجاه استخدام المعامل (إدخال أم إخراج)	Input
Value	القيمة المراد استخدامها داخل الاستعلام	123

يمكنك إضافة المعاملات إلى التجمع Parameters المصاحب للكائن Command باستخدام الوظيفة Add() كما فى الكود التالى:

```
cmd.CommandType = CommandType.StoredProcedure
cmd.CommandText = "spGetStudentByID"

Dim pmTemp As OleDbParameter
pmTemp = cmd.Parameters.Add(New OleDbParameter("@strID",
OleDbType.Char,8))
pmTemp.Direction = ParameterDirection.Input
```

```
pmTemp.Value = "120"
```

وكما ترى فقد قمنا بتعيين الخصائص Name وType وLength داخل دالة إنشاء الكائن pmTemp، كما يوضح الاتجاه Input أن المعامل عبارة عن معامل إدخال للإجراء المخزَّن. وتحدد القيمة Value رقم الطالب الذى نرغب فى البحث عنه. وعلى الرغم من أننا قمنا بتعيين خمس خصائص للمعامل، إلا أن ADO.NET يحتاج حقيقةً إلى خاصيتين فقط وهما الاسم Name والقيمة Value على أن يقوم بمعرفة بقية الخصائص بنفسه. لذا يمكنك استبدال الخمسة سطور الأخيرة فى الكود السابق بالسطر التالى فقط:

```
cmd.Parameters.Add(New OleDbParameter("@strID","120"))
```

يجب أن يتطابق الكائن Parameter الذى تقوم بإعداده داخل Visual Basic مع اسم ونوع بيانات المعامل المصاحب داخل الإجراء المخزن. كما أنك لو قمت باستخدام عدد مختلف من المعاملات أو نوع بيانات غير صحيح، فربما تم طرح الاستثناء المناسب.

على الرغم من أن معظم معاملات الإجراءات المخزَّنة عبارة عن معاملات إدخال، إلا أنك تستطيع استخدام المعامل فى إرجاع البيانات إلى البرنامج، حيث يطلق على هذا النوع من المعاملات "معاملات الإخراج" Output Parameters وتفيد حينما ترغب فى استرجاع قيمة حقل أو حقلين من قاعدة البيانات وليس الحقل بالكامل. يوضح الإجراء المخزَّن التالى مثالاً على استخدام معامل الإخراج:

```
CREATE PROCEDURE spGetMaxAge
@strState char(2),
@intResult int OUTPUT
As
Select @intResult = Max(St_Age) FROM Students
WHERE State = @strState
```

ففى هذا الكود، يستخدم الإجراء المخزَّن spGetMaxAge معامل إخراج باسم @intResult لتخزين أكبر عمر داخل الجدول Students حيث استخدمنا معه كلمة OUTPUT التى تدل على أنه معامل إخراج. يوضح الكود التالى كيفية إعداد الكائن Parameter بالمعامل @intResult:

```
pmTemp = cmd.Parameters.Add(New OleDbParameter(
"@intResult", OleDbType.Int))
pmTemp.Direction = ParameterDirection.Output
```

وكما ترى يتم تعيين معامل الإخراج داخل أمر ADO.NET بتخصيص القيمة ParameterDirection.Output للخاصية Direction. وبعد تنفيذ الأمر؛ كما سنشرح بعد قليل، يمكنك اختبار الخاصية Value المصاحبة لمعامل الإخراج للتعرف على القيمة الناتجة كما فى الكود التالى:

```
Dim intMaxAge As Integer
intMaxAge = Convert.ToInt32(cmd.Parameters("@intResult").
Value)
MessageBox.Show("The maximum age is " & intMaxAge)
```

وقد قمنا بتحويل قيمة المعامل إلى النوع Integer لتخزينها فى متغير من هذا النوع حيث تقوم الخاصية Value بتخزين قيمة من النوع Object.

تعتبر القيمة ReturnValue من القيم المستخدمة مع الخاصية Direction والتى تستخدم لاسترجاع نتيجة العبارة Return من الإجراء المخزَّن.

اختيار وظيفة تنفيذ الأمر

بعد الانتهاء من تعيين خصائص الأمر سواءً كان SQLCommand أو OleDbCommand وكذلك تعيين المعاملات الضرورية، يمكنك تنفيذ هذا الأمر باستدعاء إحدى وظائف التنفيذ الآتية:

- الوظيفة ExecuteNonQuery() وتقوم بتنفيذ عبارات SQL التى لا تقوم بإرجاع سجلات من قاعدة البيانات مثل العبارات DELETE وUPDATE وINSERT. كما يمكنك استخدامها أيضاً مع الإجراءات المخزَّنة التى تقوم بإرجاع النتيجة من خلال المعاملات.
- الوظيفة ExecuteReader() وتقوم بتنفيذ عبارة SQL وإرجاع كائن SqlDataReader يحتوى على السجلات الناتجة.

- الوظيفة **ExecuteScalar()** وتستخدم حينما تريد إرجاع قيمة واحدة (عمود) من البيانات من خلال عبارة **SQL SELECT**.
- الوظيفة **ExecuteXMLReader()** وتشبه إلى حد كبير الوظيفة **ExecuteReader()** إلا أنها متاحة مع الكائن **SQLCommand** فقط وتقوم بإرجاع كائن من النوع **XMLReader**.

ولعل السبب فى وجود أكثر من وظيفة لتنفيذ الأوامر يرجع أساساً إلى تناسب كل وظيفة من هذه الوظائف مع نوع معين من عبارات **SQL**. وعلى الرغم من حرية الاختيار من بين الوظائف الأربعة، إلا أنه من الضرورى اختيار الوظيفة المناسبة للوصول إلى أعلى كفاءة تنفيذ.

يمكنك استخدام الوظيفة **CommandTimeout()** لتعيين أكبر وقت لتنفيذ الأمر بالثوانى والتى يتم بعدها طرح الاستثناء المناسب.

- *تنفيذ العبارات التى لا تقوم بإرجاع سجلات*

لأن العبارة **DELETE** على سبيل المثال لا تقوم بإرجاع أى سجلات من قاعدة البيانات وإنما تقوم على العكس بحذف بعض السجلات منها، فإن تعريف متغير يحتوى على مجموعة السجلات الناتجة مضيعة للوقت وحسب. لذا فالخيار الأنسب هو استخدام الوظيفة **ExecuteNonQuery()** كما فى الكود التالى:

```
Private Function DeleteEmployee(ByVal strSSN As String, ByVal
strDept As String) As Integer
    Dim intRowsDeleted As Integer
    Dim strDeleteSQL As String
    Dim cmd As New OleDbCommand()
    Dim con As New OleDbConnection()

    'BUILD THE SQL STRING
    strDeleteSQL = "DELETE FROM Students WHERE
                                        St_fname='W%' "
    'SET UP COMMAND AND CONNECTION
    con.ConnectionString = "Provider= Microsoft. Jet.
```

```
            OLEDB.4.0; Data Source = d:\Students.mdb"
    con.Open()
    cmd.Connection = con
    cmd.CommandText = strDeleteSQL

    'EXECUTE COMMAND
    intRowsDeleted = cmd.ExecuteNonQuery()

    'CLOSE CONNECTION, RETURN NUMBER OF ROWS
    cn.Close()
    Return intRowsDeleted

End Function
```

فى هذا الكود، قمنا باستخدام الوظيفة **ExecuteNonQuery()** لحذف سجل من الجدول **Students** حيث تقوم هذه الوظيفة بإرجاع عدد السجلات التى تأثرت بالأمر أى التى تم حذفها.

كما يعتبر الإجراء المخزَّن **spGetMaxAge** الذى أنشأناه فى البند السابق من الأمثلة التى يكون من الأنسب معها استخدام الوظيفة **ExecuteNonQuery()** وذلك لأنها لا تقوم بإرجاع سجلات وإنما تستخدم معامل إخراج **Output Parameter**.

- ***استخدام قارئ البيانات Data Reader***

حينما تقوم بتنفيذ استعلام **SQL SELECT** أو إجراء مخزن يقوم بإرجاع سجلات، فمن الضرورى استخدام وظيفة التنفيذ التى تتيح لك الوصول إلى هذه السجلات. وتعتبر الوظيفة **ExecuteReader()** من أسهل هذه الوظائف والتى تقوم بإرجاع كائن من النوع **DataReader** والذى يشبه إلى حدٍ ما الكائن **Recordset** فى الإصدارات السابقة من **ADO** مع وجود بعض الاختلافات والتى من أهمها عدم القدرة على استخدام الكائن **DataReader** فى حالة عدم الاتصال بقاعدة البيانات.

وقد تم تصميم التصنيفات **SQLDataReader** و**OleDbDataReader** لاسترجاع السجلات من قاعدة البيانات فى أسرع وقت ممكن ومعاملة هذه السجلات بشكل

تسلسلى، لذا تقتصر عملية الانتقال بين السجلات على التحرك إلى الأمام فقط. يوضح الكود التالى كيفية استخدام قارئ البيانات.

```
Dim rdrStudent As OleDbDatatReader
Dim con As New OleDbConnection("Provider= Microsoft. Jet.
        OLEDB.4.0; Data Source = d:\Students.mdb")
Dim cmd As New OleDbCommand("SELECT St_fname,
                St_Iname, St_Age FROM Students", con)
Dim strName As String

con.Open()

rdrStudent = cmd.ExecuteReader()
While rdrStudent.Read()
        'Build string containing first and last name
        strName = rdrStudent.GetString(0).Trim & " " &
                                rdrStudent.GetString(1).Trim
        'Display Name and Age in a message box
        MessageBox.Show("Hello, " & strName & ". You are " &
                rdrStudent.Item("Age").ToString & " years old!")
End While

con.Close()
```

يقوم هذا الكود بالاستعلام عن جدول **Students** ويقوم بإرجاع كائن **OleDbDataReader**، كما يقوم كل استدعاء للوظيفة **rdrStudent.Read()** بتوجيه مؤشر قارئ البيانات إلى السجل التالى. فإذا قام الاستعلام السابق باسترجاع خمسة سجلات على سبيل المثال، فيتم تنفيذ الدوارة **While** خمس مرات أيضاً، كما تقوم الوظيفة **Read()** بإرجاع القيمة **False** بعد معالجة جميع السجلات ومن ثمَّ يتم الخروج من الدوارة **While**.

يوضح الكود السابق أيضاً طريقتين للوصول إلى قيم الأعمدة (الحقول) فى الصف الحالى لقارئ البيانات. الطريقة الأولى باستخدام الخاصية **Item** مع تعيين اسم الحقل أو دليله. أما الطريقة الثانية فمن خلال إحدى وظائف الكائن **OleDbDataReader** التى ذات

البادئة Get كالوظيفة ()GetString التى استخدمناها فى الكود السابق لأننا نعلم أن حقلى الاسم الأول والاسم الأخير يحتويان على بيانات من النوع String.

بجانب الوظائف والخصائص المصاحبة لقارئ البيانات التى استخدمناها فى الكود السابق، هناك العديد من الوظائف والخصائص الأخرى ومن أهمها ما يلى:

- الوظيفة ()GetName وتقوم بإرجاع اسم حقل البيانات. ففى الكود السابق مثلاً، تقوم الوظيفة (0)GetName بإرجاع الاسم الأول للطالب.
- الخاصية FieldCount وتقوم بإرجاع عدد حقول البيانات المستخدمة فى قارئ البيانات. فالكود السابق على سبيل المثال يحتوى على ثلاثة حقول (أعمدة).
- الوظيفة ()IsDBNull وتستخدم لاختبار وجود قيمة Null داخل أحد أعمدة قاعدة البيانات.

وبالإضافة إلى العمل مع السجلات المتعددة، يمكنك استخدام الكائن DataReader لاسترجاع أكثر من نتيجة فى صورة تسلسلية والتنقل بين هذه النتائج من خلال الوظيفة ()NextResult. وخير مثال على ذلك الإجراء المخزن الذى يحتوى على أكثر من عبارة SELECT.

- *استرجاع القيم المفردة*

نرغب أحياناً فى استرجاع قيمة واحدة فقط من خلال إجراء مخزن بدلاً من استرجاع السجل بالكامل. وقد شرحنا فيما سبق كيفية أداء ذلك من خلال معامل الإخراج فى الإجراء المخزن. إلا أنك تستطيع إرجاع قيمة واحدة أيضاً من خلال استعلام SQL. وخير مثال على ذلك، عمود الهوية (التعريف) Identity Column الذى يتشابه كثيراً مع حقل الرقم التلقائى الموجود بقاعدة البيانات Access والذى يتم إنشاؤه تلقائياً داخل قاعدة البيانات ويعمل غالباً كمفتاح أساسى. وللتعرف على مفهوم عمود التعريف، انظر معى إلى الكود التالى الذى يقوم بإنشاء جدول باسم WebUsers يحتوى على عمود تعريف وذلك من خلال العبارة CREATE TABLE كما يلى:

```
CREATE TABLE WebUsers (
```

```
UserNumber int IDENTITY (1,1) NOT NULL PRIMARY KEY,
UserID char(10) NOT NULL,
Password char(10) NOT NULL,
UserDescription varchar (50) NULL,
AccessLevel tinyint NOT NULL
)
ALTER TABLE WebUsers ADD CONSTRAINT UniqueUserID
UNIQUE NONCLUSTERED (UserID)
```

فى هذا الكود يعتبر الحقل **UserNumber** مفتاح أساسى وعمود تعريف فى نفس الوقت. وعلى ذلك حينما يتم إضافة السجلات إلى الجدول، تقوم قاعدة البيانات بتخصيص قيمة صحيحة لهذا الحقل بدءاً بالرقم ١ مع زيادة هذا الرقم مع كل سجل جديد، وبالتالى يمكنك استخدام هذا العمود كحقل أجنبى **Foreign Key** بالجداول الأخرى. والميزة من تحديد قيمة الحقل من قِبَل قاعدة البيانات بدلاً من أتقوم بتعريفها بنفسك فى أن قاعدة البيانات تلبى متطلبات المستخدم مع الاحتفاظ بأرقام فريدة فى هذا الحقل.

إذا أردت إذاً تحديث أكثر من جدول أثناء إضافة سجل للجدول **WebUsers**، تحتاج إلى معرفة قيمة الحقل **UserNumber**. يوضح الكود التالى كيفية إضافة سجل إلى الجدول ثم استرجاع قيمة الحقل **UserNumber** الخاص بهذا السجل وذلك من خلال الإجراء المخزن **spWebUserInsert**:

```
CREATE PROCEDURE spWebUserInsert
@strUserID char(10),
@strPassword char(10),
@strDescription varchar(50),
@intAccessLevel tinyint
As
INSERT INTO WebUsers
VALUES (@strUserID, @strPassword,
@strDescription,@intAccessLevel)
SELECT @@IDENTITY
```

يقوم الإجراء المخزن أولاً بتنفيذ العبارة **INSERT** باستخدام قيم المعاملات وبعد ذلك يتم اختيار المتغير النظامى **@@IDENTITY** الذى يحتوى على قيمة التعريف التى تم إضافتها

مؤخراً. ومن ثمَّ يقوم هذا الإجراء بإرجاع سجل يحتوى على حقل واحد فقط به قيمة التعريف. يمكنك بالطبع استرجاع هذه القيمة من خلال الوظيفة ExecuteReader() كما أوضحنا من قبل. ولكن كى تتجنب تعريف الكائن OleDbDataReader، يمكنك استخدام الوظيفة ExecuteScalar() المصاحبة للكائن SQLCommand أو OleDbCommand والتى تقوم بإرجاع متغير واحد فقط كما فى الكود التالى:

```
Private Function InsertWebUser(ByVal strUserId As String,
ByVal strPassword As String, ByVal strName As String, ByVal
intAccessLevel As Short) As Integer

    Dim con As New OleDbConnection("Provider= Microsoft.
      Jet.OLEDB.4.0; Data Source = d:\Students.mdb" )
    Dim cmd As OleDbCommand
    Dim intNewUserNumber As Integer

    con.Open()
    cmd = New OleDbCommand()
    cmd.Connection = con
    cmd.CommandType = CommandType.StoredProcedure
    cmd.CommandText = "spWebUserInsert"
    cmd.Parameters.Add(New OleDbParameter("@strUserID",
                                          strUserId))
    cmd.Parameters.Add(New OleDbParameter("@strPassword",
                                        strPassword))
    cmd.Parameters.Add(New
                    OleDbParameter("@strDescription", strName))
    cmd.Parameters.Add(New
            OleDbParameter("@intAccessLevel", intAccessLevel))
    intNewUserNumber = Convert.ToInt32(cmd.ExecuteScalar())

    cn.Close()
    Return intNewUserNumber

  End Function
```

يحتوى هذا الكود على الدالة InsertWebUser() التى تقوم باستخدام الوظيفة ExecuteScalar() بمصاحبة الإجراء المخزن لاسترجاع رقم المستخدم. يوضح سطر الكود التالى كيفية استدعاء هذه الدالة:

```
intUserNumber=InsertWebUser("bsiler","secret","Brian Siler",5)
```

- *استرجاع بيانات XML من خادم SQL*

قامت مايكروسوفت بتضمين معالجة XML بدءاً من SQL Server 2000، لذا فعن طريق إدخال بعض الكلمات الخاصة إلى عبارة SQL، يقوم خادم SQL بتحويل نتيجة الاستعلام إلى تنسيق XML كما فى العبارة التالية:

```
SELECT * FROM Students FOR XML AUTO, XMLDATA
```

حيث يستخدم التركيب FOR XML AUTO لإخبار خادم SQL بإرجاع السجلات بتنسيق XML، بينما تستخدم كلمة XMLDATA لإخبار الخادم أنك ترغب فى تضمين تعريفات الحقول بتنسيق XML أيضاً. وفى هذه الحالة تظهر السجلات الناتجة من الاستعلام بتنسيق XML وتكون شبيهة بالكود التالى:

```
<Students St_ID="1" St_fname="Mohamed" St_lname="Waleed"
St_Age = "16" St_Address = "13 Korba street" />
```

ولمعالجة نتائج XML بسهولة، يحتوى التصنيف SQLCommand على الوظيفة ExecuteXMLReader() التى تقوم بتفسير هذه البيانات للبرنامج من خلال إرجاع كائن من النوع XMLReader الذى يحتوى بدوره على مجموعة من الوظائف التى تمكنك من قراءة قيم وصفات XML كما فى الكود التالى:

```
cmd = New SqlCommand("SELECT * FROM Students FOR XML
                                    AUTO, XMLDATA", con)
Names = cmd.ExecuteXmlReader()
While Names.Read()
        If Names.NodeType = Xml.xmlNodeType.Element Then
                Names.MoveToAttribute("St_fname")
                If Names.HasValue Then
                        Debug.WriteLine("Current Student=" &
                                        Names.Value)
                End If
```

```
        End If
End While
```

وفى هذا الكود يتم طباعة جميع الأسماء الأولى داخل الجدول **Students** على نافذة الإخراج **Output**. وقد تم تصميم التصنيف **XMLReader** لمعالجة البيانات من خلال التنقل إلى الأمام فقط.